Constructions of advanced cryptographic primitives

Michael Reichle

PSL UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à l'École Normale Supérieure de Paris

Constructions of Advanced Cryptographic Primitives

Soutenue par
Michael Reichle
Le 9 octobre 2023

École doctorale n°386
Sciences Mathématiques de Paris Centre

Spécialité
Informatique

Composition du jury :

Dennis Hofheinz ETH Zürich	*Rapporteur*
Fabien Laguillaumie Université de Montpellier	*Rapporteur*
Michel Abdalla CNRS	*Directeur de thèse*
Brice Minaud Inria	*Co-encadrant de thèse*
Pierre-Alain Fouque Université Rennes 1	*Président du jury*
Lisa Kohl CWI Amsterdam	*Examinateur*
David Pointcheval CNRS	*Examinateur*

ENS ÉCOLE NORMALE SUPÉRIEURE

PSL RESEARCH UNIVERSITY PARIS

Résumé

Dans cette thèse, nous proposons des constructions efficaces de primitives cryptographiques avec des fonctionnalités avancées. Nous nous concentrons sur des primitives qui permettent des applications préservant la confidentialité, telles que la recherche sur des données chiffrées ou le vote électronique, avec une sécurité prouvable dans le modèle de l'oracle aléatoire (ROM). En particulier, nous construisons des schémas de *chiffrement avec recherche* (SSE) et des schémas de *signature aveugle*.

SSE permet à un client d'effectuer des requêtes par mots-clés sur une base de données chiffrée stockée sur un serveur distant. Pour obtenir le résultat, le serveur effectue souvent un grand nombre d'accès aléatoires à la mémoire. En conséquence, le débit mémoire d'un schéma SSE est souvent le principal goulot d'étranglement. Pour nos constructions, nous proposons d'abord des variantes de schémas de hachage classiques qui permettent l'allocation d'éléments pondérés. Sur la base de ces variantes, nous construisons plusieurs schémas SSE avec une bonne efficacité mémoire sur les supports de stockage modernes. Cela inclut Pluto, un schéma SSE statique avec une efficacité mémoire optimale, et Hermes, un schéma SSE dynamique avec une efficacité mémoire sous-logarithmique et sécurité persistante.

Les signatures aveugles servent d'outil fondamental pour les applications préservant la confidentialité (par exemple, le vote électronique, les jetons confidentiels d'authentification, les chaînes de blocs). Nous présentons deux cadres optimisés pour construire des signatures aveugles dans le ROM. Nous instancions chaque cadre dans le contexte des couplages et obtenons des signatures aveugles efficaces avec un nombre optimal de tours sous des hypothèses standards. La première construction est une variante hautement optimisée de la construction générique de signature aveugle de Fischlin (CRYPTO'06). Notre deuxième construction est une construction semi-générique à partir d'une classe spécifique de schémas de signature randomisables qui admet une réduction *tous-sauf-un*.

Mots clés : Chiffrement avec Recherche, Hachage Pondéré, Signature Aveugle

Abstract

In this thesis, we propose efficient constructions of cryptographic primitives with advanced functionalities. We focus on primitives that enable privacy-preserving applications, such as encrypted search or electronic voting, with provable security in the random oracle model (ROM). In particular, we construct *searchable symmetric encryption* (SSE) and *blind signature* schemes.

SSE allows a client to perform keyword queries on an encrypted database stored on a distant server. To obtain the result, the server often performs a large number of random memory accesses. In consequence, the memory throughput of an SSE scheme is often the main bottleneck. For our constructions, we first propose variants of classical hashing schemes that allow for allocation of weighted items. Based on these variants, we construct several SSE schemes with good memory efficiency on modern storage media. This includes Pluto, a static SSE scheme with optimal memory efficiency, and Hermes, a dynamic SSE scheme with sublogarithmic memory efficiency and forward security.

Blind signatures serve as a foundational tool for privacy-preserving applications (e.g., electronic voting, privacy-authentication tokens, blockchains). We present two optimized frameworks to construct blind signatures in the ROM. We instantiate each framework in the pairing setting and obtain efficient round-optimal blind signatures under standard assumptions. The first construction is a highly optimized variant of the generic blind signature construction by Fischlin (CRYPTO'06). Our second construction is a semi-generic construction from a specific class of randomizable signature schemes that admits an *all-but-one* reduction.

Keywords: Searchable Symmetric Encryption, Weighted Hashing, Blind Signature

Acknowledgments

First, I want to thank Brice Minaud. The last three years of PhD under your guidance were a great adventure. I am grateful that you always took the time to discuss any subject, answer any question, and offer your help whenever I asked. Though I am far from your technical expertise, I hope that someday mine will be within some ε-environment of yours [1]. I never (visibly) managed to drain your limitless patience despite countless weird ideas, poorly worded questions, and frustrations after rejected papers (though I wonder if I ever came close at times). Also, thank you for all your other good deeds that make this lab a great place to be: the reading group, secure votes for beverage-related articles, fun evenings at bars, and much more.

Next, I want to thank Geoffroy Couteau. You introduced me to the world of research and offered your guidance and support since we first met. The first project with you showed me how to develop vague approaches, write papers, and deal with spooky deadlines. Thanks to you I had the opportunity to go to my first conference (thanks to Dennis, Jiaxin, Lisa, and Roman on that note), and our following projects and conversations taught me much more. Your vast knowledge in any subject imaginable is inspiring, and thanks to your contagious passion and encouragement, I can call myself a cryptographer today.

Also, I want to thank Shuichi Katsumata. You took the time to discuss and improve my (often naive) ideas, helped me clean up my writing, and (admittedly) shocked me with the speed at which conjure pages of technical content and convincing introductions. Thank you for the opportunity to visit AIST in Tokyo, and thanks to Takahiro Matsuda for all the organization.

Further, I want to thank Dennis Hofheinz for introducing me to cryptography in your fantastic lecture at KIT, for inviting me to visit your team, and for accepting me as a postdoc. I am looking forward to discussing exciting topics with you and your great team.

Next, I want to thank the students and postdocs at ENS with whom I had the chance to spend the last three years: Antoine, Balthazar, Baptiste, Guirec, Hanyu, Henry, Hugo, Huy, Jianwei, Lénaïck, Nicolas, Paola, Paul, Quentin, Robert, Théo, Vincent, Xiaxi and Yann. Especially, thanks to Léonard (for the concert, fun discussions, and our workout sessions) and Ky (for our discussions on blind signatures, and tea breaks during late afternoons).

Also, thanks to Anca, Chloé, Melissa, Michele, and Romain for interesting discussions (and help in reestablishing customs forgotten during Covid). Further, thanks to the permanents members Céline Chevalier, David Pointcheval and Phong Nguyen, and also Michel Abdalla, Duong-Hieu Phan, Jaques Stern, and Hoeteck Wee, for interesting discussions at ENS or at conferences. Special thanks to David for leading the group in a kind and supportive manner, and Michel for supervising my PhD. Also, I thank the administrative team at ENS and Inria that supported me throughout the last three years, especially Meriem, Nathalie, Diana and Lise-Marie.

I also want to thank everyone I met at conferences (or prior at KIT), such as Benedikt, Calvin, Clément, Dung, Jesko, Julia, Pierre, Rachit, Roman, and many more. Also, special thanks to my coauthors: Angèle Bossuat, Raphael Bost, Geoffroy Couteau, Pierre-Alain Fouque, Dahmun Goudarzi, Shuichi Katsumata, Michael Klooß, Huang Lin, Brice Minaud, and Yusuke Sakai. Especially, thanks to Michael for guiding me during my Master thesis (and beyond), and Raphael

[1] I fear that $\varepsilon = \mathsf{poly}(\lambda)$, where λ denotes the insecurity parameter.

for interesting discussions on SSE. Also, I thank Dennis Hofheinz and Fabien Laguillaumie for agreeing to review my thesis, and Michel Abdalla, Lisa Kohl, Pierre-Alain Fouque, Brice Minaud, and David Pointcheval for agreeing to be on the jury.

Before I forget, thank you to everyone that I forgot. You were forgotten [2] but appreciated nonetheless. Before closing this section, I would like to thank everyone that supported me outside of the academic realm throughout this adventure: my mother and father, my brothers Andreas and Johannes, and my friends from Munich, Karlsruhe, Grenoble, and Paris. Finally, thanks to Dodo for your support and for sharing almost all the ups and downs of the last three years.

[2] I am sorry.

Contents

Résumé i

Abstract iii

Acknowledgments v

1 Introduction en Français 1
- 1.1 Cryptographie . . . 1
- 1.2 Confidentialité . . . 2
- 1.3 Chiffrement avec recherche . . . 3
  - 1.3.1 Nos Contributions . . . 5
- 1.4 Signatures aveugles . . . 7
  - 1.4.1 Nos Contributions . . . 9
- 1.5 Publications associées et autres contributions . . . 9
- 1.6 Organisation du Manuscrit . . . 11

2 Introduction 13
- 2.1 Cryptography . . . 13
- 2.2 Privacy . . . 14
- 2.3 Searchable Symmetric Encryption . . . 15
  - 2.3.1 Our Contributions . . . 17
- 2.4 Blind Signatures . . . 18
  - 2.4.1 Our Contributions . . . 19
- 2.5 Associated Publications and other Contributions . . . 20
- 2.6 Organization of the Manuscript . . . 22

3 Preliminaries 23
- 3.1 Notation . . . 23
- 3.2 Probability . . . 24
- 3.3 Graphs . . . 25
- 3.4 Assumptions . . . 26
  - 3.4.1 Groups . . . 26
- 3.5 Cryptographic Primitives . . . 27
  - 3.5.1 Symmetric Primitives . . . 27
  - 3.5.2 Asymmetric Primitives . . . 28
- 3.6 Searchable Symmetric Encryption . . . 34

4 Weighted Hashing 39
- 4.1 Introduction . . . 39
  - 4.1.1 Our Contributions . . . 41
- 4.2 One-Choice Allocation . . . 42
- 4.3 Two-Choice Allocation . . . 43

4.3.1 Overview . . . 43
4.3.2 Analysis . . . 45
4.4 Cuckoo Hashing . . . 49
4.4.1 Overview . . . 49
4.4.2 Analysis . . . 51

5 Page Efficiency and Constructions **61**
5.1 Introduction . . . 61
5.1.1 Our Contributions . . . 62
5.2 Page Efficiency . . . 63
5.3 Pluto . . . 64
5.3.1 Construction . . . 64
5.3.2 Security . . . 65
5.3.3 Efficiency . . . 67
5.4 LayeredSSE . . . 67
5.4.1 Construction . . . 68
5.4.2 Security . . . 69
5.4.3 Extensions . . . 70
5.4.4 Efficiency . . . 72

6 Forward Security and Page Efficiency **73**
6.1 Introduction . . . 73
6.1.1 Our Contributions . . . 75
6.1.2 Technical Overview . . . 76
6.2 SSE with Dummy Updates . . . 78
6.2.1 Security Definition . . . 78
6.2.2 A Framework to Build SSE with Dummy Updates . . . 78
6.2.3 Efficient Instantiations . . . 82
6.3 BigHermes: the Big Database Regime . . . 83
6.3.1 $\mathsf{BigHermes}_0$: Amortized BigHermes . . . 83
6.3.2 $\mathsf{BigHermes}_1$: BigHermes with Deamortized Communication . . . 84
6.3.3 $\mathsf{BigHermes}_2$: Fully Deamortized BigHermes . . . 85
6.3.4 Security . . . 86
6.3.5 Efficiency . . . 90
6.4 SmallHermes: the Small Database Regime . . . 91
6.4.1 $\mathsf{SmallHermes}_0$: Amortized SmallHermes . . . 91
6.4.2 $\mathsf{SmallHermes}_1$: SmallHermes with Deamortized Communication . . . 92
6.4.3 $\mathsf{SmallHermes}_2$: Fully Deamortized SmallHermes . . . 94
6.4.4 Heuristic Variant via Weighted 2C . . . 95
6.4.5 Security . . . 96
6.4.6 Efficiency . . . 97
6.5 The Hermes Scheme: Putting Everything Together . . . 97
6.5.1 Hermes . . . 98
6.5.2 Optimizations and Trade-offs . . . 98

7 Practical Round-Optimal Blind Signatures in the ROM **101**
7.1 Introduction . . . 101
7.1.1 Contributions . . . 103
7.1.2 Technical Overview . . . 104
7.2 Optimizing the Fischlin Blind Signature . . . 111
7.2.1 Construction . . . 111
7.2.2 Correctness and Security . . . 112

7.3 Instantiation of the Generic Construction . . . 118
7.3.1 Optimizations and Efficiency . . . 124
7.4 Blind Signatures based on Boneh-Boyen Signature . . . 124
7.4.1 Construction . . . 124
7.4.2 Correctness and Security . . . 125
7.5 Instantiation of the Framework based on Boneh-Boyen . . . 129
7.6 Frameworks for Partially Blind Signatures . . . 143
7.6.1 Partial Blindness of the Optimized Fischlin Transform . . . 143
7.6.2 Partial Blindness of Blind Signature based on Boneh-Boyen . . . 145

8 Conclusion and Open Questions **151**

A Notation **153**

B Abbreviations **155**

Chapter

1 Introduction en Français

Dans ce chapitre, nous présentons un aperçu de nos contributions et des travaux connexes. Nous donnons également un bref aperçu de l'ensemble de nos travaux et de l'organisation de ce manuscrit.

Chapter content

1.1 Cryptographie 1
1.2 Confidentialité 2
1.3 Chiffrement avec recherche 3
1.3.1 Nos Contributions 5
1.4 Signatures aveugles 7
1.4.1 Nos Contributions 9
1.5 Publications associées et autres contributions 9
1.6 Organisation du Manuscrit 11

1.1 Cryptographie

La cryptographie fournit des outils pour une communication sécurisée, souvent sous la forme de primitives cryptographiques modulaires. La primitive la plus importante est peut-être le chiffrement symétrique, où deux utilisateurs Alice et Bob partagent une clé secrète commune K et cherchent à échanger des messages privés de manière sécurisée, c'est-à-dire qu'aucun tiers (souvent appelé « adversaire ») ne doit apprendre d'informations sur les messages privés sans connaître K. Pour ce faire, Alice chiffre son message m en utilisant le clé secrète K et un algorithme de chiffrement Enc via $c \leftarrow \mathsf{Enc}(\mathsf{K}, m)$. Ensuite, Bob obtient le texte chiffré c et peut récupérer $m \leftarrow \mathsf{Dec}(\mathsf{K}, c)$ à l'aide d'un algorithme de déchiffrement Dec et de la clé K. Nous appelons souvent une instantiation d'une telle primitive un schéma. Même si l'adversaire ne peut pas exécuter l'algorithme de déchiffrement Dec sans K, il peut y avoir un autre moyen de récupérer (des parties de) le message m à partir du texte chiffré c. Ainsi, les schémas sont analysés cryptographiquement pour gagner en confiance dans leur sécurité : s'il n'existe aucune attaque faisable, il est raisonnable de supposer qu'il est sécurisé. Les exigences que nous attendons d'un schéma sécurisé sont formalisées dans des notions de sécurité, par exemple, il devrait être difficile de distinguer si un texte chiffré donné chiffre soit m_0 soit m_1, même si l'adversaire peut choisir les deux messages à sa guise. Ces notions de sécurité sont formalisées dans des jeux de sécurité, où un challenger interagit avec un adversaire (qui cherche à gagner le jeu). Si un adversaire parvient à gagner le jeu, il brise la sécurité du schéma. Ce cadre nous permet de classer les attaques, voire de prouver mathématiquement qu'un schéma est (in)sécurisé pour une notion donnée de sécurité.

Bien qu'il existe des schémas de chiffrement symétrique avec une sécurité parfaite [Sha49], c'est-à-dire que le schéma est impossible à briser même pour un adversaire ayant une puissance de calcul illimitée, ces schémas sont impraticables en raison des grandes clés secrètes qui augmentent

avec la longueur du message. Il est donc souhaitable de construire des schémas efficaces au détriment de la sécurité parfaite. En pratique, la sécurité contre les adversaires ayant une puissance de calcul limitée est suffisante.

En 1973, IBM a introduit le Data Encryption Standard (DES)—un schéma de chiffrement symétrique efficace conçu pour une utilisation pratique [oSN77]. Bien qu'il ait été démontré qu'il était non sécurisé [BS93], les principes de sa conception guident la construction des primitives modernes. De nos jours, il existe des schémas de chiffrement symétrique efficaces largement considérés comme sécurisés, tels que AES [oSN01]. D'autres primitives symétriques avec des fonctionnalités différentes sont également proposées. Un exemple important est celui des *fonctions de hachage (avec clé)*, qui créent un empreinte courte à partir de messages longs, et la sécurité garantit qu'il est difficile de trouver deux messages distincts ayant la même empreinte. Un autre exemple sont les *Message Authentication Codes* (MAC), qui peuvent créer des étiquettes sur des messages à l'aide d'une clé privée K, et la sécurité garantit qu'il est impossible de falsifier une étiquette pour des messages sans connaître K. Les MAC permettent par exemple l'authentification des utilisateurs, c'est-à-dire qu'Alice peut vérifier si elle communique avec Bob et peut garantir l'intégrité des données reçues. Cependant, il reste un inconvénient : Alice et Bob ont besoin d'une clé secrète partagée K. Pour permettre une communication sans configuration fastidieuse des clés K, une nouvelle approche est nécessaire.

Les fondements de cette direction ont été posés dans le travail fondamental de Diffie et Hellman [DH76] dans lequel ils introduisent la cryptographie *asymétrique*. Dans le chiffrement asymétrique, les deux parties communicantes n'ont pas besoin de partager une clé secrète commune K, mais Bob peut chiffrer des messages pour Alice avec la clé publique pk_A d'Alice. Alice peut ensuite déchiffrer de tels textes chiffrés avec sa clé secrète sk_A, et la sécurité garantit que les textes chiffrés ne divulguent aucune information sur le message à un adversaire n'ayant pas accès à sk_A. Cette approche nécessite (au moins) l'existence de fonctions à sens unique, c'est-à-dire de fonctions faciles à calculer mais difficiles à inverser, dont l'existence implique $\mathsf{P} \neq \mathsf{NP}$. À ce jour, la question de savoir si $\mathsf{P} \neq \mathsf{NP}$ reste non résolue, et les cryptographes s'appuient sur des hypothèses de complexité calculatoire. Ces hypothèses indiquent qu'un problème est difficile à résoudre pour un algorithme limité en puissance de calcul. Ensuite, les cryptographes construisent des schémas et démontrent qu'ils sont sécurisés sous ces hypothèses. Cette approche permet de créer divers schémas cryptographiques avec une sécurité démontrable selon certaines hypothèses. D'autres primitives comprennent l'échange de clés, qui établit une clé secrète commune entre Alice et Bob pour permettre une communication ultérieure sécurisée avec des primitives symétriques (plus efficaces), ou les signatures numériques qui sont l'équivalent asymétrique des MACs.

1.2 Confidentialité

Un objectif important de la cryptographie est de préserver la confidentialité des utilisateurs. Lorsqu'un utilisateur interagit avec un système numérique, il divulgue souvent des informations personnelles au système. Par exemple, si Alice stocke ses notes personnelles sur le serveur d'un fournisseur de services cloud, celui-ci pourrait connaître le contenu de ses notes. De même, si Bob vote en ligne pour son candidat préféré, le contenu de son bulletin de vote pourrait être divulgué au système de vote. Ces informations privées pourraient être vendues à des tiers, mais même si le fournisseur de services lui-même traite les informations privées avec soin et de bonnes intentions, de telles informations privées pourraient être récupérées par un adversaire ayant compromis le fournisseur, avec de graves conséquences. En raison de l'importance de la protection des données personnelles, celle-ci est requise par la loi dans l'Union Européenne conformément au règlement général sur la protection des données (RGPD). Le RGPD exige, par exemple, que les données personnelles soient stockées de manière confidentielle et que le système réduise au minimum la quantité d'informations personnelles qu'il traite. La cryptographie fournit les outils

nécessaires pour garantir cela : le chiffrement permet le stockage de données confidentielles, et les signatures numériques (ou les MAC) permettent l'authentification des utilisateurs qui sont autorisés à accéder aux informations privées.

Fonctionnalités plus avancées. Alors que les primitives cryptographiques de base mentionnées précédemment fournissent certains outils pour garantir la confidentialité, les systèmes complexes du monde réel nécessitent plus que simplement le chiffrement et les signatures. Par exemple, lors du chiffrement de toutes les données stockées par Alice sur le cloud, il devient difficile d'interagir avec les données ultérieurement. Alice peut souhaiter rechercher des notes contenant le mot-clé *cryptographie*, mais si les données sont chiffrées, elle doit récupérer toutes les notes et les parcourir elle-même pour les rechercher. De même, si Bob chiffre son vote et l'envoie au système de vote, celui-ci ne peut pas calculer le résultat de l'élection sans être en mesure de déchiffrer le texte chiffré. Cela pourrait conduire à l'idée que la cryptographie assure des propriétés de sécurité souhaitables mais entrave apparemment la fonctionnalité.

Cependant, Alice et Bob ne sont pas limités à l'utilisation du chiffrement et des signatures : la cryptographie moderne offre de nombreux autres outils. Des systèmes plus complexes, tels que la recherche sur des données chiffrées ou le vote électronique (comme esquissé ci-dessus), nécessitent des primitives cryptographiques avancées avec des fonctionnalités plus riches. Ces primitives sont conçues pour avoir deux propriétés importantes : l'efficacité et la sécurité. Une efficacité concrète est souhaitable car les schémas sont déployés dans des applications du monde réel. De même, une sécurité démontrable est nécessaire pour garantir que les garanties de sécurité souhaitées du système sont respectées, idéalement en se basant sur des hypothèses bien établies. L'objectif de cette thèse est la conception de telles primitives avec un accent sur l'efficacité et la sécurité basées sur des hypothèses standards. Nous présentons des constructions efficaces et démontrablement sécurisées de *chiffrement avec recherche* et de *signatures aveugles* qui sont utilisées, par exemple, dans le contexte de la recherche sur des données chiffrées et du vote électronique.

1.3 Chiffrement avec recherche

Les bases de données chiffrées sont une proposition attrayante. Une entreprise ou un hôpital peut souhaiter externaliser sa base de données clients pour une disponibilité, une évolutivité ou une persistance accrue, sans confier les données en clair à un service externe. Un service de messagerie chiffrée de bout en bout peut souhaiter stocker et rechercher des messages d'utilisateurs sans les déchiffrer. Dans une autre direction, même si une base de données sensible est stockée localement, une entreprise peut souhaiter la conserver chiffrée afin de fournir une protection supplémentaire contre les attaques et le vol de données. L'adoption par MongoDB de techniques de chiffrement avec recherche est une autre illustration récente de la demande croissante de bases de données chiffrées [Mon22]. Lors de l'externalisation du stockage d'une base de données chiffrée, une fonctionnalité minimale souhaitable est la capacité d'effectuer des recherches dans les données.

Chiffrement avec recherche. La promesse du chiffrement avec recherche (SSE) est de permettre à un client d'externaliser une base de données chiffrée de taille N vers un serveur non fiable, tout en conservant la capacité d'effectuer des recherches sur les données [SWP00]. Au minimum, le client est en mesure d'envoyer une requête de recherche pour récupérer tous les identifiants de documents correspondant à un mot-clé donné. Dans le cas du SSE *dynamique* (DSSE), le client est également en mesure de modifier le contenu de la base de données en envoyant des requêtes de mise à jour, par exemple pour insérer ou supprimer des entrées. Le serveur doit être capable de traiter correctement les requêtes, tout en apprenant le moins d'information possible sur les données et les requêtes du client.

Solutions avec une fuite minimale. Il existe des solutions bien étudiées à ce problème et nous en donnons un bref aperçu ci-dessous.

La récupération privée d'informations (PIR) a été introduite dans [CGKS95]. En résumé, le PIR permet à un utilisateur de récupérer la i-ème entrée d'une base de données de N bits détenue par un ou plusieurs serveurs sans révéler quelle position a été consultée au(x) serveur(s). Avec le PIR, le client peut chiffrer sa base de données et l'externaliser vers le serveur. Ensuite, il peut récupérer l'élément souhaité sans divulguer aucune information au serveur. Malheureusement, cette exigence de confidentialité élevée entraîne une surcharge de calcul et/ou de stockage côté client [BIM04, CK20, Yeo23], même si une indication privée est stockée par le client pour améliorer l'efficacité.

La mémoire RAM inconsciente (ORAM) permet à un client d'exécuter un programme sur un serveur sans révéler les motifs d'accès à la mémoire [GO96]. En raison de ses solides garanties de confidentialité, il est souvent utilisé dans la construction du chiffrement avec recherche (e.g., [GMP16, KMO18, AM23]). Comme dans le cas du PIR, l'utilisation de l'ORAM entraîne une surcharge importante en bande passante [GO96, LN18]. Ainsi, son utilisation est souvent réduite aux parties du protocole de recherche chiffrée complet (e.g., [MM17, DCPP20]) au prix d'une fuite supplémentaire.

Le chiffrement entièrement homomorphe (FHE) permet l'exécution de circuits arbitraires sur des données chiffrées [Gen09]. La recherche par mot-clé dans une base de données peut être modélisée comme un circuit de taille linéaire par rapport à la taille de la base de données. Comme l'évaluation des circuits sur des données chiffrées est proportionnelle à la taille du circuit, cette approche entraîne une surcharge de calcul importante du côté du serveur. Récemment, un travail passionnant a construit un chiffrement entièrement homomorphe pour les calculs en mémoire RAM [LMW22], ce qui permet l'exécution de programmes qui évolue avec la complexité du programme RAM associé. Malheureusement, cette approche est actuellement peu pratique.

Compromis entre fuite d'information et sécurité. Par rapport aux solutions mentionnées précédemment, une spécificité de la littérature sur le SSE est l'accent mis sur des solutions à haute performance, adaptées au déploiement sur de grands ensembles de données du monde réel. Des schémas de SSE efficaces sont conçus pour divulguer certaines informations sur la base de données chiffrée et les requêtes. La fuite d'informations d'un schéma est généralement exprimée par une fonction de fuite. La preuve de sécurité accompagnant un schéma de SSE fournit des garanties formelles concernant les informations divulguées au serveur lors des recherches et des mises à jour. Les approches historiques efficaces basées sur le chiffrement déterministe ou le chiffrement préservant l'ordre [BCLO09] sont sujettes à de graves attaques en raison de la grande quantité d'informations divulguées au serveur [NKW15, GLMP19]. Face à cette situation, la recherche moderne sur le chiffrement avec recherche cherche à offrir des compromis raisonnables entre performances, fonctionnalités et sécurité, adaptés au déploiement dans le monde réel. Généralement, les informations divulguées incluent la taille totale de la base de données, la répétition des requêtes et un identifiant (tel que l'adresse mémoire) des documents correspondant à une requête.

Une propriété importante d'une fonction de fuite est la sécurité persistante. La sécurité persistante exige que les mises à jour ne divulguent aucune information sur le mot-clé mis à jour au serveur [SPS14]. La motivation de la sécurité persistante est qu'elle atténue certaines attaques : les attaques les plus graves de [ZKP16] exploitent la fuite des mises à jour, et échouent sur les schémas avec sécurité persistante.

Orientations. Depuis que le SSE a été introduit par Song *et al.* [SWP00], le domaine s'est développé dans plusieurs directions différentes. En raison de l'ampleur de la littérature sur le SSE, nous mettons en évidence quelques branches importantes. Dans la mesure où le SSE

représente un compromis entre fonctionnalité, sécurité et efficacité, la recherche dans ce domaine peut être grossièrement divisée en trois volets, correspondant à chaque composant du compromis.

Alors que les schémas de SSE sont souvent conçus pour permettre des requêtes de mots-clés efficaces, certains travaux étendent la fonctionnalité fournie aux requêtes booléennes [CJJ+13], aux requêtes de plage [PKV+14] ou même à des sous-ensembles de SQL [KM18].

Les travaux portant sur la sécurité incluent les attaques et les efforts visant à réduire la fuite d'informations en réponse à ces attaques. La plupart des attaques contre le chiffrement avec recherche relèvent de la catégorie des attaques par *abus de fuite*, un terme inventé dans [CGPR15]. Les attaques par abus de fuite ne contredisent pas les garanties de sécurité d'un schéma, mais montrent comment la fuite d'informations autorisée par le modèle de sécurité permet au serveur de reconstruire de grandes parties de la base de données dans certains contextes [CGPR15, GLMP19]. Ces attaques ont motivé d'autres travaux visant à réduire ou à supprimer la fuite d'informations [GKM21], y compris des constructions avec sécurité persistante [PM21, BMO17, KMPQ21, EKPE18, DCPP22].

Parmi les travaux qui visent principalement l'efficacité, le développement le plus notable de ces dernières années concerne l'efficacité en entrée/sortie (E/S).

Efficacité E/S. Pour des raisons de performance, la plupart des conceptions de SSE s'appuient exclusivement sur des primitives cryptographiques symétriques qui ont un faible surcoût de calcul en pratique. Par conséquent, le principal goulot d'étranglement de performance est déterminé par la rapidité à laquelle les données peuvent être accédées sur le disque [CT14]. Cela a été formalisé sous la notion de localité et d'efficacité de lecture. La localité demande que les données correlées soient stockées dans un petit nombre d'emplacements disjoints sur le disque, et l'efficacité de lecture demande que les surcoûts de lecture des données soient faibles. Cette notion est motivée par le comportement d'E/S des disques durs (HDD), où les lectures disjointes sont beaucoup plus coûteuses en termes de latence et de débit que les lectures contiguës.

La nécessité de stocker des données correlées à proximité n'est pas du tout anodine pour la sécurité. Demander que les éléments de données correlées soient stockés à proximité crée une corrélation entre l'*emplacement* d'un élément de données chiffrées en mémoire et son *contenu*. Étant donné que le serveur peut observer l'emplacement des données qu'il est chargé de récupérer, et que nous ne voulons pas que le serveur déduise des informations sur le contenu de ces données, cela crée une tension entre la sécurité et l'efficacité. Autrement dit, la sécurité demande qu'il n'y ait aucune corrélation entre l'emplacement des données et leur contenu, tandis que l'efficacité E/S demande le contraire.

Cette tension a été exprimée dans un résultat d'impossibilité par Cash et Tessaro à Eurocrypt 2014 [CT14]. En bref, Cash et Tessaro montrent qu'un schéma SSE sécurisé avec un stockage linéaire du serveur ne peut pas avoir à la fois une localité constante et une efficacité de lecture constante. Cela est vrai même pour le SSE statique. Dans un autre travail phare, lors du STOC 2016, Asharov *et al.* construisent un SSE avec une localité constante et une efficacité de lecture en $\widetilde{\mathcal{O}}(\log N)$ - voire $\widetilde{\mathcal{O}}(\log \log N)$ avec une légère restriction sur la base de données en entrée [ANSS16]. Leur construction met en évidence une connexion profonde entre le hachage pondéré et le chiffrement avec recherche local. Depuis l'introduction de l'efficacité mémoire pour le SSE, de nombreuses constructions avec des accès mémoire efficaces ont été proposées [CT14, ANSS16, ASS21, DP17a, MM17, DPP18].

1.3.1 Nos Contributions

Dans cette thèse, nous introduisons une mesure différente de l'efficacité E/S, appelée *efficacité des pages*. L'efficacité des pages est le rapport entre le nombre de pages lues par le serveur pour traiter une requête et le nombre de pages nécessaires pour contenir la réponse en texte clair. Alors que la localité et l'efficacité de lecture capturent l'efficacité des schémas SSE lorsqu'ils

sont exécutés sur des disques durs (HDD), notre notion est motivée par le comportement d'E/S des disques à semi-conducteurs modernes (SSD). Nous construisons ensuite plusieurs schémas avec une bonne efficacité des pages dans le cadre introduit dans [ANSS16] basé sur le hachage pondéré.

Hachage pondéré. L'efficacité des pages consiste à stocker des identifiants liés dans un petit nombre de pages. Il est naturel de considérer les pages comme des compartiments et les listes d'identifiants correspondant à un seul mot-clé comme des balles d'un poids proportionnel à la taille de la liste. Nous nous intéressons ensuite à une borne supérieure sur le poids total lorsque n balles d'un poids total w_{tot} sont réparties aléatoirement dans $\mathcal{O}(n)$ compartiments. Une meilleure borne supérieure permet ensuite la construction de schémas SSE plus efficaces. Nous analysons des variantes *pondérées* des problèmes classiques de répartition de balles dans des compartiments non pondérés suivants :

- Dans l'allocation à choix unique (1C), chaque balle est insérée dans un seul compartiment choisi au hasard. Le compartiment le plus chargé contient au plus $\mathcal{O}(\log N)$ balles [JK77].
- Dans l'allocation à choix double (2C), chaque balle est insérée dans le compartiment le moins chargé parmi deux compartiments choisis au hasard. Le compartiment le plus chargé contient au plus $\mathcal{O}(\log \log n)$ balles [ABKU94].
- Dans le hachage coucou avec réserve [PR04, KMW10], chaque balle est insérée dans l'un des deux compartiments choisis au hasard ou dans la réserve. Après une procédure d'optimisation, chaque compartiment contient au plus une balle et la taille de la réserve est minimale.

Une analyse pondérée de 1C est donnée en espérance dans [BFHM08], et nous incluons une borne supérieure qui tient avec une probabilité écrasante pour plus de clarté. Notre variante de 2C est une généralisation de l'allocation pondérée à choix double appelée L2C, conçue pour la preuve de borne supérieure. Enfin, notre variante du hachage coucou repose sur un algorithme de flot maximum pour l'optimisation globale. Toutes nos bornes dans le cas pondéré correspondent asymptotiquement aux bornes dans le cas classique avec des poids uniformes et sont valables avec une probabilité écrasante.

Chiffrement avec recherche à efficacité des pages. Ensuite, nous construisons plusieurs schémas (D)SSE avec une bonne efficacité des pages basés sur nos variantes de hachage pondéré :

- Pluto : Du point de vue théorique, l'efficacité des pages constantes est une exigence moins contraignante que la combinaison de la localité constante et de l'efficacité de lecture constante. Fait intéressant, cette exigence moins contraignante contourne le résultat d'impossibilité de Cash et Tessaro : nous construisons un schéma SSE statique avec une efficacité des pages optimale appelé Pluto. La configuration nécessite un calcul de flot maximum sur l'ensemble des données, ce qui rend le schéma inefficace pour les mises à jour.
- LayeredSSE : Nous utilisons notre généralisation pondérée de l'allocation à choix double L2C pour construire un schéma SSE dynamique avec une efficacité des pages de $\widetilde{\mathcal{O}}(\log \log N/p)$. Le schéma révèle le schéma de requête pendant les mises à jour et n'a pas de sécurité persistante. Néanmoins, c'est le premier schéma dynamique avec une efficacité des pages sous-logarithmique.
- Hermes : Enfin, nous construisons un schéma DSSE avec sécurité persistante et une efficacité des pages de $\widetilde{\mathcal{O}}(\log \log N/p)$. Le schéma est basé sur LayeredSSE et des techniques nouvelles telles que le tamponnement contrôlé du client et les mises à jour factices.

1.4 Signatures aveugles

Les signatures aveugles ont été introduites dans [Cha82] et améliorent la fonctionnalité des signatures numériques en fournissant des garanties de confidentialité supplémentaires. Alice et Bob s'engagent dans un protocole interactif à la fin duquel Alice obtient une signature sur un message de son choix signé par Bob. Les propriétés requises sont l'aveuglement et l'incapacité de forger une signature supplémentaire. L'aveuglement stipule que si Alice présente ultérieurement la signature à Bob, celui-ci ne peut pas relier la signature à une session de signature spécifique. En particulier, Bob ne connaît pas le message signé pendant la session de signature. L'incapacité de forger une signature supplémentaire signifie qu'Alice peut obtenir des signatures pour au plus ℓ messages distincts à partir de ℓ sessions de signature.

Les signatures aveugles sont un élément fondamental dans de nombreuses applications préservant la confidentialité, telles que les e-cash [Cha82, CFN90, OO92], les attributs anonymes [Bra94, CL01], les votes électroniques [Cha88, FOO92], l'attestation anonyme directe [BCC04], les blockchains [YL19, BDE+22] et les jetons d'authentification préservant la confidentialité [VPN22, HIP+22]. Depuis l'introduction des signatures aveugles dans [Cha82], elles sont devenues un vaste domaine de recherche. Nous donnons d'abord un bref aperçu des orientations de recherche sur les signatures aveugles.

Signatures aveugles à plusieurs tours

Dans le modèle de l'oracle aléatoire (ROM), il existe des signatures aveugles en 3 étapes efficaces [Sch90, Oka93, AO00]. Il est important de noter que dans le ROM, l'adversaire a accès à un oracle représentant une fonction aléatoire idéalisée H. En pratique, H est souvent instantié de manière heuristique avec une fonction de hachage résistante aux collisions. Dans leur travail fondateur, Pointcheval et Stern [PS00] montrent que [Oka93] est sécurisé pour au plus un nombre logarithmique de sessions de signature. Leur analyse a été généralisée et affinée dans [HKL19], et un résultat similaire a été montré pour [AO00] dans [KLX22a]. Malheureusement, une attaque récente [BLL+21] montre que la borne sur les sessions de signature concurrentes est atteinte.

Cette approche a ensuite été affinée, et il existe plusieurs signatures aveugles en 3 étapes [Abe01, KLX22b, FPS20, TZ22] qui sont prouvées sûres dans le modèle générique de groupe (GGM) ou dans le modèle algébrique de groupe (AGM), ainsi que dans le ROM. Dans le GGM et l'AGM, l'adversaire est restreint à effectuer des attaques algébriques, c'est-à-dire qu'il utilise le groupe en boîte noire. Ces constructions évitent l'attaque décrite dans [BLL+21] et peuvent être instantiées de manière efficace en pratique. Notamment, [TZ22, KLX22b] offrent une sécurité concurrente complète. Ces modèles sont moins souhaitables d'un point de vue théorique car ils restreignent considérablement les capacités de l'adversaire. Dans le ROM, [BL13] montre la sécurité séquentielle de [Abe01] sans groupes génériques, mais le signataire doit veiller à ce qu'au plus une session de signature soit ouverte à tout moment.

Transformations d'amélioration. Une récente ligne de recherche [KLR21, CAHL+22] basée sur [Poi98] propose des transformations d'amélioration génériques pour les signatures aveugles avec une sécurité concurrentielle limitée. Malheureusement, l'efficacité calculatoire augmente linéairement avec le nombre de signatures effectuées. Récemment, [HLW23] propose une construction concrète basée sur les couplages dans le ROM avec une bonne efficacité qui est optimale en nombre de tours (cf. ci-dessous).

Signatures aveugles optimal en nombre de tours

Une propriété souhaitable pour les signatures aveugles est de pouvoir réaliser une session de signature en deux tours, c'est-à-dire que deux messages sont échangés pour obtenir une signature.

Il existe plusieurs résultats d'impossibilité [Lin08, FS10, Pas11] qui indiquent qu'une configuration de confiance, un modèle idéalisé ou des hypothèses non standards sont nécessaires.

Exploitation de la complexité et hypothèses interactives. Il existe des signatures aveugles optimales en termes de tours [GRS+11, GG14] qui contournent les résultats d'impossibilité pour construire des signatures aveugles dans le modèle standard en utilisant l'*exploitation de la complexité* (« complexity leveraging » en anglais). [KNYY21] parvient à éviter l'exploitation de la complexité en s'appuyant à la fois sur des hypothèses post-quantiques et des hypothèses classiques. Les schémas mentionnés ci-dessus sont principalement d'intérêt théorique et ne sont pas pratiques.

Chaum [Cha82] donne une construction simple basée sur la signature RSA classique, qui a ensuite été démontrée sécurisée dans [BNPS03] sous une hypothèse One-more RSA et dans le ROM. Plus tard, [Bol03, AKSY22] proposent des schémas similaires dans des contextes différents, sécurisés sous une hypothèse one-more comparable dans le ROM. De plus, il existe des constructions dans le modèle standard [AFG+10, FHS15, FHKS16, Gha17] qui reposent sur des hypothèses interactives adaptées en termes de difficulté. Les constructions mentionnées ci-dessus sont efficaces mais reposent sur des hypothèses de sécurité interactives. Cela est acceptable d'un point de vue pratique car les hypothèses ne sont pas encore compromises. Mais d'un point de vue théorique, cela est moins satisfaisant : les hypothèses interactives ne sont pas falsifiables et donc moins souhaitables [Nao03].

Configuration de confiance. Dans le cadre des couplages, il existe plusieurs signatures aveugles optimales en termes de tours avec une configuration de confiance [MSF10, SC12, KSD19] dans le modèle standard. Encore une fois, ces schémas sont moins pratiques et la configuration de confiance est *structurée*, c'est-à-dire non uniforme. Cela entrave le déploiement pratique car une trappe peut être intégrée dans les paramètres du schéma. Il existe d'autres schémas [BFPV13, AJOR18] avec une configuration uniforme que nous discutons ci-dessous.

Constructions dans le ROM. Il existe quelques constructions efficaces sécurisées dans le ROM avec une sécurité concurrentielle complète. Nous en donnons un bref aperçu.

Dans le cadre des réseaux de lattice, del Pino et Katsumata [dK22] proposent une signature aveugle optimale en nombre de tours avec des tailles de signature et de communication de 100 Ko et 850 Ko.

Hanzlik et al. [HLW23] optimisent l'approche des transformations de renforcement dans le cadre des couplages. Ils fournissent des signatures aveugles avec différents compromis sécurisés sous l'hypothèse CDH. Une instance donne par exemple une taille de signature de 5 Ko avec une taille de communication de 72 Ko, sous l'hypothèse CDH.

De plus, il existe des constructions dans le cadre des couplages avec une configuration de confiance composée d'éléments de groupe aléatoires [BFPV13, AJOR18]. Blazy et al. [BFPV13] construit des signatures aveugles basées sur la signature de Waters [Wat05] avec seulement deux éléments de groupe, c'est-à-dire 96 B, mais avec une communication importante, par exemple 220 KB pour des messages de 256 bits. Abeet al. [AJOR18] instancie la signature aveugle de Fischlin [Fis06] pour obtenir des signatures aveugles d'une taille de 5,8 Ko avec environ 1 Ko de communication. Avec des techniques standards, la configuration de confiance peut être supprimée dans le ROM.

Une question intéressante est jusqu'où nous pouvons optimiser les signatures aveugles (optimales en termes de tours) dans le ROM compte tenu des progrès récents.

1.4.1 Nos Contributions

Nous présentons deux signatures aveugles optimales en termes de tours basées sur des hypothèses standards liées aux groupes et le ROM dans le cadre de l'couplage asymétrique. La première construction a une taille de signature de 447 B et une taille de communication de 303 B. Dans le ROM, elle présente la plus petite taille de communication parmi toutes les constructions précédentes prouvées sous des hypothèses standards, et c'est la première construction où la somme de la taille de signature et de la taille de communication est inférieure à 1 Ko. La deuxième construction a une taille de signature de 96 B et une taille de communication de 2.2 KB. Bien qu'elle ait une taille de communication plus grande par rapport à notre première construction, la signature ne comprend que 2 éléments de groupe, correspondant à la taille précédemment la plus courte selon Blazyet al. [BFPV13], tout en améliorant simultanément leur taille de communication d'environ deux ordres de grandeur. Les deux constructions ont des variantes partiellement aveugles efficaces.

Pour notre première construction, nous revisitons la construction générique de signature aveugle de Fischlin [Fis05] et affaiblissons progressivement les composants nécessaires selon Fischlin et montrons que la signature aveugle peut être instanciée de manière beaucoup plus efficace dans le ROM que précédemment grâce à un choix judicieux des composants.

Pour notre deuxième construction, nous reprenons l'idée de Blazy et al. [BFPV13] en nous appuyant sur des signatures « randomisables ». Nous construisons une signature aveugle basée sur les signatures Boneh-Boyen [BB04a] et une preuve de connaissance nulle non interactive extrayable en ligne obtenue via la transformation de Fiat-Shamir appliquée à Bulletproofs [BBB+18] et à un Σ-protocol pour certaines déclarations liées à ElGamal.

1.5 Publications associées et autres contributions

Nous donnons un aperçu concis de nos publications personnelles.

- **Non-interactive Keyed-Verification Anonymous Credentials** [CR19]. Les anonymous credentials avec keyed-verification sont des protocoles qui permettent l'authentification d'utilisateurs autorisés auprès d'une autorité désignée sans compromettre leur vie privée. Un schéma est dit non interactif si le processus d'authentification ne nécessite qu'un seul message envoyé par l'utilisateur qui dissimule toujours son identité. Nous construisons le premier schéma de justificatif d'identité anonyme avec vérification par clé non interactif dans le modèle standard, sans pairings. Nous y parvenons en nous appuyant sur une combinaison de MAC algébrique avec une NIZK avec vérificateur désigné approprié dans le modèle standard. Pour cela, nous introduisons la notion des NIZK inconscientes, où le prouveur peut générer des preuves non interactives pour des énoncés qu'il ne peut pas vérifier lui-même, ne disposant que d'une partie du témoin correspondant, et où la preuve peut être vérifiée efficacement en fournissant la partie manquante du témoin. Nous proposons une construction optimisée d'une preuve NIZK avec vérificateur désigné insoupçonnable nécessitant une configuration de confiance dans le modèle standard.

  Cet article est le résultat de notre mémoire de licence et a été publié dans les actes de la conférence PKC en 2019.

- **Efficient Range Proofs with Transparent Setup from Bounded Integer Commitments** [CKLR21]. Nous introduisons une nouvelle approche pour construire des preuves d'intervalles efficaces. Notre approche repose sur une transformation qui transforme tout engagement sur un corps fini en un schéma d'engagement permettant de s'engager et de prouver efficacement des relations concernant des *entiers bornés*. Bien que la transformation restreigne modérément les propriétés homomorphiques, elle permet d'instancier l'approche pour des preuves d'intervalle basées sur la décomposition en carrés dans divers contextes.

Il s'agit d'un paradigme qui était auparavant limité aux preuves d'intervalle basées sur RSA. Nous instancions l'approche pour améliorer l'état de l'art dans les paradigmes du logarithme discret, des réseaux et des groupes de classes.

Cet article est le résultat de notre mémoire de Master et a été publié dans les actes de la conférence Eurocrypt en 2021.

– **SSE and SSD: Page-Efficient Searchable Symmetric Encryption** [BBF+21]. Le goulot d'étranglement des performances des schémas SSE classiques provient généralement du coût des accès mémoire. Nous constatons que pour les nouveaux supports de stockage tels que les disques SSD (Solid State Drives), la notion classique de localité n'est pas un bon prédicteur des performances pratiques. Au lieu de cela, les performances des SSD dépendent principalement de l'efficacité des pages, c'est-à-dire de la lecture du moins de pages possible. Nous définissons cette notion et identifions un problème simple d'allocation mémoire qui représente le principal défi technique nécessaire pour construire une SSE efficace en termes de pagination. Nous construisons un schéma SSE statique optimal en termes d'efficacité de pagination et d'efficacité de stockage, appelé Tethys, ainsi que les variantes Pluto et Nilus. Le cœur technique de ce résultat est une nouvelle généralisation du hachage coucou aux éléments de taille variable.

  Cet article a été publié dans les actes de la conférence Crypto en 2021.

– **Dynamic Local Searchable Symmetric Encryption** [MR22]. Nous proposons les premières constructions *dynamiques* pour les SSE avec sous-logarithmique efficacité des pages et localité. Pour les DSSE avec une taille de page p, nous construisons un schéma avec une efficacité de stockage $\mathcal{O}(1)$ et une efficacité des pages $\widetilde{\mathcal{O}}(\log\log(N/p))$, appelé LayeredSSE. L'innovation technique principale derrière LayeredSSE est une nouvelle extension pondérée du processus d'allocation à deux choix. Ensuite, nous introduisons le Generic Local Transform, qui prend en entrée un schéma DSSE efficace en termes efficacité des pages avec certaines caractéristiques, et produit un schéma SSE avec de fortes propriétés de *localité*. Nous appliquons le Generic Local Transform à LayeredSSE et obtenons un schéma DSSE avec une efficacité de stockage $\mathcal{O}(1)$, une localité $\mathcal{O}(1)$ et une efficacité de lecture $\widetilde{\mathcal{O}}(\log\log N)$, à condition que la liste la plus longue ait une taille $\mathcal{O}(N^{1-1/\log\log\lambda})$. Enfin, nous appliquons le Generic Local Transform à une variante de Tethys [BBF+21] et obtenons un SSE statique inconditionnel avec une efficacité de stockage $\mathcal{O}(1)$, une localité $\mathcal{O}(1)$ et une efficacité de lecture $\mathcal{O}(\log^{\varepsilon} N)$, pour une constante arbitrairement petite $\varepsilon > 0$.

  Cet article a été publié dans les actes de la conférence Crypto en 2022.

– **Sharp: Short Relaxed Range Proofs** [CGKR22]. Nous fournissons des techniques d'optimisation des preuves d'intervalles basées sur la décomposition en carrés et obtenons des preuves d'intervalles optimisées dans les groupes de logarithmes discrets. Ces techniques incluent des tests de courte longueur en batch pour les fractions, une Σ-protocol optimisée pour montrer qu'une valeur engagée est la somme de 3 carrés, et une méthode pour changer de groupe au sein d'une Σ-protocol pour améliorer l'efficacité. Comme nos preuves d'intervalles satisfont une notion de sécurité « relaxée », nous montrons comment renforcer leur sécurité avec un élément de groupe d'ordre caché supplémentaire. Nous esquissons également des applications des preuves d'intervalle relaxées, telles que les justificatifs d'identité anonymes et les transactions anonymes.

  Cet article a été publié dans les actes de la conférence CSS en 2022.

– **Hermes: I/O-Efficient Forward-Secure Searchable Symmetric Encryption** [MR23]. Nous construisons le premier schéma DSSE à chiffrement avec sécurité persistante et sous-logarithmique efficacité des pages appelé Hermes. Notre construction repose sur deux

nouvelles techniques. Premièrement, nous utilisons une quantité contrôlée de mise en mémoire butoir côté client, combinée à un arrangement de mises à jour déterministe. Deuxièmement, nous introduisons la notion de SSE prenant en charge des *dummy updates*. En combinant ces deux techniques, nous offrons une nouvelle voie pour réaliser la sécurité persistante, qui est compatible avec l'efficacité E/S. Hermes atteint une efficacité des pages $\widetilde{\mathcal{O}}\left(\log\log\frac{N}{p}\right)$, une efficacité de stockage constante, et présente une fuite standard avec une sécurité persistante.

Cet article va être publié dans les actes de la conférence Asiacrypt en 2023.

– **Practical Round-Optimal Blind Signatures in the ROM** [KRS23]. Nous présentons deux signatures aveugles de complexité optimale dans le ROM avec des approches différentes : l'une atteint la somme la plus petite des tailles de signature et de communication, tandis que l'autre atteint la plus petite taille de signature. Nos deux instanciations sont basées sur des hypothèses standards sur les groupes de couplage asymétriques. Notre première construction est une variante hautement optimisée de la construction générique de signature aveugle de Fischlin, avec des tailles de signature et de communication respectivement 447 B et 303 B. Nous affaiblissons progressivement les blocs de construction requis par Fischlin et obtenons la première signature aveugle dont la somme des tailles de signature et de communication est inférieure à 1KB. Notre deuxième construction est une construction semi-générique issue d'une classe spécifique de schémas de signature randomisables qui admet une réduction *tous-sauf-un*. La taille de la signature est seulement 96 B tandis que la taille de la communication est 2.2 KB. Cela correspond à la plus petite taille de signature connue précédemment tout en améliorant la taille de la communication de plusieurs ordres de grandeur. Nos deux constructions reposent sur une analyse fine (non boîte noire) du lemme de bifurcation.

  Cet article va être publié dans les actes de la conférence Asiacrypt en 2023.

– **Mergeable Searchable Encryption and Applications** [BMR23]. Nous introduisons une vision alternative de SSE dynamique, où le protocole de mise à jour est remplacé par un protocole de fusion (Merge). Le protocole Merge prend en entrée les identifiants de deux bases de données et les fusionne en une seule base de données. Le protocole de mise à jour traditionnel peut être retrouvé en tant que cas particulier de Merge, où l'une des deux bases de données d'entrée est constituée d'une seule entrée (la nouvelle mise à jour), tandis que l'autre est constituée de l'ensemble du reste de la base de données. La plus grande flexibilité d'une opération de fusion ouvre de nouvelles opportunités en termes d'efficacité et de sécurité. En pratique, cela imite le comportement des arbres de fusion log-structurés, largement déployés dans le monde des bases de données en texte clair. Nous illustrons cette polyvalence en construisant d'abord une instantiation efficace de SSE fusionnable et l'utilisons pour dériver plusieurs applications, également dans le domaine de SSE traditionnelle. Cela comprend la première DSSE avec une bonne efficacité des pages sans état client avec sécurité persistante.

  Cet article est en cours de soumission.

1.6 Organisation du Manuscrit

Cette thèse est organisée en sept chapitres comme suit.

– Le chapitre 2 est l'introduction du présent document.

– Le chapitre 3 introduit les notations utilisées tout au long de la thèse, et rappelle certaines définitions, notions et résultats fondamentaux.

– Le chapitre 4 présente des variantes pondérées des variantes classiques de hachage. Ce chapitre contient du contenu issu de [BBF+21, MR22, BMR23], bien que nous généralisions le hachage coucou avec des poids entiers (comme dans [BBF+21]) à des poids réels.

– Le chapitre 5 introduit la notion d'efficacité des pages et construit deux schémas efficaces en termes de pages, Pluto et LayeredSSE. Les constructions utilisent les variantes pondérées de hachage du chapitre 4. Ce chapitre est basé sur [BBF+21, MR22].

– Le chapitre 6 présente le premier schéma SSE à sécurité avancée avec une efficacité mémoire sub-logarithmique. La construction s'appuie sur LayeredSSE et est basée sur [MR23].

– Le chapitre 7 présente deux constructions efficaces de signatures aveugles dans le ROM basées sur des hypothèses standards. Ce chapitre est basé sur [KRS23].

– Le chapitre 8 donne un aperçu de nos résultats et discute de certains problèmes ouverts.

Chapter

2 Introduction

In this chapter, we give an overview of our contributions and related work. We also give a brief overview of all our works and the organization of this manuscript.

Chapter content

2.1 Cryptography 13
2.2 Privacy 14
2.3 Searchable Symmetric Encryption 15
2.3.1 Our Contributions 17
2.4 Blind Signatures 18
2.4.1 Our Contributions 19
2.5 Associated Publications and other Contributions 20
2.6 Organization of the Manuscript 22

2.1 Cryptography

Cryptography provides tools for secure communication, often in the form of modular cryptographic primitives. The most important primitive is perhaps symmetric encryption, where two users Alice and Bob share a common secret key K and aim to exchange private messages securely, i.e., no third party (often referred to as *adversary*) should learn information about the private messages without knowledge of K. To this end, Alice encrypts her message m using her secret key K an encryption algorithm Enc via $c \leftarrow \mathsf{Enc}(\mathsf{K}, m)$. Then, Bob obtains the ciphertext c, and can recover $m \leftarrow \mathsf{Dec}(\mathsf{K}, c)$ via some decryption algorithm Dec and the key K. We often refer to an instantiation of such a primitive as scheme. Even if the adversary cannot execute the decryption algorithm Dec without K, there might be another way to recover (parts of) the message m from the ciphertext c. Thus, schemes are cryptanalyzed to gain confidence in their security: if no attack works it is reasonable to assume that it is secure. The requirements we expect from a secure scheme are captured in security notions, e.g., it should be hard to distinguish whether a given ciphertext encrypts either m_0 or m_1, even if the adversary can choose both messages at will. Such security notions are formalized in security games, where a challenger interacts with an adversary (that intends to win the game). If an adversary manages to win the game, it breaks the security of the scheme. This framework allows us to classify attacks, or even prove mathematically that a scheme is (in)secure for a given notion of security.

While there are symmetric encryption schemes with perfect security [Sha49], i.e., the scheme is impossible to break even for an adversary with unbounded computational power, these schemes are impractical due to large secret keys that scale with the length of the message. Thus, it is desirable to construct efficient schemes at the expense of perfect security. In practice, security against adversaries with restricted computational power is sufficient.

In 1973, IBM introduced the Data Encryption Standard (DES)—an efficient symmetric encryption scheme designed for practical use [oSN77]. While it was shown to be insecure [BS93],

its design principles guide the construction of modern primitives. Nowadays, there are efficient symmetric encryption schemes that are widely believed to be secure, e.g., AES [oSN01]. Also, other symmetric primitives with different functionalities are proposed. An important example is *(keyed) hash functions* which create a short digest of long messages, and security ensures that it is hard to find two distinct messages with the same digest. Another example is *Message Authentication Codes* (MACs) which can create tags on messages given a private key K, and security ensures that it is not possible to forge a tag for messages without knowing K. MACs enable for example authentication of users, i.e., Alice can check if she is talking to Bob and can ensure the integrity of received data. Nevertheless, one caveat remains: Alice and Bob need a shared secret key K. To enable communication without the tedious setup of keys K a new approach is needed.

The foundation of this direction was laid in the seminal work of Diffie and Hellman [DH76] in which they introduce *asymmetric* cryptography. In asymmetric encryption, both communicating parties do not have to share a common secret key K, but Bob can encrypt messages for Alice with the public key pk_A of Alice. Alice can then decrypt such ciphertexts with her secret key sk_A, and security ensures that ciphertexts leak no information about the message to an adversary without access to sk_A. This approach requires (at least) the existence of one-way functions, i.e., functions that are easy to compute but hard to invert, whose existence implies $\mathsf{P} \neq \mathsf{NP}$. As to this date the question of whether $\mathsf{P} \neq \mathsf{NP}$ remains unresolved, cryptographers rely on computational hardness assumptions. These assumptions state that some problem is hard to solve for a computationally bound algorithm. Then, cryptographers construct schemes and show that it is secure such assumptions. This approach enables the creation of diverse cryptographic schemes with provable security under some assumption(s). Additional primitives include *key exchange* which establishes a common secret key between Alice and Bob to enable subsequent secure communication with (more efficient) symmetric primitives, or *digital signatures* which are the asymmetric counterpart of MACs.

2.2 Privacy

An important goal of cryptography is to preserve the privacy of users. When a user interacts with a digital system, it often leaks personal information to the system. For example, if Alice stores her personal notes on the server of a cloud provider, the cloud provider might learn the content of her notes. Similarly, if Bob votes online for his favorite candidate, the content of his ballot might be leaked to the voting system. This private information might be sold to third parties, but even if the service provider itself treats the private information with care and good intentions, such private information might be recovered by an adversarial third party after a data breach with grave consequences. Due to the great importance of the protection of personal data, it is required by law in the European Union under the General Data Protection Regulation (GDPR). The GDPR demands for example that personal data is stored confidentially and that the system minimizes the amount of personal information it processes. Cryptography provides tools required to ensure this: encryption enables the storage of confidential data, and digital signatures (or MACs) enable the authentication of users that are entrusted to access private information.

Richer Functionalities. While the aforementioned basic cryptographic primitives provide some tools to ensure privacy, complex real-world systems require more than just encryption and signatures. For example, when encrypting all data stored by Alice on the cloud, it becomes difficult to interact with the data later on. Alice might want to search for notes that contain the keyword *cryptography* but if the data is encrypted, she has to retrieve all notes and then search through them herself. Similarly, if Bob encrypts his vote and sends it to the voting system, the voting system cannot calculate the result of the election without being able to decrypt the

ciphertext. This might lead to the assumption that cryptography ensures desirable security properties but seemingly hinders functionality.

But Alice and Bob are not restricted to the use of encryption and signatures: modern cryptography provides many other tools. More involved systems, such as encrypted search or electronic voting (as sketched above), require more advanced cryptographic primitives with richer functionalities. These primitives are designed to have two important properties: efficiency and security. Concrete efficiency is desirable since the schemes are deployed in real-world applications. Similarly, provable security is required to ensure that the desired security guarantees of the system hold, ideally under well-established assumptions. The focus of this thesis is the design of such primitives with a focus on efficiency and security based on standard assumptions. We present efficient and provably secure constructions of *searchable symmetric encryption* and *blind signatures* which are used, e.g., in the context of encrypted search and electronic voting.

2.3 Searchable Symmetric Encryption

Encrypted databases are an attractive proposition. A business or hospital may want to outsource its customer database for higher availability, scalability, or persistence, without entrusting plaintext data to an external service. An end-to-end encrypted messaging service may want to store and search user messages, without decrypting them. In a different direction, even if a sensitive database is stored locally, a company may want to keep it encrypted to provide a layer of protection against security breaches and data theft. The adoption by MongoDB of searchable encryption techniques is another recent illustration of the growing demand for encrypted databases [Mon22]. When outsourcing the storage of an encrypted database, a minimal desirable functionality is the ability to search the data.

Searchable Encryption. The promise of searchable symmetric encryption (SSE) is to allow a client to outsource an encrypted database of size N to an untrusted server while retaining the ability to search the data [SWP00]. At a minimum, the client is able to issue a search query to retrieve all document identifiers that match a given keyword. In the case of *Dynamic* SSE (DSSE), the client is also able to modify the contents of the database by issuing update queries, for example, to insert or remove entries. The server must be able to correctly process the queries while learning as little information as possible about the client's data and queries.

Solutions with Minimal Leakage. There are some well-studied solutions to this problem and we give a brief overview below.

Private information retrieval (PIR) was introduced in [CGKS95]. Roughly, PIR allows a user to retrieve the i-th entry from a N-bit database held by one or more servers without revealing which position was accessed by the server(s). With PIR, the client can encrypt its database and outsource it to the server. Then, she can retrieve the desired item without leaking any information to the server. Unfortunately, this strong privacy requirement inherits heavy computational and/or client storage overheads [BIM04, CK20, Yeo23], even if a private hint is stored by the client to improve efficiency.

Oblivious RAM (ORAM) allows a client to run a program on a server without revealing memory access patterns [GO96]. Due to its strong privacy guarantees it is often used in the construction searchable encryption (e.g., [GMP16, KMO18, AM23]). As in the case of PIR, the usage of ORAM comes at a large overhead in bandwidth [GO96, LN18]. Thus, the use is often minimized to parts of the full encrypted search protocol (e.g., [MM17, DCPP20]) at the cost of additional leakage.

Fully homomorphic encryption (FHE) allows for the execution of arbitrary circuits over encrypted data [Gen09]. Keyword search in a database can be modeled as a circuit that is linear

in the size of the database. As the evaluation of circuits over encrypted data scales linearly with the circuit size, this approach requires large computational overhead on the server side. Recently, an exciting work constructs homomorphic encryption for RAM computation [LMW22] which allows for the execution of programs that scales with the complexity of the associated RAM program. Unfortunately, this approach is impractical for now.

Tradeoffs between Leakage and Security. Compared to the aforementioned solutions, a specificity of SSE literature is the focus on high-performance solutions, suitable for deployment on large real-world datasets. Efficient SSE schemes are designed that leak some information about the encrypted database and the queries. The leakage of a scheme is typically expressed via a leakage function. The security proof accompanying an SSE scheme provides formal guarantees regarding what information is leaked to the server during searches and updates. Efficient historical approaches based on deterministic encryption or order-preserving encryption [BCLO09] are subject to severe attacks, due to the large amount of information leaked to the server [NKW15, GLMP19]. In view of this situation, modern research on searchable encryption seeks to offer workable trade-offs between performance, functionality, and security, suited for real-world deployment. Typically, this leaked information includes the total size of the database, the repetition of queries, and an identifier (such as the memory address) of the documents that match a query.

An important property of a leakage function is forward security. Forward security asks that updates should leak no information on the updated keyword to the server [SPS14]. The motivation for forward security is that it mitigates certain attacks: the more severe attacks from [ZKP16] exploit update leakage, and fail on forward-secure schemes.

Directions. Since SSE was introduced by Song *et al.* [SWP00], the area has developed in several different directions. Because of the breadth of literature on SSE, we highlight a few important branches. Insofar as SSE is a trade-off between functionality, security, and efficiency, research in the area can be roughly divided into three avenues, one for each component of the trade-off.

While SSE schemes are often designed to allow for efficient keyword queries, some works extend the provided functionality for boolean queries [CJJ+13], range queries [PKV+14], or even subsets of SQL [KM18].

Works that deal with security include attacks, and efforts to reduce leakage in response to those attacks. Most attacks against searchable encryption fall in the category of *leakage-abuse* attacks, a term coined in [CGPR15]. Leakage-abuse attacks do not contradict the security claims of a scheme, but show how the leakage allowed by the security model enables the server to reconstruct large parts of the database in certain settings [CGPR15, GLMP19]. These attacks have motivated further works that reduce or suppress leakage [GKM21], including forward-secure schemes [PM21, BMO17, KMPQ21, EKPE18, DCPP22].

Among works that mainly target efficiency, perhaps the most notable development in recent years is I/O efficiency.

I/O Efficiency. For performance reasons, most SSE designs rely exclusively on symmetric cryptographic primitives which have small computational overhead in practice. As a result, the main performance bottleneck is instead determined by how quickly the data can be accessed on disk [CT14]. This was formalized under the notion of locality and read efficiency. Locality asks that related data be stored in a small number of disjoint locations on disk, and read efficiency asks that the overhead of read data is small. This notion is motivated by the I/O behavior of Hard Disk Drives (HDDs), where disjoint reads are much more expensive in latency and throughput than contiguous reads.

The need to store related data in close proximity is not at all innocuous for security. Asking that related data items should be stored in close proximity creates a correlation between the

location of an encrypted data item in memory, and its *contents*. Since the server can observe the location of data it is asked to retrieve, and we do not want the server to infer information about the contents of that data, this creates tension between security and efficiency. That is, security asks that there is no correlation between the location of data and its content, while I/O efficiency asks for the opposite.

This tension was captured in an impossibility result by Cash and Tessaro at Eurocrypt 2014 [CT14]. In brief, Cash and Tessaro show that a secure SSE scheme with linear server storage cannot have both constant locality and constant read efficiency. This holds true even for static SSE. In another seminal work, at STOC 2016, Asharov *et al.* build an SSE with constant locality and $\widetilde{\mathcal{O}}(\log N)$ read efficiency — even $\widetilde{\mathcal{O}}(\log \log N)$ with a mild restriction on the input database [ANSS16]. Their construction uncovers a deep connection between weighted hashing and local searchable encryption. Since the introduction of memory efficiency for SSE, many constructions with efficient memory accesses were proposed [CT14, ANSS16, ASS21, DP17a, MM17, DPP18].

2.3.1 Our Contributions

In this thesis, we introduce a different measure of I/O efficiency, called *page efficiency*. Page efficiency is the ratio of memory pages read by the server to process a query, divided by the number of pages necessary to hold the plaintext answer. While locality and read efficiency capture the efficiency of SSE schemes when run on HDDs, our notion is motivated by the I/O behavior of modern Solid State Drives (SSDs). Then, we construct several schemes with good page efficiency in the framework introduced in [ANSS16] based on weighted hashing.

Weighted Hashing. Page efficiency asks to store related identifiers in a small number of pages. It is natural to view pages as bins and identifier lists matching a single keyword as balls of weight proportional to the size of the list. Then, we are interested in an upper bound on the total weight when n balls of total weight w_{tot} are allocated into $\mathcal{O}(n)$ bins at random. A better upper bound later allows for the construction of more efficient SSE schemes. We analyze *weighted* variants of the following unweighted balls-into-bins problems:

- In one-choice allocation (1C), each ball is inserted into a single bin chosen at random. The most loaded bin contains at most $\mathcal{O}(\log N)$ balls [JK77].
- In two-choice allocation (2C), each ball is inserted into the least loaded bin amongst two bins chosen at random. The most loaded bin contains at most a $\mathcal{O}(\log \log n)$ balls [ABKU94].
- In cuckoo hashing with stash [PR04, KMW10], each ball is inserted into one of two bins chosen at random or the stash. After an optimization procedure, each bin contains at most one ball and the size stash is minimal.

An analysis of weighted 1C is given in expectation in [BFHM08], and we include an upper bound that holds with overwhelming probability for completeness. Our 2C variant is a generalization of weighted two-choice allocation called L2C which is designed for the upper bound proof. Lastly, our cuckoo hashing variant relies on a max flow algorithm for global optimization. All our bounds in the weighted setting asymptotically match the bounds in the classical setting with uniform weights and hold with overwhelming probability.

Page-efficient Searchable Encryption. Then, we construct several (D)SSE schemes with good page efficiency based on our weighted hashing variants:

- Pluto: From a theoretical standpoint, constant page efficiency is a weaker requirement than the combination and constant locality and constant read efficiency. Interestingly, this

weaker requirement sidesteps the impossibility result of Cash and Tessaro: we construct a static SSE scheme with optimal page efficiency called Pluto. The setup requires a max flow calculation over all data which renders the scheme inefficient for updates.

– LayeredSSE: We use our weighted generalization of two-choice allocation L2C to construct a dynamic SSE scheme with $\widetilde{\mathcal{O}}(\log\log N/p)$ page efficiency. The scheme leaks the query pattern during updates and is not forward secure. Nevertheless, it is the first dynamic scheme with sublogarithmic page efficiency.

– Hermes: Finally, we construct a DSSE scheme with forward security and $\widetilde{\mathcal{O}}(\log\log N/p)$ page efficiency. The scheme is based on LayeredSSE and novel techniques such as controlled client buffering and dummy updates.

2.4 Blind Signatures

Blind signatures were introduced in [Cha82] and enhance the functionality of digital signatures to provide additional privacy guarantees. Alice and Bob engage in an interactive protocol at the end of which Alice obtains a signature on a message of her choice signed by Bob. The required properties are blindness and one-more unforgeability. Blindness states that if Alice later presents the signature to Bob, Bob cannot link the signature to a specific signing session. In particular, Bob does not learn the signed message during the signing session. One-more unforgeability states that Alice can obtain signatures on at most ℓ distinct messages from ℓ signing sessions.

Blind signatures are a fundamental building block in many privacy-preserving applications such as e-cash [Cha82, CFN90, OO92], anonymous credentials [Bra94, CL01], e-voting [Cha88, FOO92], direct anonymous attestation [BCC04], blockchains [YL19, BDE+22] and privacy-preserving authentication tokens [VPN22, HIP+22]. Since blind signatures were introduced in [Cha82], they have become a vast area of research. We first give a short overview of research directions on blind signatures.

Multi-round Blind Signatures

In the Random Oracle Model (ROM), there are efficient 3-move blind signatures [Sch90, Oka93, AO00]. Note that in the ROM, the adversary has oracle access to an idealized random function H. In practice, H is often heuristically instantiated with a collision-resistant hash function. In their seminal work, Pointcheval and Stern [PS00] show that [Oka93] is secure for at most polylog-many signing sessions. Their analysis was generalized and refined in [HKL19] and a similar result was shown for [AO00] in [KLX22a]. Unfortunately, a recent attack [BLL+21] shows that the polylog upper bound on concurrent signing sessions is tight.

This approach was later refined and there are several 3-move blind signatures [Abe01, KLX22b, FPS20, TZ22] that are proven secure in the Generic Group Model (GGM) or Algebraic Group Model (AGM), and the ROM. In the GGM and AGM, the adversary is restricted to performing algebraic attacks, i.e., it uses the group in a black-box manner. These constructions avoid the attack given in [BLL+21] and can be instantiated efficiently in practice. Notably, [TZ22, KLX22b] provide full concurrent security. Such models are less desirable from a theoretical perspective as they restrict the capabilities of the adversary substantially. In the ROM, [BL13] shows sequential security of [Abe01] without generic groups, but the signer needs to ensure that at most one signing session is open at all times.

Boosting Transforms. A recent line of work [KLR21, CAHL+22] based on [Poi98] provides generic boosting transformations for blind signatures with limited concurrent security. Unfortunately, the computational efficiency scales linearly with the number of signed signatures.

Recently, [HLW23] provides a concrete pairing-based construction in the ROM with good efficiency that is round optimal (cf. below).

Round-Optimal Blind Signatures

A desirable property for blind signatures are signing session in two rounds, i.e., two messages are exchanged to obtain a signature. There are several impossibility results [Lin08, FS10, Pas11] that indicate that either a trusted setup, an idealized model, or non-standard assumptions are required.

Complexity Leveraging and Interactive Assumptions. There are round-optimal blind signatures [GRS+11, GG14] that circumvent impossibility results to construct standard model blind signatures via complexity leveraging. [KNYY21] manages to avoid complexity leveraging by relying on both post-quantum assumptions and classical assumptions. The aforementioned schemes are mainly of theoretical interest and not practical.

Chaum [Cha82] gives a simple construction based on the classical RSA signature, which was later shown secure in [BNPS03] under a One-more RSA and in the ROM. Later, [Bol03, AKSY22] propose similar schemes in different settings secure under a comparable one-more assumption in the ROM. Further, there are constructions in the standard model [AFG+10, FHS15, FHKS16, Gha17] that rely on tailored interactive hardness assumptions. The above constructions are efficient but rely on interactive security assumptions. This is acceptable from a practical viewpoint as the assumptions are not yet broken. But it is less satisfactory from a theoretical viewpoint: interactive assumptions are non-falsifiable and thus less desirable [Nao03].

Trusted Setup. In the pairing setting, there are several round-optimal blind signatures with trusted setup [MSF10, SC12, KSD19] in the standard model. Again, these schemes are less practical and the trusted setup is *structured*, i.e., non-uniform. This hinders practical deployment as a trapdoor might be embedded in the parameters of the scheme. There are other schemes [BFPV13, AJOR18] with a uniform setup that we discuss below.

Constructions in the ROM. There are some efficient constructions secure in the ROM with full concurrent security. We give a brief overview.

In the lattice setting, del Pino and Katsumata [dK22] propose a round-optimal blind signature with signature and communication sizes 100 KB and 850 KB.

Hanzlik et al. [HLW23] optimize the approach of boosting transforms in the pairing setting. They provide blind signatures with different tradeoffs secure under the CDH assumption. One instantiation yields for example a signature size of 5 KB with a communication size 72 KB, under the CDH assumption.

Further, there are constructions in the pairing setting with a trusted setup consisting of random group elements [BFPV13, AJOR18]. Blazy et al. [BFPV13] constructs blind signatures based on Waters signature [Wat05] of mere 2 group elements, i.e., 96 B, but with large communication, e.g., 220 KB for 256 bit messages. Abe et al. [AJOR18] instantiates the Fischlin blind signature [Fis06] to obtain blind signatures of size 5.8 KB with around 1 KB of communication. With standard techniques, the setup can be removed in the ROM.

An interesting question is how far we can optimize (round-optimal) blind signatures in the ROM given the recent progress.

2.4.1 Our Contributions

We present two round-optimal blind signatures based on standard group-based assumptions and the ROM in the asymmetric pairing setting. The first construction has signature and

communication sizes 447 B and 303 B, respectively. In the ROM, it has the smallest communication size among all prior schemes proven under standard assumptions and is the first construction where the sum of the signature and communication sizes fit below 1 KB. The second construction has signature and communication sizes 96 B and 2.2 KB, respectively. While it has a larger communication size compared to our first construction, the signature only consists of 2 group elements, matching the previously shortest by Blazy et al. [BFPV13] while simultaneously improving their communication size by around two orders of magnitude. Both constructions have efficient partially blind variants.

For our first construction, we revisit the generic blind signature construction by Fischlin [Fis05] and progressively weaken the building blocks required by Fischlin and show that the blind signature can be instantiated much more efficiently in the ROM than previously thought by a careful choice of the building blocks.

For our second construction, we revisit the idea by Blazy et al. [BFPV13] relying on randomizable signatures. We construct a blind signature based on Boneh-Boyen signatures [BB04a] and an online-extractable non-interactive zero-knowledge proof obtained via the Fiat-Shamir transform applied to Bulletproofs [BBB+18] and a Σ-protocol for some ElGamal related statements.

2.5 Associated Publications and other Contributions

We give a concise overview of our personal publications.

– **Non-interactive Keyed-Verification Anonymous Credentials** [CR19]. Anonymous credentials with keyed-verification are protocols are protocols that allow for the authentication of authorized users to a designated authority without compromising their privacy. A scheme is said to be non-interactive if the authentication process only requires the user to send a single message that still conceals its identity. We construct the first non-interactive keyed-verification anonymous credential scheme in the standard model, without pairings. We achieve this by building upon a combination of algebraic MAC with an appropriate designated-verifier NIZK in the standard model. For this, we introduce the notion of oblivious non-interactive zero-knowledge proof systems, where the prover can generate non-interactive proofs for statements that he cannot check by himself, having only a part of the corresponding witness, and where the proof can be checked efficiently given the missing part of the witness. We provide an optimized construction of an oblivious designated-verifier NIZK with a trusted setup in the standard model.

  This paper is the result of our Bachelor thesis and has been published in the proceedings of PKC in 2019.

– **Efficient Range Proofs with Transparent Setup from Bounded Integer Commitments** [CKLR21]. We introduce a new approach for constructing efficient range proofs. Our approach relies on a transformation that transforms any commitment over a finite field into a commitment scheme that allows to commit to and efficiently prove relations about *bounded integers*. While the transformation restricts the homomorphic properties moderately, it allows us to instantiate the approach for range proofs based on square decomposition in various settings. This is a paradigm that was previously limited to RSA-based range proofs. We instantiate the approach to improve over the state of the art in the discrete logarithm, lattice, and class group setting.

  This paper is the result of our Master thesis and has been published in the proceedings of Eurocrypt in 2021.

– **SSE and SSD: Page-Efficient Searchable Symmetric Encryption** [BBF+21]. The performance bottleneck of classic SSE schemes typically comes from the cost of memory

accesses. We observe that for newer storage media such as Solid State Drives (SSDs) the classical notion of locality is not a good predictor of practical performance. Instead, SSD performance mainly depends on *page efficiency*, that is, reading as few pages as possible. We define this notion and identify a simple memory allocation problem that captures the main technical challenge required to build page-efficient SSE. We construct an optimal page-efficient and storage-efficient static SSE scheme Tethys, and variants Pluto and Nilus. The technical core of the result is a new generalization of cuckoo hashing to items of variable size.

This paper has been published in the proceedings of Crypto in 2021.

– **Dynamic Local Searchable Symmetric Encryption** [MR22]. We provide the first *dynamic* constructions for page efficient and local SSE. For page-efficient DSSE with page size p, we build a scheme with storage efficiency $\mathcal{O}(1)$ and page efficiency $\widetilde{\mathcal{O}}(\log\log(N/p))$, called LayeredSSE. The main technical innovation behind LayeredSSE is a novel weighted extension of the two-choice allocation process. Then, we introduce the Generic Local Transform, which takes as input a *page-efficient* DSSE scheme with certain features and outputs an SSE scheme with strong *locality* properties. We apply the Generic Local Transform to LayeredSSE, and obtain a DSSE scheme with storage efficiency $\mathcal{O}(1)$, locality $\mathcal{O}(1)$, and read efficiency $\widetilde{\mathcal{O}}(\log\log N)$, under the condition that the longest list is of size $\mathcal{O}(N^{1-1/\log\log\lambda})$. Finally, we apply the Generic Local Transform to a variant of Tethys [BBF+21], and obtain an unconditional static SSE with storage efficiency $\mathcal{O}(1)$, locality $\mathcal{O}(1)$, and read efficiency $\mathcal{O}(\log^{\varepsilon} N)$, for an arbitrarily small constant $\varepsilon > 0$.

This paper has been published in the proceedings of Crypto in 2022.

– **Sharp: Short Relaxed Range Proofs** [CGKR22]. We provide techniques to optimize range proofs based on square decomposition and obtain optimized range proofs in discrete logarithm groups. These techniques include batched shortness tests for fractions, an optimized Σ-protocol for showing that a committed value is the sum of 3 squares and a method to switch groups within a Σ-protocol to improve efficiency. As our range proofs satisfy a *relaxed* notion of security, we show how to enhance their security with one additional hidden order group element. We also sketch applications of relaxed range proofs, such as anonymous credentials and anonymous transactions.

This paper has been published in the proceedings of CSS in 2022.

– **Hermes: I/O-Efficient Forward-Secure Searchable Symmetric Encryption** [MR23]. We construct the first forward-secure and page-efficient DSSE scheme called Hermes. Our construction relies on two novel techniques. First, we make use of a controlled amount of client buffering, combined with a deterministic update schedule. Second, we introduce the notion of SSE supporting *dummy updates*. In combination, those two techniques offer a new path to realizing forward security, which is compatible with I/O efficiency. Hermes achieves $\widetilde{\mathcal{O}}\left(\log\log\frac{N}{p}\right)$ page efficiency, constant storage efficiency, and has standard leakage with forward security.

This paper will be published in the proceedings of Asiacrypt in 2023.

– **Practical Round-Optimal Blind Signatures in the ROM** [KRS23]. We present two round-optimal blind signatures in the ROM with different approaches: one achieves the smallest sum of the signature and communication sizes, while the other achieves the smallest signature size. Both of our instantiations are based on standard assumptions over asymmetric pairing groups. Our first construction is a highly optimized variant of the generic blind signature construction by Fischlin and has signature and communication sizes 447 B and 303 B, respectively. We progressively weaken the building blocks required

by Fischlin and result in the first blind signature where the sum of the signature and communication sizes fit below 1 KB. Our second construction is a semi-generic construction from a specific class of randomizable signature schemes that admits an *all-but-one* reduction. The signature size is only 96 B while the communication size is 2.2 KB. This matches the previously known smallest signature size while improving the communication size by several orders of magnitude. Both of our constructions rely on a (non-black box) fine-grained analysis of the forking lemma.

This paper will be published in the proceedings of Asiacrypt in 2023.

– **Mergeable Searchable Encryption and Applications** [BMR23]. We introduce an alternative view of dynamic SSE, where the update protocol is replaced by a Merge protocol. The Merge protocol takes as input the identifiers of two databases and merges them into a single database. The traditional Update protocol can be recovered as the special case of Merge, where one of the two input databases consists of a single entry (the new update), while the other consists of the entire rest of the database. The greater flexibility of a Merge operation opens up new opportunities for both efficiency and security. In practice, it mimics the behavior of log-structured merge trees, which are widely deployed in the world of plaintext databases. We illustrate this versatility by first building an efficient instantiation of mergeable SSE and using it to derive several applications, also in the realm of traditional SSE. This includes the first *stateless* page-efficient DSSE with forward security.

This paper is in submission.

2.6 Organization of the Manuscript

This thesis is organized into seven chapters as follows.

– Chapter 2 is the present introduction.

– Chapter 3 introduces the notations used throughout and recalls some definitions, notions, and fundamental results.

– Chapter 4 presents weighted variants of classical hashing variants. This chapter contains content from [BBF+21, MR22, BMR23], though we generalize cuckoo hashing with integer weights (as in [BBF+21]) to real weights.

– Chapter 5 introduces the notion of *page efficiency* and constructs two page efficient schemes, Pluto and LayeredSSE. The constructions use the weighted hashing variants of Chapter 4. This chapter is based on [BBF+21, MR22].

– Chapter 6 provides the first forward secure SSE scheme with sublogarithmic memory efficiency. The construction relies on LayeredSSE and is based on [MR23].

– Chapter 7 presents two efficient constructions of blind signatures in the ROM based on standard assumptions. This chapter is based on [KRS23].

– Chapter 8 gives a brief overview of our results and discusses some open problems.

Chapter

3 Preliminaries

In this chapter, we establish the notation that we use throughout the thesis. Also, we recall some probabilistic lemmata, computational hardness assumptions, and definitions of cryptographic primitives used in subsequent chapters.

Chapter content

3.1 Notation **23**
3.2 Probability **24**
3.3 Graphs **25**
3.4 Assumptions **26**
3.4.1 Groups 26
3.5 Cryptographic Primitives **27**
3.5.1 Symmetric Primitives 27
3.5.2 Asymmetric Primitives 28
3.6 Searchable Symmetric Encryption **34**

3.1 Notation

Let $\lambda \in \mathbb{N}$ be the security parameter. A probabilistic polynomial time (PPT) algorithm $\mathcal{A}$ runs in time polynomial in the (implicit) security parameter λ. We write $\mathsf{Time}(\mathcal{A})$ for the runtime of $\mathcal{A}$. A function $f(\lambda)$ is *negligible* in λ if it is $\mathcal{O}(\lambda^{-c})$ for every $c \in \mathbb{N}$. We write $f = \mathsf{negl}(\lambda)$ for short. Similarly, a function $f(\lambda)$ is *overwhelming* in λ if $1 - f(\lambda) = \mathsf{negl}(\lambda)$. Also, we write $f = \mathsf{poly}(\lambda)$ if $f(\lambda)$ is a polynomial with variable λ. If D is a probability distribution, $x \leftarrow D$ means that x is sampled from D and if S is a set, $x \leftarrow S$ means that x is sampled uniformly and independently at random from S. We also write $|S|$ for the cardinality of set S.

Further, we write $D_0 \overset{c}{\approx} D_1$ for distributions D_0, D_1, if for all PPT adversaries $\mathcal{A}$, we have $|\Pr[x_0 \leftarrow D_0 : \mathcal{A}(1^\lambda, x_0) = 1] - \Pr[x_1 \leftarrow D_1 : \mathcal{A}(1^\lambda, x_1) = 1]| = \mathsf{negl}(\lambda)$. Similarly, we write $D_0 \overset{s}{\approx} D_1$ if the above holds even for unbounded adversaries. For some PPT algorithm $\mathcal{A}$, we write $\mathcal{A}^{\mathcal{O}}$ if $\mathcal{A}$ has oracle access to the oracle $\mathcal{O}$. If $\mathcal{A}$ performs some check, and the check fails, we assume that $\mathcal{A}$ outputs $\perp$ immediately. Generally, we assume that adversaries are implicitly stateful.

We denote with $[n]$ the set $\{1, ..., n\}$ for $n \in \mathbb{N}$. We write $[a, b]$ for the integer interval $\{x \in \mathbb{N} : a \leq x \leq b\}$ and $[a, b]_{\mathbb{R}}$ for the real interval $\{x \in \mathbb{R} : a \leq x \leq b\}$. For any $\vec{h} = (h_1, ..., h_q)$ and $i \in [q]$, we denote $\vec{h}_{<i}$ as $(h_1, ..., h_{i-1})$ and $\vec{h}_{\geq i}$ as $(h_i, ..., h_q)$, where $\vec{h}_{<1}$ denotes an empty vector. Moreover, for any two vectors $\vec{h}$, $\vec{h'}$ of arbitrary length, we use $\vec{h} \| \vec{h'}$ to denote the concatenation of the two vectors. In particular, for any $i \in [q]$ and $\vec{h} \in \mathcal{H}^q$, we have $\vec{h} = \vec{h}_{<i} \| \vec{h}_{\geq i}$. We denote by $\oplus$ the bitwise XOR operation.

Protocols. Let $\mathsf{prot} = (\mathsf{prot}_A, \mathsf{prot}_B)$ be a protocol between two parties A and B. We denote an execution of protocol prot between A and B with input in_A and in_B respectively by $\mathsf{prot}_{\mathsf{A}}(\mathsf{in}_A) \longleftrightarrow$

$\mathsf{prot}_{\mathsf{B}}(\mathsf{in}_B)$. We may write $\mathsf{prot}(\mathsf{in}_A; \mathsf{in}_B)$ for short, if both executing parties can be inferred from the context.

Data Structures. For concision, in algorithmic descriptions, tables and arrays are implicitly assumed to be initialized with zeros if they contain integers or $\perp$'s otherwise, if it is clear by context (unless stated otherwise). Our algorithms will frequently make use of *bins*, which can be thought of as disjoint memory segments of some fixed size $s = f(p, \lambda)$ for some function f with some additional input p. Bins can contain arbitrary data up to their capacity s. Bins are always implicitly assumed to be padded with zeros up to their full capacity, so that their size remains fixed. In particular, the encryption of a bin reveals no information about the amount of real data contained in the bin.

3.2 Probability

In this paragraph, we recall some basic probabilistic results that we require later on. The following variant of the Chernoff bound gives an upper bounds on the probability that certain random variables attain values far from their expectation.

Lemma 3.1 (Chernoff's Bound)**.** *Suppose that $X_1, ..., X_n$ are independent random variables taking values in $\{0, 1\}$. Let X denote their sum and let $\mu = \mathsf{Exp}[X]$ denote the expectancy of X. Then for any $\delta > 0$, it holds that*

$$\Pr[X \geq (1+\delta)\mu] \leq \exp\left(-\frac{\delta^2 \mu}{2+\delta}\right)$$

The next lemma allows us to switch between expectation and probability when analyzing whether some event occurs with negligible probability in some scenarios.

Lemma 3.2. *Let $N \in \mathbb{N}$ with $N = \mathsf{poly}(\lambda)$. For any integer-valued random variable $X \in [0, N]$ and any $R_N > 0$:*

$$\Pr[X > R_N] = \mathsf{negl}(\lambda) \quad \textit{iff} \quad \mathsf{Exp}[\max(X - R_N, 0)] = \mathsf{negl}(\lambda).$$

Proof. We denote $Y = \max(X - R_N, 0)$. Note that Y is a non-negative integer-valued random variable. Thus, we have that

$$\mathsf{Exp}[Y] = \sum_{i \geq 0} i \Pr[Y = i] = \sum_{i \geq 0} \Pr[Y > i]$$

The above implies that $\Pr[Y > 0] \leq \mathsf{Exp}[Y]$. Thus, if $\mathsf{Exp}[Y] = \mathsf{negl}(\lambda)$, we have that

$$\Pr[X > R_N] = \Pr[Y > 0] \leq \mathsf{Exp}[Y] = \mathsf{negl}(\lambda).$$

For the other direction, assume that $\Pr[X > R_N] = \mathsf{negl}(\lambda)$. We have as above that $\mathsf{Exp}[Y] = \sum_{i \geq 0} \Pr[Y > i]$. As $Y \leq N$, we have $\sum_{i \geq 0} \Pr[Y > i] = \sum_{i=0}^{N} \Pr[Y > i]$. Finally, observe that $\Pr[Y > i] \leq \Pr[X > R_N + i] \leq \Pr[X > R_N] = \mathsf{negl}(\lambda)$. As $N = \mathsf{poly}(\lambda)$, the statement follows. □

Finally, we recall the standard splitting lemma from Pointcheval and Stern [PS00].

Lemma 3.3 (Splitting Lemma)**.** *Let $\epsilon \in (0, 1]$ and $A \subseteq X \times Y$ such that*

$$\Pr_{(x,y) \leftarrow X \times Y}[(x, y) \in A] \geq \epsilon.$$

For any $\alpha \in [0, \epsilon)$ define

$$B = \left\{ (x,y) \in X \times Y \,\middle|\, \Pr_{(x,y') \leftarrow X \times Y}[(x,y') \in A] \geq \epsilon - \alpha \right\}.$$

Then we have

$$\Pr_{(x,y) \leftarrow X \times Y}[(x,y) \in B \mid (x,y) \in A] \geq \frac{\alpha}{\epsilon}.$$

3.3 Graphs

We present a brief overview of graphs and flows in networks (see [Sch03] for more details).

Directed Graphs.

A directed graph $G = (V, E)$ is a tuple vertices V and edges $E \subseteq V \times V$. Sometimes, we label edges of a graph with real values via a function $\ell : E \to \mathbb{R}$. Sometimes, we write $G = (V, E, \ell)$ for labeled graphs.

Max Flow and Min Cuts

A network $N = (G, c, s, t)$ consists of a directed graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{R}^+$, a source $s \in V$, as well as a sink $t \in V$. For each edge $(u, v) \in E$, we assume that its corresponding back edge (v, u) is in E, with $c(v, u) = 0$ if necessary.

A flow assigns a value, its flow, to each edge which has to satisfy two properties. First, the flow of an edge cannot exceed its capacity. Second, the sum of the flows entering a node must equal the sum of the flows exiting that node, except for the source and the sink. The value of the flow is its outgoing flow from source s. Note that this value is equal to the incoming flow at sink t, as the flow has to be conserved throughout the network.

Definition 3.4 (Flows, Flow Value)**.** A flow $f : E \to \mathbb{R}^+$ is an assignment of edge weights satisfying the following constraints:

- Capacity constraint: $\forall (u, v) \in E : f(u, v) \leq c(u, v)$
- Conservation of flows:

$$\forall v \in V \setminus \{s, t\} : \sum_{u \in V : \{u,v\} \in E} f(u, v) = \sum_{u \in V : \{v,u\} \in E} f(v, u).$$

The value of the flow is defined as

$$|f| = \sum_{u \in V : \{s,u\} \in E} f(s, u) - \sum_{u \in V : \{u,s\} \in E} f(u, s).$$

A cut is a partition (S, T) of the nodes V such that the source resides in S and the sink resides in T. The capacity of a cut is the sum of the weights of edges leaving S.

Definition 3.5 (Cuts, Capacity)**.** A cut $C = (S, T)$ is a partition of V such that $s \in S$ and $t \in T$. The capacity of a cut C is defined as $\mathsf{cap}(C) = \sum_{(u,v) \in E \cap (S \times T)} c(u, v)$.

The following theorem shows that finding a maximum flow and a minimal cut are closely related.

Theorem 3.6 (Max-Flow Min-Cut)**.** *For any network, the maximal flow value from s to t is equal to the minimal cut capacity over cuts separating s and t. Further, there is a max flow f and a min cut (S, T) of V such that the flow of edges between S and T is at capacity, and there is no flow from T to S.*

Algorithms. Finding a max flow of a network is a well-studied problem, and many efficient solutions exist. The Ford-Fulkerson algorithm [FF56] finds the max flow f_{max} of a network in running time $\mathcal{O}(|E|f_{max})$. The algorithm looks (in a greedy manner) for augmenting paths in the graph. As the name suggests, an augmenting path is a path (under capacity) that increases the max flow by sending more flow along its edges. Since the greedy decision might not be optimal, the algorithm considers the residual graph which allows for reallocation of some flow from previous rounds. If no more augmenting path can be found, the solution is optimal. Note that the Ford-Fulkerson might not terminate if the capacities are irrational, but there are algorithms that work for real weights (e.g., [SW97]).

3.4 Assumptions

We introduce the group-based hardness assumptions we require in Chapter 7. We also give a brief overview of explainable sampling of group elements.

3.4.1 Groups

Throughout this work, write $1_{\mathbb{G}}$ for the neutral element of some group $\mathbb{G}$ and use multiplicative notation. Also, we assume a PPT algorithm $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow \mathsf{PGen}(1^\lambda)$ that on input 1^λ outputs descriptions of the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order p and a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, as well as generators g_1, g_2 of $\mathbb{G}_1, \mathbb{G}_2$, respectively. Recall that e is a pairing if e is non-degenerate (i.e., $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$) and e is bilinear (i.e., $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ for $a, b \in \mathbb{Z}_p$). We sometimes use implicit notation $[x]_k = g_k^x$ for $k \in [1, 2, T]$, $x \in \mathbb{Z}_p$ and $g_T = e(g_1, g_2)$. We extend the notation to matrices naturally, i.e., we write $[\boldsymbol{A}]_k = ([a_{i,j}]_k)$ for $\boldsymbol{A} = (a_{i,j}) \in \mathbb{Z}_p^{n \times m}$ and $k \in [1, 2, T]$. Note that while we mainly consider the asymmetric pairing setting (i.e., $\mathbb{G}_1 \neq \mathbb{G}_2$), all instantiations have a natural variant in the symmetric pairing setting with similar efficiency. Similarly, we assume a PPT algorithm $(\widehat{\mathbb{G}}, g) \leftarrow \mathsf{GGen}(1^\lambda, p)$ that on input 1^λ and prime order p, outputs a description of a group $\widehat{\mathbb{G}}$ of order p with generator g that is not equipped with a pairing. Generally, we assume that given the description(s), group operations, pairing evaluation and membership tests are efficient, and write $g \leftarrow \mathbb{G}$ for drawing elements from some group $\mathbb{G}$ at random. For readability, we leave PGen and GGen implicit in the rest of the work.

Instantiation For our instantiations in Chapter 7, we assume that the modulus p is of size 256 bit, and an element of $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ is of size $(382, 763, 4572)$ bit, respectively. These are common sizes of standard BLS curves [BLS03] with security parameter $\lambda = 128$, in particular BLS12-381 [Bow17]. For groups that require no pairing operation, we use a curve of order p and assume that elements are of size 256 bit. We generally write $\widehat{\mathbb{G}}$ for such groups.

Assumptions

Throughout, we use the following hardness assumptions. Let $\mathbb{G}$ be an arbitrary group with generator g and $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow \mathsf{PGen}(1^\lambda)$ be a pairing description.

The discrete logarithm (DLOG) assumption in $\mathbb{G}$ states that it is hard to compute the discrete logarithm x of some random $h = g^x \in \mathbb{G}$. The decisional Diffie-Hellman (DDH) assumption states that it is hard to distinguish tuples (g^a, g^b, g^{ab}) from tuples (g^a, g^b, g^c) with random $a, b, c \leftarrow \mathbb{Z}_p$. The symmetric external Diffie-Hellman (SXDH) assumption holds if the DDH assumption holds in $\mathbb{G}_1$ and in $\mathbb{G}_2$. Finally, the (asymmetric) computational Diffie-Hellman assumption states that given $(g_1^a, g_2^a, g_1^b, g_2^b)$, it is hard to computes g_1^{ab}.

Definition 3.7 (DLOG)**.** The discrete logarithm (DLOG) assumption in group $\mathbb{G}$ with generator g holds if for any PPT adversary $\mathcal{A}$, it holds that

$$\Pr[x \leftarrow \mathbb{Z}_p : \mathcal{A}(g, g^x) = x] = \mathsf{negl}(\lambda)$$

Definition 3.8 (DDH)**.** The decisional Diffie-Hellman (DDH) assumption holds in group $\mathbb{G}$ with generator g if for any PPT adversary $\mathcal{A}$, it holds that

$$|\Pr[a, b \leftarrow \mathbb{Z}_p : \mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - \Pr[a, b, c \leftarrow \mathbb{Z}_p : \mathcal{A}(g, g^a, g^b, g^c) = 1]| = \mathsf{negl}(\lambda).$$

Definition 3.9 (SXDH)**.** The symmetric external Diffie-Hellman (SXDH) assumption holds in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ if the DDH assumption holds in $\mathbb{G}_1$ and in $\mathbb{G}_2$.

Definition 3.10 (CDH)**.** The computational Diffie-Hellman (CDH) assumption holds in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ if for any PPT adversary $\mathcal{A}$, it holds that

$$\Pr[a, b \leftarrow \mathbb{Z}_p : \mathcal{A}(g_1, g_2, g_1^a, g_2^a, g_1^b, g_2^b) = g^{ab}] = \mathsf{negl}(\lambda).$$

Explaining Group Elements as Random Strings

Our frameworks in Chapter 7 generally require that public parameters pp (of commitment schemes) and common random strings crs (of NIZKs) are random bit strings. For readability, we allow that pp and crs contain random group elements $g \leftarrow \mathbb{G}$ for some group $\mathbb{G}$. This is without loss of generality, as using explainable sampling, we can explain these elements as random strings.

Concretely, explainable sampling for $\mathbb{G}$ allows us to sample from the uniform distribution $\mathcal{U}_{\mathbb{G}}$ over $\mathbb{G}$ via $g \leftarrow \mathsf{SampleG}(1^\lambda; r)$ using ℓ_G bits of randomness, where $r \leftarrow \{0,1\}^{\ell_G}$. Importantly, there exists an algorithm $\mathsf{ExplainG}$ that given uniformly random $g \in \mathbb{G}$ outputs randomness $r' \leftarrow \mathsf{ExplainG}(1^\lambda, g)$ such that $g = \mathsf{SampleG}(1^\lambda; r')$ and $r \overset{s}{\approx} r'$. Note that such sampling techniques are known for elliptic curves, see for example [BCI+10].

Using this notation, a random string $\mathsf{crs} = (r_1, \ldots, r_n)$ of size $\{0,1\}^{n \cdot \ell_G}$ represents n group elements $g_i \leftarrow \mathsf{SampleG}(1^\lambda; r_i)$. Further, as long as all $g_i' \in \mathbb{G}$ are also distributed independently and uniformly at random, the bit string $\mathsf{crs}' = (r_1', \ldots, r_n')$ will have a negligible statistical distance to crs, where $r_i' \leftarrow \mathsf{ExplainG}(1^\lambda, g_i')$. Thus, we can safely replace crs with crs' in security reductions. The above also applies for $\mathsf{pp} = (r_1, \ldots, r_n)$. Similarly, we can instantiate random oracles that map into the group $\mathbb{G}$ with random oracles mapping into $\{0,1\}^{\ell_G}$ this way.

Throughout, we write $g \leftarrow \mathbb{G}$ short for $g \leftarrow \mathsf{SampleG}(1^\lambda)$. Also, we write $\mathsf{crs} = (g_1, \ldots, g_n)$ for short, where g_i are drawn uniformly at random. Note that we can replace crs with $\mathsf{crs}' = (g_1', \ldots, g_n')$ in proofs, if all g_i' are also distributed independently uniform over $\mathbb{G}$. We extend the notation above to tuples of random elements drawn from different groups. These techniques also apply for mixed crs or pp over different groups.

3.5 Cryptographic Primitives

We give a brief overview of the primitives used in Chapters 5 and 6. We also discuss the random oracle model used throughout this manuscript.

3.5.1 Symmetric Primitives

We introduce symmetric encryption and pseudorandom function used in Chapters 5 and 6. We recall the random oracle model used throughout this manuscript.

Symmetric Encryption

A symmetric encryption scheme E is a tuple of PPT algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ such that

- $\mathsf{E.KeyGen}(1^\lambda)$: outputs a secret key K,
- $\mathsf{E.Enc}(\mathsf{K}, m)$: given secret key K and message m, outputs a ciphertext c,

– $\mathsf{E.Dec}(\mathsf{K}, c)$: given secret key K and ciphertext c, outputs a message m.

Below, we define the notions of correctness and IND-CPA security. Later, we assimilate the encryption scheme E with its encryption algorithm Enc, and sometimes leave key generation implicit. That is, we say Enc is an encryption scheme (with decryption algorithm Dec), and some key $\mathsf{K}_{\mathsf{Enc}}$ is generated for security parameter λ. We often write $\mathsf{Enc}_{\mathsf{K}_{\mathsf{Enc}}}(m)$ short for $\mathsf{Enc}(\mathsf{K}_{\mathsf{Enc}}, m)$.

Definition 3.11 (Correctness)**.** A symmetric encryption scheme E is *correct* if for all keys $\mathsf{K} \leftarrow \mathsf{E.KeyGen}(1^\lambda)$, messages m, and $c \leftarrow \mathsf{E.Enc}(\mathsf{K}, m)$, we have that $\mathsf{E.Dec}(\mathsf{K}, c) = m$.

Definition 3.12 (IND-CPA)**.** A symmetric encryption scheme is IND-CPA secure if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathrm{ind}}_{\mathcal{A}}(\lambda) = \left| \Pr \begin{bmatrix} \mathsf{K} \leftarrow \mathsf{E.KeyGen}(1^\lambda),\ (m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda), \\ \mathsf{coin} \leftarrow \{0,1\},\ c \leftarrow \mathsf{E.Enc}(\mathsf{K}, m_{\mathsf{coin}}) \end{bmatrix} : \begin{matrix} |m_0| = |m_1|, \\ \mathsf{coin} = \mathcal{A}^{\mathcal{O}}(c) \end{matrix} \right] - \frac{1}{2} \right| = \mathsf{negl}(\lambda),$$

where $\mathcal{O}$ is an oracle that on input m outputs $\mathsf{Enc}(\mathsf{K}, m)$.

Pseudorandom Function

A pseudorandom function (PRF) family $\mathcal{F} = \{F_{\mathsf{K}} : A \to B \mid \mathsf{K} \in \{0,1\}^\lambda\}$ is a family of functions. Below, we define PRF security.

Definition 3.13 (PRF security)**.** A PRF is secure if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathcal{A}}(\lambda) = \left| \Pr \begin{bmatrix} \mathsf{K} \leftarrow \{0,1\}^\lambda : \\ 1 = \mathcal{A}^{F_{\mathsf{K}}(\cdot)}(1^\lambda), \end{bmatrix} - \Pr \begin{bmatrix} R \leftarrow \{F : A \to B\} : \\ 1 = \mathcal{A}^{R(\cdot)}(1^\lambda), \end{bmatrix} \right| = \mathsf{negl}(\lambda).$$

Hash Function

A (keyed) hash function H is of the form $\mathsf{H}\colon \mathcal{K} \times \{0,1\}^* \mapsto \{0,1\}^\ell$. The key (i.e., the first input) to H is usually implicit, and part of the public parameters. We call H a *collision-resistant* hash function (CRHF), if it is hard to find a collision, i.e., two inputs m, m' such that $\mathsf{H}(m) = \mathsf{H}(m')$.

Random Oracle Model

Later, we often prove security of a scheme using in the random oracle model [BR93]. In the random oracle model, a hash function $\mathsf{H} : \{0,1\}^* \to D$ is idealized as truly random function. That is, for each fresh query x, the output $y \leftarrow \mathsf{H}(x)$ is sampled uniformly and independently at random from its output domain D. For repeated queries, the same output is provided.

While we cannot instantiate random oracles in a sound manner in the standard model [CGH98], H is usually instantiated with a hash function heuristically. This approach has become standard to prove security for efficient cryptographic schemes.

3.5.2 Asymmetric Primitives

Next, we define the asymmetric primitives we use in Chapter 7.

Commitment Scheme

A *commitment scheme* is a PPT algorithm $\mathsf{C} = \mathsf{C.Commit}$ such that

– $\mathsf{C.Commit}(\mathsf{pp}, m; r)$: given the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{C}}}$, message $m \in \mathcal{C}_{\mathsf{msg}}$ and randomness $r \in \mathcal{C}_{\mathsf{rnd}}$, computes a commitment $c \in \mathcal{C}_{\mathsf{com}}$, and outputs the pair (c, r),

Here, $\{0,1\}^{\ell_{\mathsf{C}}}$, $\mathcal{C}_{\mathsf{msg}}$, $\mathcal{C}_{\mathsf{rnd}}$, $\mathcal{C}_{\mathsf{com}}$, are public parameter, message, commitment randomness, and commitment spaces, respectively.[1] We do not explicitly define the opening algorithm since we can use the commitment randomness r as the decommitment (or opening) information and check if $c = \mathsf{Commit}(\mathsf{pp}, m; r)$ holds to verify that c is a valid commitment to message m.

Below, we first define the standard notions of binding and hiding, where note that correctness is implicit since we define the decommitment algorithm via Commit. While we define the computational variants below, we obtain the statistical variants by allowing the adversary $\mathcal{A}$ to be unbounded.

Definition 3.14 (Hiding)**.** A commitment scheme is *hiding* if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathrm{hide}}_{\mathcal{A}}(\lambda) = \left|\Pr\begin{bmatrix}\mathsf{pp} \leftarrow \{0,1\}^{\ell_{\mathsf{C}}},\ (m_0,m_1) \leftarrow \mathcal{A}(\mathsf{pp}), & \\ \mathsf{coin} \leftarrow \{0,1\}, (c,r) \leftarrow \mathsf{Commit}(\mathsf{pp}, m_{\mathsf{coin}}) & : \begin{matrix}\mathsf{coin} = \mathcal{A}(c) \wedge \\ m_0, m_1 \in \mathcal{C}_{\mathsf{msg}}\end{matrix}\end{bmatrix} - \frac{1}{2}\right| = \mathsf{negl}(\lambda).$$

Definition 3.15 (Binding)**.** A commitment scheme is *binding* if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathrm{bind}}_{\mathcal{A}}(\lambda) = \Pr\begin{bmatrix}\mathsf{pp} \leftarrow \{0,1\}^{\ell_{\mathsf{C}}},\ (m_0,m_1,r_0,r_1) \leftarrow \mathcal{A}(\mathsf{pp}), & \\ m_0 \neq m_1 \in \mathcal{C}_{\mathsf{msg}} \wedge r_0, r_1 \in \mathcal{C}_{\mathsf{rnd}} & : c_0 = c_1 \\ \wedge (c_b, r_b) = \mathsf{Commit}(\mathsf{pp}, m_b; r_b), b \in \{0,1\} & \end{bmatrix} = \mathsf{negl}(\lambda).$$

We further define rerandomizability. This allows rerandomizing a commitment to a new commitment on the same message. Moreover, we require that the new commitment has enough min-entropy for a random rerandomization randomness.

Definition 3.16 (Rerandomizability)**.** A commitment scheme is *rerandomizable* if there exist PPT algorithms $(\mathsf{RerandCom}, \mathsf{RerandRand})$ such that

- $\mathsf{RerandCom}(\mathsf{pp}, c, \Delta r)$: given the public parameter pp, a commitment $c \in \mathcal{C}_{\mathsf{com}}$, and a rerandomization randomness $\Delta r \in \mathcal{C}_{\mathsf{rnd}}$, *deterministically* outputs a rerandomized commitment $c' \in \mathcal{C}_{\mathsf{com}}$,
- $\mathsf{RerandRand}(\mathsf{pp}, c, m, r, \Delta r)$: on input of the public parameter pp, a commitment $c \in \mathcal{C}_{\mathsf{com}}$, a message $m \in \mathcal{C}_{\mathsf{msg}}$, a randomness r, and a rerandomization randomness $\Delta r \in \mathcal{C}_{\mathsf{rnd}}$, outputs a rerandomized randomness r',

and the following holds:

- for all $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{C}}}$, $m \in \mathcal{C}_{\mathsf{msg}}$, $(c,r) \leftarrow \mathsf{Commit}(\mathsf{pp}, m)$, and $\Delta r \in \mathcal{C}_{\mathsf{rnd}}$, if we compute $c' = \mathsf{RerandCom}(\mathsf{pp}, c, \Delta r)$ and $r' \leftarrow \mathsf{RerandRand}(\mathsf{pp}, c, m, r, \Delta r)$, then it holds that $c' = c''$, where $(c'', r') = \mathsf{Com}(\mathsf{pp}, m; r')$, and
- we have

$$\max_{c, c' \in \mathcal{C}_{\mathsf{com}}} \Pr[\mathsf{pp} \leftarrow \{0,1\}^{\ell_{\mathsf{C}}}, \Delta r \leftarrow \mathcal{C}_{\mathsf{rnd}} : c' = \mathsf{RerandCom}(\mathsf{pp}, c, \Delta r)] = \mathsf{negl}(\lambda).$$

We note that any natural additive homomorphic commitment scheme satisfies rerandomizability if we define $\mathsf{RerandCom}(\mathsf{pp}, c, \Delta r) = c + \mathsf{Commit}(\mathsf{pp}, 0; \Delta r) = c'$. Observe that if $c = \mathsf{Commit}(\mathsf{pp}, m; r)$, the rerandomziaed randomness is $r' = r + \Delta r$ since $c' = \mathsf{Commit}(\mathsf{pp}, m; r')$ by the homomorphic property. Moreover, c' has high min-entropy since $\mathsf{Commit}(\mathsf{pp}, 0)$ has high min-entropy for most natural commitment schemes. Finally, we note that while a computational variant of the high min-entropy property suffices for our generic construction, we use the statistical variant for simplicity and because our instantiation satisfies it.

[1] We assume uniform public parameters to improve readability. In our context, it is sufficient that the distribution of the public parameters is explainable (see section 3.4.1).

Signature Scheme

We consider *deterministic* signature schemes; a scheme where the randomness of the signing algorithm is derived from the secret key and message. We can derandomize any signature scheme by using a pseudorandom function for generating the randomness used in the signing algorithm (see for example [Kat10]). Formally, a signature scheme is a tuple of PPT algorithms $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ such that

- $\mathsf{KeyGen}(1^\lambda)$: generates a verification key vk and a signing key sk,
- $\mathsf{Sign}(\mathsf{sk}, m)$: given a signing key sk and a message $m \in \mathcal{S}_{\mathsf{msg}}$, *deterministically* outputs a signature σ,
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: given a verification key pk and a signature σ on message m, *deterministically* outputs a bit $b \in \{0, 1\}$.

Here, $\mathcal{S}_{\mathsf{msg}}$ is the message space. We define the standard notion of correctness and `EUF-CMA` security

Definition 3.17 (Correctness)**.** A signature scheme is *correct*, if for all $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $m \in \mathcal{S}_{\mathsf{msg}}$, and $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$, it holds that $\mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1$.

Definition 3.18 (EUF-CMA)**.** A signature scheme is *`EUF-CMA`* if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathrm{euf}}_{\mathcal{A}}(\lambda) = \Pr\left[\begin{matrix}(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(\mathsf{vk})\end{matrix} \quad : m \notin L \wedge \ \mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1\right] = \mathsf{negl}(\lambda),$$

where L is the list of messages $\mathcal{A}$ queried to the Sign-oracle.

(Partially) Blind Signature Scheme

A partially blind signature scheme is a tuple of PPT algorithms $\mathsf{PBS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ such that

- $\mathsf{KeyGen}(1^\lambda)$: generates the verification key bvk and signing key bsk,
- $\mathsf{User}(\mathsf{bvk}, \boxed{t}, m)$: given verification key bvk, common message $t \in \mathcal{BS}_t$, and message $m \in \mathcal{BS}_{msg}$, outputs a first message ρ_1 and a state st,
- $\mathsf{Signer}(\mathsf{bsk}, \boxed{t}, \rho_1)$: given signing key bsk, common message $t \in \mathcal{BS}_t$, and first message ρ_1, outputs a second message ρ_2,
- $\mathsf{Derive}(\mathsf{st}, \boxed{t}, \rho_2)$: given state st, common message $t \in \mathcal{BS}_t$, and second message ρ_2, outputs a signature σ,
- $\mathsf{Verify}(\mathsf{bvk}, \boxed{t}, m, \sigma)$: given verification key bvk, common message $t \in \mathcal{BS}_t$, and signature σ on message $m \in \mathcal{BS}_{msg}$, outputs a bit $b \in \{0, 1\}$.

Here, $\mathcal{BS}_t$ and $\mathcal{BS}_{msg}$ are the message and tag spaces, respectively. In case the common message $t \in \mathcal{BS}_t$ is omitted from the syntax (or alternatively, always set to a fixed value), then we call the scheme to be a blind signature BS. We consider the standard security notions for blind signatures [JLO97]. Below, we define correctness, partial blindness under *malicious keys*, and one-more unforgeability of a (partially) blind signature scheme. The definition for blind signature can be recovered by ignoring t, denoted with a box. Moreover, we assume the state is kept implicit in the following for better readability.

Definition 3.19 (Correctness)**.** A partially blind signature scheme is *correct*, if for all messages $(\boxed{t}, m) \in \mathcal{BS}_t \times \mathcal{BS}_{msg}$, $(\mathsf{bvk}, \mathsf{bsk}) \leftarrow \mathsf{KeyGen}(1^\lambda), (\rho_1, \mathsf{st}) \leftarrow \mathsf{User}(\mathsf{bvk}, \boxed{t}, m), \rho_2 \leftarrow \mathsf{Signer}(\mathsf{bsk}, \boxed{t}, \rho_1), \sigma \leftarrow \mathsf{Derive}(\mathsf{st}, \boxed{t}, \rho_2)$, it holds that $\mathsf{Verify}(\mathsf{bvk}, \boxed{t}, m, \sigma) = 1$.

Definition 3.20 (Partial Blindness Under Malicious Keys)**.** A partially blind signature scheme is *blind under malicious keys* if for any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathrm{blind}}_{\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{l|l} & (\mathsf{bvk}, \boxed{t}, m_0, m_1) \leftarrow \mathcal{A}(1^\lambda),\ \mathsf{coin} \leftarrow \{0,1\}, \\ & (\rho_{1,b}, \mathsf{st}_b) \leftarrow \mathsf{User}(\mathsf{bvk}, \boxed{t}, m_b) \text{ for } b \in \{0,1\}, \\ & (\rho_{2,\mathsf{coin}}, \rho_{2,1-\mathsf{coin}}) \leftarrow \mathcal{A}(\rho_{1,\mathsf{coin}}, \rho_{1-\mathsf{coin}}), \\ & \sigma_b \leftarrow \mathsf{Derive}(\mathsf{st}_b, \boxed{t}, \rho_{2,b}) \text{ for } b \in \{0,1\}, \\ & \textit{if } \exists b \text{ s.t. } \mathsf{Verify}(\mathsf{bvk}, \boxed{t}, m_b, \sigma_b) = 0\text{:} \\ & \quad \textit{then } \sigma_0 = \sigma_1 = \bot, \end{array} : \mathsf{coin} = \mathcal{A}(\sigma_0, \sigma_1) \right] - \frac{1}{2} = \mathsf{negl}(\lambda).$$

Definition 3.21 (One-more Unforgeability)**.** A blind signature scheme is *one-more unforgeable* if for any $Q = \mathsf{poly}(\lambda)$ and PPT adversary $\mathcal{A}$ that for each common message $\boxed{t}$ makes at most Q signing queries containing the same $\boxed{t}$, we have

$$\mathsf{Adv}^{\mathrm{omuf}}_{\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{l} (\mathsf{bvk}, \mathsf{bsk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \{\boxed{t}, (m_i, \sigma_i)\}_{i \in [Q+1]} \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{bvk}) \end{array} : \begin{array}{l} \forall i \neq j \in [Q+1] : m_i \neq m_j \\ \wedge\ \mathsf{Verify}(\mathsf{bvk}, \boxed{t}, m_i, \sigma_i) = 1 \end{array} \right] = \mathsf{negl}(\lambda),$$

where $\mathcal{O} = \mathsf{Signer}(\mathsf{bsk}, \boxed{\cdot}, \cdot)$ is a signing oracle.

Σ-Protocol

Let R be an NP relation with statements x and witnesses w. We denote by $\mathscr{L}_{\mathsf{R}} = \{x \mid \exists w \text{ s.t. } (x,w) \in \mathsf{R}\}$ the language induced by R. A Σ-protocol for an NP relation R for language $\mathscr{L}_{\mathsf{R}}$ is a tuple of PPT algorithms $\Sigma = (\mathsf{Init}, \mathsf{Chall}, \mathsf{Resp}, \mathsf{Verify})$ such that

- $\mathsf{Init}(x, w)$: given a statement $x \in \mathscr{L}_{\mathsf{R}}$, and a witness w such that $(x,w) \in \mathsf{R}$, outputs a first flow message (i.e., commitment) α and a state st, where we assume st includes x, w,
- $\mathsf{Chall}(1^\lambda)$: samples a challenge $\beta \leftarrow \mathcal{CH}$,
- $\mathsf{Resp}(\mathsf{st}, \beta)$: given a state st and a challenge $\beta \in \mathcal{CH}$, outputs a third flow message (i.e., response) γ,
- $\mathsf{Verify}(x, \alpha, \beta, \gamma)$: given a statement $x \in \mathscr{L}_{\mathsf{R}}$, a commitment α, a challenge $\beta \in \mathcal{CH}$, and a response γ, outputs a bit $b \in \{0,1\}$.

Here, $\mathcal{CH}$ denotes the challenge space. We call the tuple (α, β, γ) the *transcript* and say that they are *valid for* x if $\mathsf{Verify}(x, \alpha, \beta, \gamma)$ outputs 1. When the context is clear, we simply say it is valid and omit x. Also, we omit the parameter 1^λ in the following from Chall for readability.

We first define the standard notions of correctness, honest-verifier zero-knowledge, and 2-special soundness.

Definition 3.22 (Correctness)**.** A Σ-protocol is *correct*, if for all $(x,w) \in \mathsf{R}$, $(\alpha, \mathsf{st}) \leftarrow \mathsf{Init}(x,w)$, $\beta \in \mathcal{CH}$, and $\gamma \leftarrow \mathsf{Resp}(\mathsf{st}, \beta)$, it holds that $\mathsf{Verify}(x, \alpha, \beta, \gamma) = 1$.

Definition 3.23 (High Min-Entropy)**.** A Σ-protocol has *high min-entropy* if for all $(x,w) \in \mathsf{R}$ and (possibly unbounded) adversary $\mathcal{A}$, it holds that

$$\Pr[(\alpha, \mathsf{st}) \leftarrow \mathsf{Init}(x,w), \alpha' \leftarrow \mathcal{A}(1^\lambda) : \alpha = \alpha'] = \mathsf{negl}(\lambda).$$

Definition 3.24 (HVZK)**.** A Σ-protocol is *honest-verifier zero-knowledge* (HVZK), if there exists a PPT zero-knowledge simulator Sim such that the distributions of $\mathsf{Sim}(x, \beta)$ and the honestly generated transcript with Init initialized with (x, w) are computationally indistinguishable for any $x \in \mathscr{L}_{\mathsf{R}}$, and $\beta \in \mathcal{CH}$, where the honest execution is conditioned on β being used as the challenge.

Definition 3.25 (2-Special Soundness)**.** A Σ-protocol is 2-*special sound*, if there exists a *deterministic* PT extractor Ext such that given two valid transcripts $\{(\alpha, \beta_b, \gamma_b)\}_{b\in[2]}$ for statment x with $\beta_0 \neq \beta_1$, along with x, outputs a witness w such that $(x, w) \in \mathsf{R}$.

Note that in the above, two valid transcripts for x with the same commitment and different challenges imply that statement x is in $\mathscr{L}_{\mathsf{R}}$. That is, we do not guarantee x to lie in $\mathscr{L}_{\mathsf{R}}$ when invoking Ext. While subtle, this allows us to invoke Ext properly within the security proof even if the reduction cannot decide if the statement x output by the adversary indeed lies in $\mathscr{L}_{\mathsf{R}}$.

In the following, we propose a new notion of f*-unique extraction*. The notion is similar to the *unique response* property [Fis05, Unr12] which requires that given an incomplete transcript (α, β), there is at most one response γ such that the transcript $\tau = (\alpha, \beta, \gamma)$ is valid. We relax this in two ways. First, we require that given a transcript τ and another challenge β', it is impossible to find two different responses γ_0, γ_1, such $w_0 \neq w_1$, where w_b is the witness extracted from τ and $\tau_b = (\alpha, \beta', \gamma_b)$. We further relax this by only requiring this property for a portion of the witness, defined by a function f, i.e., we require $f(w_0) \neq f(w_1)$ instead of $w_0 \neq w_1$.

While it may seem like an unnatural property, this is satisfied by many natural Σ-protocols. In particular, if the first flow α contains a perfectly binding commitment $c = \mathsf{Commit}(f(w); r)$ to $f(w)$, and the extractor extracts the appropriate r, then the Σ-protocol has f-unique extraction. We remark also that a statistical variant of f-unique extraction is sufficient for our purpose. We choose the definition below for simplicity and because our instantiation satisfies it. See section 7.2 for more details and concrete example of f-unique extraction.

Definition 3.26 (f-Unique Extraction)**.** For a (possibly non-efficient) function f, a Σ-protocol Σ has f*-unique extraction* if for any statement x, any transcript $\tau = (\alpha, \beta, \gamma)$ and challenge $\beta' \neq \beta$, there is no γ_0, γ_1, such that for $\tau_b = (\alpha, \beta', \gamma_b)$, we have

$$f(\mathsf{Ext}(x, \tau, \tau_0)) \neq f(\mathsf{Ext}(x, \tau, \tau_1)).$$

Non-Interactive Zero Knowledge

Given a witness w for statement x, a non-interactive zero-knowledge (NIZK) proof system allows a prover to generate a proof π that attests that she *knows* some w' such that $(w', x) \in \mathsf{R}$. Proofs π can be verified for statement x *without* revealing anything but that the statement is true. Here, we quantify "knowledge of the witness" either via *adaptive knowledge soundness* or *online-extractability*. The former informally states that if an algorithm $\mathcal{A}$ can generate a valid proof-statement pair (x, π), then there exists some extractor that when given black-box access to $\mathcal{A}$, can extract some witness w s.t. $(x, w) \in \mathsf{R}$. The latter requires that the witness w can be extracted from (x, π) "on-the-fly" without disrupting $\mathcal{A}$. In this context, we require some random oracle H on which proving and verification rely. Further, we assume that the prover and verifier are supplied with a common random string crs. As we later aim to avoid such a crs in our blind signature framework, the crs will be the output of a random oracle.

More formally, an NIZK for a relation R is a tuple of PPT algorithms $(\mathsf{Prove}^{\mathsf{H}}, \mathsf{Verify}^{\mathsf{H}})$ with oracle access such that:

- $\mathsf{Prove}^{\mathsf{H}}(\mathsf{crs}, x, w)$: receives a common random string $\mathsf{crs} \in \{0,1\}^{\ell}$, a statement x and a witness w, and outputs a proof π,

– $\mathsf{Verify}^{\mathsf{H}}(\mathsf{crs}, x, \pi)$: receives a statement x and a proof π, and outputs a bit $b \in \{0,1\}$.

An NIZK satisfies correctness and zero-knowledge, where we call it *adaptive knowledge sound* if it satisfies definition 3.29 and *online-extractable* if it satisfies definition 3.31.

Definition 3.27 (Correctness)**.** An NIZK is *correct* if for any $\mathsf{crs} \in \{0,1\}^\ell$, $(x,w) \in \mathsf{R}$, and $\pi \leftarrow \mathsf{Prove}^{\mathsf{H}}(\mathsf{crs}, x, w)$, it holds that $\mathsf{Verify}^{\mathsf{H}}(\mathsf{crs}, x, \pi) = 1$.

Definition 3.28 (Zero-Knowledge)**.** An NIZK is *zero-knowledge* if there exists a PPT simulator $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{H}}, \mathsf{Sim}_{\pi})$ such that for any PPT adversary $\mathcal{A}$, it holds that

$$\mathsf{Adv}^{\mathrm{zk}}_{\mathcal{A}}(\lambda) = \left| \Pr\left[\mathcal{A}^{\mathsf{H},\mathcal{P}}(\mathsf{crs}) = 1\right] - \Pr\left[\mathcal{A}^{\mathsf{Sim}_{\mathsf{H}},\mathcal{S}}(\mathsf{crs}) = 1\right] \right| = \mathsf{negl}(\lambda),$$

where $\mathcal{P}$ and $\mathcal{S}$ are oracles that on input (x,w) return $\perp$ if $(x,w) \notin \mathsf{R}$, and else output $\mathsf{Prove}^{\mathsf{H}}(\mathsf{crs}, x, w)$ or $\mathsf{Sim}_{\pi}(\mathsf{crs}, x)$ respectively. Note that the probability is taken over $\mathsf{crs} \leftarrow \{0,1\}^\ell$ and the random choices of H, and both $\mathsf{Sim}_{\mathsf{H}}$ and Sim_{π} have a shared state.

We define adaptive knowledge soundness. We remark that the soundness relation $\mathsf{R}_{\mathsf{lax}}$ can be different from the (correctness) relation R. We are typically interested in $\mathsf{R} \subseteq \mathsf{R}_{\mathsf{lax}}$ and call $\mathsf{R}_{\mathsf{lax}}$ the *relaxed* relation.

Definition 3.29 (Adaptive Knowledge Soundness)**.** An NIZK is *adaptively knowledge sound* for relation $\mathsf{R}_{\mathsf{lax}}$ if there exists positive polynomials $p_{\mathsf{T}}, p_{\mathsf{P}}$ and a PPT algorithm Ext such that for any $\mathsf{crs} \in \{0,1\}^\ell$, given oracle access to any PPT adversary $\mathcal{A}$ (with explicit random tape ρ) that makes $Q_H = \mathsf{poly}(\lambda)$ random oracle queries with

$$\Pr[(x,\pi) \leftarrow \mathcal{A}^{\mathsf{H}}(\mathsf{crs}; \rho) : \mathsf{Verify}^{\mathsf{H}}(\mathsf{crs}, x, \pi) = 1] \geq \mu(\lambda),$$

we have

$$\Pr\begin{bmatrix} (x,\pi) \leftarrow \mathcal{A}^{\mathsf{H}}(\mathsf{crs}; \rho), \\ w \leftarrow \mathsf{Ext}(\mathsf{crs}, x, \pi, \rho, \vec{h}) \end{bmatrix} : (x,w) \in \mathsf{R}_{\mathsf{lax}} \Bigg] \geq \frac{\mu(\lambda) - \mathsf{negl}(\lambda)}{p_{\mathsf{P}}(\lambda, Q_H)},$$

where $\vec{h}$ are the outputs of H, and the probability is over the random tape ρ and the random choices of H. Also, we require that the runtime of Ext is bounded by $p_{\mathsf{T}}(\lambda, Q_H) \cdot \mathsf{Time}(\mathcal{A})$.

Remark 3.30 (Fiat-Shamir)**.** The Fiat-Shamir transformation can applied to a Σ-protocol (or more generally an interactive proof system) to compile it into an NIZK. Non-interactivity is achieved by computing the challenge $\beta \leftarrow \mathsf{H}(x, \alpha)$ deterministically from the output of the first message α of the Σ-protocol and the statement x. Then, the last message γ is computed given β, and the transcript $\pi = (\alpha, \beta, \gamma)$ is prover's output. Roughly, special soundness of a Σ-protocol translates to adaptive knowledge soundness of the NIZK obtained via Fiat-Shamir in the ROM.

Next, we define (multi)-online extractability similarly to [dK22]. We consider a slightly simplified definition where the runtime of the extractor Ext does not depend on the advantage μ of the adversary $\mathcal{A}$.

Definition 3.31 ((Multi)-Online Extractability)**.** An NIZK is *online-extractable* if for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator SimCRS and extractor Ext, such that

CRS Indistinguishability. For any PPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathrm{crs}}_{\mathcal{A}}(\lambda) = |\Pr[\mathcal{A}^{\mathsf{H}}(\mathsf{crs}) = 1] - \Pr[\mathcal{A}^{\mathsf{H}}(\overline{\mathsf{crs}}) = 1]| = \mathsf{negl}(\lambda),$$

where $\mathsf{crs} \leftarrow \{0,1\}^\ell$ and $(\overline{\mathsf{crs}}, \mathsf{td}) \leftarrow \mathsf{SimCRS}(1^\lambda)$.

Online Extractability. There exists positive polynomials $p_{\mathsf{T}}, p_{\mathsf{P}}$ such that for any $Q_H = \mathsf{poly}(\lambda)$ and PPT adversary $\mathcal{A}$ that makes at most Q_H random oracle queries with

$$\Pr\left[(\overline{\mathsf{crs}}, \mathsf{td}) \leftarrow \mathsf{SimCRS}(1^\lambda), \{(x_i, \pi_i)\}_i \leftarrow \mathcal{A}^{\mathsf{H}}(\overline{\mathsf{crs}}) : \forall i : \mathsf{Verify}^{\mathsf{H}}(\mathsf{crs}, x_i, \pi_i) = 1\right] \geq \mu(\lambda),$$

where $i \in [Q_S]$, it holds that

$$\Pr\left[\begin{array}{c}(\overline{\mathsf{crs}}, \mathsf{td}) \leftarrow \mathsf{SimCRS}(1^\lambda), \{(x_i, \pi_i)\}_i \leftarrow \mathcal{A}^{\mathsf{H}}(\overline{\mathsf{crs}}), \\ \{w_i \leftarrow \mathsf{Ext}(\overline{\mathsf{crs}}, \mathsf{td}, x_i, \pi_i)\}_i\end{array} : \begin{array}{c}\forall i : (x_i, w_i) \in \mathsf{R} \\ \wedge\ \mathsf{Verify}^{\mathsf{H}}(\overline{\mathsf{crs}}, x_i, \pi_i) = 1\end{array}\right] \geq \frac{\mu(\lambda) - \mathsf{negl}(\lambda)}{p_{\mathsf{P}}(\lambda, Q_H)},$$

where the runtime of Ext is upper bounded by $p_{\mathsf{T}}(\lambda, Q_H) \cdot \mathsf{Time}(\mathcal{A})$ and again $i \in [Q_S]$.

3.6 Searchable Symmetric Encryption

We recall the notion of searchable symmetric encryption (SSE). Throughout, a *database* $\mathsf{DB} = \{(w_i, (\mathsf{id}_1, ..., \mathsf{id}_{\ell_i}))\}_{i=1}^K$ is a set of K pairs $(w_i, (\mathsf{id}_1, ..., \mathsf{id}_{\ell_i}))$, where w_i is a *keyword*, and $(\mathsf{id}_1, ..., \mathsf{id}_{\ell_i})$ is a tuple of ℓ_i document identifiers matching keyword w_i. The number of distinct keywords is K. We write $\mathsf{DB}(w_i) = (\mathsf{id}_1, ..., \mathsf{id}_{\ell_i})$ for the list of identifiers matching keyword w_i. The *size* of the database DB is the number of distinct keyword-identifier pairs (w_i, id_j), with $\mathsf{id}_j \in \mathsf{DB}(w_i)$. It is equal to $\sum_{i=1}^K \ell_i$.

We will usually assume that there is an upper bound W on the total number of keywords: $K \leq W$; and an upper bound N on the size of the database: $\sum_{i=1}^k \ell_i \leq N$. Throughout the thesis, the integer p denotes the page size. We treat p as a variable independent of the size of the database N, in line with previous work. Our complexity analysis holds in the RAM model of computation, where accessing a random memory word costs unit time. Memory is counted in the number of memory words, which are assumed to be of size $\mathcal{O}(\lambda)$ bits, as is common in the literature.

Before giving the formal definition, let us sketch how a dynamic SSE scheme operates. The client stores a small state st, while the server stores the encrypted database EDB. The client calls $\Sigma.\mathsf{Setup}$ to initialize both states, on input an initial plaintext database DB. To search the database on keyword w, the client initiates the protocol $\Sigma.\mathsf{Search}$ on input w, and eventually obtains the list of matching identifiers $\mathsf{DB}(w)$. As a side effect, $\Sigma.\mathsf{Search}$ may also update the client and server states. Similarly, to add a new keyword-identifier pair (w, id) to the encrypted database, the client initiates $\Sigma.\mathsf{Update}$ on the corresponding input.

A *dynamic searchable symmetric encryption* scheme Σ is a 4-tuple of PPT algorithms (KeyGen, $\mathsf{Setup}, \mathsf{Search}, \mathsf{Update}$):

- $\Sigma.\mathsf{KeyGen}(1^\lambda)$: Takes as input the security parameter λ and outputs client secret key K.
- $\Sigma.\mathsf{Setup}(\mathsf{K}, N, W, \mathsf{DB})$: Takes as input the client secret key K, upper bounds N on the database size and W on the number of keywords, and a database DB. Outputs an encrypted database EDB and client state st.
- $\Sigma.\mathsf{Search}(\mathsf{K}, w, \mathsf{st}; \mathsf{EDB})$: The client receives as input the secret key K, keyword w and state st. The server receives as input the encrypted database EDB. Outputs some data d, an updated state st' for the client, and an updated encrypted database EDB' for the server.
- $\Sigma.\mathsf{Update}(\mathsf{K}, w, \mathsf{id}, \mathsf{op}, \mathsf{st}; \mathsf{EDB})$: The client receives as input the secret key K, a keyword-identifier pair (w, id), an operation $\mathsf{op} \in \{\mathsf{del}, \mathsf{add}\}$, and state st. The server receives the encrypted database EDB. Outputs an updated state st' for the client, and an updated encrypted database EDB' for the server.

We denote by Search_C (resp. Update_C) the client side of the protocol Search (resp. Update), and by Search_S (resp. Update_S) its server counterpart. We may omit W in the input of Setup if it is not used.

For concision, in the remainder, the client state st will be omitted in the notation. As is standard in SSE literature, we sometimes assume that keywords are preprocessed via a PRF by the client to avoid clutter in the description of our schemes. That is, $w = \mathsf{PRF}_K(\mathsf{H}(k))$ where k is the actual keyword, for some PRF key K known only to the client.

Epochs. To facilitate the description of our schemes in Chapter 6, it is convenient to conceptually partition update queries issued by the client into sequences of W consecutive updates. The time frame corresponding to one such sequence of W updates is called an *epoch*. More precisely, the k-th epoch comprises all updates and searches that are performed between the $((k-1)\cdot W+1)$-th and $(k\cdot W)$-th update. Looking ahead, update queries belonging to the current epoch will typically be preprocessed together on the client side, then progressively pushed to the server during the next epoch.

Correctness. Informally, correctness asks that at the outcome of a Search protocol on keyword w, the client should obtain exactly the identifiers of documents matching w. The following definition asks for perfect correctness. In some cases, we may allow correctness to fail as long as the probability of failure is negligible.

Definition 3.32 (Correctness)**.** An SSE scheme Σ is *correct* if for all sufficiently large $W, N \in \mathbb{N}$, for all databases DB, and all sequences of search and update operations, provided at most K keywords are used, and the size of the database remains at most N at all times, letting $\mathsf{K} \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$ and $\mathsf{EDB} \leftarrow \Sigma.\mathsf{Setup}(\mathsf{K}, N, W, \mathsf{DB})$, at the outcome of a search query on keyword w, the client obtains exactly the identifiers of documents matching keyword w at query time. (That is, documents matching w in the initial database DB or added by an update query matching w, and not subsequently deleted.)

Security. We use the standard semantic security notion from [CGKO06]. The server is modeled as an honest-but-curious adversary. Intuitively, security asks that the information learned by the server in the course of the scheme's execution is no more than a specified leakage. The allowed leakage is expressed by a *leakage function*, composed of setup leakage $\mathcal{L}_{\mathsf{Stp}}$, search leakage $\mathcal{L}_{\mathsf{Srch}}$, and update leakage $\mathcal{L}_{\mathsf{Updt}}$. The intent is that, when executing Setup on input x, the server should learn no more than $\mathcal{L}_{\mathsf{Stp}}(x)$. To formally capture that requirement, the security definition asks that there exists a PPT simulator that can simulate the view of the server, taking as input only $\mathcal{L}_{\mathsf{Stp}}(x)$. The same goes for Search and Update.

Formally, we define two games, SSEReal and SSEIdeal. In both games, the adversary first chooses a database DB. In SSEReal, the encrypted database EDB is then generated by $\mathsf{Setup}(\mathsf{K}, N, \mathsf{DB})$. In SSEIdeal, EDB is instead generated by a stateful PPT algorithm Sim called the *simulator*, on input $\mathcal{L}_{\mathsf{Stp}}(\mathsf{DB}, N)$. After receiving EDB, the adversary adaptively issues search and update queries. In SSEReal, all queries are answered honestly. In SSEIdeal, search queries on keyword w are simulated by Sim on input $\mathcal{L}_{\mathsf{Srch}}(w)$, and update queries for operation op, keyword w, and identifier id are simulated by Sim on input $\mathcal{L}_{\mathsf{Updt}}(\mathsf{op}, w, \mathsf{id})$. At the end of the game, the adversary outputs a bit b.

Definition 3.33 (Semantic Security)**.** Let Σ be an SSE scheme and let $\mathcal{L} = (\mathcal{L}_{\mathsf{Stp}}, \mathcal{L}_{\mathsf{Srch}}, \mathcal{L}_{\mathsf{Updt}})$ be a leakage function. The scheme Σ is *$\mathcal{L}$-adaptively secure* if for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator Sim such that:

$$|\Pr[\mathrm{SSEReal}_{\Sigma,\mathcal{A}}(\lambda) = 1] - \Pr[\mathrm{SSEIdeal}_{\Sigma,\mathsf{Sim},\mathcal{L},\mathcal{A}}(\lambda) = 1]| = \mathsf{negl}(\lambda).$$

Leakage Patterns

To facilitate the description of leakage functions, we make use of the following standard notions from the literature (see [Bos16] for more details).

- The *search pattern* $\mathsf{sp}(w)$ for keyword w is the sequence of identifiers of previous search queries on w.
- The *update pattern* $\mathsf{up}(w)$ for keyword w is the sequence of identifiers of previous update queries on w.
- The *query pattern* $\mathsf{qp}(w) = (\mathsf{sp}(w), \mathsf{up}(w))$ is the combination of search and update patterns.

Forward Security and Backward Security. Forward security was introduced and formalized in [SPS14] and [Bos16], respectively. In this manuscript, we consider a strictly stronger notion of forward security where updates leak no information.

Definition 3.34 (Forward Security)**.** An SSE scheme with leakage $\mathcal{L} = (\mathcal{L}_{\mathsf{Stp}}, \mathcal{L}_{\mathsf{Srch}}, \mathcal{L}_{\mathsf{Updt}})$ is *forward-secure* if $\mathcal{L}_{\mathsf{Updt}}(\mathsf{op}, w, \mathsf{id}) = \bot$.

The notion of backward security was formalized in [BMO17], and restricts the leakage incurred by deletions. Backward security is not the main focus of this manuscript, and we refer the reader to [BMO17] for the formal definition. We consider type-II backwards security in Chapter 6, which requires that search queries leak the documents currently matching w, when they were inserted, and when all the updates on w happened (but not their content). Note that schemes with leakage $\mathcal{L}^{\mathsf{fs}}$ are forward-secure and type-II backward-secure.

Remark on Deletions

Generally, we present our SSE constructions without deletions to improve readability. This however does not reduce their functionality, as all schemes can be extended to support deletions with the generic framework of [Bos16]. Intuitively, given an SSE scheme Σ_{add} that supports additions, it allows us to construct a scheme Σ that supports additions and deletions with the same leakage. For this, two instantiations of Σ_{add} are used, Σ_0 for additions and Σ_1 for deletions. Added identifiers are inserted into Σ_0 while deleted keyword-identifier pairs are inserted into Σ_1. For each search request, the client queries Σ_0 and Σ_1 and retrieves identifier lists L_0 and L_1 respectively. The result of the search request then is $L_0 \setminus L_1$. Clearly, if Σ_{add} has leakage $\mathcal{L}^{\mathsf{fs}}$, then Σ also has leakage $\mathcal{L}^{\mathsf{fs}}$.

Efficiency Metrics

We recall common efficiency measures for SSE introduced in [CT14]. Locality and read efficiency quantify the memory efficiency of SSE schemes on hard disk drives. Storage efficiency quantifies the storage overhead compared to a plaintext database.

Definition 3.35 (Read Pattern)**.** Regard server-side storage as an array of memory locations, containing the encrypted database EDB. When processing search query $\mathsf{Search}(\mathsf{K}, w_i, \mathsf{st}_i; \mathsf{EDB}_i)$ or update query $\mathsf{Update}(\mathsf{K}, (w_i, L_i), \mathsf{op}_i, \mathsf{st}_i; \mathsf{EDB}_i)$, the server accesses memory locations $m_1, \ldots, m_h$. We call these locations the *read pattern* and denote it with $\mathsf{RdPat}(\mathsf{op}_i, \mathsf{in}_i)$.

Definition 3.36 (Locality)**.** A SSE scheme has locality L if for any λ, DB, N, sequence S, and any i, $\mathsf{RdPat}(\mathsf{op}_i, \mathsf{in}_i)$ consists of at most L disjoint intervals.

Definition 3.37 (Read Efficiency)**.** A SSE scheme has read efficiency R if for any λ, DB, N, sequence S, and any i, $|\mathsf{RdPat}(\mathsf{op}_i, \mathsf{in}_i)| \leq R \cdot P$, where P is the number of memory locations needed to store all (added and deleted) document indices matching keyword w_i in plaintext (by concatenating indices).

Definition 3.38 (Storage Efficiency)**.** A SSE scheme has storage efficiency E if for any λ, DB, N, sequence S, and any i, $|\mathsf{EDB}_i| \leq E \cdot |DB_i|$.

Chapter

4 Weighted Hashing

In this chapter, we propose and analyze hashing variants based on classical balls-into-bins processes in the weighted setting, including one-choice allocation, two-choice allocation and cuckoo hashing. We prove upper bounds on the size of the most loaded bin that hold with overwhelming probability.

Chapter content

4.1 Introduction 39
4.1.1 Our Contributions 41
4.2 One-Choice Allocation 42
4.3 Two-Choice Allocation 43
4.3.1 Overview 43
4.3.2 Analysis 45
4.4 Cuckoo Hashing 49
4.4.1 Overview 49
4.4.2 Analysis 51

4.1 Introduction

In this chapter, we investigate various properties of single-choice games as well as multiple-choice games in the context of weighted balls. That is, we want to allocate n balls into m bins, where each ball b_i has a weight $w_i \in [0,1]_{\mathbb{R}}$, and its destination bins are chosen at random. We denote by $w_{\mathsf{tot}} = \sum_i w_i$ the total weight. If the weight distribution can change throughout the process, the value w_{tot} denotes an upper bound on the total weight. We consider one-choice allocation (1C), two-choice allocation (2C), and cuckoo hashing. These algorithms have found many applications in cryptography, especially in the literature related to oblivious algorithms.

In what follows, *with overwhelming probability* is synonymous with *except with negligible probability*, whereas *with high probability* simply means with probability close to 1 in some sense, but not necessarily overwhelming.

One-choice allocation. In one-choice allocation, n balls with uniform weight are thrown into n bins. Each ball is inserted into a single bin chosen independently and uniformly at random, e.g., by hashing an identifier of the ball. A standard analysis using Chernoff bounds shows that, at the outcome of the insertion process, the most loaded bin contains at most $\mathcal{O}(\log n)$ balls with high probability [JK77], and at most $\mathcal{O}(\delta(n)\log n)$ balls with overwhelming probability, for any $\delta = \omega(1)$.

In the weighted setting, each *weighted* ball is placed in a random bin. Again, we are interested in the load of the most loaded bin, where the load is the total weight of the balls allocated into a bin. [BFHM08] shows that if balls have weights in the real interval $[0,1]_{\mathbb{R}}$ and the sum of weights of all balls is fixed to some $w_{\mathsf{tot}} \in \mathbb{N}$, then inserting w_{tot} balls of weight 1 is the worst case for the maximum load of a bin (in expectation).

Two-choice allocation. Once again, n balls are thrown into n bins. For each ball, two bins are chosen independently and uniformly at random. The ball is inserted into whichever of the two bins contains the fewest balls at the time of insertion. A celebrated result by Azar et al. shows that, at the outcome of the insertion process, the most loaded bin contains $\mathcal{O}(\log \log n)$ balls with high probability [ABKU94]. It was later shown that the result holds with overwhelming probability [RMS01]. Thus, having two choices per ball instead of one yields an exponential improvement in the load of the most loaded bin. This result has found application in many areas of computer science. A survey may be found in [RMS01], which also presents some of the underlying analytical techniques.

The weighted variant of the two-choice process inserts each ball into whichever of two uniformly random bins currently contains the least weight in total (rather than the least number of balls, in the unweighted case). If balls have weights in $[0, 1]_{\mathbb{R}}$, then one would hope that the load of the most loaded bin is $\mathcal{O}(\log \log w_{\mathsf{tot}})$, where $w_{\mathsf{tot}} = \sum w_i$ is the total weight of the balls.

At STOC 2007, Talwar and Wieder analyzed the weighted two-choice process [TW07]. They showed that when the weights of the balls are drawn *independently* from a fixed distribution of expectation 1, with some mild smoothness assumptions, the load of the most loaded bin is indeed $\mathcal{O}(\log \log n)$ (which is also $\mathcal{O}(\log \log W)$, if W is defined to be the expectation of the total weight). In fact, they show a much stronger result that the gap between the expected load of each bin and the load of the most loaded bin is bounded by $\mathcal{O}(\log \log m)$ with high probability, even when inserting an unbounded number of balls $m \gg n$ into n bins, inspired by a seminal paper by Berenbrink *et al.* showing the same result in the unweighted case [BCSV06]. A simpler proof of a variant of the result was later given in [TW14], again assuming weights drawn independently from a suitable distribution. To our knowledge, all existing analyses of the weighted two-choice process rely on a distributional assumption of that form[1].

It seems quite natural to want to consider the case that there is no distributional assumption: the sequence of ball weights is instead an *arbitrary* sequence, subject only to an upper bound on the weight of an individual ball. Indeed, in many cases, the weights of the balls (corresponding, e.g., to the cost of a job in a job allocation application, or the size of an object in a memory management application) may be determined by a client, and need not be drawn from a consistent distribution, and may not be independent of each other. To our knowledge, a distribution-free result of that form is only known for 1C [BFHM08] (in expectation), but the same article argues that their analysis technique cannot extend to the two-choice process.

Cuckoo hashing. Cuckoo hashing is introduced by Pagh and Rodler [PR04]. It has found many applications within cryptography: among others, oblivious algorithms (cf. [CGLS17], and the references therein), private set intersection [PSSZ15], and more recently, searchable encryption [PPYY19, BBF+21]. In cuckoo hashing, n balls are inserted into $(2+\varepsilon)n$ bins, where $\varepsilon > 0$ is an arbitrarily small constant. Each bin can contain at most one ball. For each ball, two bins are chosen independently and uniformly at random. The ball is inserted into one of the two bins. If the bin was already occupied, the occupying ball is moved to its other possible destination bin, possibly creating a chain reaction. Pagh and Rodler have shown that insertion terminates in expected $\mathcal{O}(\log n)$ time [PR04] (including the amortized cost of rebuilding the whole table with a new hash function in case of insertion failure). In the end, similar to two-choice allocation, each ball is stored in one of two possible locations. Thanks to the more complex insertion algorithm, which allows moving already placed balls, the most loaded bin has (by definition) a load of 1, instead of $\mathcal{O}(\log \log n)$ for two-choice allocation. To achieve a negligible probability of failure, cryptographic applications typically use cuckoo hashing with a stash [KMW10].

[1] A partial exception may exist: the result on the so-called $(1+\beta)$-choice process studied in [PTW10] is written in the same distributional form, but upon closer inspection, it appears that the core potential function argument could be written without the need of a distributional assumption, as long as the weights are bounded. However, as noted also in [TW14], this technique can only prove a logarithmic bound, rather than the $\mathcal{O}(\log \log)$.

4.1.1 Our Contributions

In this chapter, we provide weighted variants for the aforementioned allocation schemes and prove upper bounds for the load of the most loaded bin. All our upper bounds match the upper bounds in the classical setting with unweighted balls.

One-choice allocation. As a warm-up, we extend the results in [BFHM08] and show that for weighted 1C, a logarithmic bound of $\widetilde{\mathcal{O}}(\log w_{\mathsf{tot}})$ holds with overwhelming probability for $m = o(w_{\mathsf{tot}})$ bins.

Two-choice allocation. We introduce layered two-choice (L2C), a weighted adaption of 2C. L2C has the same basic behavior as a (weighted) two-choice algorithm: for each ball, two bins are chosen uniformly at random as possible destinations. The only difference is how the bin where the ball is actually inserted is selected among the two destination bins. The most natural choice would be to store the ball in whichever bin currently has the least load, where the *load* of a bin is the sum of the weights of the balls it currently contains. Instead, we use a slightly more complex decision process. In a nutshell, we partition the possible weights of balls into $\mathcal{O}(\log\log\lambda)$ subintervals, and the decision process is performed independently for balls in each subinterval. For the first subinterval (holding the smallest weights), we use the aforementioned weighted 1C process, while for the other subintervals, we use an unweighted two-choice process.

The point of this construction is that its analysis reduces to the analysis of the weighted one-choice process and the unweighted two-choice process, for which powerful analytical techniques are known. We leverage those techniques to show that in L2C, the most loaded bin has a load at most $\mathcal{O}(\log\log w_{\mathsf{tot}})$. This bound is the asymptotically identical to the bound in classical two-choice allocation without requiring that ball weights be drawn independently from a fixed distribution (unlike past results in [TW07, TW14]). Our bound requires only than ball weights lie in $[0,1]_{\mathbb{R}}$ and that an upper bound w_{tot} on the total weight is known. In practice, what this means is that we have an allocation algorithm that, for most intents and purposes, behaves like a weighted variant of two-choice allocation, and for which updates and distribution-free guarantees can be obtained relatively painlessly.

Note that in our SSE constructions in Chapters 5 and 6, the adversary is in control the weight distribution, so such a distribution-free bound is required for our applications.

Cuckoo hashing. We introduce a natural generalization WCuckoo of cuckoo hashing with a stash that allows for the allocation of weighted balls. Notably, WCuckoo is the first variant of cuckoo hashing for weighted balls. Given n balls (b_i, w_i) with total weight w_{tot} and $m = (2+\varepsilon)w_{\mathsf{tot}}$ bins, we initially allocate each ball into the first bin u_i of the two bins (u_i, v_i) chosen at random. Each bin is identified with a vertex $v_i \in V$ and each ball (b_i, w_i) is identified with an edge $(u_i, v_i) \in E$ of weight w_i, where $V = [m]$ and $E \subseteq V \times V$. This gives rise to a graph $G = (V, E)$ with m vertices and n edges labeled with weights, similar to the (unweighted) standard cuckoo graph. Our algorithm WCuckoo extends the graph with a source s and sink t, both connected with the bins V depending on the bin loads. Then, the allocation is optimized according to a max flow calculated in the extended graph.

In our analysis, we first show that this allocation strategy is optimal in the sense that it minimizes the *overflow*, i.e., the total weight moved to the stash. In short, we view WCuckoo as an algorithm that operates on graphs G (as sketched above) and apply the max-flow min-cut theorem to partition the vertices (i.e., bins) into two sets S (containing the source) and T (containing the sink). Then, we use that the partition (S, T) is minimal to show that the allocation of WCuckoo cannot be improved.

Finally, we show that for any input, a stash of size $1 + \omega(\log\lambda)/\log m$ suffices with overwhelming probability. Roughly, we use a convexity argument to reduce to results on cuckoo

hashing. The first step is to prove that the expectancy of the stash size for an arbitrary input is upper-bounded by its expectancy when balls are of uniform weight. The core of that step is a convexity argument: we prove that the minimal stash size, as a function of the underlying graph, is convex. The result then follows using some majorization techniques (inspired by the analysis of weighted balls-and-bins problems in [BFHM08]). In the second step, we extend a result by Wieder [Wie17] on cuckoo hashing with uniform balls. In particular, we show that known stash bounds for cuckoo hashing (with uniform balls) hold with overwhelming probability. The third and final step is to slightly extend the above convexity argument, and combine it with some particular features of the problem and the uniform stash bound, to deduce a tail bound on the stash size, as desired.

4.2 One-Choice Allocation

We recall the weighted one-choice allocation process. An arbitrary number of balls $\{b_i\}_i$ of weight $w_i \in [0,1]_{\mathbb{R}}$ with total weight at most w_{tot} are thrown into m bins at random. We show that the maximum loaded bin has a load of $\widetilde{\mathcal{O}}(\log w_{\mathsf{tot}})$ with overwhelming probability.

We start with a description of the process. The balls with weights in $[0,1]_{\mathbb{R}}$ are thrown uniformly and independently at random into m bins. We model the random choice with a hash function $\mathsf{H} : \{0,1\}^* \mapsto \{1,\ldots,m\}$, where H is uniformly random among functions mapping into $\{1,\ldots,m\}$. We set $\delta(\lambda) = \log\log\lambda$ throughout this section.

Algorithm 1 One-Choice Allocation (1C)

$\underline{\mathsf{1C.Setup}(w_{\mathsf{tot}})}$

1: Set $m \leftarrow \lceil w_{\mathsf{tot}}/(\delta(\lambda)\log(w_{\mathsf{tot}}))\rceil$
2: Initialize m empty bins $B_0,\ldots,B_{m-1}$
3: Return $B_0,\ldots,B_{m-1}$

$\underline{\mathsf{1C.UpdateBall}(b,w,B_0,\ldots,B_{m-1})}$

1: Receive bins $B_1,\ldots,B_m$, and ball b with weight w
2: Set $\alpha \leftarrow \mathsf{H}(b) \bmod m$
3: **if** $b \notin B_\alpha$ **then**
4: Insert b into B_α
5: Update weight of b to w in B_α

We now prove an upper bound on the load of the most loaded bin. First, we recall a lemma (proven in [BFHM08]) that upper bounds the expectation of a weighted one-choice process.

Lemma 4.1 ([BFHM08], Corollary 3.5)**.** *Let $w_{\mathsf{tot}} \in \mathbb{N}$ and $c \in [0,1]_{\mathbb{R}}$. Let $\tilde{w} = (c)_{i\in[\tilde{n}]}$ and $w = (w_i)_{i\in[n]}$ be non-negative vectors. Let $\sum_{i=1}^{n} w_i \le c\cdot\tilde{n}$ and $w_i \le c$ for all $i \in [n]$.*

Denote by X_{mlb} (resp. $\tilde{X}_{\mathsf{mlb}}$) the random variable indicating the load of the most loaded bin after throwing n balls with weights w (resp. w_{tot} balls with weights $\tilde{x}$) uniformly and independently at random into m bins. For any positive $R \in \mathbb{R}$, it holds that

$$\mathsf{Exp}[\max(X_{\mathsf{mlb}} - R, 0)] \le \mathsf{Exp}[\max(\tilde{X}_{\mathsf{mlb}} - R, 0)].$$

Note that we simplified the formulation of [BFHM08] and adapted it to our needs. Roughly, [BFHM08] requires that $\tilde{w}$ *majorizes* w, which holds if both are chosen as in Lemma 4.1. Then, [BFHM08] shows that $\mathsf{Exp}[\tilde{X}_{\mathsf{mlb}}] \ge \mathsf{Exp}[X_{\mathsf{mlb}}]$ using a convexity argument. As $f(X) = \max(X-R,0)$ is increasing, it is straightforward to adapt their proof to our formulation. Note that we use a similar argument in Section 4.4, where we also define majorization formally. For the upper bound on the most loaded bin in weighted 1C, our formulation is sufficient.

Theorem 4.2 (1C). *Let $w_{\mathsf{tot}} = \mathsf{poly}(\lambda)$ and $m \in \mathbb{N}$ such that $\exp(-w_{\mathsf{tot}}/m) = \mathsf{negl}(\lambda)$. Then, the most loaded bin during the execution of an (initially fixed) sequence of 1C.UpdateBall operations has load at most $1+3\lceil w_{\mathsf{tot}}\rceil/m$ except with negligible probability, if the total weight of the inserted balls does not exceed w_{tot}.*

Proof. Let X_{mlb} be the random variable denoting the load of the most loaded bin at some point during the sequence. Let $R = 3\lceil w_{\mathsf{tot}}\rceil/m$. We show that $\Pr[X_{\mathsf{mlb}} > R+1] = \mathsf{negl}(\lambda)$. Note that this is implied by $\Pr[\lceil X_{\mathsf{mlb}}\rceil > R+1] = \mathsf{negl}(\lambda)$.

As $\lceil X_{\mathsf{mlb}}\rceil < \lceil w_{\mathsf{tot}}\rceil$ is integer-valued, we can apply Lemma 3.2. Now, we need to show that $\mathsf{Exp}[\max(\lceil X_{\mathsf{mlb}}\rceil - (R+1), 0)] = \mathsf{negl}(\lambda)$. As $\lceil X_{\mathsf{mlb}}\rceil < X_{\mathsf{mlb}} + 1$, it suffices to show that the expression $\mathsf{Exp}[\max(X_{\mathsf{mlb}} - R, 0)]$ is negligible in λ.

By Lemma 4.1, this quantity is upper bounded by $\mathsf{Exp}[\max(\tilde{X}_{\mathsf{mlb}} - R, 0)]$, where $\tilde{X}_{\mathsf{mlb}}$ is the load of the most loaded bin for $\lceil w_{\mathsf{tot}}\rceil$ balls of weight 1. Note that $\tilde{X}_{\mathsf{mlb}} \leq \lceil w_{\mathsf{tot}}\rceil$ is a non-negative integer variable. After a final application of Lemma 3.2, it remains to show that $\Pr[\tilde{X}_{\mathsf{mlb}} \geq R] = \mathsf{negl}(\lambda)$. For the last quantity, it is sufficient to analyze classical 1C without weights.

Let X_i be the random variable that denotes the number of balls in bin B_i. Every ball has a chance of $1/m$ to be assigned to B_i and there are $\lceil w_{\mathsf{tot}}\rceil$ balls. Thus, we have $\mathsf{Exp}[X_i] = \lceil w_{\mathsf{tot}}\rceil/m$. Note that X_i can be expressed as sum of independent random variables taking values in $\{0,1\}$. Applying Lemma 3.1 with constant 2, we obtain:

$$\begin{aligned} &\Pr[X_i \geq 3\mathsf{Exp}[X_i]] \leq \exp\left(-\frac{4\cdot\mathsf{Exp}[X_i]}{4}\right) \\ \implies &\Pr[X_i \geq R] \leq \exp(-\lceil w_{\mathsf{tot}}\rceil/m) = \mathsf{negl}(\lambda), \end{aligned}$$

by assumption. Finally, a union bound over all bins yields that $\Pr[\tilde{X}_{\mathsf{mlb}} \geq R] = \mathsf{negl}(\lambda)$. □

Inserting the parameters of the two-dimensional one-choice algorithm of [ANSS16], we obtain a quasi-logarithmic upper bound for weighted 1C via Theorem 4.2.

Corollary 4.3. *Let $\lambda \leq w_{\mathsf{tot}} = \mathsf{poly}(\lambda)$, and $m = \lceil w_{\mathsf{tot}}/(\delta(\lambda)\log w_{\mathsf{tot}})\rceil$ with $\delta(\lambda) = \omega(1)$*

Then, the most loaded bin during the execution of an (initially fixed) sequence of 1C.UpdateBall operations has load at most $\widetilde{\mathcal{O}}(\log w_{\mathsf{tot}})$ except with negligible probability, if the total weight of the inserted balls does not exceed w_{tot}.

Proof. This follows as $\exp(-\delta(\lambda)\log w_{\mathsf{tot}}) = \mathsf{negl}(\lambda)$ if $w_{\mathsf{tot}} \geq \lambda$ and $\delta(\lambda) = \omega(1)$. □

4.3 Two-Choice Allocation

In this section, we describe layered two-choice allocation (L2C), a variant of two-choice allocation that allows to allocate n weighted balls (b_i, w_i) into m bins, where b_i is a unique identifier and $w_i \in [0,1]_{\mathbb{R}}$ is the weight of the ball. (We often write ball b_i for short.) First, let $1 \leq \delta(\lambda) \leq \log(\lambda)$ be a function. We denote by $w_{\mathsf{tot}} = \sum_{i=1}^{n} w_i$ the sum of all weights and set $m = w_{\mathsf{tot}}/(\delta(\lambda)\log\log w_{\mathsf{tot}})$. We will later choose $\delta(\lambda) = o(\log\log\lambda)$ such that allocation has negligible failure probability. In the overview, we set $\delta(\lambda) = 1$ and assume that $m = \Omega(\lambda)$ for simplicity (which suffices for negligible failure probability).

4.3.1 Overview

L2C is based on both *weighted* one-choice allocation (1C) and *unweighted* two-choice allocation (2C). On a high level, we split the set of possible weights $[0,1]_{\mathbb{R}}$ into $\log\log m$ subintervals

$$[0, 1/\log m]_{\mathbb{R}}, (1/\log m, 2/\log m]_{\mathbb{R}}, \ldots, (2^{\log\log m - 1}/\log m, 1]_{\mathbb{R}}.$$

In words, the first interval is of size $1/\log m$ and the boundaries between intervals grow by a factor 2 every time. We will allocate balls with weights in a given subinterval independently from the others.

Balls in the first subinterval have weights $w_i \leq \log m$ and are thus small enough to apply weighted 1C. This suffices because one-choice performs worst for uniform weights of maximal size $1/\log m$. In that case, there are at most $n' = w_{\mathsf{tot}} \log m$ balls and we expect a bin to contain $n'/m = \log m \cdot \log\log w_{\mathsf{tot}}$ balls of uniform weight, since $m = w_{\mathsf{tot}}/(\log\log w_{\mathsf{tot}})$. As each ball has weight $1/\log m$, the expected load per bin is $\log\log w_{\mathsf{tot}}$. This translates to a $\mathcal{O}(\log\log w_{\mathsf{tot}})$ bound with overwhelming probability. Formally, this follows via Theorem 4.2 after scaling the weights with a factor $\log m$.

For the other intervals, applying unweighted and independent 2C per interval suffices, as the weights of balls differ at most by a factor 2 and there are only $\log\log m$ intervals. More concretely, let n_i be the number of balls in the i-th subinterval $A_i = (2^{i-1}/\log m, 2^i/\log m]_{\mathbb{R}}$ for $i \in \{1, \ldots, \log\log m\}$. Balls with weights in subinterval A_i fill the bins with at most $\mathcal{O}(n_i/m + \log\log m)$ balls, independent of other subintervals. Note that we are working with small weights, and thus potentially have $\omega(m)$ balls. Thus, we need to extend existing 2C results to negligible failure probability in m for the heavily-loaded case (cf. lemma 4.5). As there are only $\log\log m$ subintervals, and balls in interval A_i have weight at most $2^i/\log m$, we can just sum the load of each subinterval and receive a bound

$$\sum_{i=1}^{\log\log m} \frac{2^i}{\log m}\mathcal{O}(n_i/m + \log\log m) = \mathcal{O}(w_{\mathsf{tot}}/m + \log\log m).$$

In total, we have $\mathcal{O}(w_{\mathsf{tot}}/m + \log\log m) = \mathcal{O}(\log\log w_{\mathsf{tot}})$ bounds for the first and the remaining intervals. Together, this shows that all bins have load at most $\mathcal{O}(\log\log w_{\mathsf{tot}})$ after allocating all n items. This matches the bound of standard 2C with unweighted balls if $m = \Omega(\lambda)$. For our SSE application (cf. Section 5.4), we want to allow for negligible failure probability with the least number of bins possible. We can set $\delta(\lambda) = \log\log\log(\lambda)$ and obtain a bin size of $\widetilde{\mathcal{O}}(\log\log w_{\mathsf{tot}})$ with overwhelming probability, if $m = \frac{w_{\mathsf{tot}}}{\delta(\lambda)\log\log w_{\mathsf{tot}}}$. The analysis is identical in this case.

Handling Updates. The described variant of L2C is static. That is, we have not shown a bound on the load of the most loaded bin if we add balls or update the weight of balls. Fortunately, inserts of new balls are trivially covered by the analysis sketched above, if m was chosen large enough initially in order to compensate for the added weight. Thus, we assume there is some upper bound w_{tot} on the total weights of added balls which is used to initially set up the bins. We can also update weights if we proceed with care.

For this, let b_i be some ball with weight w_{old}. We want to update its weight to $w_{\mathsf{new}} > w_{\mathsf{old}}$. If w_{old} and w_{new} reside in the subinterval, we can directly update the weight of b_i, as L2C ignores the concrete weight of balls inside a given subinterval for the allocation. Indeed, in the first interval, the bin in which b_i is inserted is determined by a single random choice, and for the remaining subintervals, the 2C process only considers the number of balls inside the same subinterval, ignoring concrete weights.

When w_{new} is larger than the bounds of the current subinterval, we need to make sure that the ball is inserted into the correct bin of its two choices. For this, the ball b_i is inserted into the bin with the lowest number of balls with weights inside the new subinterval. Even though the bin of b_i might change in this process, we still need to consider b_i as a ball of weight w_{old} in the old bin for subsequent ball insertions in the old subinterval. Thus, we mark the ball as *residual* ball but do not remove it from its old bin. That is, we consider it as ball of weight w_{old} for the 2C process but assume it is not identified by b_i anymore. As there are only $\log\log m$ different subintervals, storing the residual balls has a constant overhead. The full algorithm L2C is given in Algorithm 2. We parameterize it by a hash function H mapping uniformly into $\{1, \ldots, m\}^2$. The random bin choices of a ball b_i are given by $\alpha_1, \alpha_2 \leftarrow \mathsf{H}(b_i)$.

Algorithm 2 Layered 2-Choice Allocation (L2C)

L2C.Setup($\{(b_i, w_i)\}_{i=1}^n, w_{\mathsf{tot}}$)

1: Receive n balls (b_i, w_i), and maximal total weight w_{tot}
2: Initialize $m = \lceil w_{\mathsf{tot}}/(\delta(\lambda)\log\log w_{\mathsf{tot}})\rceil$ empty bins $B_1, \ldots, B_m$
3: **for all** $i \in \{1, \ldots, n\}$ **do**
4: Set $\alpha_1, \alpha_2 \leftarrow \mathsf{H}(b_i)$
5: InsertBall$(b_i, w_i, B_{\alpha_1}, B_{\alpha_2})$
6: Return $B_1, \ldots, B_m$

L2C.InsertBall($b_{\mathsf{new}}, w_{\mathsf{new}}, B_{\alpha_1}, B_{\alpha_2}$)

1: Receive bins $B_{\alpha_1}, B_{\alpha_2}$, and ball $(b_{\mathsf{new}}, w_{\mathsf{new}})$
2: Assert that α_1, α_2 are the choices given by $\mathsf{H}(b_{\mathsf{new}})$
3: Split the set of possible weights $[0,1]_{\mathbb{R}}$ into $\log\log m$ sub-intervals

$$[0, 1/\log m]_{\mathbb{R}}, (1/\log m, 2/\log m]_{\mathbb{R}}, \ldots, (2^{\log\log m - 1}/\log m, 1]_{\mathbb{R}}$$

4: Choose $k \in \mathbb{N}$ minimal such that $w_{\mathsf{new}} \leq 2^k/\log m$
5: **if** k = 1 **then**
6: Set $\alpha \leftarrow \alpha_1$
7: **else**
8: Let B_α be the bin with the least number of balls of weight in $\left(\frac{2^{k-1}}{\log m}, \frac{2^k}{\log m}\right]_{\mathbb{R}}$ among B_{α_1} and B_{α_2}
9: Insert ball b_{new} into bin B_α

L2C.UpdateBall($b_{\mathsf{old}}, w_{\mathsf{old}}, w_{\mathsf{new}}, B_{\alpha_1}, B_{\alpha_2}$)

1: Receive bins $B_{\alpha_1}, B_{\alpha_2}$ that contain ball $(b_{\mathsf{old}}, w_{\mathsf{old}})$, and new weight $w_{\mathsf{new}} \geq w_{\mathsf{old}}$
2: Assert that α_1, α_2 are the choices given by $\mathsf{H}(b_{\mathsf{old}})$
3: **if** $w_{\mathsf{old}}, w_{\mathsf{new}} \in \left(\frac{2^{k-1}}{\log m}, \frac{2^k}{\log m}\right]_{\mathbb{R}}$ for some k **then**
4: Update the weight of b_{old} to w_{new} directly
5: **else**
6: Mark b_{old} as residual ball (it is still considered as a ball of weight w_{old})
7: InsertBall$(b_{\mathsf{old}}, w_{\mathsf{new}}, B_{\alpha_1}, B_{\alpha_2})$

4.3.2 Analysis

Let $\delta(\lambda) = \log\log\log\lambda$ and m sufficiently large such that $m^{-\Omega(\delta(\lambda)\log\log w_{\mathsf{tot}})} = \mathsf{negl}(\lambda)$. Alternatively, we can also set $\delta(\lambda) = 1$ and $m = \Omega(\lambda)$. (Note that this is the probability that allocation of 1C or 2C fails.)

We need to show that after setup and during a (selective) sequence of operations, the most loaded bin has a load of at most $\mathcal{O}(\delta(\lambda)\log\log w_{\mathsf{tot}})$, where w_{tot} is an upper bound on the total weight of the inserted balls. Let us sketch the proof. First, we modify the sequence S such that we can reduce the analysis to only (sufficiently independent) L2C.InsertBall operations, while only increasing the final bin load by a constant factor. This is constant factor of the load is due to the additional weight of residual balls. Then, we analyze the load of the most loaded bin for the each subinterval independently. This boils down to an analysis of a 1C process in the first subinterval and a 2C process in the remaining subintervals as in the overview of L2C (cf. Section 4.3.1). Summing up the independent bounds yields the desired result.

Theorem 4.4. *Let $w_{\mathsf{tot}} = \mathsf{poly}(\lambda)$, and $m = \lceil w_{\mathsf{tot}}/(\delta(\lambda)\log\log w_{\mathsf{tot}})\rceil$ with $\delta(\lambda) = \log\log\log\lambda$. (Alternatively, we can also set $\delta(\lambda) = 1$ and $m = \Omega(\lambda)$.) Assume that $m \geq \lambda^{1/\log\log\lambda}$, and model H as random oracle.*

Then it holds that throughout the execution of some sequence S of L2C operations, where updates exclusively increase the weight of balls, the most loaded bin has at most load $\mathcal{O}(\delta(\lambda)\log\log w_{\mathsf{tot}})$ except with negligible probability.

Before proving Theorem 4.4, we introduce some additional preliminaries. In the next lemma, we consider a sequence of ball insertions and deletions of arbitrary length, such that the total number of balls in the bins at any point in time is bounded by $n = h \cdot m$. A ball insertion is a standard 2-choice insertion: pick two bins i.u.r., and insert the ball into the least loaded bin. A deletion removes one previously inserted ball. The sequence of additions and deletions is fixed at the input of the problem.

Lemma 4.5 (2C)**.** *Let $\delta(m)$ be an arbitrary map such that $1 \leq \delta(m) \leq \log m$ for all $m \geq 1$. At the outcome of the sequence of additions and deletions, the most loaded bin contains $O(h + \delta(m)\log\log m)$ items, except with probability $m^{-\Omega(\delta(m)\log\log m)}$.*

In particular, by setting $\delta(m) = 1$, we get that if $m \geq \lambda$, then the failure probability from the claim is negligible. Similarly, by setting $\delta(m) = \log\log\log(m)$, we get that if $m \geq \lambda^{1/\log\log\lambda}$, then the failure probability from the claim is negligible. Also, note that $\delta(\lambda) = \Theta(\delta(m))$ for $m = \mathsf{poly}(\lambda)$ in both cases.

Proof. We adapt the proof of [Vöc03], which proves a bound $O(h) + \log\log m$ with probability $m^{-\alpha}$, for an arbitrary constant α. The proof uses *witness trees*. The existence of a bin containing more than $Ch + L$ items implies the existence of a witness tree of height $L + C'$, for some suitable constants C, C'. Thus, in order to bound the probability that a bin contains more than $Ch + L$ items, it suffices to bound the probability that a witness tree of height $L + C'$ exists. In more detail, the proof shows that the probability that a witness tree of height $L + 3$ exists is upper-bounded by

$$m^{-\kappa+1+o(1)} + m^{-\alpha}$$

where κ, α are certain parameters (to be discussed later), with:

$$L \leq \log\log m + \log(1+\alpha) + \kappa.$$

The proof sets α and κ to be constants. The fact that γ and κ are constant is not essential to the argument, and is only used in two places in the proof.

The first place is the end of Section 2.3, when upper-bounding the probability of activation of a pruned witness tree by $m^{-\kappa+1+o(1)}$. The final step of that upper-bound requires $\alpha \cdot \kappa = m^{o(1)}$, which is obviously true for a pair of constants.

The other, more important place where the choice of having constant α and κ comes into play is in the final derivation. The proof shows that, except with probability at most $m^{-\kappa+1+o(1)} + m^{-\alpha}$, the number of items in the most loaded bin is at most:

$$\begin{aligned} L + O(h) &\leq \log\log m + \log(1+\alpha) + \kappa + O(h) \\ &= \log\log m + O(1) + O(h) \\ &= \log\log m + O(h). \end{aligned}$$

In that final computation, the fact that α and κ are constant makes it possible to absorb the $\log(1+\alpha) + \kappa$ term into the $O(h)$ term. The other term is only $\log\log m$, which is optimal. If we set $\alpha = \kappa = \delta(m)\log\log m$ instead, we get:

$$\begin{aligned} L + O(h) &\leq \log\log m + \log(1+\alpha) + \kappa + O(h) \\ &\leq 3\delta(m)\log\log m + O(h). \end{aligned}$$

In the case $\delta = 1$, this worsens the constant in front of the $\log\log$ term, which is likely why the authors chose α and κ to be constant. (A better constant than 3 is possible, we choose 3 for simplicity.) On the other hand, the probability of failure becomes at most

$$m^{-\kappa+1+o(1)} + m^{-\alpha} = m^{-\Omega(\delta(m)\log\log m)}$$

as claimed. Note that the condition $\alpha \cdot \kappa = m^{o(1)}$ is still fulfilled. □

We are now ready to prove Theorem 4.4.

Proof. Let $\{(b_i, w_i)_{i=1}^n\}$ be balls with (pair-wise unique) identifier b_i and weight $w_i \in [0,1]_{\mathbb{R}}$. Further, let $S = (\mathsf{op}_i, \mathsf{in}_i)_{i=n+1}^{s+n}$ be a sequence of s insert or update operations $\mathsf{op}_i \in \{\mathsf{L2C.InsertBall}, \mathsf{L2C.UpdateBall}\}$ with input $\mathsf{in}_i = (b_i, w_i, B_{\alpha_{i,1}}, B_{\alpha_{i,2}})$ for inserts and $\mathsf{in}_i = (b_i, o_i, w_i, B_{\alpha_{i,1}}, B_{\alpha_{i,2}})$ for updates. Here, b_i denotes the identifier of a ball with weight w_i and old weight $o_i \leq w_i$ before the execution of op_i. Also, the bins are chosen via $\alpha_{i,1}, \alpha_{i,2} \leftarrow \mathsf{H}(b_i)$.

Execute $(B_i)_{i=1}^m \leftarrow \mathsf{L2C.Setup}(\{(b_i, w_i)_{i=1}^n\})$ and the operations $\mathsf{op}_i(\mathsf{in}_i)$ with input in_i for all $i \in [n+1, n+s]$. Also, assume that $\sum_{i=1}^{n+s} w_i - o_i \leq w_{\mathsf{tot}}$, i.e., the total weight after all operations is at most w_{tot}. We need to give an upper bound on the load of the most loaded bin throughout the execution of the process.

Note that the load of a bin is never decreasing, so it is sufficient to analyze the final load of bins $B_1, \ldots, B_m$. Also, note that we can replace Setup with n $\mathsf{InsertBall}$ operations. Thus, we can assume without loss of generality that bins $B_1, \ldots B_m$ are initially empty after $\mathsf{L2C.Setup}$. Also, note that $m^{-\Omega(\delta(\lambda)\log\log w_{\mathsf{tot}})} = \mathsf{negl}(\lambda)$ under the given requirements (cf. lemma 4.5). As H is modeled as a random oracle, we assume that the bin choices α_1, α_2 of ball b are chosen independently and uniformly at random from $[1, m]^2$. We split the proof into three parts:

(1) First, we will modify the sequence S such that we can reduce the analysis to only (sufficiently independent) $\mathsf{L2C.InsertBall}$ operations, while only increasing the final bin load by a constant factor.

(2) Second, we analyze the maximal bin load when only considering balls of weight at most $1/\log m$. Here, $\mathsf{L2C.InsertBall}$ behaves exactly as weighted $\mathsf{1C}$, and we can upper bound the bin load via Theorem 4.2.

(3) Last, we inspect the maximal bin load considering items in the remaining subintervalls $(2^{i-1}/\log m, 2^i/\log m]_{\mathbb{R}}$ for $i \in \{1, \ldots, \log\log m\}$. Per interval, $\mathsf{L2C.InsertBall}$ behaves like unweighted two-choice (independent of other subintervals) and inherits the $\log\log m$ bin load direclty, as balls with different weights differ only by a constant factor per interval. Summing up the maximal bin load per interval will yield the desired result.

Part 1 — Adapting the sequence: Observe that update operations that with old and new weights inside the same subinterval can be ignored. More concretely, let $\mathsf{op}_i = \mathsf{UpdateBall}$ be some update operation on ball b_i with old weight o_i and new weight w_i. If o_i both w_i are in the same subinterval $(\frac{2^{k-1}}{\log m}, \frac{2^k}{\log m}]_{\mathbb{R}}$ for some k, the operation op_i replaces the old weight o_i of ball b_i with the new weight w_i directly (inside the same bin). Thus, we can simply remove op_i and replace the previous operation $\mathsf{op}_j = (b_i, o_j, o_i) = (b_j, o_j, w_j)$ on the same ball with $\mathsf{op}'_j = (b_i, o_j, w_i)$ directly. This does not change the final load of the bins. (Note that operations between op_j and op_i make the same choices as the concrete weight inside a subinterval never impacts which bin is chosen.)

Now, let $(\mathsf{op}_i)_{i \in I}$ be all remaining update operations for some fixed ball b_*, so $b_i = b_*$ and $\mathsf{op}_i = \mathsf{UpdateBall}$ for $i \in I$. As we removed consecutive update operations in the same subinterval, operation op_i marks the ball b_* as residual ball and calls $\mathsf{InsertBall}(b_*, w_i, B_{\alpha_{*,1}}, B_{\alpha_{*,2}})$. Let $j = \max(I)$ be the index of the last update operation op_j on b_* and k be minimal such that $w_j \leq 2^k/\log m$. As there are only k subintervals below the last interval $(2^{k-1}/\log m, 2^k/\log(m)]_{\mathbb{R}}$, there are at most k such update operations, i.e., $|I| \leq k$, and one insert ball operation. Assume without loss of generality that all $k+1$ operations exist. The residual ball left by the i-th update

operation has at most size $2^{i-1}/\log m$ and thus, this UpdateBall operation can be replaced by an InsertBall$(b_*, 2^{i-1}/\log m, (B_{\alpha_{*,1}}, B_{\alpha_{*,2}}))$ operation. Thus, for ball b_* with final weight w_j, we have to insert k additional balls in order to replace all update operations on ball b_* with inserts. The total weight of these additional balls is

$$\sum_{i=1}^{k} 2^{i-1}/\log m \leq 2^k/\log(m) \leq 2w_j,$$

since $w_j \geq 2^{k-1}/\log(m)$. Thus, the total weight is increased by at most $2w_j$ per ball when we replace UpdateBall operation with InsertBall operations as above, where w_j is the final weight.

This way, we can iteratively transform all UpdateBall operations into InsertBall operations at the cost of a factor 3 in the total weight. The remaining InsertBall operations have as input some ball b_i that is inserted at most once per subinterval. Also, the bin choices are drawn uniformly and independently random per ball and subinterval. Thus, if we transform the sequence S as above into a sequence S' of InsertBall operations, and show the desired upper bound $\mathcal{O}(\delta(\lambda)\log\log w_{\mathsf{tot}})$ for S' and $w'_{\mathsf{tot}} = 3w_{\mathsf{tot}}$, then the same upper bound (up to a constant factor) holds for the initial sequence S. Without loss of generality, we only consider sequences with InsertBall operations and $w'_{\mathsf{tot}} = 3w_{\mathsf{tot}}$ weight in the following. Note that the other parameters remain unchanged.

Part 2 — Light balls: Here, we show that the most loaded bin has load at most $\mathcal{O}(\delta(\lambda)\log\log w_{\mathsf{tot}})$ when only considering balls of at most weight $1/\log m$. Assume that all balls are of size at most $1/\log m$ for simplicity. In that case, observe that InsertBall behaves like weighted 1C with balls of weight $w_i \leq 1/\log m$ and total weight at most w'_{tot}. Recall that $m = \lceil w_{\mathsf{tot}}/(\delta(\lambda)\log\log w_{\mathsf{tot}})\rceil$.

First, observe that 1C with balls of weight $w_i \cdot \log m \in [0,1]_{\mathbb{R}}$, m bins and maximal total weight $w'_{\mathsf{tot}} \cdot \log m$ has an upper bound of

$$\mathcal{O}(\frac{w'_{\mathsf{tot}} \cdot \log m}{m}) = \mathcal{O}(\log m \cdot \delta(\lambda)\log\log w_{\mathsf{tot}})$$

on the load of the most loaded bin except with negligible probability. This follows from Theorem 4.2, as $\exp(-w'_{\mathsf{tot}} \cdot \log(m)/m) = \mathcal{O}(m^{-\delta(\lambda)\log\log w_{\mathsf{tot}}}) = \mathsf{negl}(\lambda)$.

Scaling back the sequence to their original weights w_i by a factor $1/\log m$ yields the desired bound. That is, the sequence with weights w_i has an upper bound $1/\log m \cdot \mathcal{O}(\log m \cdot \delta(\lambda)\log\log w_{\mathsf{tot}}) = \mathcal{O}(\delta(\lambda)\log\log w_{\mathsf{tot}})$ as desired.

Part 3 — Heavy balls: So far, we have shown that the most loaded bin has load at most $\mathcal{O}(\delta(\lambda)\log\log w_{\mathsf{tot}})$ with overwhelming probability, when only considering balls of weight smaller or equal to $1/\log m$. We will now show that when considering the remaining balls of weight in $(1/\log m, 1]_{\mathbb{R}}$, a maximal load of $\mathcal{O}(w_{\mathsf{tot}}/m + \delta(\lambda)\log\log w_{\mathsf{tot}}) = \mathcal{O}(\delta(\lambda)\log\log w_{\mathsf{tot}})$ is preserved.

For $i \in [\log\log m]$, let n_i be the number of balls in each subinterval $A_i = (2^{i-1}/\log m, 2^i/\log m]_{\mathbb{R}}$. Recall that each ball b_i has two bin choices that are drawn uniformly and independently random at the first insertion. These choices are reutilized across the subintervals A_i, if b_i is inserted in multiple subintervals. But note that per subinterval, b_i is only inserted once. Thus, L2C behaves like unweighted 2C for each subinterval A_i (independent from the balls in other subintervals). By Lemma 4.5, the bin with the highest number of balls (amongst the n_i balls with weights in A_i) contains at most $\mathcal{O}(n_i/m + \delta(\lambda)\log\log m)$ balls with overwhelming probability. (Note that there are at most $w_{\mathsf{tot}}\log m$ balls and that $m^{-\Omega(\delta(\lambda)\log\log w_{\mathsf{tot}})} = \mathsf{negl}(\lambda)$.)

Each ball with weight $w_i \in A_i$ has weight at most $\max(A_i) = 2^i/\log m$ and thus, the load of the most loaded bin is at most $2^i/\log m \cdot \mathcal{O}(n_i/m + \delta(\lambda)\log\log m)$ when considering balls with weights in A_i. Summing over all A_i's, when considering only balls with weights in $(1/\log m, 1]$,

the load of the most loaded bin is at most

$$\begin{aligned}
&\sum_{i=1}^{\log\log m} \frac{2^i}{\log m}\mathcal{O}(n_i/m + \delta(\lambda)\log\log m)\\
=&\sum_{i=1}^{\log\log m} \mathcal{O}(2\frac{n_i 2^{i-1}}{m\log m}) + \sum_{i=1}^{\log\log m} \frac{2^i}{\log m}\mathcal{O}(\delta(\lambda)\log\log m)\\
\leq&\mathcal{O}(\frac{w_{\mathsf{tot}}}{m} + \delta(\lambda)\log\log w_{\mathsf{tot}}),
\end{aligned}$$

as $m = \mathcal{O}(w_{\mathsf{tot}})$ and $\sum_i n_i 2^{i-1} \leq 3w_{\mathsf{tot}}$, since $w'_{\mathsf{tot}} = 3w_{\mathsf{tot}}$ is an upper bound on the total weight. The above holds with overwhelming probability, since the probability that 2C fails is $\mathsf{negl}(\lambda)$, and there are only $\log\log m$ subintervals.

As we showed in the first part that is suffices to look at the modified sequence (with only InsertBall operations) and we gave an upper bound for the remaining balls of weight at most $1/\log m$ in the first part, we have that the load of the most loaded bin is at most $\mathcal{O}(\delta(\lambda)\log\log w_{\mathsf{tot}})$. This concludes the proof. □

4.4 Cuckoo Hashing

In this section, we explore weighted cuckoo hashing. We present a natural generalization of static cuckoo hashing with a stash and prove upper bounds on the stash size. As a reminder, cuckoo hashing allocates the balls into m bins $B_1, ..., B_m$ with capacity 1 or a stash. The main differences to 2C are that balls can be moved to a stash, and that the allocation can be optimized globally to minimize the stash size. This allows for a constant bin size of 1.

Each weighted ball b_i is mapped to two random bins via hash functions H_1, $H_2 : \{0,1\}^* \mapsto \{1, ..., m\}$. We assume that each ball has weight $w_i \in [0,1]_{\mathbb{R}}$, and that H_1 (resp. H_2) is uniformly random among functions mapping into $\{1, ..., m/2\}$ (resp. $\{m/2+1, ..., m\}$). Two possible destination bins $B_{H_1(b_i)}$ and $B_{H_1(b_i)}$ are associated to the i-th ball b_i. This is a direct generalization of unweighted cuckoo hashing, where all balls have size 1. Throughout, we assume that balls b_i can be split into parts. Each part can be allocated to either B_{α_1}, bin B_{α_2} or the stash. Later, we show how to allocate balls without splitting them into parts at the cost of having bins of capacity 2 instead of 1.

The goal of the WCuckoo algorithm is to allocate the balls into the bins and the stash, while respecting the capacity of every bin and while minimizing the stash size. It determines weights $w_{i,j}$ for each ball b_i such that $w_{i,1} + w_{i,2} < w_i$. These weights determine how each ball is allocated. Each ball b_i is split into three parts of weight $w_{i,1}, w_{i,2}$ and $w_{i,3} = w_i - \sum_b w_{i,b}$ The parts of weight $w_{i,1}$ and $w_{i,2}$ of ball b_i are allocated into bins $B_{H_1(b_i)}$ and $B_{H_1(b_i)}$, respectively. If $w_{i,3} > 0$, the remaining weight $w_{i,3}$ is allocated to the stash. It ensures that each bin receives a total weight of at most 1, and that the stash is as small as possible.

4.4.1 Overview

We present our weighted cuckoo hashing algorithm WCuckoo in Algorithm 3. Below, we give a brief description. The algorithm takes as input the number of bins m and the balls $(b_i, w_i)_{i\in[n]}$. It outputs bins $\{B_i\}_i$ that contain the balls b respecting the following conditions:

1. Each part of ball b is allocated into either B_{α_1}, B_{α_2} or the stash, where $\alpha_1 = H_1(b_i)$ and $\alpha_2 = H_2(b_i)$.

2. All bins do not exceed their capacity.

Algorithm 3 WCuckoo

WCuckoo.Setup$((b_i, w_i)_{i=1}^n), w_{\mathsf{tot}}$

$w_{\mathsf{tot}} \leftarrow \sum_{i=1}^n w_i, m \leftarrow (2+\varepsilon)w_{\mathsf{tot}}$
Initialize m empty bins $B_1, ..., B_m$ and an empty stash S.
Create an oriented graph G with m vertices numbered $\{1, \ldots, m\}$
Initialize empty table wgt to store edge labels
for all balls (b_i, w_i) **do**
    Set $\alpha_1 \leftarrow H_1(b_i), \alpha_2 \leftarrow H_2(b_i)$
    if $(\alpha_1, \alpha_2) \notin G$ **then**
        Add (α_1, α_2) to G with label $\mathsf{wgt}[\alpha_1, \alpha_2] \leftarrow 0$
    Update label $\mathsf{wgt}[\alpha_1, \alpha_2] \leftarrow \mathsf{wgt}[\alpha_1, \alpha_2] + w_i$
Add separate source vertex s and sink vertex t to G
for all vertex u **do**
    Compute its outgoing weight $\mathsf{out}_u = \sum_{(u,v)\in E} \mathsf{wgt}[u, v]$.
    if $\mathsf{out}_u > 1$ **then**
        Add edge from the source s to u with $\mathsf{wgt}[s, u] \leftarrow \mathsf{out}_u - 1$
    else if $\mathsf{out}_u < 1$ **then**
        Add edge from u to the sink t with $\mathsf{wgt}[u, t] \leftarrow 1 - \mathsf{out}_u$
    Add missing back edges initialized with capacity 0 in G
Compute a max flow f from s to t with capacity labels wgt
for all edges (u, v) in G **do**
    Update $\mathsf{wgt}[u, v] \leftarrow \mathsf{wgt}[u, v] - f[u, v]$
    Update $\mathsf{wgt}[v, u] \leftarrow \mathsf{wgt}[v, u] + f[u, v]$
for all balls (b_i, w_i) **do**
    Set $\alpha_1 \leftarrow H_1(b_i), \alpha_2 \leftarrow H_2(b_i)$
    Set $w_{i,1} \leftarrow \min(\mathsf{wgt}[\alpha_1, \alpha_2], w_i)$
    Set $w_{i,2} \leftarrow \min(\mathsf{wgt}[\alpha_2, \alpha_1], w_i - w_{i,1})$
    Set $w_{i,3} \leftarrow w_i - (w_{i,1} + w_{i,2})$
    Allocate $w_{i,1}, w_{i,2}, w_{i,3}$ weight of b_i to $B_{\alpha_1}, B_{\alpha_2}, S$, respectively
    Update label $\mathsf{wgt}[\alpha_1, \alpha_2] \leftarrow c[\alpha_1, \alpha_2] - w_{i,1}$
    Update label $\mathsf{wgt}[\alpha_2, \alpha_1] \leftarrow c[\alpha_2, \alpha_1] - w_{i,2}$
return $(B_1, ..., B_m, S)$

The algorithm first creates a graph similar to the cuckoo graph in cuckoo hashing: vertices are the bins, and for each ball, there is an edge (α_1, α_2) is drawn between its two possible destination bins $\alpha_1 = H_1(b_i)$ and $\alpha_2 = H_2(b_i)$. Here, each edge (α_1, α_2) is associated with the weight of the ball, and we collapse multiple directed edges between two bins into a directed single edge associated with the sum of their weights.

Note that edges are initially oriented in an arbitrary way. Ultimately, each ball will be assigned to the bin α_1 at the origin of its corresponding edge. If the bin already contains more than $1 - \mathsf{wgt}[\alpha_1, \alpha_2]$ weight, then the ball is split and the remaining weight is moved to α_2. If the total weight of α_2 also exceeds its capacity after insertion, the ball is split again and the excess weight is moved to the stash. This means that the load of a bin is the total outgoing weight of the associated vertex, and at most 1.

Efficiency. We now analyze the efficiency of WCuckoo. WCuckoo allocates a total number of $m = (2+\varepsilon)w_{\mathsf{tot}}$ bins. We show that a stash size $1 + \omega(\log \lambda)/\log m$ suffices to allocate all balls. In particular, the stash size does not grow with the total weight.

Theorem 4.6 (Main bound)**.** *The probability that the allocation fails is bounded by: Let $L \in [0,1]^n_{\mathbb{R}}$ such that $\sum_i L[i] = w_{\mathsf{tot}} \geq 4, m = (2+\varepsilon)w_{\mathsf{tot}}$ for some constant $\varepsilon > 0$, $s = \mathcal{O}(w_{\mathsf{tot}}^{1/c})$ for some sufficiently large constant c.*

$$\Pr[\mathsf{Fail}_{m,s}(L,H)] = \mathcal{O}(m^{-(s-1)/2}).$$

The computational cost is dominated by the max flow algorithm in setup. We refer to Section 3.3 for an overview of possible algorithms.

Indivisible balls. In our analysis, we assume that we can split balls and allocate each part into either of the two bins or the stash. This is not a problem for our applications in Chapter 5. But if in a specific application, we need to allocate each ball entirely without splitting, it is not hard to see that a bin size of 3 (instead of 1) is sufficient (with logarithmic stash as before). This follows immediately from Theorem 4.6. Given an allocation where the balls are split, move each ball into the data structure (i.e., bins or stash) that contains the most weight of the ball. Then, each data structure receives at most 3 times its original weight (with respect to the allocation with split balls).

4.4.2 Analysis

In this section, we prove Theorem 4.6. First, we show that the algorithm $\mathsf{WCuckoo}$ is optimal, in the sense that it minimizes the size of the stash, among all possible ways of allocating the balls between their two destination bins and the stash. Then, we upper bound the stash size via a convexity argument.

Setup and Notation

Let us recall some notation. Let $(b_i, w_i)_{i=1}^n$ be a tuple of weighted balls, let m be the total number of bins, and let s be the size of the stash (counted as the total weight allocated to the stash). Recall that $w_i \in [0,1]_{\mathbb{R}}$ for all $i \in [n]$. We always assume m is a multiple of 2; otherwise, an extra bin is added. The hash functions are chosen as follows: H_1 is uniformly random among functions mapping into $\{1,\dots,m/2\}$; H_2 is uniformly random among functions mapping into $\{m/2+1,\dots,m\}$. We denote the total weight of all balls by $w_{\mathsf{tot}} = \sum_{i=1}^n w_i$.

Graphs. For a vertex u in a directed graph $G = (V, E, \mathsf{wgt})$ with edge labels wgt, we say that $\mathsf{wgt}(u,v)$ is the weight of edge $(u,v) \in E$. Below, all graphs are directed with edge labels, and we assume that all for all $(u,v) \in E$ we have $(v,u) \in E$ (with $\mathsf{wgt}(v,u) = 0$ if the back edge was added). Let $\mathsf{out}_G(u) \coloneqq \sum_{(u,v)\in E} \mathsf{wgt}(u,v)$ denote the total weight of outgoing edges. We consider graphs with non-negative weights $\mathsf{wgt}(u,v) \geq 0$ for all edges $(u,v) \in E$. We write $W_{\mathsf{total}}(G) = \sum_{(u,v)\in E} \mathsf{wgt}(u,v)$ for the total weight of all edges in G.

We say that a graph $D = (V_D, E_D, \mathsf{wgt}_D)$ *arises* from G if $E_D = E, V_D = V$ and for all edges $(u,v) \in E$, we have $\mathsf{wgt}(u,v) + \mathsf{wgt}(v,u) = \mathsf{wgt}_D(u,v) + \mathsf{wgt}_D(v,u)$ and $\mathsf{wgt}_D(u,v) \geq 0$.

A (vertex-induced) *subgraph* of G is a graph $G' = (V', E', \mathsf{wgt}')$ such that $V' \subseteq V$ and $E' = (V' \times V') \cap E$ with weights $\mathsf{wgt}'(u,v) = \mathsf{wgt}(u,v)$ for all $(u,v) \in E'$.

Let $P = ((u,v_1),\dots,(v_k,v))$ be a path from u to v in G. We write $\mathsf{inv}(P) = ((v,v_k),\dots,(v_1,u))$ for the path from v to u following the back edges.

When the set of vertices V and edges E is fixed, we sometimes view directed graphs as vector spaces over $\mathbb{R}$, in the natural way. That is, let $G_0 = (V, E, \mathsf{wgt}_0), G_1 = (V, E, \mathsf{wgt}_1)$ be undirected graphs. Let $x \in \mathbb{R}$. We define the addition of two graphs by $G_0 + G_1 = (V, E, \mathsf{wgt})$, where $\forall (u,v) \in E, \mathsf{wgt}(u,v) = \mathsf{wgt}_0(u,v) + \mathsf{wgt}_1(u,v)$. Similarly, scalar multiplication is defined by $x \cdot G_0 = (V, E, \mathsf{wgt})$, where $\forall (u,v) \in E, w(u,v) = x \cdot \mathsf{wgt}_0(u,v)$.

Proof of Optimality

We regard WCuckoo as an algorithm that takes as input a graph $G = (V, E, \mathsf{wgt})$ with $|V| = m$ vertices and total weight $W_{\mathsf{total}}(G) = \sum_{(u,v)\in E} \mathsf{wgt}(u, v)$. WCuckoo outputs a graph D arising from G. The input graph corresponds to the graph constructed in lines 3 to 9 in Algorithm 3. The output graph is the graph $D = (V, E, \mathsf{wgt}_D)$, where the weights were adapted as in line 19 to 21 in Algorithm 3 according to the max flow. Note the graphs arising from G correspond to the possible allocations of the balls into their destination bins, and the output graph D arises from G. The allocation is optimal, if the total weight of each bin over capacity is globally as small as possible. This motivates the notion of overflow.

Definition 4.7. Let $G = (V, E, \mathsf{wgt})$ be a directed graph. The *overflow* $\mathsf{over}(G)$ is defined as:

$$\mathsf{over}(G) = \sum_{v\in V} \max(0, \mathsf{out}_G(v) - 1)$$

Observe that the overflow is exactly the weight that cannot fit into the bins. Hence, the the stash size is the overflow of the graph D output by WCuckoo. The main result of this section is Theorem 4.9, which states that WCuckoo is optimal, in the sense that it minimizes the overflow. This holds regardless of the hash functions: we do not care how they are picked for our proof of optimality. WCuckoo takes as input a graph and bin capacity, and always orients the edges optimally for the given metric, regardless of what the graph looks like.

Definition 4.8. Let G be a graph. The *optimal overflow* $\mathsf{opt}(G)$ is the infimum, taken over the overflow of all directed graphs G' arising from G:

$$\mathsf{opt}(G) = \inf_{G' \text{ arises from } G} \mathsf{over}(G').$$

Note that as $\mathsf{over}(G') \in \mathbb{R}$ and at least 0, the above notion of overflow is well-defined. The core result of this section is the following theorem.

Theorem 4.9 (Optimality of WCuckoo)**.** *Let $G = (V, E, \mathsf{wgt})$ be a directed graph with weights wgt. Let D be the graph output by* WCuckoo *on input G. Then $\mathsf{over}(D) = \mathsf{opt}(G)$.*

Before proving theorem 4.9, we begin with a few definitions and lemmas. First, we show that the there exists some graph D such that $\mathsf{opt}(G) = \mathsf{over}(G)$. Consequently, we write $\mathsf{opt}(G) = \max_{G \text{ arises from } G} \mathsf{over}(G')$ in the following. This follows from compactness of $\mathbb{R}$ and continuity of over.

Lemma 4.10 (Existence)**.** *Let $G = (V, E, \mathsf{wgt})$ be a directed graph. Then there exists some graph D such that $\mathsf{opt}(G) = \mathsf{over}(D)$.*

Proof. Let (G_i) be a sequence of graphs arising from G such that $\lim_{i\to\infty} \mathsf{over}(G_i) = \mathsf{opt}(G)$. Let wgt_i be the weights of graph $G_i = (V, E, \mathsf{wgt}_i)$. There are finitely many edges E, so we can write $\mathsf{wgt}_i = \mathsf{wgt}_i^{(u,v)}$ as a vector in $\mathbb{R}^{|E|}$. As $\mathbb{R}^{|E|}$ is compact as finite cartesian product of a compact space $\mathbb{R}$, it is compact [2]. Thus, there is a converging sub-sequence $(\mathsf{wgt}_{f(i)})$ of (wgt_i) which converges to $\mathsf{wgt}_D \in \mathbb{R}^{|E|}$. Let $D = (V, E, \mathsf{wgt}_D)$. As G_i arises from G, we have for all $i \in \mathbb{N}$ that

$$\mathsf{wgt}_{f(i)}(u, v) + \mathsf{wgt}_{f(i)}(v, u) = \mathsf{wgt}(u, v) + \mathsf{wgt}(v, u).$$

Thus, the series $(\mathsf{wgt}_{f(i)}(u, v) + \mathsf{wgt}_{f(i)}(v, u) - \mathsf{wgt}(u, v) - \mathsf{wgt}(v, u))$ converges to 0, and D arises from G as addition is continuous. Similarly, over is continuous as a composition of continuous functions, and we have

$$\mathsf{opt}(G) = \lim_{i\to\infty} \mathsf{over}(G_i) = \mathsf{over}((V, E, \lim_{i\to\infty} \mathsf{wgt}_i)) = \mathsf{over}(D).$$

□

[2] This is known as Tychonoff's theorem.

Definition 4.11. Let $G = (V, E)$ be an undirected graph. The *unavoidable overflow* $\mathsf{unav}(G)$ of G is defined as $\mathsf{unav}(G) = W_{\mathsf{total}}(G) - |V|$. If $\Delta \subseteq V$, we write $\mathsf{unav}_G(\Delta)$ for $\mathsf{unav}(G')$, where G' is the subgraph of G induced by Δ.

Lemma 4.12. *Let $G = (V, E)$ be an undirected graph. Then $\mathsf{opt}(G) = \max_{\Delta \subseteq V} \mathsf{unav}_G(\Delta)$.*

Proof. We first show that $\mathsf{opt}(G) \geq \max_{\Delta \subseteq V} \mathsf{unav}_G(\Delta)$. Pick $\Delta \subseteq V$. Let D be a graph arising from G such that $\mathsf{over}(D) = \mathsf{opt}(G)$, and let $D' = (\Delta, E')$ be the subgraph of D induced by Δ. We have:

$$\begin{aligned}\mathsf{over}(D) \geq \mathsf{over}(D') &= \sum_{v \in \Delta} \max(0, \mathsf{out}_{D'}(v) - 1) \\ &\geq \sum_{v \in \Delta} (\mathsf{out}_{D'}(v) - 1) = W_{\mathsf{total}}(D') - |\Delta| \\ &= \mathsf{unav}_G(\Delta).\end{aligned}$$

This shows that $\mathsf{opt}(G) \geq \max_{\Delta \subseteq V} \mathsf{unav}_G(\Delta)$ as desired. Next, we show that $\mathsf{opt}(G) \leq \max_{\Delta \subseteq V} \mathsf{unav}_G(\Delta)$.

Again, let D be a directed graph arising from G such that $\mathsf{over}(D) = \mathsf{opt}(G)$. Define $\Gamma \subseteq V$ to be the set of vertices of D whose weight $\mathsf{out}_G(v)$ is strictly more than 1. Let $\Delta \supseteq \Gamma$ be the set of vertices that can be reached from Γ by following a directed path P in D with edges of non-zero weight.

Note that we have $\mathsf{out}_D(v) \geq 1$ for all $v \in \Delta$. We can show this as follows. Assume for the sake of contradiction that there is some $v \in \Delta$ with $\mathsf{out}_D(v) < 1$. Let $P = (u, v_1), \ldots, (v_k, v)$ be a directed path (with no repeated vertices) to $v \in \Delta$ with $u \in \Gamma$ and $\mathsf{wgt}_D(e) > 0$ for all e in P. Let $\mathsf{wgt}_P = \min_{e \in P} \mathsf{wgt}_D(e) > 0$. We can decrease the weight of all edges in P by wgt_P and increase the weight of all edges in $\mathsf{inv}(P)$ by wgt_P. This defines a graph $\overline{D} = (V, E, \mathsf{wgt}_{\overline{D}})$ with $\mathsf{wgt}_{\overline{D}}(e) = \mathsf{wgt}_D(e) + \mathsf{wgt}_P$ for $e \in P$, $\mathsf{wgt}_{\overline{D}}(e) = \mathsf{wgt}_D(e) - \mathsf{wgt}_P$ if $e \in \mathsf{inv}(P)$, and all other weights remain unchanged. Thus, the graph $\overline{D}$ arises from D and the overflow of intermediate vertices remains unchanged. Notably, we have $\mathsf{out}_{\overline{D}}(u) = \mathsf{out}_D u - \mathsf{wgt}_P$ and $\mathsf{out}_{\overline{D}}(v) = \mathsf{out}_D(v) + \mathsf{wgt}_P$, so the overflow in $\overline{D}$ is smaller than in D. This contradicts optimality of D.

Observe that all outgoing edges with non-zero weight of vertices in Γ are included in the subgraph D' induced by Δ. Thus, we have $\mathsf{over}(D') = \mathsf{over}(D)$. Finally, we have

$$\mathsf{over}(D) = \mathsf{over}(D') = \sum_{v \in \Delta} (\mathsf{out}_{D'}(v) - 1) = W_{\mathsf{total}}(D') - |\Delta| = \mathsf{unav}(D') = \mathsf{unav}_G(\Delta).$$

Hence $\mathsf{opt}(G) \leq \max_{\Delta \subseteq V} \mathsf{unav}_G(\Delta)$. □

We are now ready to prove Theorem 4.9.

Proof of Theorem 4.9. Let $G = (V, E, \mathsf{wgt})$ be the directed graph at the input of WCuckoo. Let $\overline{G} = (V_{st}, \overline{E}, \mathsf{wgt}_{\overline{G}})$ with $V_{st} = V \cup \{s, t\}$ be the directed graph obtained within WCuckoo right after running the max flow algorithm, and before adapting the edge weights. So $\overline{G}$ includes the source and sink vertices s and t, as well as a number of edges from the source to V and from V to the sink. Concretely, for each vertex $v \in V$ at most one edge is added in $\overline{E}$. If $\mathsf{out}_G(v) > 1$, an edge $(s, v) \in \overline{E}$ with $\mathsf{wgt}_{\overline{G}}(s, v) = \mathsf{out}_G(v) - 1$ is added, and if $\mathsf{out}_G(v) < 1$, there is an edge $(v, t) \in \overline{E}$ with $\mathsf{wgt}_{\overline{G}}(v, t) = 1 - \mathsf{out}_G(v)$. Without loss of generality, we assume that $\mathsf{wgt}_{\overline{G}} u, v = 0$ if $(u, v) \notin \overline{E}$.

After building $\overline{G}$, the WCuckoo algorithm computes a max flow f. Let $F = |f|$ denote the value of the max flow. Finally, let $\overline{D} = (V_{st}, \overline{E}, \mathsf{wgt}_{\overline{D}})$ denote the directed graph obtained *after* updating the weights; and $D = (V, E, \mathsf{wgt}_D)$ the subgraph of $\overline{D}$ induced by V. The

graph D is the graph that is returned by WCuckoo. Note that by construction, we have $\mathsf{wgt}_D(u,v) = \mathsf{wgt}(u,v) - f(u,v) + f(v,u)$ for all $(u,v) \in E_D$, with $f(u,v) < \mathsf{wgt}(u,v)$. Thus, D arises from G. Our goal is to prove $\mathsf{over}(D) = \mathsf{opt}(G)$.

To do so, the strategy is to use Lemma 4.12, and exhibit a subset of vertices $\Delta \subseteq V$ such that $\mathsf{unav}_G(\Delta) = \mathsf{over}(D)$. This is enough to conclude, because using Lemma 4.12, $\mathsf{opt}(G) \geq \mathsf{unav}_G(\Delta) = \mathsf{over}(D) \geq \mathsf{opt}(G)$, so $\mathsf{opt}(G) = \mathsf{over}(D)$ as desired. We now build such a Δ.

By the Max-Flow Min-Cut Theorem (cf. Theorem 3.6), there exists a partition C of V_{st} into disjoint subsets S and T such that $s \in S$, $t \in T$, and the total flow of edges going from S to T in C is equal to $F = \mathsf{cap}(C)$. Furthermore, the flow of each edge is at capacity, i.e., $f(u,v) = \mathsf{wgt}(u,v)$ for all (u,v) between S and T.

We claim that $\Delta = S \setminus \{s\}$ is our witness: that is, $\mathsf{unav}_G(\Delta) = \mathsf{over}(D)$. First, we have that

$$\begin{aligned}
\mathsf{out}_D(u) &= \sum_{(u,v)\in E} \mathsf{wgt}_D(u,v) = \sum_{(u,v)\in E} \mathsf{wgt}_G(u,v) - f(u,v) + f(v,u) \\
&= \sum_{(u,v)\in E} \mathsf{wgt}_G(u,v) + \sum_{(u,v)\in E} f(v,u) - f(u,v) \\
&= \mathsf{out}_G(u) + \sum_{(u,v)\in \overline{E}} f(v,u) - f(u,v) - \sum_{(u,v)\in \overline{E}\setminus E} f(v,u) - f(u,v) \\
&= \mathsf{out}_G(u) - \sum_{(u,v)\in \overline{E}\setminus E} f(v,u) - f(u,v) \\
&= \mathsf{out}_G(u) - f(s,u) + f(u,t)
\end{aligned}$$

Here, used the *conservation of flows* property in the second to last equation, and in the last equation that $f(u,s) = f(t,u) = 0$ due to the capacity constraint.

We show that $\mathsf{over}(D) = \mathsf{over}(G) - F$. Let $u \in V$. Indeed, if $\mathsf{out}_G(u) = 1$, then there is no edge between s or t and u in $\overline{G}$, and thus $f(s,u) = f(u,t) = 0$. Thus, we have $\mathsf{out}_D(u) = \mathsf{out}_G(u) = 1$. If $\mathsf{out}_G(u) < 1$, then $\mathsf{out}_D(u) = \mathsf{out}_G(u) + f(u,t) \leq 1$, and if $\mathsf{out}_G(u) > 1$, then $\mathsf{out}_D(u) = \mathsf{out}_G(u) - f(s,u) \geq 1$, where we used the *capacity constraint* of f for both inequalities. Let $\Gamma = \{u \in V \mid \mathsf{out}_G(u) \geq 1\}$. Note that $\sum_{u\in\Gamma} f(s,u) = F$ because all edges with non-zero capacity from the source end in Γ by construction of $\overline{G}$. In total, we have

$$\begin{aligned}
\mathsf{over}(D) &= \sum_{v\in V} \max(0, \mathsf{out}_D(v) - 1) = \sum_{u\in\Gamma} \mathsf{out}_D(u) - 1 \\
&= \sum_{u\in\Gamma} \mathsf{out}_G(u) - f(s,u) - 1 = \mathsf{over}(G) - F.
\end{aligned}$$

Next, observe that the invariants of edge weights from the source to V and from V to the sink are kept in $\overline{D}$. That is if $\mathsf{out}_D(v) > 1$, then

$$\begin{aligned}
\mathsf{wgt}_{\overline{D}}(s,v) &= \mathsf{wgt}_{\overline{G}}(s,v) - f(s,v) + f(v,s) \\
&= \mathsf{out}_G(v) - 1 - f(s,v) = \mathsf{out}_D(v) - 1.
\end{aligned}$$

Thus, there is an edge (s,v) in $\overline{D}$ with $\mathsf{wgt}_{\overline{D}}(s,v) = \mathsf{out}_D(v) - 1$. Similarly, if $\mathsf{out}_D(v) < 1$, then there is an edge (v,t) in $\overline{D}$ with $\mathsf{wgt}_{\overline{D}}(v,t) = 1 - \mathsf{out}_D(v)$.

Further, for all $(u,v) \in S \times T$ we have $\mathsf{wgt}_{\overline{D}}(u,v) = \mathsf{wgt}_{\overline{G}}(u,v) - f(u,v) + f(v,u) = 0$, because the flow of the edges between S and T in $\overline{G}$ is at capacity and there is no flow from T to S. In words, all edges between S and T have weight 0 in $\overline{D}$.

Thus, in D all $v \in \Delta$ are at least at capacity, as otherwise there would be an edge from S to T with positive weight in $\overline{D}$. Similarly, if $v \in V$ is over capacity, then $v \in \Delta$. Otherwise, there would be an edge from s to T with positive weight.

Finally, we are ready to conclude. All edges from Δ to $V \setminus \Delta$ have 0 weight, so $\sum_{v \in \Delta} \mathsf{out}_D(u) = W_{\mathsf{total}}(D_\Delta)$, where D_Δ is the subgraph of D induced by Δ. Also, all vertices in D that are over capacity are in Δ, so

$$\mathsf{over}(D) = \sum_{v \in V} \max(0, \mathsf{out}_D(v) - 1) = \sum_{v \in \Delta} \mathsf{out}_D(v) - 1 = W_{\mathsf{total}}(D_\Delta) - |V| = \mathsf{unav}(D).$$

□

Upper Bound of the Stash Size

We now show that for any algorithm that is optimal in the sense of Theorem 4.9 (including WCuckoo), a stash of size $\omega(\log \lambda)/\log m$ suffices to ensure a negligible probability of failure. A failure occurs when the stash is too small to receive all reassigned values.

Recall that we want to balls b_i of weight $w_i \in [0,1]_{\mathbb{R}}$ with total weight $\sum w_i = w_{\mathsf{tot}}$ into $m = \mathcal{O}(w_{\mathsf{tot}}/p)$ bins of capacity 1. Fix an arbitrary stash size s. The ball b_i must be stored in bins $H_1(b_i)$, $H_2(b_i)$, or in the stash, where the hash functions are modeled as uniformly random over the ranges $\{1, \ldots, m/2\}$ and $\{m/2+1, \ldots, m\}$ respectively. Allocation succeeds if every bin receives at most a total weight of 1, the stash receives at most a total weight of s, and all balls are allocated entirely. Algorithmically, we already know how to proceed: we have proved that WCuckoo is optimal, in the sense that it minimizes the number of values stored in the stash, among all possible assignments. This means that if it is *possible* to succeed (for a given pair of hash functions), then WCuckoo always succeeds. What we want to do now is to determine m and s such that it is in fact possible to succeed, except with negligible probability. Note that WCuckoo already fixes m according to the following analysis, for now we assume that it is a variable. Here and in the remainder, to ease notation, we assume that w_{tot} and s are integers (which can be enforced by adding a ball with weight at most 1).

As in Section 4.4.2, we view WCuckoo as an algorithm with input $G = (V, E, \mathsf{wgt})$, a directed graph with $|V| = m$ vertices, and outputs a graph D arising from G. Recall that WCuckoo minimizes the overflow $\mathsf{over}(G) = \sum_{v \in V} \max(0, \mathsf{out}_G(v) - 1)$, which is equal to the stash size s. Our goal in this section is to find a suitable upper bound on that quantity, which holds with overwhelming probability.

Our proof strategy is to reduce the analysis of the stash size to a cuckoo hashing problem with w_{tot} balls and m bins. We divide the proof into three steps. In the first step, we show that having w_{tot} balls of weight 1 is a worst case for the *expected* stash size. With the right view of the problem, the result follows from a convexity argument, using a *majorization* technique. In the second step, we show that in that worst case, our problem becomes equivalent to a cuckoo hashing problem, and hence inherits the same upper bound as in [KMW10]. However, since our reduction to the worst case was only in expectancy, we are not done. In the third step, we derive an upper bound that holds with overwhelming probability from the previous results. Let us start with some notation.

Input graph. Again, We regard WCuckoo as an algorithm that takes as input a directed graph $G = (V, E, \mathsf{wgt})$ with $|V| = m$ vertices and total weight $W_{\mathsf{total}}(G) = \sum_{(u,v) \in E} \mathsf{wgt}(u, v)$. The input graph is induced by the tuple of ball weights $L := (w_i)_{i=1}^n$, the number of bins m, and a pair of hash functions $H = (H_1, H_2)$ mapping into $\{1, \ldots, m/2\}$ and $\{m/2, \ldots, m\}$, respectively. We call $G = (V, E, \mathsf{wgt}) = \mathsf{graph}_H^m(L)$ the *input graph* of WCuckoo, defined as follows: $V = \{1, \ldots, m\}$, and for $u, v \in V$:

$$w(u,v) = \sum_{\substack{i \in [1,k], \\ (H_1(i), H_2(i)) = (u,v)}} w_i.$$

In the following, we write $L[i] = w_i$. Also, we assume that $E = V \times V$ and write $G = (V, \mathsf{wgt})$ for short, where we assume that $\mathsf{wgt}(u, v) = 0$ if there is no i with $(H_1(i), H_2(i)) = (u, v)$.

Probability of Failure. Given m, s, a tuple L of weights with $\max_i L[i] \leq 1$, and a pair of hash functions $H = (H_1, H_2)$ mapping into $\{1, ..., m/2\}$ and $\{m/2, ..., m\}$, respectively, define $\mathsf{Fail}_{m,s}(L, H)$ as the event that it is *impossible* to assign the weight of every ball i into the bins $H_1(\mathrm{id}_i)$, $H_2(\mathrm{id}_i)$, or in the stash, such that no bin receives more than 1 weight, and the stash receives no more than s weight. Note that the total weight w_{tot} is implied by the parameter L: $w_{\mathsf{tot}} = \sum_i L[i]$. We want an upper bound on the probability of $\mathsf{Fail}_{m,s}(L, H)$, over the random choice of the hash functions H, as a function of (m, w_{tot}, s), and independently of the choice of L (as long as L satisfies the constraints of our problem: $\max_i L[i] \leq 1$, and $\sum_i L[i] = w_{\mathsf{tot}}$).

The probability of failure $\mathsf{Fail}_{m,s}(L, H)$ can be expressed expressed differently as follows. Because WCuckoo is optimal in the sense of minimizing the stash size, $\mathsf{opt}(G)$ is equal to the stash size after running WCuckoo on a graph G. Thus, the probability of failure is equal to:

$$\Pr[\mathsf{Fail}_{m,s}(L, H)] = \Pr[\mathsf{opt}(\mathsf{graph}^m_H(L)) > s],$$

where the probability is over the uniformly random choice of H_1, H_2. We are ultimately interested in upper bounding this probability, but for now, we focus on the expected stash size $\mathsf{Exp}[\mathsf{opt}(\mathsf{graph}^m_H(L))]$.

Step 1: Majorization. The main result of this step is the following lemma. Below, the list L_D corresponds to an arbitrary tuple (that is provided as input to WCuckoo) and L_M to a tuple, where every ball has weight 1. After proving this lemma for some function f_H, we show that the stash size fulfills the requirements.

Lemma 4.13 (Bound of Expectancy)**.** *Let $L_M \in \{0,1\}^n, L_D \in [0,1]^n_{\mathbb{R}}$ with $\sum_i L_D[i] = \sum_i L_M[i]$. Let $f_H : [0,1]^n_{\mathbb{R}} \mapsto \mathbb{R}$ be convex and such that $\mathsf{Exp}[f_H(L)]$ is invariant under list permutation. It holds that:*

$$\mathsf{Exp}[f_H(L_M)] \geq \mathsf{Exp}[f_H(L_D)],$$

where the expectancy is over the (uniformly random) choice of $H = (H_1, H_2)$.

Because the hash functions (H_1, H_2) are uniform, composing their input with any fixed permutation still yields a uniform distribution. It follows that the expected stash size $\mathsf{Exp}[\mathsf{opt}(\mathsf{graph}^m_H(\cdot))]$ is invariant under list permutation. We will later show that the stash size is a convex function, and thus, the lemma above shows that the list L_M is the worst case for the expectation of the stash size.

The proof of Lemma 4.13 follows the structure of [BFHM08], and uses the so-called *majorization* technique. We now introduce the notions and tools necessary for our proof. We refer to [BFHM08] for more details.

Definition 4.14 (Majorization)**.** For two non-increasing vectors $L_1, L_2 \in \mathbb{R}^n$ with $\sum_i L_1[i] = \sum_i L_2[i]$, we say that L_1 *majorizes* L_2, written $L_1 \succ L_2$, if

$$\forall k \in [n] : \sum_{i \in [k]} L_1[i] \geq \sum_{i \in [k]} L_2[i].$$

Definition 4.15 (T-Transformation)**.** A *T-transformation* matrix $\boldsymbol{T}$ is a matrix of the form $\boldsymbol{T} = \delta \boldsymbol{I} + (1-\delta)\boldsymbol{P}$, where $\delta \in [0,1]_{\mathbb{R}}$, $\boldsymbol{I}$ is the identity matrix, and $\boldsymbol{P}$ is a permutation matrix that swaps exactly two coordinates. We write $L_1 \overset{\mathsf{T}}{\Longrightarrow} L_2$, if L_2 can be derived form L_1 by applying one T-transformation.

Lemma 4.16. *For $L_1, L_2 \in \mathbb{R}^k, L_1 \succ L_2$ if and only if L_2 can be derived from L_1 by successive applications of at most $k-1$ T-Transformations.*

The above lemma allows us to transform one list into another, given that the target list is majorized by the original list. For L_M and L_D defined as in Lemma 4.13, it holds that L_M majorizes L_D, so we can transform L_M into L_D with $k-1$ T-transformations. This allows us to prove Lemma 4.13.

Proof of Lemma 4.13. Without loss of generality (since f is invariant under list permutation), we assume that L_M and L_D are non-increasing vectors. Under the given constraints, it holds that $L_M \succ L_D$. Thus, L_D can be derived from L_M by $k-1$ T-transformations:

$$L_M \overset{\mathsf{T}}{\Longrightarrow} L_1 \overset{\mathsf{T}}{\Longrightarrow} L_2 \overset{\mathsf{T}}{\Longrightarrow} \cdots \overset{\mathsf{T}}{\Longrightarrow} L_{k-2} \overset{\mathsf{T}}{\Longrightarrow} L_D.$$

Setting $L_M = L_0$ and $L_D = L_{k-1}$, we have $L_{i+1} = \delta_i L_i + (1-\delta_i) L_i \boldsymbol{P}_i$, for all $i \in [0, k-2]$. We receive

$$\mathsf{Exp}[f_H(L_{i+1})] \leq \delta_i \mathsf{Exp}[f_H(L_i)] + (1-\delta_i)\mathsf{Exp}[f_H(L_i\boldsymbol{P}_i)] = \mathsf{Exp}[f_H(L_i)],$$

since f_H is convex and $\mathsf{Exp}[f_H(L_i\boldsymbol{P}_i)] = \mathsf{Exp}[f_H(L_i)]$ by assumption. The result follows by induction. □

We now show that the stash size $\mathsf{opt}(\mathsf{graph}_H^m(\cdot))$ is convex. First, we show that opt is convex. We will later conclude that the expected stash size is convex, since graph_H^m is linear.

Lemma 4.17. *The function $\mathsf{opt}(\cdot)$ is convex, meaning that for graphs $G = (V, \mathsf{wgt}), G_0 = (V, \mathsf{wgt}_0)$ and $G_1 = (V, \mathsf{wgt}_1)$ with $G = \delta G_0 + (1-\delta)G_1$ for all $\delta \in [0,1]$, it holds that:*

$$\mathsf{opt}(G) \leq \delta \mathsf{opt}(G_0) + (1-\delta)\mathsf{opt}(G_1).$$

Proof. Let $G = \delta G_0 + (1-\delta)G_1$ with weight $\mathsf{wgt} = \delta \mathsf{wgt}_0 + (1-\delta)\mathsf{wgt}_1$. With Lemma 4.12, we have that $\mathsf{opt}(D) = \max_{\Delta \subseteq V} \mathsf{unav}_D(\Delta)$ for $D \in \{G, G_0, G_1\}$. The convexity follows from the following calculation:

$$\begin{aligned}
\mathsf{opt}(G) &= \max_{\Delta\subseteq V} \mathsf{unav}_G(\Delta) = \max_{\Delta\subseteq V} \sum_{(u,v)\in\Delta^2} \mathsf{wgt}(u,v) - |\Delta| \\
&= \max_{\Delta\subseteq V} \sum_{(u,v)\in\Delta^2} \delta\mathsf{wgt}_0(u,v) + (1-\delta)\mathsf{wgt}_1(u,v) - |V| \\
&= \max_{\Delta\subseteq V} \sum_{(u,v)\in\Delta^2} \delta(\mathsf{wgt}_0(u,v) - |V|) + (1-\delta)(\mathsf{wgt}_1(u,v) - |V|) \\
&\leq \max_{\Delta\subseteq V} \sum_{(u,v)\in\Delta^2} \delta(\mathsf{wgt}_0(u,v) - |V|) + \max_{\Delta\subseteq V} \sum_{(u,v)\in\Delta^2} (1-\delta)(\mathsf{wgt}_1(u,v) - |V|) \\
&= \delta(\max_{\Delta\subseteq V} \mathsf{unav}_{G_0}(\Delta)) + (1-\delta)(\max_{\Delta\subseteq V} \mathsf{unav}_{G_1}(\Delta)) \\
&= \delta\mathsf{opt}(G_0) + (1-\delta)\mathsf{opt}(G_1).
\end{aligned}$$

□

Lemma 4.18. *Let $H = (H_1, H_2)$ be fixed hash functions, $L \in \mathbb{R}^n$. Then $\mathsf{opt}(\mathsf{graph}_H^m(L))$ is convex (as a function of L).*

Proof. We first show that graph_H^m is linear. Let $L_1, L_2 \in \mathbb{R}^n$ be two lists and $\delta \in \mathbb{R}$. The weight of edge (u,v) in $\mathsf{graph}_H^m(L_1 + \delta L_2)$ is

$$\begin{aligned}
\mathsf{wgt}(u,v) &= \sum_{\substack{i\in[n],\\ (H_1(i),H_2(i))=(u,v)}} (L_1 + \delta L_2)[i] \\
&= \sum_{\substack{i\in[n],\\ (H_1(i),H_2(i))=(u,v)}} L_1[i] + \delta \sum_{\substack{i\in[n],\\ (H_1(i),H_2(i))=(u,v)}} L_2[i] \\
&= \mathsf{wgt}_1(u,v) + \delta\mathsf{wgt}_2(u,v),
\end{aligned}$$

where wgt_1 and wgt_2 are the weights of $\mathsf{graph}_H^m(L_1)$ and $\mathsf{graph}_H^m(L_2)$, respectively. Thus graph_H^m is linear. Now let $L = \delta L_1 + (1-\delta)L_2$ for $\delta \in [0,1]_{\mathbb{R}}$. Linearity implies that

$$\mathsf{graph}_H^m(L) = \delta \mathsf{graph}_H^m(L_1) + (1-\delta)\mathsf{graph}_H^m(L_2),$$

and Lemma 4.17 yields the desired result. □

Thus, we can apply Lemma 4.13 on the stash size $\mathsf{opt}(\mathsf{graph}_H^m(\cdot))$. As consequence, lists of the form L_M, i.e.balls with weight 1, are a worst case for the expected stash size.

Step 2: Cuckoo Hashing. The previous step shows that it suffices to bound the expectancy for the particular list L_M. This means we have "gotten rid" of the parameter L, in the sense that we have reduced our problem to a version where this parameter is fixed. Thus, it is enough to bound the probability of failure in the case where all weights are 1, and bins have capacity 1. This is exactly a "cuckoo with a stash" problem, in the sense of Kirsch *et al.* [KMW10]. In that setting, Kirsch *et al.* prove an upper bound $\mathcal{O}(w_{\mathsf{tot}}^{-s})$ on the failure probability. That bound cannot be used directly, because it requires that the stash size s should be constant. In particular, it can only yield a polynomially low failure probability. A negligible failure probability is crucial in our setting, since the failure probability depends on the list length distribution, which we aim to hide. An extension of the original proof from [KMW10] to variable s is also given in [GM11], but that extension requires that the table size m should be polylogarithmic in the security parameter.

An $\mathcal{O}(w_{\mathsf{tot}}^{-s/2})$ bound for variable s was shown in [ADW14], with only the restriction that $s = \mathcal{O}(w_{\mathsf{tot}}^{1/c})$ for some constant c. The proof targets explicit hash families, but also extends to uniformly random functions, as noted in the introduction of the same article. In fact, a simpler variant of the proof directly targeting uniformly random hash functions is given in [Wie17]. For efficiency reasons, in practice, we wish to use cryptographic hash functions, modeled as uniformly random functions, so we are mainly interested in the uniform case. For that reason, we use the result from [Wie17] (extending it slightly to get a closed-form expression).

Lemma 4.19 (Corollary of [Wie17], Theorem 5.5)**.** *Consider the setting of cuckoo hashing where w_{tot} items are inserted into two tables of size $m \geq (1+\varepsilon)w_{\mathsf{tot}}$ each, for some constant $\varepsilon > 0$, with a stash of size s. Assume the hash functions used for cuckoo hashing are uniformly random. Then the probability that a valid cuckoo assignment fails to exist*[3] *is at most*

$$w_{\mathsf{tot}}^{-s} \cdot \left(\frac{2}{1+\varepsilon}\right)^s \cdot \sum_{t=0}^{\infty} \frac{t^{8s}}{(1+\varepsilon)^t} = \mathcal{O}\left(\sqrt{s} \cdot \left(\frac{Cs^8}{w_{\mathsf{tot}}}\right)^s\right)$$

for some constant $C \leq 2 \cdot (8(1+1/\varepsilon)/e)^8$.

Proof. Let f denote the probability that a valid cuckoo assignment fails to exist. The first bound on f is proved in [Wie17, Theorem 5.5]. Technically, the theorem is written in a setting where s is constant, but as already observed in [PSWW18, Appendix C], the proof is purely combinatorial, and holds for arbitrary s.

The $\mathcal{O}\left(\sqrt{s} \cdot \left(\frac{Cs^8}{w_{\mathsf{tot}}}\right)^s\right)$ upper bound can be derived as follows. Letting A_k be the k-th Eulerian polynomial, by a classic identity [Pet15],

$$\sum_{t=0}^{\infty} t^k x^t = \frac{x \cdot A_k(x)}{(1-x)^{k+1}}.$$

[3] That is, letting T_1, T_2 denote the two tables, and h_1, h_2 the hash functions, it is not possible to assign every item x into either index $h_1(x)$ in T_1, index $h_2(x)$ in T_2, or the stash, without either assigning two items to the same table location, or exceeding the stash size s.

Reinjecting into the first bound with $x = 1/(1+\varepsilon)$ yields

$$f \leq \left(\frac{2}{(1+\varepsilon)w_{\mathsf{tot}}}\right)^s \cdot \frac{x \cdot A_{8s}(x)}{(1-x)^{8s+1}}.$$

Since $x < 1$ and A_k is increasing, $A_{8s}(x) < A_{8s}(1) = (8s)!$. Using the upper-bound variant of Stirling's formula $k! \leq e\sqrt{k}(k/e)^k$, this yields

$$\begin{aligned} f &\leq \left(\frac{2}{(1+\varepsilon)w_{\mathsf{tot}}}\right)^s \cdot \frac{x \cdot 2e\sqrt{2s}(8s/e)^{8s}}{(1-x)^{8s+1}} \\ &\leq \left(\frac{2}{w_{\mathsf{tot}}}\right)^s \cdot \frac{x}{1-x} \cdot 2e\sqrt{2s}\left(\frac{8s}{e(1-x)}\right)^{8s} \\ &= \mathcal{O}\left(\sqrt{s} \cdot \left(\frac{Cs^8}{w_{\mathsf{tot}}}\right)^s\right). \end{aligned}$$

□

Lemma 4.19 says something about the *existence* of a solution to a cuckoo hashing problem with certain parameters. This is enough in the scope of this article, because WCuckoo outputs an optimal solution. Thus, using Lemma 4.19, we get that, for some constant C':

$$\Pr[\mathsf{Fail}_{m,s}(L_M, H)] = \mathcal{O}\left(\sqrt{s} \cdot \left(\frac{C's^8}{w_{\mathsf{tot}}}\right)^s\right).$$

A straightforward computation yields the following corollary.

Corollary 4.20. *Let $m = (2+\varepsilon)w_{\mathsf{tot}}$ for some constant $\varepsilon > 0$, and $s = \mathcal{O}(w_{\mathsf{tot}}^{1/c})$ for some sufficiently large constant c. Then:*

$$\Pr[\mathsf{Fail}_{m,s}(L_M, H)] = \mathcal{O}\left(m^{-s/2}\right).$$

Step 3: Bounding the Probability of Failure. We will now use the results from the previous two steps to compute a bound for the probability of failure $\Pr[\mathsf{Fail}_{m,s}(L, H)]$, for an arbitrary lists L. From Step 2, we know a bound on the probability of failure for the list L_M. From Step 1, we know that L_M is a worst case for the *expectancy* of the stash size, but not necessarily the failure probability. The remaining arguments bridge that gap.

We our now ready to show the main result. In particular, a stash of size $s = 1 + \omega(\log\lambda)/\log w_{\mathsf{tot}}$ suffices to ensure that WCuckoo succeeds, except with negligible probability.

We are now ready to prove Theorem 4.6.

Proof. We denote $f_H(\cdot) = \mathsf{opt}(\mathsf{graph}_H^m(\cdot))$ in the following. We need to show that the probability of failure $\Pr[\mathsf{Fail}_{m,s}(L, H)] = \Pr[f_H(L) > s]$ is negligible. Note that $\lceil f_H(L)\rceil \leq w_{\mathsf{tot}}$ is integer-valued, so due to Lemma 3.2, it suffices to show that $\mathsf{Exp}[\max(\lceil f_H(L)\rceil - s, 0)] = \mathsf{negl}(\lambda)$. As $\lceil f_H(L)\rceil < f_H(L) + 1$, it remains to show

$$\mathsf{Exp}[\max(f_H(L) - (s-1), 0)] = \mathsf{negl}(\lambda).$$

Lemma 4.18 shows that $\mathsf{opt}(\mathsf{graph}_H^m(\cdot))$ is convex. We know that $\max(f_H(\cdot) - (s-1), 0)$ is convex as composition of the increasing and convex function $g(x) = \max(x - s, 0)$ and convex function $\mathsf{opt}(\mathsf{graph}_H^m(\cdot))$. It is also invariant under list permutation. Thus, we can apply Lemma 4.13 on $\max(f_H(\cdot) - (s-1), 0)$:

$$\mathsf{Exp}[\max(f_H(L) - (s-1), 0)] \leq \mathsf{Exp}[\max(f_H(L_M) - (s-1), 0)],$$

where $L_M \in \{0,1\}^w_{\mathsf{tot}}$ is the vector consisting of w_{tot} copies of the value 1. We now upper bound this expectancy. To do so, we apply (a variant of) the definition of the expectancy (as in the proof of Lemma 3.2), split the resulting sum into three parts and bound each part separately. Here, note that $f_H(L_M)$ is an integer variable [4]. Denote $P_H(i) = \Pr[\max(f_H(L_M) - (s+i-1), 0) > 0]$. We have

$$\begin{aligned}
\mathsf{Exp}[\max(f_H(L_M) - (s-1), 0)] &= \sum_{i \geq 0} \Pr[\max(f_H(L_M) - (s-1), 0) > i] \\
&= \sum_{i \geq 0} \Pr[\max(f_H(L_M) - (s+i-1), 0) > 0] \\
&= \sum_{i=0}^{\lceil \log(m) \rceil} P_H(i) + \sum_{i=\lceil \log(m) \rceil + 1}^{m-1} P_H(i) + \sum_{i \geq m} P_H(i)
\end{aligned}$$

We consider each of the three terms above in turn. For the first term, we can apply Lemma 4.19, as the probability $P_H(i)$ is the probability of failure of cuckoo hashing for stash size $s + i - 1 \leq s + \lceil \log(m) \rceil - 1 = \mathcal{O}(w_{\mathsf{tot}}^{1/c})$, for some sufficiently large constant c. Thus:

$$\sum_{i=0}^{\lceil \log(m) \rceil} P_H(i) = \mathcal{O}\Big(\sum_{i=0}^{\lceil \log(m) \rceil} m^{-(s+i-1)/2} \Big) = \mathcal{O}\Big(m^{-(s-1)/2} \cdot \sum_{i=0}^{\lceil \log(m) \rceil} m^{-i/2} \Big) = \mathcal{O}(m^{-(s-1)/2}).$$

The last equation holds because

$$\sum_{i=0}^{\log(m)} m^{-i/2} \leq \sum_{i=0}^{\infty} m^{-i/2} = \frac{1}{1 - m^{-1/2}} = \mathcal{O}(1).$$

Let us turn to the second term. Observe that $P_H(\cdot)$ is necessarily non-increasing. In particular, $P_H(i) \leq P_H(\lceil \log m \rceil)$ for $i \geq \log m$. Applying the cuckoo hashing bound, we get $P_H(\lceil \log m \rceil) \leq m^{-(s+\lceil \log m \rceil - 1)/2}$. Using this bound in the second sum yields:

$$\begin{aligned}
\sum_{i=\lceil \log m \rceil + 1}^{m-1} P_H(i) &\leq m \cdot \mathcal{O}(m^{-(s+\lceil \log m \rceil - 1)/2}) \\
&= \mathcal{O}(m^{-(s-1)/2} \cdot m^{1 - \lceil \log m \rceil / 2}) \\
&= \mathcal{O}(m^{-(s-1)/2}).
\end{aligned}$$

Finally, let us consider the third term. If the stash size s is larger than the total weight w_{tot} of all balls to allocate, it is trivially not possible for the stash to overflow, and the probability of failure is 0. Thus, the third term vanishes, as $P_H(i) = 0$ for $i \geq m$. Putting everything together, we get

$$Pr[\mathsf{Fail}_{m,p,s}(L, H)] = \mathcal{O}(m^{-(s-1)/2}).$$

This concludes the proof. □

[4] When all balls have weight 1, we consider the case where balls are not split into parts. In that case, the optimal overflow is an integer. Also, any upper bound for the stash size in this case yields an upper bound for when we allow for splitting (as we consider the optimal stash size). To avoid clutter in the notation, we still write $f_H(L_M)$ for short.

Chapter

5 Page Efficiency and Constructions

The performance bottleneck of classic SSE schemes typically does not come from their fast, symmetric cryptographic operations, but rather from the cost of memory accesses. To address this issue, many works in the literature have considered the notion of locality, a simple design criterion that helps capture the cost of memory accesses in traditional storage media, such as Hard Disk Drives.

In this chapter, we propose a notion that captures efficiency on new storage media, such as Solid State Drives, which have become increasingly common. Then, we provide two constructions with good page efficiency based on L2C and WCuckoo (cf. Chapter 4).

Chapter content

5.1 Introduction 61
5.1.1 Our Contributions 62
5.2 Page Efficiency 63
5.3 Pluto 64
5.3.1 Construction 64
5.3.2 Security 65
5.3.3 Efficiency 67
5.4 LayeredSSE 67
5.4.1 Construction 68
5.4.2 Security 69
5.4.3 Extensions 70
5.4.4 Efficiency 72

5.1 Introduction

In Searchable Symmetric Encryption (SSE), a client holds a collection of documents and wishes to store them on an untrusted cloud server. The client also wishes to be able to issue search queries to the server and retrieve all documents matching the query. Meanwhile, the server should learn as little information as possible about the client's data and queries. Searchable Encryption is an important goal in the area of cloud storage since the ability to search over an outsourced database is often a critical feature. The goal of SSE is to enable that functionality while offering precise guarantees regarding the privacy of the client's data and queries with respect to the host server.

Compared to other settings related to computation over encrypted data, such as Fully Homomorphic Encryption, a specificity of SSE literature is the focus on high-performance solutions, suitable for deployment on large real-world datasets. To achieve this performance, SSE schemes accept the leakage of some information on the plaintext dataset, captured in security proofs by a leakage function. The leakage function is composed of setup leakage and query

leakage. The setup leakage is the total leakage prior to query execution and typically reveals the size of the index, and possibly the number of searchable keywords. For a static scheme, the query leakage usually reveals the repetition of queries, and the set of document indices matching the query. Informally, the security model guarantees that the adversary does not learn any information about the client's data and queries, other than the previous leakages.

In particular, the allowed leakage typically reveals nothing about keywords that have *not yet* been queried. Although this requirement may seem natural and innocuous, it has deep implications for the storage and memory accesses of SSE schemes. At Eurocrypt 2014, Cash and Tessaro [CT14] proved an impossibility result that may be roughly summarized as follows. If an SSE scheme reveals nothing about the number of documents matching *unqueried* keywords, then it cannot satisfy the following three efficiency properties simultaneously: (1) constant storage efficiency: the size of the encrypted database is at most linear in the size of the plaintext data; (2) constant read efficiency: the amount of data read by the server to answer a query is at most linear in the size of the plaintext answer; (3) constant locality: the memory accesses made by the server to answer a query consist of a constant number of contiguous accesses. Thus, a secure SSE scheme with constant storage efficiency and read efficiency cannot be *local*: it must perform a superconstant number of disjoint memory accesses.

In practice, many SSE schemes (e.g.[CGKO06, CJJ+13, Bos16]) make one random memory access per entry matching the search query. As explained in [CJJ+14, MM17], making many small random accesses hampers performance: hard disk drives (HDD) were designed for large sequential accesses, and solid-state drives (SSD) use a leveled design that does not accommodate small reads well. As discussed (e.g., in [BMO17]), this results in the fact that in many settings, the performance bottleneck for SSE is not the cost of cryptographic operations (which rely on fast, symmetric primitives), but the cost of memory accesses.

As a consequence, SSE scheme designers have tried to reduce the number of disk accesses needed to process a search query, e.g.by grouping entries corresponding to the same keywords in blocks [CJJ+14, MM17], or by using more complex allocation mechanisms [ANSS16, ASS18, DPP18]. However, no optimal solution is possible, due to the previously mentioned impossibility result of Cash and Tessaro. In the static case, the first construction by Asharov *et al.* [ANSS16] from STOC 2016 achieves linear server storage and constant locality, at the cost of logarithmic read efficiency (the amount of data read by the server to answer a query is bounded by the size of the plaintext answer times $\mathcal{O}(\log N)$, where N is the size of the plaintext database). The logarithmic factor was reduced to $\log^{\gamma} N$ for $\gamma < 1$ by Demertzis *et al.* at Crypto 2018 [DPP18]. Further, [ANSS16] provides a construction with $\mathcal{O}(\log \log N)$ read efficiency but the maximal number of matching keywords is slightly restricted.

An interesting side-effect of this line of research is that it has highlighted the connection between Searchable Encryption and hash-based memory allocation schemes (cf. Chapter 4). The construction from [ANSS16] relies on a two-dimensional variant of the classic balls-and-bins allocation problem. Likewise, the construction from [DPP18] uses several memory allocation schemes tailored to different input sizes.

5.1.1 Our Contributions

As discussed above, memory accesses are a critical bottleneck for SSE performance. This has led to the notion of locality, and the construction of many SSE schemes aiming to improve locality, such as [CT14, ANSS16, MM17, DP17b, DPP18]. The motivation behind the notion of locality is that it is a simple criterion that captures the performance of traditional storage media such as HDDs. In recent years, other storage media, and especially SSDs, have become more and more prevalent. To illustrate that point, the number of SSDs shipped worldwide is projected to overtake HDD shipments in 2021 [Sta21].

However, locality as a design target, was proposed assuming an implementation on a HDD.

The starting observation is that for SSDs, the notion of locality is no longer a good predictor of practical performance. This raises two questions: first, is there a simple SSE design criterion to capture SSD performance, similar to locality for HDDs? And can we design SSE schemes that fulfill that criterion?

The answer to the first question is straightforward: for SSDs, performance is mainly determined by the number of memory pages that are accessed, regardless of whether they are contiguous. This leads us to introduce the notion of page efficiency. The page efficiency of an SSE scheme is simply the number of pages that the server must access to process a query, divided by the number of pages of the plaintext answer to the query [1].

To answer the second question, we use the weighted hashing variants in Chapter 4 to build Pluto and LayeredSSE. The scheme Pluto is an SSE scheme with optimal page efficiency, optimal storage efficiency, and standard leakage pattern: no information is leaked about unqueried keywords. Pluto is based on weighted cuckoo hashing and is static, i.e., it does not support updates. In the dynamic setting, our scheme LayeredSSE has optimal storage efficiency and $\mathcal{O}(\log\log N/p)$ page efficiency with standard leakage (in the above sense), though it is not forward secure. While LayeredSSE is less efficient than Pluto as it relies on L2C, it allows for efficient updates. Note that the asymptotic notation treats both the database size and the page size as variables. The efficiency of both constructions is summarized in Table 5.1.

Table 5.1: Page-efficient SSE schemes. N denotes the total size of the database, W denotes the number of keywords, p is the number elements per page, $\varepsilon > 0$ is an arbitrarily small constant, and λ is the security parameter.

Schemes	Client st.	Page eff.	Storage eff.	Dynamism
$\Pi_{\text{pack}}, \Pi_{\text{2lev}}$ [CJJ+14]	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(p)$	Static
OCA [ANSS16]	$\mathcal{O}(1)$	$\widetilde{\mathcal{O}}(\log N)$	$\mathcal{O}(1)$	Static
TCA [ANSS16]	$\mathcal{O}(1)$	$\widetilde{\mathcal{O}}(\log\log N)$	$\mathcal{O}(1)$	Static
IO-DSSE [MM17]	$\mathcal{O}(W)$	$\mathcal{O}(\log W)$	$\mathcal{O}(1)$	Dynamic
Pluto	$\mathcal{O}(p\log\lambda)$	3	$3+\varepsilon$	Static
LayeredSSE	$\mathcal{O}(1)$	$\widetilde{\mathcal{O}}\left(\log\log\frac{N}{p}\right)$	$\mathcal{O}(1)$	Dynamic

5.2 Page Efficiency

Optimizing an SSE scheme for locality requires that each read query accesses few non-contiguous memory locations, thus making this operation efficient for HDDs. In the case of SSDs, it is sufficient to optimize for few page accesses (as SSDs efficiently read entire pages of memory). For this reason, we introduce the notions *page cost* and *page efficiency* to measure the efficiency of read queries performed on SSDs. We introduce the notion of page efficiency.

In the following definitions, we set $\mathsf{K} \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $\mathsf{EDB} \leftarrow \mathsf{Setup}(\mathsf{K}, N, \mathsf{DB})$ given database DB and upper bound N on the number of document identifiers. Also, $S = (\mathsf{op}_i, \mathsf{in}_i)_{i=1}^s$ is a sequence of search and update queries, where $\mathsf{op}_i \in \{\mathsf{add}, \mathsf{del}, \perp\}$ is a operation and $\mathsf{in}_i = (\mathsf{op}_i, w_i, \mathsf{id}_i, \mathsf{st}_i, \mathsf{EDB}_i)$ its input. Here, w_i is a keyword and id_i is a (added or deleted) identifier, and after executing all previous operations op_j for $j \le i$, st_i is the client state and EDB_i the encrypted database. We denote by DB_i the database after i operations. We assume that the

[1] We note that experiments in [BBF+21] confirm that page efficiency is an excellent predictor of SSD performance.

total number of identifiers never exceeds N. (If $\mathsf{op}_i = \perp$, the query is a search query and L_i is empty.)

Definition 5.1 (Page Pattern)**.** Regard server-side storage as an array of pages, containing the encrypted database EDB. When processing search query $\mathsf{Search}(\mathsf{K}, w_i, \mathsf{st}_i; \mathsf{EDB}_i)$ or update query $\mathsf{Update}(\mathsf{K}, (w_i, L_i), \mathsf{op}_i, \mathsf{st}_i; \mathsf{EDB}_i)$, the read pattern $\mathsf{RdPat}(\mathsf{op}_i, \mathsf{in}_i)$ induces a number of page accesses $p_1, \ldots, p_{h'}$. We call these pages the *page pattern*, denoted by $\mathsf{PgPat}(\mathsf{op}_i, \mathsf{in}_i)$.

Definition 5.2 (Page Cost)**.** A SSE scheme has page cost $aX + b$, where a, b are real numbers, and X is a fixed symbol, if for any λ, DB, N, sequence S, and any i, $|\mathsf{PgPat}(\mathsf{op}_i, \mathsf{in}_i)| \leq aX + b$, where X is the number of pages needed to store document indices matching keyword w_i in plaintext.

Definition 5.3 (Page Efficiency)**.** A SSE scheme has page efficiency P if for any λ, DB, N, sequence S, and any i, $|\mathsf{PgPat}(\mathsf{op}_i, \mathsf{in}_i)| \leq P \cdot X$, where X is the number of pages needed to store document indices matching keyword w_i in plaintext.

5.3 Pluto

In this section, we construct a static page-efficient SSE scheme called Pluto. Let p be the page size. Roughly, Pluto uses $\mathsf{WCuckoo}$ (cf. Section 4.4) to allocate lists L of at most p identifiers matching a keyword w close together. As Algorithm 3 is kept purely combinatorial, balls technically have no content. We still need to retrieve lists L given the keyword w in the context of searchable encryption. Thus, we say that the pair (w, L) is a ball identified by w and scaled weight $|L|/p$. We assume that we can retrieve the list L given w from the bin that contains ball (w, L). Clearly, this does not change the behavior of $\mathsf{WCuckoo}$ and the stash bound still holds. We adapt the notation of $\mathsf{WCuckoo}$ to such lists:

– $\mathsf{WCuckoo.Setup}(\{(w_i, L_i)\}_{i=1}^{W}, w_{\mathsf{tot}})$: The input is (w_i, L_i) and some maximal weight w_{tot}, where L_i is a list of (at most p) identifiers matching keyword w_i. We say that (w_i, L_i) is a ball with identifier w_i and weight $|L_i|/p \in [0, 1]_{\mathbb{R}}$. The bin choices for (w_i, L_i) are given by $\alpha_1, \alpha_2 \leftarrow \mathsf{H}(w_i)$. The setup algorithm of Algorithm 2 is performed given these balls, and the bins $B_1, \ldots, B_m$ and the stash S is output.

Note that given the bins $B_{\mathsf{H}_1(w_i)}, B_{\mathsf{H}_2(w_i)}$ and the stash S, we can recover the list L_i of matching identifiers. Note that due to the scaling, a bin contains at most p identifiers.

5.3.1 Construction

We now describe Pluto in detail. Let PRF be a secure PRF mapping to $\{0,1\}^{2\lambda+\lceil\log(N)\rceil}$. Let Enc be an IND-CPA secure symmetric encryption scheme (assimilated with its encryption algorithm in the notation). We split the output of the PRF into a key of 2λ bits and a mask of $\lceil \log N \rceil$ bits. The first 2λ bits are used to preprocess the keywords w_i, and the last $\lceil \log N \rceil$ bits m_i are used for encryption. We write $m_i \leftarrow \mathsf{PRF}_{\mathsf{K}_{\mathsf{PRF}}}(w_i)$ for short, and keep the preprocessing of w_i implicit (cf. Section 3.5.1). Also, we denote by H_1 and H_2 the hash functions of $\mathsf{WCuckoo}$. Pseudo-code is provided in Algorithm 4.

– $\mathsf{Pluto.KeyGen}(1^\lambda)$: generates a PRF key $\mathsf{K}_{\mathsf{PRF}}$ and an encryption key $\mathsf{K}_{\mathsf{Enc}}$, and outputs $\mathsf{K} = (\mathsf{K}_{\mathsf{PRF}}, \mathsf{K}_{\mathsf{Enc}})$.

– $\mathsf{Pluto.Setup}(\mathsf{DB}, \mathsf{K})$: takes as input a database DB, and the client's master secret key $\mathsf{K} = (\mathsf{K}_{\mathsf{PRF}}, \mathsf{K}_{\mathsf{Enc}})$. For each keyword w_i, we have a list $\mathsf{DB}(w_i)$ of ℓ_i identifiers corresponding to the documents that match w_i. First, samples $m_i \leftarrow \mathsf{PRF}_{K_{\mathsf{PRF}}}(w_i)$ which will serve as token for w_i later on. First, the lists $\mathsf{DB}(w_i)$ are partitioned into sublists $L_{i,1}, \ldots, L_{i,x_i}$,

where L_{i,x_i} has at most p identifiers and the other sublists have exactly p identifiers. Note that $x_i = \lceil \ell_i/p \rceil$. Then, a table T_{full} (resp. T_{len}) that can store up to $\lfloor N/p \rfloor$ (resp. N) entries is allocated. All full lists $L_{i,j}$ with identifiers matching w_i are stored in $T_{\mathsf{full}}[w_i, j] \leftarrow \mathsf{Enc}_{\mathsf{K_{Enc}}}(L_{i,j})$, and the number of lists x_i is stored in $T_{\mathsf{len}}[w_i] \leftarrow m_i \oplus x_i$. The remaining incomplete lists are allocated via WCuckoo. That is, bins $B_1, \ldots, B_m$ and a stash S are created via $\mathsf{WCuckoo.Setup}((w_i, L_{i,x_i}), w_{\mathsf{tot}})$ for $w_{\mathsf{tot}} = N/p$, and all bins are padded to full capacity p. Then, all bins are encrypted and $\mathsf{EDB} = (B_i^{\mathsf{enc}}, T_{\mathsf{full}}, T_{\mathsf{len}})$ and client state S is output.

– $\mathsf{Pluto.Search}(\mathsf{K}, w_i, S; \mathsf{EDB})$: sets $m_i \leftarrow \mathsf{PRF}_{K_{\mathsf{PRF}}}(w_i)$ and sends (w_i, m_i) to the server. The server retrieves $x_i \leftarrow T_{\mathsf{len}}[w_i] \oplus m_i$, and sends $B_{\alpha_1}, B_{\alpha_2}$ and $T_{\mathsf{full}}[w_i, j]$ for $j \in [x_i - 1]$ to the client, where $\alpha_i \leftarrow \mathsf{H}_i(w_i)$. Then, the client retrieves $\mathsf{DB}(w_i)$ from S and the decrypted bins and full lists.

Algorithm 4 Pluto

$\mathsf{Pluto.KeyGen}(1^\lambda)$

1: Sample keys $\mathsf{K_{PRF}}$, $\mathsf{K_{Enc}}$ for PRF, Enc
2: **return** $\mathsf{K} = (\mathsf{K_{PRF}}, \mathsf{K_{Enc}})$

$\mathsf{Pluto.Search}(\mathsf{K}, w_i, S; \mathsf{EDB})$

Client:

1: Set $m_i \leftarrow \mathsf{PRF}_{\mathsf{K_{PRF}}}(w_i)$
2: **return** (w_i, m_i)

Server:

1: Initialize empty set R
2: Set $x_i = T_{\mathsf{len}}[K_i] \oplus m_i$
3: **for all** $j \in [x_i - 1]$ **do**
4: Add $T_{\mathsf{full}}[w_i, j]$ to R
5: Add $B_{\mathsf{H}_1(w_i)}$ and $B_{\mathsf{H}_2(w_i)}$ to R
6: **return** R

Client:

1: Recover $\mathsf{DB}(w_i)$ from S and R
2: **return** $\mathsf{DB}(w_i)$

$\mathsf{Pluto.Setup}(\mathsf{K}, \mathsf{DB})$

1: Initialize table T_{len} of size N
2: Initialize table T_{full} of size $\lfloor N/p \rfloor$
3: Initialize empty tuple $\mathcal{L}$
4: **for all** keywords w_i **do**
5: $m_i \leftarrow \mathsf{PRF}_{\mathsf{K_{PRF}}}(w_i)$
6: $x_i \leftarrow \lceil |\mathsf{DB}(w_i)|/p \rceil$
7: $T_{\mathsf{len}}[w_i] \leftarrow x_i \oplus m_i$
8: Split $\mathsf{DB}(w_i)$ into sublists $L_{i,1}, \ldots, L_{i,x_i}$
9: Add (w_i, L_{i,x_i}) to $\mathcal{L}$
10: **for all** $j \in [x_i - 1]$ **do**
11: Set $T_{\mathsf{full}}[w_i, j] \leftarrow L_{i,j}$
12: Set $w_{\mathsf{tot}} = \lceil N/p \rceil$
13: $B_1, \ldots, B_m, S \leftarrow \mathsf{WCuckoo.Setup}(\mathcal{L}, w_{\mathsf{tot}})$
14: Set $B_i^{\mathsf{enc}} \leftarrow \mathsf{Enc}_{\mathsf{K_{Enc}}}(B_i)$
15: Store the stash S on the client
16: **return** $\mathsf{EDB} = (B_1^{\mathsf{enc}}, \ldots, B_m^{\mathsf{enc}}, T_{\mathsf{full}}, T_{\mathsf{len}})$

5.3.2 Security

The leakage profile of the construction is the standard leakage profile of a static SSE scheme. Recall that x_i is the minimal number of pages for the list of documents matching keyword w_i. The leakage during setup is $\mathcal{L}_{\mathsf{Setup}}(\mathsf{DB}) = |\mathsf{DB}| = N$ and the leakage during search is $\mathcal{L}_{\mathsf{Search}}(\mathsf{DB}, w_i) = (\ell_i, \mathsf{sp})$, where sp is the search pattern and $\ell_i = |\mathsf{DB}(w_i)|$. Let $\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}})$.

Before detailing the security proof, we give a brief sketch. For Setup, the simulator creates the required number m of bins, derived from $N = \mathcal{L}_{\mathsf{Setup}}(\mathsf{DB})$, and fills each one with the encryption of arbitrary data using Enc. Similarly, it creates a table T_{len} of size N with random entries, and a table T_{full} with $\lfloor N/p \rfloor$ entries of p encrypted zeroes. It then creates the simulated database EDB consisting of the bins and the tables. The IND-CPA security of Enc guarantees that the adversary cannot distinguish the simulated bins from the real ones. Also, the simulated table is indistinguishable from the real table, since the concrete values x_i are masked with a random mask m_i. Thus, the unqueried table entries appear random.

For a (new) search query, the simulator obtains from the leakage function the number x_i, and simulates the token $m_i = \ell_i \oplus T[w_i]$ and chooses $w_i \leftarrow \{0,1\}^{2\lambda}$ at random. The PRF security of

PRF guarantees that the adversary cannot distinguish the simulated values from the real ones. Note that the adversary recovers the correct value $\ell_i = T[w_i] \oplus (\ell_i \oplus T[w_i])$. This concludes the proof.

While the proof is simple, it relies heavily on WCuckoo. Namely, the bins accessed to retrieve the incomplete lists are independent on the database distribution. Another subtle but important point is that the security argument requires that the correctness of WCuckoo holds with overwhelming probability over the random coins choice of H_1 and H_2. Indeed, the probability of a correctness failure may be dependent on the input of WCuckoo.Setup, and thus leak information. Moreover, if a correctness failure occurs, it is not acceptable to run WCuckoo.Setup again with fresh hash functions, as the choice of H_1 and H_2 would become dependent on the database distribution.

Theorem 5.4. *Let* $\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}})$ *with* $\mathcal{L}_{\mathsf{Setup}}(\mathsf{DB}) = |\mathsf{DB}| = N$ *and* $\mathcal{L}_{\mathsf{Search}}(\mathsf{DB}, w_i) = (\ell_i, \mathsf{sp})$. *Assume that* Enc *is an IND-CPA secure encryption scheme and* PRF *is a secure pseudo-random function. Then* Pluto *is an* $\mathcal{L}$*-adaptively semantically secure SSE scheme.*

Proof. Recall that the leakage during setup is $\mathcal{L}_{\mathsf{Setup}}(\mathsf{DB}) = N$, and the leakage during search is $\mathcal{L}_{\mathsf{Search}}(w_i) = (\ell_i, \mathsf{sp})$, where ℓ_i is the number of sublists for the list of documents matching keyword w_i, and sp is the search pattern. Our goal is to show that SSE is $\mathcal{L}$-adaptively semantically secure.

Let $\mathcal{S}$ denote the simulator. During setup, $\mathcal{S}$ receives $\mathcal{L}_{\mathsf{Setup}}(\mathsf{DB}) = |\mathsf{DB}| = N$, and is tasked with simulating the database EDB. To do this, $\mathcal{S}$ samples a fresh key $\mathsf{K}'_{\mathsf{Enc}}$ for Enc, and generates $B'_1 = \mathsf{Enc}_{\mathsf{K}'_{\mathsf{Enc}}}(z), \ldots, B'_m = \mathsf{Enc}_{\mathsf{K}'_{\mathsf{Enc}}}(z)$, where z is an all-zero input of length p. Note that m can be computed via N. Then, the simulator creates a simulated table T'_{len} filled with N random entries $\chi \leftarrow \{0,1\}^{\lceil \log N \rceil}$. Similarly, it creates a simulated table T'_{full} filled with $\lfloor N/p \rfloor$ entries of the form $\mathsf{Enc}_{\mathsf{K}'_{\mathsf{Enc}}}(z)$. The simulator returns the simulated database $\mathsf{EDB}' = (B'_1, \ldots, B'_m), T'_{\mathsf{len}}, T'_{\mathsf{full}}$.

During a search query, the simulator receives $\mathcal{L}_{\mathsf{Search}}(w_i) = (\ell_i, \mathsf{sp})$, and is tasked with simulating the search token m'_i and w'_i (which is preprocessed by a PRF). If the search pattern sp reveals that the same keyword has already been searched in the past, the simulator simply replies the same token. Otherwise, w'_i is chosen uniformly at random. The simulated mask is set to $m'_i = T'[w'_i] \oplus \ell_i$. The simulated search token is (w'_i, m'_i). Observe that the server recovers the correct value ℓ_i.

We need to prove that the real game is indistinguishable from the ideal game. We proceed with a sequence of hybrid games.

- Hybrid 0 is the real experiment.
- Hybrid 1 is the same as Hybrid 0 except each pair (w_i, m_i) is replaced with the simulated pair (w'_i, m'_i). The table T_{len} and the search tokens are still generated as in the real game, using this new sampling of (w'_i, m'_i). The PRF security of PRF implies that the advantage of an adversary trying to distinguish Hybrid 0 from Hybrid 1 is negligible.
- Hybrid 2 is the same as Hybrid 1 except the table T_{len} is replaced by the simulated table T'_{len}, and the search token (w_i, m_i) by the simulated search token (w'_i, m'_i). That is, those values are generated as in the ideal game. It is easy to see that the view of the adversary in Hybrid 2 is identical to its view in Hybrid 1. The advantage of an adversary trying to distinguish between Hybrid 1 and Hybrid 2 is zero.
- Hybrid 3 is the same as Hybrid 2, except a flag FAIL is raised if the input of Setup is not valid, which occurs if two distinct lists have the same identifier, i.e., there exist $i \neq j$ such that $w'_i = w'_j$. By a standard birthday argument, this probability is negligible. More precisely, there are at most $N = \mathsf{poly}(\lambda)$ values w'_i, so there are less than N^2 pairs (i, j), and each pair has a probability of collision $2^{-2\lambda}$, hence by the union bound, the probability

of collision is upper-bounded by $N^2 2^{-2\lambda} = \mathsf{negl}(\lambda)(\lambda)$. It follows that the advantage of an adversary trying to distinguish Hybrid 2 from Hybrid 3 is negligible.

– Hybrid 4 is the same as Hybrid 3 except the real database EDB is replaced by the simulated database EDB'. The real key $\mathsf{K}_{\mathsf{Enc}}$ and the simulated key $\mathsf{K}'_{\mathsf{Enc}}$ used to generated the two databases are identically distributed, so the only difference between the two databases lies in the plaintext messages being encrypted. Since Enc is IND-CPA secure, it follows that the advantage of an adversary trying to distinguish Hybrid 3 from Hybrid 4 is negligible.

– Hybrid 5 is the ideal experiment. The server's view in the ideal experiment and in Hybrid 5 are identically distributed, so the advantage of an adversary trying to distinguish Hybrid 4 from Hybrid 5 is zero.

This concludes the proof. □

Remark 5.5. In the proof, we assumed that T_{len} and T_{full} do not leak the keys w_i at setup. This can be achieved, e.g., with cuckoo hashing. Nevertheless, it is not difficult to adapt the proof if the keys are leaked, though different PRF evaluations of the keyword to index T_{len} and T_{full} are required.

5.3.3 Efficiency

With the above notation, each search requires x_i page accesses for T_{full}, 1 page access for T_{len} and 2 page accesses for both bins. Thus, page efficiency is $\mathcal{O}(1)$. Also, as T_{len}, T_{full} and $(B_1, ..., B_m)$ all require $\mathcal{O}(N)$ space, the storage efficiency is $\mathcal{O}(1)$. The client requires permanent storage of $1 + \frac{\omega(\log \lambda)}{\log((2+\varepsilon)N/p)}$ pages. The runtime is dominated by setup due to the invocation of $\mathsf{WCuckoo}$.

5.4 LayeredSSE

In this section, we introduce the SSE scheme $\mathsf{LayeredSSE}$ based on $\mathsf{L2C}$. Essentially, we interpret lists L_i of identifiers matching keyword w_i as balls of a certain weight and use $\mathsf{L2C}$ to manage the balls in m bins. Let N be the maximal size of the database, p be the page size and H be a hash function mapping into $\{1, ..., m\}^2$ for $m = \lceil w_{\mathsf{tot}}/(\log\log\log\lambda \cdot \log\log w_{\mathsf{tot}})\rceil$ and $w_{\mathsf{tot}} = N/p$. Assume for now that $|L_i| \leq p$, i.e.each keyword has at most p associated keywords. Let $p \leq N^{1-1/\log\log\lambda}$. (This is needed for the requirement $m \geq \lambda^{1/\log\log\lambda}$ of $\mathsf{L2C}$, see Theorem 4.4.) As for Pluto, we first adapt the notation of $\mathsf{L2C}$ to identifier lists as follows.

– $\mathsf{L2C.Setup}(\{(w_i, L_i)\}_{i=1}^{W}, w_{\mathsf{tot}})$: We interpret the pair (w_i, L_i) as a ball with identifier w_i and weight $|L_i|/p \in [0,1]_{\mathbb{R}}$, where L_i is a list of (at most p) identifiers matching keyword w_i. The bin choices for (w_i, L_i) are given by $\alpha_1, \alpha_2 \leftarrow \mathsf{H}(w_i)$. Run the setup defined in Algorithm 2 given these balls.

– $\mathsf{L2C.InsertBall}((w, L), B_{\alpha_1}, B_{\alpha_2})$: Insert ball (w, L) into either bin B_{α_1} or bin B_{α_2} as in Algorithm 2.

– $\mathsf{L2C.Update}((w, L), L', B_{\alpha_1}, B_{\alpha_2})$: Update the weight of ball (w, L) to weight $|L \cup L'|/p$ as in Algorithm 2 and add identifiers L' to list L. One of the bins now contains the ball $(w, L \cup L')$. If the new weight lies in a different subinterval, one bin contains a residual ball (w, L) that we consider to not match w anymore.

5.4.1 Construction

Here, we describe the dynamic page efficient symmetric searchable encryption scheme LayeredSSE based on L2C. For a concise overview, we assume that $\ell_i \leq p$ and ignore delete operations for now. Also, we present a version of the scheme with an update that requires 2 RTTs. Later, we show how to treat arbitrary list sizes, introduce delete operations and show how to obtain updates in 1 RTT. A detailed description of LayeredSSE is given in Algorithm 5.

- $\mathsf{LayeredSSE.KeyGen}(1^\lambda)$: Sample encryption key $\mathsf{K}_{\mathsf{Enc}}$ for Enc with the given security parameter λ. Return the client's master secret key $\mathsf{K} = \mathsf{K}_{\mathsf{Enc}}$.
- $\mathsf{LayeredSSE.Setup}(\mathsf{K}, N, \mathsf{DB})$: Receive as input the client's secret key K, an upper bound N on the number of identifiers and the initial database $\mathsf{DB} = (\mathsf{DB}(w_i))_{i=1}^W$. Recall that $\mathsf{DB}(w_i) = (\mathsf{id}_1, \ldots, \mathsf{id}_{\ell_i})$ is a list of ℓ_i document identifiers and that $\sum_{i=1}^W \ell_i \leq N$. interpret $(w_i, \mathsf{DB}(w_i))$ as a ball of weight $\ell_i/p \in [0,1]$ and call L2C.Setup with maximal weight N/p and balls $(w_i, \mathsf{DB}(w_i))_{i=1}^W$ as input. The two random choices $(\alpha_{i,1}, \alpha_{i,2}) \leftarrow \mathsf{H}(w_i)$ in L2C.Setup are drawn by evaluating H on w_i. The result are m bins $(B_i)_{i=1}^m$ filled with the balls such that each bin has load at most $c \log\log\log(\lambda) \log\log(N/p)$ (see Theorem 4.4). Thus, each bin contains at most $p \cdot c \log\log\log(\lambda) \log\log(N/p)$ identifiers as weights are scaled by a factor p. (The constant $c \in \mathbb{N}$ only depends on N but not the output of L2C.Setup.) Next, each bin is filled up to maximal size with dummy items. Finally, encrypt the bins $B_i^{\mathsf{enc}} \leftarrow \mathsf{Enc}_{\mathsf{K}_{\mathsf{Enc}}}(B_i)$ and return $\mathsf{EDB} = (B_i^{\mathsf{enc}})_{i=1}^m$.
- $\mathsf{LayeredSSE.Search}(\mathsf{K}, w; \mathsf{EDB})$: The client receives its secret key K and keyword w. She sends w to the server, and in return receives bins $B_{\alpha_1}^{\mathsf{enc}}, B_{\alpha_2}^{\mathsf{enc}}$, where $(\alpha_1, \alpha_2) \leftarrow \mathsf{H}(w)$.
- $\mathsf{LayeredSSE.Update}(\mathsf{K}, (w, L'), \mathsf{add}; \mathsf{EDB})$: The client receives its secret key K, keyword w and a list L' of new identifiers matching w. She sends w to the server and again receives bins $B_{\alpha_1}^{\mathsf{enc}}, B_{\alpha_2}^{\mathsf{enc}}$ in return, where $(\alpha_1, \alpha_2) \leftarrow \mathsf{H}(w)$. Next, the client decrypts $B_{\alpha_1}^{\mathsf{enc}}, B_{\alpha_2}^{\mathsf{enc}}$ to $B_{\alpha_1}, B_{\alpha_2}$ and retrieves ball (w, L) from the corresponding bin $B_\alpha \in \{B_{\alpha_1}, B_{\alpha_2}\}$. Then, she calls L2C.UpdateBall with old ball (w, L), new identifiers L' and bins $B_{\alpha_1}, B_{\alpha_2}$ to insert the new identifiers L' into one of the bins. Finally, she reencrypts the bins and sends them to the server. The server then replaces the old bins with the updated bins.

Algorithm 5 LayeredSSE

$\mathsf{LayeredSSE.KeyGen}(1^\lambda)$

1: Sample key $\mathsf{K}_{\mathsf{Enc}}$ for Enc
2: **return** $K = \mathsf{K}_{\mathsf{Enc}}$

$\mathsf{LayeredSSE.Setup}(\mathsf{K}, N, \mathsf{DB})$

1: Set $\mathsf{in} \leftarrow \{(w_i, \mathsf{DB}(w_i))\}_{i=1}^W$
2: Set $(B_1, \ldots, B_m) \leftarrow \mathsf{L2C.Setup}(\mathsf{in}, N/p)$
3: Set $B_i^{\mathsf{enc}} \leftarrow \mathsf{Enc}_{\mathsf{K}_{\mathsf{Enc}}}(B_i)$ for $i \in [1, m]$
4: **return** $\mathsf{EDB} = (B_1^{\mathsf{enc}}, \ldots, B_m^{\mathsf{enc}})$

$\mathsf{LayeredSSE.Search}(\mathsf{K}, w; \mathsf{EDB})$

Client:

1: **return** w

Server:

1: Set $\alpha_1, \alpha_2 \leftarrow \mathsf{H}(w)$
2: **return** $B_{\alpha_1}^{\mathsf{enc}}, B_{\alpha_2}^{\mathsf{enc}}$

$\mathsf{LayeredSSE.Update}(\mathsf{K}, (w, L'), \mathsf{add}; \mathsf{EDB})$

Client:

1: **return** w

Server:

1: Set $\alpha_1, \alpha_2 \leftarrow \mathsf{H}(w)$
2: **return** $B_{\alpha_1}^{\mathsf{enc}}, B_{\alpha_2}^{\mathsf{enc}}$

Client:

1: Set $B_{\alpha_i} \leftarrow \mathsf{Dec}_{\mathsf{K}_{\mathsf{Enc}}}(B_{\alpha_i}^{\mathsf{enc}})$ for $i \in \{1, 2\}$
2: Retrieve ball (w, L) from B_α for appropriate $\alpha \in \{\alpha_1, \alpha_2\}$
3: Run $\mathsf{L2C.UpdateBall}((w, L), L', B_{\alpha_1}, B_{\alpha_2})$
4: Set $B_{\alpha_i}^{\mathsf{new}} \leftarrow \mathsf{Enc}_{\mathsf{K}_{\mathsf{Enc}}}(B_{\alpha_i})$ for $i \in \{1, 2\}$
5: **return** $B_{\alpha_2}^{\mathsf{new}}, B_{\alpha_2}^{\mathsf{new}}$

Server:

1: Replace $B_{\alpha_i}^{\mathsf{enc}}$ with $B_{\alpha_i}^{\mathsf{new}}$ for $i \in \{1, 2\}$

5.4.2 Security

The scheme LayeredSSE is correct as each keyword has two bins that contain its identifiers associated to it (and these bins are consistently retrieved and updated with L2C). If the hash function is modeled as a random oracle, the bin choices are uniformly random and Theorem 4.4 guarantees that bins do not overflow.

Also, LayeredSSE is selectively secure and has standard setup leakage N, such as search and update leakage qp, where qp is the query pattern. This can be shown with a simple hybrid argument. We first sketch the proof. For setup, the simulator Sim receives N, recomputes m and initializes m empty bins $B_1, \ldots, B_m$ of size $p \cdot c \log\log\log(\lambda) \log\log(N/p)$ each. Sim then outputs $\mathsf{EDB}' = (\mathsf{Enc}_{\mathsf{K}'_{\mathsf{Enc}}}(B_i)_{i=1}^m)$ for some sampled key $\mathsf{K}'_{\mathsf{Enc}}$. As Enc is IND-CPA secure (and bins do not overflow in the real experiment except with negligible probability), the output EDB' is indistinguishable from the output of Setup in the real experiment. For a search query on keyword w, Sim checks the query pattern qp whether w was already queried. If w was not queried before, Sim a new uniformly random keyword w'. Otherwise, Sim responds with the same keyword w' from the previous query. As we assume that keywords are preprocessed by the client via a PRF, the keywords w and w' are indistinguishable. For an update query on keyword w, the client output in the first flow is the same as in a search query and thus, Sim can proceed as in search. For the second flow, Sim receives two bins $B_{\alpha_1}, B_{\alpha_2}$ from the adversary, directly reencrypts them and sends them back to the adversary. This behavior is indistinguishable, as the bins are encrypted and again, bins do not overflow except with negligible probability. A formal analysis is given below.

Lemma 5.6 (Correctness)**.** *The scheme* LayeredSSE *is correct if at most p identifiers are associated to each keyword and* H *is modeled as a random oracle.*

Proof. We use L2C to insert (and update) the lists of identifiers $\mathsf{DB}(w)$ of length $\ell \leq p$ into m bins. Each list is interpreted as a ball of weight $\ell/p \in [0,1]$. Theorem 4.4 implies that the maximal loaded bin has load at most $c \log\log\log(\lambda) \log\log(N/p)$ for some appropriate constant $c \in \mathbb{N}$ (for $\delta(\lambda) = \log\log\log(\lambda)$), since the bin choices via H are uniformly and independently random by assumption. That means, it contains at most $p \cdot c \log\log\log(\lambda) \log\log(N/p)$ identifiers (as we scaled weights by a factor p). Consequently, the bins only overflow with negligible probability. Further, it follows from inspection that one of the two bins returned by the search algorithm on input w contains all the identifiers matching keyword w. □

Lemma 5.7 (Selective Security)**.** *Let $\mathcal{L}_{\mathsf{Stp}}(\mathsf{DB}, N) = N$, $\mathcal{L}_{\mathsf{Srch}}(w) = \mathsf{qp}$ and $\mathcal{L}_{\mathsf{Updt}}(\mathsf{op}, w, L') = \mathsf{qp}$, where qp is the query pattern and $\mathsf{op} = \mathsf{add}$. Let $\mathcal{L} = (\mathcal{L}_{\mathsf{Stp}}, \mathcal{L}_{\mathsf{Srch}}, \mathcal{L}_{\mathsf{Updt}})$. The scheme* LayeredSSE *is $\mathcal{L}$-selectively semantically secure if at most p identifiers are associated to each keyword,* Enc *is* IND-CPA *secure and* H *is modeled as a random oracle.*

Proof. Let Sim denote the simulator and $\mathcal{A}$ an abitrary honest-but-curious PPT the adversary.

Initially, Sim receives $\mathcal{L}_{\mathsf{Stp}}(\mathsf{DB}, N) = N$ and a series of search and update requests with input $\mathcal{L}_{\mathsf{Updt}}(\mathsf{op}_i, w_i, L'_i) = \mathcal{L}_{\mathsf{Srch}}(w_i) = \mathsf{qp}$. First, Sim initializes $m = \lceil (N/p)/(\log\log(N/p) \log\log\log(\lambda)) \rceil$ bins $B_1, \ldots, B_m$ zeroed out up to size $p \cdot c \log\log\log(\lambda) \log\log(N/p)$, and outputs $\mathsf{EDB}' = (\mathsf{Enc}_{\mathsf{K}'_{\mathsf{Enc}}}(B_1), \ldots, \mathsf{Enc}_{\mathsf{K}'_{\mathsf{Enc}}}(B_m))$ for some encryption key $\mathsf{K}'_{\mathsf{Enc}}$ sampled by Sim. Next, Sim simulates the search and update queries.

For search queries, Sim receives sp. If the query pattern sp indicates that the keyword was already queried, Sim outputs the keyword w' from the previous query. Otherwise, Sim outputs a new uniformly random keyword w' (that has not been queried yet).

For update queries, Sim receives sp. First, Sim proceeds as in search for generating the first output w'. After sending w' to the adversary $\mathcal{A}$, Sim receives two encrypted bins. Sim simply reencrypts both bins and sends them back to the server.

We now show that the real game is indistinguishable from the ideal game. For this, we define four hybrid games.

– Hybrid 0 is identical to the real game.

– Hybrid 1 is the same as Hybrid 0 except the simulated keywords w' are output. By assumption Hybrid 0 and Hybrid 1 are indistinguishable.

– Hybrid 2 is the same as Hybrid 1 except a flag FAIL is raised when a bin overflows (i.e.contains more than $p \cdot c \log \log N/p$ identifiers after Setup or Update). Theorem 4.4 implies that this happens only with negligible probability. Thus, Hybrid 1 and Hybrid 2 are indistinguishable.

– Hybrid 3 is the same as Hybrid 2 except that the encrypted database EDB is replaced with the simulated EDB' and bins are just reencrypted and sent back to the adversary in the second flow of Update. Since Enc is IND-CPA secure (and a flag FAIL is only raised with negligible probability), it follows that Hybrid 2 and Hybrid 3 are indistinguishable.

– Hybrid 4 is the same as the ideal experiment. The server's view in the ideal experiment and in Hybrid 3 are identically distributed, so we conclude inductively that the ideal game and the real game are indistinguishable. □

For adaptive security, the adversary can issue search and update queries that depend on previous queries. As Theorem 4.4 assumes selectively chosen InsertBall and UpdateBall operations, there is no guarantee that bins do not overflow anymore in the real game. Thus, the adversary can potentially distinguish update queries of the simulated game from real update queries if she manages to overflow a bin in the real game, as she would receive bins with increased size only in latter case. Fortunately, we can just add a check in Update whether one of the bins overflows after the L2C.UpdateBall operation. In that case, the client reverts the update and send back the (reencrypted) original bins. Now, Theorem 4.4 still guarantees that bins overflow only with negligible probability after Setup and we can show that the simulated game is indistinguishable from the real game as before. (Note that LayeredSSE is still correct after this modification, since queries are chosen selectively for correctness.) Note that when the client remarks that a bin overflowed in an Update in a real world environment, this is due malicious Update operations. The client can adapt his reaction accordingly, whereas the server learns no information about the attack without being notified by the client.

We can show that LayeredSSE with the adjustment of Update is correct and $\mathcal{L}$-adaptively secure, where $\mathcal{L}$ is defined as in Lemma 5.7. The same simulator Sim suffices and we omit the details.

5.4.3 Extensions

In this section, we discuss some extensions such as handling lists of arbitrary size and improved update round trip times. Note that while we did not describe how to handle deletes explicitly, this can be done generically (cf. Section 3.6).

Handling Long Lists

We now adapt LayeredSSE to handle arbitrary lists L (with potentially more than p identifiers). We proceed similarly to the static scheme Pluto from [BBF+21] and extend the ideas to updates. For this, we split L into sublists of size at most p. The (encrypted) full sublists of size p can be stored in a hash table T_{full} on the server and the incomplete sublists are handled by LayeredSSE as before. For search, the client needs to know the number of sublists in order to fetch the right amount from the server. This information is also required for update queries in order to know when to insert another full list into T_{full}. This information can be outsourced in a table T_{len}. Here, the client stores for each keyword w (with ℓ matching identifiers) the number of sublists

$T_{\mathsf{len}}[w] = \lceil \ell/p \rceil$ in encrypted format. In the following, we describe the updated $\mathsf{Setup}, \mathsf{Search}$ and Update of $\mathsf{LayeredSSE}$ in more detail.

Setup. For setup, let L_i be a list of ℓ_i identifiers matching keyword w_i and PRF be a secure pseudo-random function mapping to $\{0,1\}^{\lceil \log(N) \rceil}$. We set $x_i = \lceil \ell_i/p \rceil$. The client splits L_i into sublists $L_{i,1}, \ldots, L_{i,x_i-1}$ of size p and sublist L_{i,x_i} of size at most p. She evaluates $m_i \leftarrow \mathsf{PRF}_{\mathsf{K}_{\mathsf{PRF}}}(w_i)$, where $\mathsf{K}_{\mathsf{PRF}}$ is a key for PRF sampled in KeyGen. The mask m_i is used to encrypt the content of T_{len}. After initializing the table T_{len} with N random entries of size $\log(N)$ bits and T_{full} with N/p (arbitrary) lists of size p, she sets $T_{\mathsf{len}}[w_i] = x_i \oplus m_i$ and $T_{\mathsf{full}}[w \mid\mid i] = L_{i,j}$ for $j \in [1, x-1]$. Next, she generates $(B_i)_{i=1}^m$ as before with the incomplete lists L_{i,x_i} except that the bin choices for list L_{i,x_i} are $(\alpha_{i,1}, \alpha_{i,2}) \leftarrow \mathsf{H}(w_i \mid\mid x_i)$. Note that we need to also hash the counter x_i, as after some updates, the incomplete sublist of w_i might become full and a new incomplete sublist has to be started. When the new incomplete sublist gets inserted with $\mathsf{L2C}$, it is interpreted as a new ball and new bins need to be chosen. Finally, she encrypts the content of T_{full} and returns $\mathsf{EDB} = (T_{\mathsf{len}}, T_{\mathsf{full}}, (B_i^{\mathsf{enc}})_{i=1}^m)$.

Search. For search queries on keyword w, the client outputs mask $m \leftarrow \mathsf{PRF}_{\mathsf{K}_{\mathsf{PRF}}}(w)$ in addition to w. The server uses this mask to decrypt the number of sublists $x \leftarrow T_{\mathsf{len}}[w] \oplus m$, retrieves $x-1$ encrypted sublists $L_i \leftarrow T_{\mathsf{full}}[w \mid\mid i]$ from the table for $i \in [1, x-1]$ and the two bins $B_{\alpha_1}^{\mathsf{enc}}$ and $B_{\alpha_2}^{\mathsf{enc}}$ via $(\alpha_1, \alpha_2) \leftarrow \mathsf{H}(w \mid\mid x)$. Finally, the server sends the encrypted bins and sublists to the client. Clearly, the client obtains all matching identifiers after decrypting the received lists and bins.

Update. For update queries on keyword w and list L' of (at most p) new identifiers[2], the client generates mask m as before and sends (w, m) to the server. The server again decrypts x from T_{len} and sends $B_{\alpha_1}^{\mathsf{enc}}, B_{\alpha_2}^{\mathsf{enc}}$ to the client. In addition, the server already sends the bins $B_{\alpha_3}^{\mathsf{enc}}, B_{\alpha_4}^{\mathsf{enc}}$ for $\alpha_3, \alpha_4 \leftarrow \mathsf{H}(w \mid\mid x+1)$ to the client (in case the incomplete list overflows). The client now retrieves the old (incomplete) list L of identifiers matching w from the decrypted bins $B_{\alpha_1}, B_{\alpha_2}$. We distinguish two cases:

1. If $L \cup L'$ contains more than p identifiers, the client sets $L^{\mathsf{new}} = L \cup L'$ and marks (w, L) as a residual ball inside $B_{\alpha_1}, B_{\alpha_2}$. Then, she splits L^{new} into two sublists $L^{=p}$ with p identifiers and $L^{\leq p}$ of at most p identifiers. The client then inserts list $L^{\leq p}$ into bins $B_{\alpha_3}, B_{\alpha_4}$ via $\mathsf{L2C.InsertBall}((w, L^{\leq p}), B_{\alpha_3}, B_{\alpha_4})$ and sends the updated (reencrypted) bins $\{B_i^{\mathsf{enc}}\}_{i=1}^4$ such as encrypted list $L^{\mathsf{enc}} = \mathsf{Enc}_{\mathsf{K}_{\mathsf{Enc}}}(L^{=p})$ to the server.

2. Otherwise, the client proceeds as before, i.e.adds the new identifiers L' to ball (w, L) via $\mathsf{UpdateBall}$ and reencrypts the received bins.

Finally, the server replaces the old bins with the reencrypted bins, and if she received an encrypted list L^{new}, she stores the received list in $T_{\mathsf{full}}[w \mid\mid x+1] = L^{\mathsf{enc}}$ and updates $T_{\mathsf{len}}[w] = x+1$.

Leakage profile. Now, search and update queries $\mathsf{LayeredSSE}$ leak the number of sublists $x = \lceil \ell/p \rceil$ for a given keyword w with ℓ matching identifiers. Further, updates leak when a list was completed (so also the value $\lceil (\ell + |L'|)/p \rceil$). This is captured in the leakage function $\mathcal{L}_{\mathsf{l2c}} = (\mathcal{L}_{\mathsf{Stp}}, \mathcal{L}_{\mathsf{Srch}}, \mathcal{L}_{\mathsf{Updt}})$, defined as follows. The setup leakage is $\mathcal{L}_{\mathsf{Stp}}(\mathsf{DB}, N) = N$ is the maximal size N of the database, the search leakage $\mathcal{L}_{\mathsf{Srch}}(w) = (\mathsf{qp}, \lceil \ell_i/p \rceil, \lceil d_i/p \rceil)$ is the query pattern and the number of pages required to store the inserted and deleted items, and the update leakage $\mathcal{L}_{\mathsf{Updt}}(\mathsf{op}, w, L) = (\mathsf{op}, \mathsf{qp}, \lceil (\ell_i + |L|)/p \rceil, \lceil (d_i + |L|)/p \rceil, \lceil \ell_i/p \rceil, \lceil d_i/p \rceil)$ is the operation, the query pattern and the number of pages required to store the inserted and deleted items

[2]For updates with more than p identifiers, the client can perform multiple updates.

(before and after the update). As tables T_{len} and T_{full} are encrypted, it is straightforward to adapt the security analysis in Section 5.4.2 to the extended scheme with respect to leakage function $\mathcal{L}$.

Optimized RTT

Search queries of LayeredSSE need only 1 RTT, whereas update queries unfortunately require 2 RTTs. We can use "piggybacking" in order to reduce the update RTT to 1 as follows. Instead of sending the second flow of the update query directly to the server, the client stashes the response and waits for the next query (either update or search). On the next query, the client sends the stashed response in addition to the query. The server then finishes the pending update query (by storing the received bins and updating the tables) and responds the query subsequently.

5.4.4 Efficiency

We now inspect the efficiency of LayeredSSE. Let $w_{\mathsf{tot}} = N/p$. The server stores $m = \lceil w_{\mathsf{tot}}/(\log\log\log\lambda \cdot \log\log w_{\mathsf{tot}})\rceil$ bins of size $\mathcal{O}(p\log\log\log(\lambda)\log\log(w_{\mathsf{tot}})) \cdot \mathcal{O}(\lambda)$ each, tables T_{len} with N entries of size $\log(N)$ and T_{full} with N/p entries of size $p \cdot \mathcal{O}(\lambda)$ each. (Recall that a single identifier has size $\mathcal{O}(\lambda)$.) As $N = \mathsf{poly}$, the storage efficiency is $\mathcal{O}(1)$ in total. There is no client stash required[3]. Further, the server looks up 4 bins of capacity $\widetilde{\mathcal{O}}(p\log\log(N/p))$ and $x-1$ encrypted lists of p identifiers from T_{full} for a search query on word w, where x is the number of pages needed to store the document indices matching keyword w in plaintext. Thus, the page efficiency is $\widetilde{\mathcal{O}}\left(\log\log\frac{N}{p}\right)$. This further implies that LayeredSSE has $\mathcal{O}(1)$ locality if only lists up to size p are inserted.

Theorem 5.8 (LayeredSSE)**.** *Let N be an upper bound on the size of database* DB *and p be the page size. Let $p \leq N^{1-1/\log\log\lambda}$. The scheme* LayeredSSE *is correct and $\mathcal{L}_{\mathsf{l2c}}$-adaptively semantically secure if* Enc *is* IND-CPA *secure and* H *is modeled as a random oracle. It has constant storage efficiency and $\widetilde{\mathcal{O}}(\log\log N/p)$ page efficiency.*

Proof. Efficiency and security follow from the discussions above. □

[3]The version of LayeredSSE with 1 RTT updates requires a stash of size $\widetilde{\mathcal{O}}(p\log\log(N/p))$ to temporarily store the second flow of the update query until the next query.

Chapter

6 Forward Security and Page Efficiency

Since the seminal work of Bost [Bos16], forward security has become a de facto standard in SSE. In the same article, Bost conjectures that forward security and I/O efficiency are incompatible. This explains the current status quo, where users are forced to make a difficult choice between security and efficiency. In this chapter we show that forward security and I/O efficiency can be realized simultaneously. That is, we construct an SSE scheme with I/O efficiency $\widetilde{\mathcal{O}}\left(\log\log\frac{N}{p}\right)$, storage efficiency $\mathcal{O}(1)$, with standard leakage and forward security.

Chapter content

6.1 Introduction . . . 73
6.1.1 Our Contributions . . . 75
6.1.2 Technical Overview . . . 76
6.2 SSE with Dummy Updates . . . 78
6.2.1 Security Definition . . . 78
6.2.2 A Framework to Build SSE with Dummy Updates . . . 78
6.2.3 Efficient Instantiations . . . 82
6.3 BigHermes: the Big Database Regime . . . 83
6.3.1 BigHermes_0: Amortized BigHermes . . . 83
6.3.2 BigHermes_1: BigHermes with Deamortized Communication . . . 84
6.3.3 BigHermes_2: Fully Deamortized BigHermes . . . 85
6.3.4 Security . . . 86
6.3.5 Efficiency . . . 90
6.4 SmallHermes: the Small Database Regime . . . 91
6.4.1 SmallHermes_0: Amortized SmallHermes . . . 91
6.4.2 SmallHermes_1: SmallHermes with Deamortized Communication . . . 92
6.4.3 SmallHermes_2: Fully Deamortized SmallHermes . . . 94
6.4.4 Heuristic Variant via Weighted 2C . . . 95
6.4.5 Security . . . 96
6.4.6 Efficiency . . . 97
6.5 The Hermes Scheme: Putting Everything Together . . . 97
6.5.1 Hermes . . . 98
6.5.2 Optimizations and Trade-offs . . . 98

6.1 Introduction

In the previous chapter, we give page-efficient constructions for static and dynamic schemes. The focus of this chapter is to construct page-efficient SSE with forward security.

Forward Security

Forward security was introduced in [SPS14] and asks that updates leak no information to the server. Since the first efficient construction of forward-secure SSE [Bos16], forward security has become a de facto standard in SSE [Bos16, PM21, BMO17, KMPQ21, EKPE18, DCPP22]. Forward secure SSE is useful in practice since it mitigates certain file-injection attacks [ZKP16] that exploit update leakage. Further, forward security ensures that building the database online leaks no information (except the number of update queries) to the server.

Memory Efficiency

In Chapter 5, we introduce a new notion of memory efficiency, called page efficiency. Page efficiency is the ratio of memory pages read by the server to process a query, divided by the number of pages necessary to hold the plaintext answer. From a practical standpoint, page efficiency is a very good predictor of performance on modern Solid State Drives (SSDs), whereas locality is mainly relevant for older Hard Disk Drives. From a theoretical standpoint, constant page efficiency is a weaker requirement than the combination and constant locality and constant read efficiency.

Since the work of Cash and Tessaro [CT14], many I/O-efficient schemes have been proposed. Of these, the vast majority hold in the *static* setting [CT14, ANSS16, ASS21, DP17a, DPP18, BBF+21], where the database is fixed at setup and does not support updates. To our knowledge, the only construction in the *dynamic* setting is IO-DSSE [MM17] and our scheme LayeredSSE (cf. Section 5.4).

IO-DSSE. We give a brief overview of IO-DSSE. IO-DSSE opened the way in a new area but suffers from significant limitations. Intuitively, I/O efficiency requires that document identifiers that match the same keyword should be stored in close proximity. For that purpose, IO-DSSE groups identifiers matching the same keyword into *blocks* of a fixed size p. Since the number of documents matching a given keyword need not be a multiple of p, it is usually the case that one of the blocks is *incomplete*, i.e., it contains less than p identifiers. As in other works on I/O-efficient SSE, the main technical issue is how to efficiently handle incomplete blocks. Especially, when a new document matching a given keyword is added to the database, the document identifier needs to be appended to the incomplete block associated with the keyword. In that process, it is unclear how to hide to the server which incomplete block is being modified.

The solution proposed by IO-DSSE is to store incomplete blocks in an optimized Oblivious RAM (ORAM) construction. ORAM is a generic solution to hide memory access patterns but is notoriously expensive in practice. In IO-DSSE, this approach is viable, because IO-DSSE focuses on use cases with few searchable keywords: they target a messaging app scenario and assume in their experiments that no more than 350 keywords are searchable. Because the number of keywords is small, and there can be at most one incomplete block per keyword, the ORAM overhead remains manageable. By contrast, if we were to run IO-DSSE on the English Wikipedia database (a classic target in SSE literature), which contains millions of keywords, IO-DSSE blows up the size of the database by a factor of more than 100. A second limitation of IO-DSSE is that its security claim is incorrect. It does not appear that the issue can be fixed without a significant penalty for performance. We refer to [MR23] for more details. A third limitation of IO-DSSE is that, independently of the previous security issue, IO-DSSE is not forward-secure. This puts it at odds with most of the rest of modern (non-I/O-efficient) SSE literature, where forward security has become a standard requirement [Bos16, PM21, BMO17, KMPQ21, EKPE18, DCPP22].

LayeredSSE. Similarly, our construction LayeredSSE is not forward secure. Unfortunately, LayeredSSE leaks which keyword is being updated, which is the worst case with regard to certain file-injection attacks [ZKP16].

Hierarchical Approach. Before detailing our contributions, we note that dynamic I/O-efficient SSE could in principle be built using a folklore hierarchical construction, which generically builds dynamic SSE from a static SSE scheme. That construction was sketched in [SPS14, DP17a], and studied in more detail in [DCPP22]. Following that approach, one could theoretically build a dynamic I/O-efficient SSE scheme by using a static I/O-efficient SSE scheme as an underlying static scheme. However, this quickly proves impractical. First, the approach inherently incurs a logarithmic factor in both locality and page efficiency, on top of the I/O cost of the underlying static SSE. As a result, it cannot hope to match the page efficiency of Hermes, which is sublogarithmic. The most natural candidate for the underlying static SSE is the Pluto scheme (cf. Section 5.3): it is the only one to achieve constant page efficiency, hence the only one that would not further deteriorate page efficiency beyond the log factor inherent to the approach. However, Pluto has a quadratic $\mathcal{O}(N^2)$ setup time. This is problematic, because for every N insertions, the generic construction requires building a fresh static SSE instance of size N. This implies that the average computational cost of *one* update would be $\mathcal{O}(N^2/N) = \mathcal{O}(N)$. Last but not least, the hierarchical approach requires periodically rebuilding a static SSE scheme of size N. Generically, this implies storing the *entire* database on the client side during the rebuilding phase.

6.1.1 Our Contributions

Both SSE design goals, I/O efficiency, and forward security, date back to 2014 [CT14, SPS14]. Almost a decade later, there is no satisfactory solution to achieving both at once. This is not a coincidence: the two goals seem to be fundamentally at odds with each other, and it was even conjectured in [Bos16] that both notions are incompatible. While [Bos16] argues only about locality for SSE (page efficiency did not exist at the time), the same argument directly translates to page efficiency. To sum up the argument: a correlation between the identifiers $\mathsf{DB}(w)$ matching keyword w and a newly inserted identifier also matching w breaks forward security. On the other hand, identifiers matching the same keyword w need to be stored in close proximity to satisfy page efficiency. Note that accessing correlated pages still breaks forward security. Consequently, an identifier cannot be written to the server, unless $\mathsf{DB}(w)$ is partially rewritten for each access, in an ORAM-like manner.

This state of affairs raises some troubling questions for SSE. Since I/O efficiency is the main performance bottleneck, if it is mutually exclusive with forward security, then forward security comes at a heavy performance price — hinting at a stronger form of Cash and Tessaro's impossibility result when forward security comes into play. It would also imply that the rich literature on I/O-efficient SSE will have to remain confined to static SSE, or at least non-forward-secure SSE. So far, the conjecture is supported by current work: [Bos16] conjectures that I/O efficiency and forward security are "irreconcilable notions", except via expensive constructions such as ORAM. [MM17] builds the first dynamic I/O-efficient SSE scheme to date with low update leakage (although still not quite forward-secure), but relies on ORAM and has a flawed security proof. Our construction LayeredSSE from Section 5.4 has great page efficiency but leaks the query pattern and size of the response during updates.

As our main contribution, we go against the prevailing wisdom and show that the previous conjecture is incorrect. We build the first *forward-secure* SSE scheme, Hermes, with linear server storage and sublogarithmic page efficiency $\widetilde{\mathcal{O}}(\log\log(N/p))$. Notably, Hermes is the first I/O-efficient SSE scheme with forward security. We note that this holds with regard to page efficiency: the question of building a similar scheme with respect to locality remains an intriguing open question. A brief comparison with existing schemes is given in Table 6.1. Hermes does not rely on ORAM, or any ORAM-like structure. Instead, it makes use of two new technical ideas.

First, we introduce the notion of SSE schemes supporting *dummy updates*. A dummy update can be triggered by the client at any time, and must look indistinguishable from a real update

Table 6.1: An overview of relevant dynamic SSE schemes. N is an upper bound on database size, W is an upper bound on the number of keywords, p is the page size.

SSE	Page Efficiency	St. Efficiency	Client St.	Forward Sec.
$\Sigma o\phi o\varsigma$ [Bos16]	$\mathcal{O}(p)$	$\mathcal{O}(1)$	$\mathcal{O}(W)$	✓
Π_{pack}, Π_{2lev} [CJJ+14]	$\mathcal{O}(1)$	$\mathcal{O}(p)$	$\mathcal{O}(W)$	✗
IO-DSSE [MM17]	$\mathcal{O}(\log W)$	$\mathcal{O}(1+\frac{p \cdot W}{N})$	$\mathcal{O}(W)$	✗
LayeredSSE [Section 5.4]	$\widetilde{\mathcal{O}}\left(\log\log\frac{N}{p}\right)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	✗
Hermes	$\widetilde{\mathcal{O}}\left(\log\log\frac{N}{p}\right)$	$\mathcal{O}(1)$	$\mathcal{O}(W)$	✓

in the server's view. On the other hand, we require that server storage should only grow with an upper bound N on the number of *real* entries in the database: dummy updates create no storage overhead. We present a simple framework to build an SSE scheme $\mathsf{Dummy}(\Sigma)$ supporting dummy updates, based on an underlying suitable SSE scheme Σ. The framework is based on an application of the two-choice allocation process.

The second main technical idea is a form of "deamortized" trivial ORAM. An explanation of this technique is deferred to the technical overview below. For now, we note that it involves buffering $\mathcal{O}(W)$ updates on the client side before pushing them to the server, where W is an upper bound on the number of searchable keywords. It was proved in [BF19] that single-round forward-secure SSE requires $\Omega(W)$ client storage. As a consequence, the new buffer does not increase client storage beyond a constant factor[1]. Interestingly, it is is the use of client storage that circumvents the proof sketched in [Bos16] for the incompatibility of forward security and I/O efficiency.

6.1.2 Technical Overview

I/O efficiency and forward security are two important goals of SSE research, but seemingly incompatible. On an intuitive level, this is because I/O efficiency requires that identifiers matching the same keyword should be stored close to each other, so that they can be read together efficiently when the keyword is queried. Forward security requires that when a *new* identifier matching some keyword is added, it cannot be stored close to previous identifiers for the same keyword, since that would leak information about the new identifier to the server. One way to resolve this apparent contradiction is to use ORAM, as was suggested in [Bos16], and later realized in [MM17]. We start by sketching the ORAM approach, which will serve to explain what Hermes does differently.

A natural way to build page-efficient SSE is to maintain an array of bins of capacity one page each, one bin for each keyword. When the client wishes to insert a new document identifier matching some keyword w, the new document identifier is added to the bin associated with w. Once the bin for keyword w is full, i.e. it contains a full page of identifiers matching w, the page of identifiers is inserted into a separate SSE scheme Σ that only contains full pages, and the bin is emptied. Because Σ only contains *full* pages, each page can be stored in an arbitrary location, and the scheme remains page-efficient; hence page efficiency is easy to realize. In order to search for the list of document identifiers that match the keyword w, the client needs only to fetch the bin associated with w, and query Σ on w.

The problem with this naive approach is that it is not forward-secure: during an update, the server can see which bin is accessed, hence which keyword is being updated. A generic

[1]Although $\mathcal{O}(W)$ client storage is inherent to sublogarithmic forward-secure SSE as just noted, in practice, this cost may be too high for some applications. Practical tradeoffs to reduce client storage are discussed in Section 6.5.2.

way to circumvent this leakage is to store the bins in an ORAM, which completely hides access patterns. (Roughly speaking, this approach is the one of IO-DSSE.) Despite the use of ORAM, the approach still has two issues. First, it is *still* not forward-secure: when a page becomes full, the server can observe that a new element is inserted into Σ, hence updates are not leakage-free (this leakage suffices to break the security game of forward-secure SSE). Second, ORAM incurs an $\Omega(\log W)$ overhead, where W is the number of searchable keywords, and is costly in practice.

Challenge 1: achieving forward security. To avoid leaking when a bin becomes full, we could insert a new item in Σ for every client update, regardless of whether the bin is actually full. A bin becomes full after it receives p client updates, where p is the page size (counted in number of memory words, identified here with the size of a document identifier). Asymptotically, we would have storage efficiency $\mathcal{O}(p)$ instead of the desired $\mathcal{O}(1)$. In practice, for 64-bit identifiers and 4kB memory pages, $p = 512$: blowing the size of Σ by a factor p is not acceptable. Instead, we introduce the idea of SSE supporting dummy updates. When a bin is full, the full page is inserted into Σ; when it is not full, the client issues a dummy update to Σ. The promise of dummy updates is that they should be indistinguishable from real updates in the server's view. At the same time, we arrange that they cost nothing in storage: in our actual construction dummy updates do not alter the contents of the encrypted database.

Challenge 2: realizing dummy updates. At a high level, building SSE with dummy updates comes down to building a key-value store that is amenable to fake key queries. For that purpose, we use the two-choice allocation process. In a two-choice process, n values are stored in m bins of fixed capacity c. Each value for key k is stored in the bin $H_1(k)$ or $H_2(k)$, where H_1, H_2 are hash functions mapping into $[1, m]$. We pad the bins to their full capacity c, and encrypt them with an IND-CPA scheme. Intuitively, simulating a dummy query is simple: the client fetches two uniformly random bins, re-encrypts them, and re-uploads them to the server. Setting $c = \widetilde{\mathcal{O}}(\log \log n)$, $m = \widetilde{\mathcal{O}}(n/\log \log n)$ suffices to ensure a negligible probability of overflow, with constant storage efficiency. This idea can be adapted to the SSE setting. We realize it as a framework that builds a scheme $\mathsf{Dummy}(\Sigma)$ supporting dummy updates, based on an underlying suitable forward-secure scheme Σ.

Challenge 3: dispensing with ORAM. Recall that our scheme uses W bins, each of capacity one page. As noted in the introduction, single-round forward-secure SSE requires $\Omega(W)$ client storage [BF19]. If we buffer W updates on the client before pushing them to the server, we can afford to scan all W bins. This costs W page accesses per W updates, hence $\mathcal{O}(1)$ *amortized* page efficiency. Another way to view this process is that we are performing a trivial ORAM (i.e. reading the entire array of bins), amortized over W updates. This basic idea can be deamortized, in such a way that each client update generates $\mathcal{O}(1)$ page accesses in the worst case. The deamortization is somewhat subtle and proceeds differently depending on the regime of global parameters N, W, and p. For that reason, we introduce two schemes, BigHermes and SmallHermes, respectively for the case $N \geq pW$ and $N \leq pW$. Hermes is the combination of those two schemes, one for each regime.

In the end, the storage and page efficiency of Hermes reduce to those of the underlying scheme supporting dummy updates. Because that scheme relies on the two-choice process, this works out to $\widetilde{\mathcal{O}}(\log \log(N/p))$ page efficiency, and constant storage efficiency. While the ideas of buffering and deamortization, as well as dummy updates, may be natural in hindsight, we view them as the key contributions of this chapter: to our knowledge, they realize forward security in a way that is fundamentally different from how it was realized in prior works — and one that happens to be compatible with I/O efficiency. We note that both new techniques (the use of dummy updates, for forward security; and the replacement of standard ORAM with "deamortized" trivial ORAM, for efficiency), are modular: we could have introduced intermediate schemes that realize one without the other, although we did not see a compelling reason for it.

6.2 SSE with Dummy Updates

A first key technique for our results is the introduction of the notion of *dummy updates*. An SSE schemes supports dummy updates if its interface is equipped with a new operation $\mathsf{DummyUpdate}$, taking as input only the client's master key K. For technical reasons, we also require that Setup receives an additional parameter D, an upper bound on the total number of dummy updates.

6.2.1 Security Definition

Informally, an SSE scheme supporting dummy updates is said to be secure if it is secure in the same sense as a normal SSE scheme, with the added requirement that dummy updates should be indistinguishable from real updates from the server's perspective. (A subtle but important point is that later constructions will ensure that server storage does not depend on D: dummy updates will look indistinguishable from real updates, without actually affecting server storage.)

The security definition for SSE supporting dummy updates is given in Definition 6.1, with the associated security game in Algorithm 6. Note that this definition naturally extends the standard definition (Definition 3.33).

Definition 6.1 (Adaptive Semantic Security with Dummy Updates)**.** Let Σ be an SSE scheme supporting dummy updates, $\mathcal{A}$ a stateful PPT adversary, and Sim a stateful PPT simulator. Let $q \in \mathbb{N}$, and let $\mathcal{L} = (\mathcal{L}_{\mathsf{Stp}}, \mathcal{L}_{\mathsf{Srch}}, \mathcal{L}_{\mathsf{Updt}})$ be a leakage function. The games $\text{SSEREAL}^{\mathsf{dum}}_{\Sigma,\mathcal{A}}$ and $\text{SSEIDEAL}^{\mathsf{dum}}_{\Sigma,\mathcal{A},\mathcal{L},\mathcal{A}}$ are defined in Algorithm 6. Σ is *$\mathcal{L}$-adaptively secure with support for dummy updates* if $\mathcal{L}_{\mathsf{Updt}}$ does not depend on its first input op, and if for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator Sim such that:

$$|\Pr[\text{SSEREAL}^{\mathsf{dum}}_{\Sigma,\mathcal{A}}(\lambda) = 1] - \Pr[\text{SSEIDEAL}^{\mathsf{dum}}_{\Sigma,\mathsf{Sim},\mathcal{L},\mathcal{A}}(\lambda) = 1]| = \mathsf{negl}(\lambda).$$

Algorithm 6 Security games for SSE supporting dummy updates.

$\text{SSEREAL}^{\mathsf{dum}}_{\Sigma,\mathcal{A}}$

$\mathsf{K} \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$
$(\mathsf{DB}, N, D, W, \mathsf{st}_\mathcal{A}) \leftarrow \mathcal{A}(1^\lambda)$
$\mathsf{EDB} \leftarrow \Sigma.\mathsf{Setup}(\mathsf{K}, N, D, W, \mathsf{DB})$
send EDB to $\mathcal{A}$
for all $1 \le i \le q$ **do**
  $(\mathsf{op}_i, \mathsf{in}_i, \mathsf{st}_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A})$
  if $\mathsf{op}_i = \mathsf{srch}$ **then**
    Parse $\mathsf{in}_i = w_i$
    $\Sigma.\mathsf{Search}_C(\mathsf{K}, w_i) \leftrightarrow \mathcal{A}(\mathsf{st}_\mathcal{A})$
  else if $\mathsf{op}_i \in \{\mathsf{add}, \mathsf{del}\}$ **then**
    Parse $\mathsf{in}_i = (w_i, \mathsf{id}_i)$
    $\Sigma.\mathsf{Update}_C(\mathsf{K}, w_i, \mathsf{id}_i, \mathsf{op}_i) \leftrightarrow \mathcal{A}(\mathsf{st}_\mathcal{A})$
  else
    $\Sigma.\mathsf{DummyUpdate}_C(\mathsf{K}) \leftrightarrow \mathcal{A}(\mathsf{st}_\mathcal{A})$
output bit $b \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A})$

$\text{SSEIDEAL}^{\mathsf{dum}}_{\Sigma,\mathsf{Sim},\mathcal{L},\mathcal{A}}$

$(\mathsf{DB}, N, D, W, \mathsf{st}_\mathcal{A}) \leftarrow \mathcal{A}(1^\lambda)$
$\mathsf{EDB} \leftarrow \mathsf{Sim}(\mathcal{L}_{\mathsf{Stp}}(\mathsf{DB}, N, D, W))$
send EDB to $\mathcal{A}$
for all $1 \le i \le q$ **do**
  $(\mathsf{op}_i, \mathsf{in}_i, \mathsf{st}_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A})$
  if $\mathsf{op}_i = \mathsf{srch}$ **then**
    $\mathsf{Sim}(\mathcal{L}_{\mathsf{Srch}}(\mathsf{in}_i)) \leftrightarrow \mathcal{A}(\mathsf{st}_\mathcal{A})$
  else
    $\mathsf{Sim}(\mathcal{L}_{\mathsf{Updt}}(\mathsf{op}_i, \mathsf{in}_i)) \leftrightarrow \mathcal{A}(\mathsf{st}_\mathcal{A})$
output bit $b \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A})$

6.2.2 A Framework to Build SSE with Dummy Updates

We now describe a framework that constructs an SSE scheme $\mathsf{Dummy}(\Sigma)$ supporting dummy updates, based on an underlying SSE scheme Σ. Provided Σ is *suitable* in a sense that will be defined shortly, the resulting scheme $\mathsf{Dummy}(\Sigma)$ achieves the following features:

- It is a secure SSE scheme supporting dummy updates (cf. Definition 6.1).
- Server storage grows only with the number of *real* updates; in particular, it does not depend on the upper bound D on the number of dummy updates.
- Relative to the base scheme Σ, $\mathsf{Dummy}(\Sigma)$ only incurs a $\widetilde{\mathcal{O}}(\log\log N)$ overhead in communication.

The definition of a *suitable* SSE is given next. Essentially, it requires that server storage should behave like a key-value store. This is how most forward-secure SSE schemes operate. It also requires that running Setup on a non-empty initial database DB is equivalent to running Setup on an empty database, then performing updates to add the contents of DB. Here, "equivalent" means that the client and server states at the outcome of either process are distributed identically. This condition can be fulfilled trivially by any SSE scheme. It is not strictly necessary, but makes the description of the framework, and its security proof, more consise.

Definition 6.2 (Suitable SSE)**.** We say that an SSE scheme Σ is *suitable* if there exist a key space $\mathcal{K}$, a token space $\mathcal{T}$, and a map $\mathsf{keys} : \mathcal{T} \mapsto \mathcal{K}$ such that: (1) $\Sigma.\mathsf{Setup}(\mathsf{K}, N, W, \mathsf{DB})$ outputs an encrypted database EDB in the form of an encrypted key-value store that maps a key to encrypted identifiers. (2) $\Sigma.\mathsf{Search}(K, w; \mathsf{EDB})$ is a two-step protocol in which the client first sends a token $\tau \in \mathcal{T}$ and the server responds with $\mathsf{EDB}[k_1], \ldots, \mathsf{EDB}[k_q]$ for $k_1, \ldots, k_q \leftarrow \mathsf{keys}(\tau)$. (3) $\Sigma.\mathsf{Update}(K, (w, \mathsf{id}), \mathsf{op}; \mathsf{EDB})$ is a one-step protocol in which the client sends a key $k \in \mathcal{K}$ and value $v = \mathsf{Enc}_{\mathsf{K}_{\mathsf{Enc}}}(\mathsf{id})$ and the server stores v in EDB at position k, i.e.sets $\mathsf{EDB}[k] = v$. (4) Running the setup routine $\mathsf{Setup}(\mathsf{K}, N, W, \mathsf{DB})$ is equivalent to running the setup $\mathsf{Setup}(\mathsf{K}, N, W, \emptyset)$ with an empty database and subsequently performing an update operation for each keyword-identifier pair $(w, \mathsf{id}) \in \mathsf{DB}$ locally. (5) Σ is forward-secure.

Construction

A detailed description is given in Algorithm 7. Let us explain it here in text. Let Σ be a suitable SSE scheme. First, observe that it is easy to add dummy updates to Σ if we are willing to let server storage grow linearly with the number of dummy updates: we could simply let $\mathsf{DummyUpdate}$ perform a real update with a fresh keyword-identifier pair. Because Σ is forward-secure, the leakage of either type of update would be $\perp$. The problem is that the server would have to store the keyword-identifier pairs arising from dummy updates, potentially blowing up storage overhead. This is what we wish to avoid.

Instead, the idea is to wrap Σ inside an encrypted two-choice allocation scheme (cf. Chapter 4). First, $\mathsf{Dummy}(\Sigma)$ initializes a two-choice scheme with $m = N/\widetilde{\mathcal{O}}(\log\log N)$ bins $B_1, \ldots, B_m$, each of capacity $\widetilde{\mathcal{O}}(\log\log N)$ (where one unit corresponds to the storage cost of one identifier in Σ). Because Σ is suitable, we know server storage in Σ behaves like a key-value store: for each update with token τ, the server stores the corresponding data items under the keys $\mathsf{keys}(\tau)$. Let $\mathsf{H} : \mathcal{K} \to \{1, \ldots, m\}^2$ map keys to pairs of bins. In $\mathsf{Dummy}(\Sigma)$, whenever Σ would store a data item under a key k, the same item is instead stored in one of the two bins B_α, B_β, where $(\alpha, \beta) \leftarrow \mathsf{H}(k)$. The destination bin is chosen among B_α, B_β according to the two-choice process: the item is inserted into whichever bin currently contains fewer items.

In $\mathsf{Dummy}(\Sigma)$, the client always downloads and sends back full bins, padded to their maximal capacity $\widetilde{\mathcal{O}}(\log\log N)$, and encrypted under a key $\mathsf{K}_{\mathsf{Enc}}$ known only to the client. This is where the $\widetilde{\mathcal{O}}(\log\log N)$ overhead in communication comes from. On the other hand, thanks to the properties of the two-choice process, dummy updates can be realized easily: the client simply asks to access two bins $(\alpha, \beta) \leftarrow \mathsf{H}(k)$ for a fresh key k, re-encrypts them, and re-uploads them. This matches the behavior of real updates (up to the IND-CPA security of Enc, and the pseudo-randomness of H, modeled as a random oracle). In more detail, the key k for dummy

updates is generated by $\Sigma.\mathsf{Update}$ using a reserved dummy keyword w_{dum}, and a fresh identifier id chosen by the client.

Remark. Although the outline given above is natural, the detailed construction in Algorithm 7 involves some subtlety. During setup, Σ receives as upper bound for the size of the database $N+D$, rather than just N. This is useful for the security proof. It also means that the encrypted database *generated by* Σ may scale with D. However, that database is not needed to run $\mathsf{Dummy}(\Sigma)$, so this has no impact on storage efficiency. To see that the database generated by Σ is not needed by $\mathsf{Dummy}(\Sigma)$, observe in Algorithm 7 that $\mathsf{Dummy}(\Sigma)$ crucially only makes use of the *client-side* part of the protocols of Σ; the only relevant aspect of the server-side part of those protocols is the keys map. This is made possible by the suitability assumption on Σ, which is used very strongly.

Algorithm 7 $\mathsf{Dummy}(\Sigma)$

$\mathsf{Dummy}(\Sigma).\mathsf{Setup}(\mathsf{K}, N, W, D, \mathsf{DB})$

1: Pick dummy keyword w_{dum}
2: $\mathsf{EDB}_\Sigma \leftarrow \Sigma.\mathsf{Setup}(\mathsf{K}_\Sigma, N+D, W+1, \emptyset)$
3: Initialize $m = N/\widetilde{\mathcal{O}}(\log\log N)$ empty bins $B_1, \ldots, B_m$ of capacity $\widetilde{\mathcal{O}}(\log\log N)$.
4: **for all** $(w, \mathsf{id}) \in \mathsf{DB}$ **do**
5: $\quad (k, v) \leftarrow \Sigma.\mathsf{Update}_C(\mathsf{K}, (w, \mathsf{id}), \mathsf{add})$
6: $\quad (\alpha, \beta) \leftarrow \mathsf{H}(k)$
7: $\quad$ Insert id into the bin B_γ with fewest items among B_α, B_β
8: **return** $\mathsf{EDB} = (\mathsf{Enc}_{\mathsf{K}_{\mathsf{Enc}}}(B_i))_{i=1}^m$

$\mathsf{Dummy}(\Sigma).\mathsf{Update}(\mathsf{K}, (w, \mathsf{id}), \mathsf{op}; \mathsf{EDB})$

Client:

1: $(k, v) \leftarrow \Sigma.\mathsf{Update}_C(\mathsf{K}_\Sigma, (w, \mathsf{id}), \mathsf{add})$
2: **send** k

Server:

1: $(\alpha, \beta) \leftarrow \mathsf{H}(k)$
2: **send** $B_\alpha^{\mathsf{enc}}, B_\beta^{\mathsf{enc}}$

Client:

1: Decrypt $B_\alpha^{\mathsf{enc}}, B_\beta^{\mathsf{enc}}$ to B_α, B_β
2: Insert id into the bin B_γ with fewest identifiers among B_α, B_β
3: **send** re-encrypted B_α, B_β

Server:

1: Replace $B_\alpha^{\mathsf{enc}}, B_\beta^{\mathsf{enc}}$ with received bins

$\mathsf{Dummy}(\Sigma).\mathsf{KeyGen}(1^\lambda)$

1: Sample $\mathsf{K}_\Sigma \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$ and key $\mathsf{K}_{\mathsf{Enc}}$ for Enc
2: **return** $\mathsf{K} = (\mathsf{K}_\Sigma, \mathsf{K}_{\mathsf{Enc}})$

$\mathsf{Dummy}(\Sigma).\mathsf{Search}(\mathsf{K}, w; \mathsf{EDB})$

Client:

1: $\tau \leftarrow \Sigma.\mathsf{Search}_C(\mathsf{K}_\Sigma, w)$
2: **send** τ

Server:

1: $k_1, \ldots, k_q \leftarrow \mathsf{keys}(\tau)$
2: $(\alpha_i, \beta_i) \leftarrow \mathsf{H}(k_i)$ for $i \in [1, q]$
3: **send** $\{B_{\alpha_i}^{\mathsf{enc}}, B_{\beta_i}^{\mathsf{enc}}\}_{i=1}^q$

$\mathsf{Dummy}(\Sigma).\mathsf{DummyUpdate}(\mathsf{K}; \mathsf{EDB})$

Client:

1: Pick fresh identifier id
2: $(k, v) \leftarrow \Sigma.\mathsf{Update}_C(\mathsf{K}_\Sigma, (w_{\mathsf{dum}}, \mathsf{id}), \mathsf{add})$
3: **send** k

Server:

1: $(\alpha, \beta) \leftarrow \mathsf{H}(k)$
2: **send** $B_\alpha^{\mathsf{enc}}, B_\beta^{\mathsf{enc}}$

Client:

1: **send** re-encrypted B_α, B_β

Server:

1: Replace $B_\alpha^{\mathsf{enc}}, B_\beta^{\mathsf{enc}}$ with received bins

Security

The security of $\mathsf{Dummy}(\Sigma)$ follows naturally from that of Σ. Indeed, $\mathsf{Dummy}(\Sigma)$ essentially amounts to running Σ inside encrypted bins, with the difference that Σ is scaled for size $N+D$. As a result, $\mathsf{Dummy}(\Sigma)$ intuitively has at most the same leakage as Σ for Setup, Search and Update, except that D is additionally leaked. Regarding $\mathsf{DummyUpdate}$, the only difference between real and dummy updates is that a new identifier is inserted in the bin with the former, while the bins are re-encrypted without modifying their content with the latter. Since Enc is

IND-CCA-secure, and bins are always padded to ther full size (cf. Section 3.1), the two behaviors are indistinguishable.

Theorem 6.3. *Let Σ be a suitable, $\mathcal{L}^{\mathsf{fs}}$-adaptively secure SSE scheme. Let* Enc *be an IND-CPA secure encryption scheme. Let* H *be a random oracle. Let $N \geq \lambda$, and $D = \mathsf{poly}(N)$.* $\mathsf{Dummy}(\Sigma)$ *is a correct and secure SSE scheme supporting dummy updates with respect to leakage $\mathcal{L}_{\mathsf{dum}} = (\mathcal{L}_{\mathsf{Stp}}, \mathcal{L}_{\mathsf{Srch}}, \mathcal{L}_{\mathsf{Updt}})$, where $\mathcal{L}_{\mathsf{Stp}}(\mathsf{DB}, N, D, W) = (N, D, W), \mathcal{L}_{\mathsf{Srch}}(w) = (\mathsf{qp}, \ell)$, and $\mathcal{L}_{\mathsf{Updt}}(\mathsf{op}, w, \mathsf{id}) = \bot$.*

Before we give a formal proof, we give some intuition. Because $\mathsf{2C}$ guarantees a maximum load of $\widetilde{\mathcal{O}}(\log\log N)$ with overwhelming probability if $N \geq \lambda$ (cf. Lemma 4.5), correctness follows from the correctness of Σ. We turn to security. Let Sim_Σ be a simulator for Σ. During setup, the client initializes the encrypted database EDB_Σ of Σ and outputs $m = N/\widetilde{\mathcal{O}}(\log\log N)$ encrypted bins. The output leaks nothing but N, since the bins are encrypted. Note that for subsequent updates and searches, the state of Sim_Σ still needs to be initialized by simulating EDB_Σ. (Here, we use the fourth property of Definition 6.2.) This potentially leaks N, D and W. For search queries, the search token can be sampled via Sim_Σ. This requires the query pattern qp_Σ of Σ and the length ℓ of the identifier list matching the searched keyword. Note that each search and update query induces a corresponding query on Σ. Thus, the query pattern of $\mathsf{Dummy}(\Sigma)$ and Σ are equivalent. Consequently, ℓ and qp are leaked. Similarly, all updates (including dummy updates) can be simulated via Sim_Σ. As updates leak nothing in Σ, neither dummy nor real updates have any leakage.

Now, we show semantic security of $\mathsf{Dummy}(\Sigma)$ with dummy updates, if Σ is suitable and forward secure. Note that correctness follows from Lemma 4.5 and the proof is straight-forward.

Proof. Let Sim denote the simulator and $\mathcal{A}$ an arbitrary honest-but-curious PPT adversary. Further, let Sim_Σ be a simulator for Σ with leakage $\mathcal{L}^{\mathsf{fs}}$. Initially, Sim receives $\mathcal{L}_{\mathsf{Stp}}(\mathsf{DB}, N, D, W) = (N, D, W)$ and later, an series of search and update queries with input $\mathcal{L}_{\mathsf{Srch}}(w) = (\mathsf{qp}, \ell)$ and $\mathcal{L}_{\mathsf{Updt}}(op, w, \mathsf{id}) = \bot$ respectively. First, Sim generates an encryption key $\mathsf{K}'_{\mathsf{Enc}}$ and initializes $m = N/\widetilde{\mathcal{O}}(\log\log N)$ bins $B_1, \ldots, B_m$ zeroed out up to size $\widetilde{\mathcal{O}}(\log\log N)$. Next, she sets $\mathsf{EDB}_\Sigma \leftarrow \mathsf{Sim}_\Sigma(N, W)$ and outputs $\mathsf{EDB}' = (\mathsf{Enc}_{\mathsf{K}'_{\mathsf{Enc}}}(B_1), \ldots, \mathsf{Enc}_{\mathsf{K}'_{\mathsf{Enc}}}(B_m))$. Next, Sim simulates the search and update queries.

For search queries, Sim receives the query pattern qp and the length ℓ of the identifier list matching the searched keyword. Sim outputs $\tau \leftarrow \mathsf{Sim}_\Sigma(\mathsf{qp}, \ell)$. For update queries, Sim receives no input. Sim sets $(k, v) \leftarrow \mathsf{Sim}_\Sigma(\bot)$ and outputs k. We now show that the real game is indistinguishable from the ideal game. For this, we define five hybrid games.

- Hybrid 0 is identical to the real game.
- Hybrid 1 is the same as Hybrid 0 raises a flag flag when a bin overflows its capacity during setup or update. As the client never inserts an identifier if the bin were to overflow, the probability of a flag being raised is 0. Thus, Hybrid 1 and Hybrid 0 are indistinguishable.
- Hybrid 2 is the same as Hybrid 1 except in Setup, the encrypted database EDB replaced with EDB', and in Update, the client sends back fresh encryptions of $\log\log(m)$ zeros. Since Enc is IND-CPA secure, it follows that the advantage of an adversary trying to distinguish Hybrid 2 from Hybrid 1 is negligible.
- Hybrid 3 is given in Algorithm 8 and is the same as Hybrid 2 except in Setup, the client performs the setup of Σ with DB directly instead of performing the updates locally. As Σ is suitable, Hybrid 3 and Hybrid 2 are identically distributed and thus indistinguishable.
- Hybrid 4 is given in Algorithm 9 and is the same as Hybrid 3, except the scheme Σ is simulated via Sim_Σ. Hybrid 4 and Hybrid 3 are indistinguishable, as Σ is $\mathcal{L}^{\mathsf{fs}}$-adaptively secure.

– Hybrid 5 is identical to the ideal game. Hybrid 5 and Hybrid 4 are identically distributed and thus, indistinguishable. □

Algorithm 8 Hybrid 3

$\underline{\mathsf{Dummy}(\Sigma).\mathsf{Setup}(\mathsf{K}, N, W, \mathsf{DB})}$

1: Set $N_\Sigma \leftarrow N + D$ and $W_\Sigma \leftarrow W + 1$
2: Generate random $\mathsf{K}'_{\mathsf{Enc}}$
3: Generate dummy keyword w_{dum} and identifier $\mathsf{id}_{\mathsf{dum}}$
4: Let $\mathsf{EDB}_\Sigma \leftarrow \mathsf{Dummy}(\Sigma).\mathsf{Setup}(\mathsf{K}, N_\Sigma, W_\Sigma, \mathsf{DB})$
5: Initialize $m = N/\widetilde{\mathcal{O}}(\log\log N)$ bins $B_1, ..., B_m$ containing $\widetilde{\mathcal{O}}(\log\log N)$ zeros
6: **return** output $\mathsf{EDB} = (\mathsf{Enc}_{\mathsf{K}'_{\mathsf{Enc}}}(B_i))_{i=1}^m$

$\underline{\mathsf{Dummy}(\Sigma).\mathsf{Search}(\mathsf{K}, w; \mathsf{EDB})}$

Client:
1: Set $\tau \leftarrow \Sigma.\mathsf{Search}_C(\mathsf{K}, w)$
2: **send** τ

$\underline{\mathsf{Dummy}(\Sigma).\mathsf{Update}(\mathsf{K}, (w, \mathsf{id}), \mathsf{op}; \mathsf{EDB})}$

Client:
1: Set $(k, v) \leftarrow \Sigma.\mathsf{Update}_C(\mathsf{K}, (w, \mathsf{id}), \mathsf{add})$
2: **send** k

Client:
1: Receive $B^{\mathsf{enc}}_\alpha, B^{\mathsf{enc}}_\beta$
2: **send** reencrypted B_α, B_β

$\underline{\mathsf{Dummy}(\Sigma).\mathsf{DummyUpdate}(\mathsf{K}; \mathsf{EDB})}$

Client:
1: Set $d \leftarrow (w_{\mathsf{dum}}, \mathsf{id}_{\mathsf{dum}})$
2: Set $(k, v) \leftarrow \Sigma.\mathsf{Update}_C(\mathsf{K}, d, \mathsf{add})$
3: **send** k

Client:
1: Receive $B^{\mathsf{enc}}_\alpha, B^{\mathsf{enc}}_\beta$
2: **send** reencrypted B_α, B_β

Algorithm 9 Hybrid 4

$\underline{\mathsf{Setup}(\mathcal{L}_{\mathsf{Stp}}(N, W, \mathsf{DB}))}$

1: Set $N_\Sigma \leftarrow N + D$ and $W_\Sigma \leftarrow W + 1$
2: Generate random $\mathsf{K}'_{\mathsf{Enc}}$
3: Let $\mathsf{EDB}_\Sigma \leftarrow \mathsf{Sim}_\Sigma(\mathcal{L}^{\mathsf{fs}}_{\mathsf{Stp}}(N_\Sigma, W_\Sigma, \emptyset))$
4: Initialize m empty bins $B_1, ..., B_m$
5: Fill each bin up to capacity $\widetilde{\mathcal{O}}(\log\log N)$ with zeros
6: **return** output $\mathsf{EDB} = (\mathsf{Enc}_{\mathsf{K}'_{\mathsf{Enc}}}(B_i))_{i=1}^m$

$\underline{\mathsf{Search}_C(\mathcal{L}_{\mathsf{Srch}}(w))}$

Client:
1: Simulate $\tau \leftarrow \mathsf{Sim}_\Sigma(\mathcal{L}_{\mathsf{Srch}}(w))$
2: **send** τ

$\underline{\mathsf{Update}_C(\mathcal{L}_{\mathsf{Updt}}(\mathsf{op}, w, \mathsf{id}) = \perp)}$

Client:
1: Simulate $(k, v) \leftarrow \mathsf{Sim}_\Sigma(\perp)$
2: **send** k

Client:
1: Receive $B^{\mathsf{enc}}_\alpha, B^{\mathsf{enc}}_\beta$
2: **send** reencrypted B_α, B_β

$\underline{\mathsf{DummyUpdate}_C()}$

Client:
1: Simulate $(k, v) \leftarrow \mathsf{Sim}_\Sigma(\perp)$
2: **send** k

Client:
1: Receive $B^{\mathsf{enc}}_\alpha, B^{\mathsf{enc}}_\beta$
2: **send** reencrypted B_α, B_β

6.2.3 Efficient Instantiations

We now evaluate the efficiency of our framework. Notably, if Σ is *efficient* in a certain sense, so is $\mathsf{Dummy}(\Sigma)$.

Definition 6.4. A suitable SSE scheme Σ is said to be *efficient* if:

– Σ has $\mathcal{O}(1)$ storage efficiency;

– If EDB contains ℓ values matching keyword w, then for $\tau \leftarrow \Sigma.\mathsf{Search}_C(\mathsf{K}, w)$, $|\mathsf{keys}(\tau)| = \ell$.

Later, we will only use Σ to store full pages. That is, the atomic items stored in Σ will be identifier lists of size p, rather than single identifiers. Each key will map to one list of size p. Each access to the encrypted database EDB then translates to one page access, and retrieves p identifiers. On the other hand, if Σ is efficient in the sense above, the number of accesses is minimal, hence Σ with full-page items has page efficiency $\mathcal{O}(1)$. $\mathsf{Dummy}(\Sigma)$ then has page efficiency $\widetilde{\mathcal{O}}(\log\log N)$. Moreover, if Σ has $\mathcal{O}(1)$ storage efficiency, so does $\mathsf{Dummy}(\Sigma)$.

Putting both remarks together, we see that if Σ is efficient per Definition 6.4, then $\mathsf{Dummy}(\Sigma)$ has storage efficiency $\mathcal{O}(1)$, and when used to store full-page items as outlined above, it has page efficiency $\widetilde{\mathcal{O}}(\log\log N)$. To instantiate this idea, the $\Sigma o\phi o\varsigma$ [Bos16] and Diana [BMO17] schemes are good choices: they are both suitable, efficient, and $\mathcal{L}^{\mathsf{fs}}$-adaptively secure (assuming identifiers are encrypted before being stored on the server).

Remark 6.5. The $\mathsf{Dummy}(\Sigma)$ framework technically does not exclude the possibility that the sizes of individual search tokens and keys could scale with D, insofar as they are produced by Σ, and $N + D$ is an input of $\Sigma.\mathsf{Setup}$. In practice, the overhead is at most constant for $\Sigma o\phi o\varsigma$ and Diana, and can be eliminated entirely with a careful instantiation. To simplify the presentation, we have not added formal requirements for that purpose at the framework level.

6.3 BigHermes: the Big Database Regime

We are now ready to present our main construction, Hermes. The construction differs depending on whether $N \geq pW$ or $N < pW$. This section presents $\mathsf{BigHermes}$, which deals with the case $N \geq pW$. Throughout the section, we assume $N \geq pW$, and let Σ_{dum} be an efficient forward-secure SSE supporting dummy queries, in the sense of Section 6.2.

The final $\mathsf{BigHermes}$ construction is rather involved. To simplify the explanation, we build $\mathsf{BigHermes}$ progressively. We introduce three variants: $\mathsf{BigHermes}_0$, $\mathsf{BigHermes}_1$, $\mathsf{BigHermes}_2$. The three variants are gradually more complex, but achieve gradually stronger properties. The difference lies in the efficiency guarantees. $\mathsf{BigHermes}_0$ uses the idea of dummy updates from Section 6.2 to achieve sublogarithmic page efficiency, communication and time complexity overheads; but only in an amortized sense. $\mathsf{BigHermes}_1$ shows how $\mathsf{BigHermes}_0$ can be deamortized in page efficiency and communication, notably without the use of ORAM found in prior work [MM17]. We provide an overview of the data structures used in $\mathsf{BigHermes}_1$ in Table 6.2. Finally, $\mathsf{BigHermes}_2$ builds on $\mathsf{BigHermes}_1$ to deamortize time complexity, and completes the construction.

Table 6.2: Overview of the data structures used in $\mathsf{BigHermes}_1$ (see Algorithm 10).

Data Structure	Comments
Bins $B_{w_1}, \ldots, B_{w_W}$	Each bin B_w stores up to p identifiers matching keyword w
$\mathsf{CB}_{\mathsf{new}} : w \mapsto L$	Table that buffers for each keyword w fresh identifiers L matching w (up to W identifiers in total)
$\mathsf{CB}_{\mathsf{out}} : w \mapsto (L, L)$	Table that buffers for keyword w up to 2 identifier lists of size at most p
$\mathsf{CFP} : [1, W/p] \mapsto L$	Table that maps an index to either a list L of p identifiers or $\perp$
$T_{\mathsf{len}} : w \mapsto \ell$	Stores for keyword w the number ℓ of identifiers that match w.

6.3.1 $\mathsf{BigHermes}_0$: Amortized BigHermes

If a list of identifiers matching the same keyword contains exactly p identifiers, let us say that the list is *full*. If it contains less than p identifiers, it is *underfull*. In $\mathsf{BigHermes}_0$, for each keyword w, the server stores exactly one underfull list of identifiers matching w. For that purpose, the

server stores W bins $(B_{w_i})_{i=1}^{W}$ of capacity p, one for each keyword. Full lists are stored separately in an instance of Σ_{dum}. When searching for keyword w, the client will retrieve the corresponding bin, and call $\Sigma_{\mathsf{dum}}.\mathsf{Search}$ to fetch full lists matching w (if any).

Naively, to perform an update on keyword w, the client could simply fetch the corresponding bin and add the new identifier, emptying the bin into Σ_{dum} if it is full. However, this would trivially break forward security, because the server would learn information about which keyword is being updated. Instead, the scheme proceeds in epochs (cf. Section 3.6). Each epoch corresponds to W consecutive updates. The client buffers all updates arising during the current epoch, until the buffer contains W updates, and the epoch ends. At the end of the epoch, the client downloads all bins $(B_{w_i})_{i=1}^{W}$ from the server, updates them with the W new identifiers from its buffer, and pushes the updated bins back to the server.

If one of the bins becomes full during this end-of-epoch update, it would be tempting to immediately insert the full list into Σ_{dum}. However, this would again break forward security, as the server would learn how many bins became full during the epoch. To hide that information, $\mathsf{BigHermes}_0$ takes advantage of dummy updates. Observe that during an end-of-epoch update, at most W lists can become full. To hide how many bins become full, $\mathsf{BigHermes}_0$ always performs exactly W updates on Σ_{dum} at the end of the epoch, padding real updates with dummy updates as necessary. From a security standpoint, this approach works because real updates and dummy updates are indistinguishable. From an efficiency standpoint, dummy updates have no impact on the efficiency of Σ_{dum}, as discussed in Section 6.2 (in short, while dummy updates are indistinguishable from real ones for the server, they effectively do nothing). The only cost of dummy updates is in communication complexity, and page efficiency. For both quantities, note that W updates to Σ_{dum} are performed per epoch of W client updates, thus at most one dummy update per client update. Hence, we can instantiate Σ_{dum} for at most $D_{\mathsf{dum}} = N$ dummy updates, and $\mathsf{BigHermes}_0$ directly inherits the same page efficiency and communication overhead as Σ_{dum}.

It remains to discuss the order of (real and dummy) updates on Σ_{dum} at the end of an epoch. This order has an impact on the security of the scheme. To see this, suppose that the following process is used: at the end of an epoch, push full lists to Σ_{dum} for each keyword where this is needed, taking keywords in a fixed order $w_1, w_2, \ldots$; then pad with dummy updates. Now imagine that during an epoch, the client fills a list for keyword w_2, but not for w_1. When the client subsequently performs a search on w_2, the server can see that the locations accessed in Σ_{dum} during the search (partially) match the locations accessed during the first update on Σ_{dum} at the end of the previous epoch (since the first update was for w_2). Since it was the first update, this implies that no update was needed for w_1. The server deduces that no list for w_1 was full at the end of the previous epoch. This breaks security. To avoid that issue, at the end of an epoch, $\mathsf{BigHermes}_0$ first computes all W updates that will need to be issued to Σ_{dum}, then permutes them uniformly at random, before sending the updates to the server. As the security proof will show, this is enough to obtain security.

In the end, $\mathsf{BigHermes}_0$ achieves storage efficiency $\mathcal{O}(1)$, inherited from the same property of Σ_{dum}, and because of the assumption $N \geq pW$, the storage cost of the W bins is $\mathcal{O}(N)$. As noted earlier, $\mathsf{BigHermes}_0$ also inherits the page efficiency and communication overhead of Σ_{dum}. Because the number of entries in the database of Σ_{dum} is at most N/p, it follows that $\mathsf{BigHermes}_0$ has page efficiency and communication overhead $\widetilde{\mathcal{O}}(\log\log(N/p))$. However, this is only in an amortized sense, since batches of W updates are performed together at the end of each epoch.

6.3.2 BigHermes$_1$: BigHermes with Deamortized Communication

The reason $\mathsf{BigHermes}_0$ successfully hides which underfull list requires an update when the client wishes to insert a new keyword-document pair is simple: all underfull lists (bins) are updated at the same time, at the end of an epoch. This approach may be interpreted as hiding the access

pattern to bins using a trivial ORAM: the entire set of bins is downloaded, updated locally, and uploaded back to the server. Dummy updates are then used to hide how many full bins are pushed to Σ_{dum}. This approach is possible due to amortization: a trivial ORAM access only occurs once every W client updates, and updates all W bins simultaneously.

In short, $\mathsf{BigHermes}_1$ deamortizes $\mathsf{BigHermes}_0$ by no longer updating bins all at once at the end of an epoch, and instead updating them one by one over the course of the next epoch. Thus, $\mathsf{BigHermes}_1$ may be understood as a "deamortized" trivial ORAM, which turns out to be much more efficient in our setting than directly using a standard ORAM, as in prior work [MM17]. Among other benefits, this is what allows Hermes to achieve sublogarithmic efficiency, avoiding the logarithmic overhead inherent in ORAM [LN18]. Let us now explain the algorithm. Pseudo-code is available in Algorithm 10. A visual representation of the update procedure is also given in Figure 6.1.

At a high level, at the end of an epoch, the client pre-computes where the W new identifiers from the epoch should be stored on the server, without actually pushing them to the server. To that end, the client maintains a (client-side) table T_{len}, that maps each keyword to the number of matching identifiers currently in the server-side database. Using T_{len}, at the end of an epoch, for each keyword w, the client splits the list of new identifiers matching the keyword into three (possibly empty) sublists: (1) a sublist that completes the content of B_w to a full list (if possible); (2) full sublists of size p; and (3) an underfull sublist of remaining identifiers (if any). Let $\mathsf{CB}_{\mathsf{out}}$ be a (client-side) buffer that maps each keyword to sublists (1) and (3). All sublists of type (2) are stored in another buffer CFP that maps an integer in $[1, W/p]$ to either a full list or $\perp$. (Note that there are at most W/p such sublists in total.) Once all keywords are processed in that manner, the content of CFP is shuffled randomly.

Over the course of the next epoch, the contents of CFP and $\mathsf{CB}_{\mathsf{out}}$ are pushed to the server according to a fixed schedule. In more detail, during the k-th update operation of the next epoch, the client inserts the new keyword-identifier pair into $\mathsf{CB}_{\mathsf{new}}$. This new keyword-identifier pair will not be processed until the end of the current epoch. The client then moves on to pushing updates that were buffered from the end of the previous epoch, proceeding as follows. She downloads the bin B_{w_k} for the k-th keyword from the server. The client then retrieves from $\mathsf{CB}_{\mathsf{out}}[w_k]$ the list L_1 that completes the content of B_{w_k} to a list of size p, and the new underfull list L_x. If there are enough new identifiers in L_1 to complete the content of B_{w_k} to a full list, the new full list is written to Σ_{dum}, and the contents of B_{w_k} is replaced with L_x. Otherwise, the client performs a dummy update to Σ_{dum}, and adds the identifiers of L_1 to B_{w_k}. In either case, B_{w_k} is then re-encrypted and uploaded to the server. Finally, if $k \leq W/p$, the client also retrieves $L_S \leftarrow \mathsf{CFP}[k]$. Recall that L_S is either a full list buffered from the previous epoch, or $\perp$. If $L_S = \perp$ the client performs a dummy update, otherwise she writes L_S to Σ_{dum}. In total, from the point of view of the server, during the k-th client update in a given epoch, the bin B_{w_k} is accessed, and if $k \leq W/p$ (resp. $k > W/p$), two (resp. one) updates are performed in Σ_{dum}. Thus, the access pattern during a client update is fully predictable, and reveals no information to the server. Also note that during each epoch, at most $2W$ dummy updates are performed. Hence, a number of at most $D_{\mathsf{dum}} = 2N$ dummy updates are performed on Σ_{dum}.

6.3.3 $\mathsf{BigHermes}_2$: Fully Deamortized BigHermes

In summary, $\mathsf{BigHermes}_1$ achieves worst-case sublogarithmic page efficiency, communication complexity, as well as server-side time and memory complexities. It also achieves sublogarithmic client-side time complexity, but only amortized over an epoch, since the last update of an epoch triggers an end-of-epoch computation that runs in time $\mathcal{O}(W)$ on the client. Although the computations are simple, this behavior may be undesirable, and one may wish for *worst-case* sublogarithmic time complexity on the client side. Since every other aspect of the scheme is already deamortized, this would result in a fully deamortized scheme. That is what we set out to

do with $\mathsf{BigHermes}_2$.

Here, standard deamortization techniques suffice. The main new techniques underpinning BigHermes (SSE supporting dummy updates, and the idea of "deamortizing" a trivial ORAM) were already present in $\mathsf{BigHermes}_1$. That is why we have focused the presentation on $\mathsf{BigHermes}_1$, in both Algorithm 10 and Figure 6.1. Nevertheless, we now show that client-side computation can also be deamortized.

$\mathsf{BigHermes}_2$ makes use of a pipeline precomputation in two steps. For this, we require separate copies of $\mathsf{CB}_{\mathsf{new}}, \mathsf{CFP}$ and $\mathsf{CB}_{\mathsf{out}}$ for both steps. We denote by $\mathsf{CB}^{(i)}_{\mathsf{new}}, \mathsf{CFP}^{(i)}$ and $\mathsf{CB}^{(i)}_{\mathsf{out}}$ the copies of $\mathsf{CB}_{\mathsf{new}}, \mathsf{CFP}$ and $\mathsf{CB}_{\mathsf{out}}$ respectively for the step i. Additionally, we require a table T_{x} that stores for each keyword w the number of underfull sublists, i.e.sublists of type (1) or (2), during the current epoch; and a counter ctr that counts the total number of (full) sublists of type (2). The tables T_{len} and T_{x} are shared by both steps. Further, we assume that at the beginning of each epoch, a new permutation π is drawn, $\mathsf{CB}^{(1)}_{\mathsf{new}}, \mathsf{CFP}^{(1)}$ and $\mathsf{CB}^{(1)}_{\mathsf{out}}$ is copied to $\mathsf{CB}^{(2)}_{\mathsf{new}}, \mathsf{CFP}^{(2)}$ and $\mathsf{CB}^{(2)}_{\mathsf{out}}$ respectively, and that $\mathsf{CB}^{(1)}_{\mathsf{new}}, \mathsf{CFP}^{(1)}, \mathsf{CB}^{(1)}_{\mathsf{out}}, \mathsf{ctr}$ and T_{x} are reinitialized. The client then performs the following operations, for each pipeline step.

1. During the first step, the data structures are prepared such that in step 2, the client can directly write the content to the server. That is, each update, the new keyword-identifier pair (w, id) is added to the identifier list $\mathsf{CB}_{\mathsf{new}}[w]$, and $T_{\mathsf{len}}[w]$ is incremented. Then, the client checks whether the current identifier list $\mathsf{CB}_{\mathsf{new}}[w]$ is equal to the sublist of type (1), i.e.if $T_{\mathsf{x}}[w] = 0$ and $T_{\mathsf{len}}[w] = 0 \bmod p$. In that case, the sublist is complete and $T_{\mathsf{x}}[w]$ is incremented. Further, she sets $\mathsf{CB}_{\mathsf{out}}[w] \leftarrow \mathsf{CB}_{\mathsf{new}}[w]$ and empties $\mathsf{CB}_{\mathsf{new}}[w]$ thereafter.

   If $T_{\mathsf{x}}[w] \neq 0$, she checks whether $|\mathsf{CB}_{\mathsf{new}}[w]| = p$, i.e.whether the sublist is of type (2). In that case, $T_{\mathsf{x}}[w]$ and ctr is incremented. Then, she sets $\mathsf{CFP}[\pi(\mathsf{ctr})] \leftarrow \mathsf{CB}_{\mathsf{new}}[w]$ and empties $\mathsf{CB}_{\mathsf{new}}[w]$ thereafter.

   Note that at the end of an epoch, all full lists of size p are written to CFP in a random location. Further, for each keyword w, $\mathsf{CB}_{\mathsf{new}}[w]$ contains the underfull sublist of type (3) that will be written to B_w in the next step and $\mathsf{CB}_{\mathsf{out}}[w]$ contains the sublist of type (1) that completes the current list of B_w (if that is possible).

2. During the second step, the content of the data structures is written to the server as before. The only difference is that $\mathsf{CB}_{\mathsf{out}}$ only contains L_1, whereas $\mathsf{CB}_{\mathsf{new}}$ contains L_x (with the notation of Algorithm 10).

6.3.4 Security

BigHermes is forward-secure with standard leakage $\mathcal{L}^{\mathsf{fs}}$ defined below. In words, Setup leaks an upper bound on the size of the database and on the number of keywords; Search reveals the query pattern, and the number of identifiers matching the searched keyword; and updates leak nothing.

Theorem 6.6. *Let N be an upper bound on the size of the database, let p be the page size, and let W be an upper bound on the number of keywords. Let $N \geq pW$, and assume $N/p \geq \lambda$. Let Σ_{dum} be a forward-secure* SSE *supporting dummy updates, let* Enc *be an IND-CPA secure encryption scheme, and let* PRF *be a secure pseudorandom function (for the preprocessing of keywords).*

Let $\mathcal{L}^{\mathsf{fs}} = (\mathcal{L}^{\mathsf{fs}}_{\mathsf{Stp}}, \mathcal{L}^{\mathsf{fs}}_{\mathsf{Srch}}, \mathcal{L}^{\mathsf{fs}}_{\mathsf{Updt}})$, with $\mathcal{L}^{\mathsf{fs}}_{\mathsf{Stp}}(\mathsf{DB}, W, N) = (W, N)$, $\mathcal{L}^{\mathsf{fs}}_{\mathsf{Srch}}(w_i) = (\mathsf{qp}, \ell_i)$ where ℓ_i is the number of identifiers matching w_i, and $\mathcal{L}^{\mathsf{fs}}_{\mathsf{Updt}}(\mathsf{op}, w, \mathsf{id}) = \perp$. Then BigHermes *is correct and $\mathcal{L}^{\mathsf{fs}}$-adaptively semantically secure.*

We first sketch the security proof. BigHermes stores all full identifier lists of size p in Σ_{dum}, and one underfull sublist per keyword w in the bin B_w. Each search, the corresponding bin is

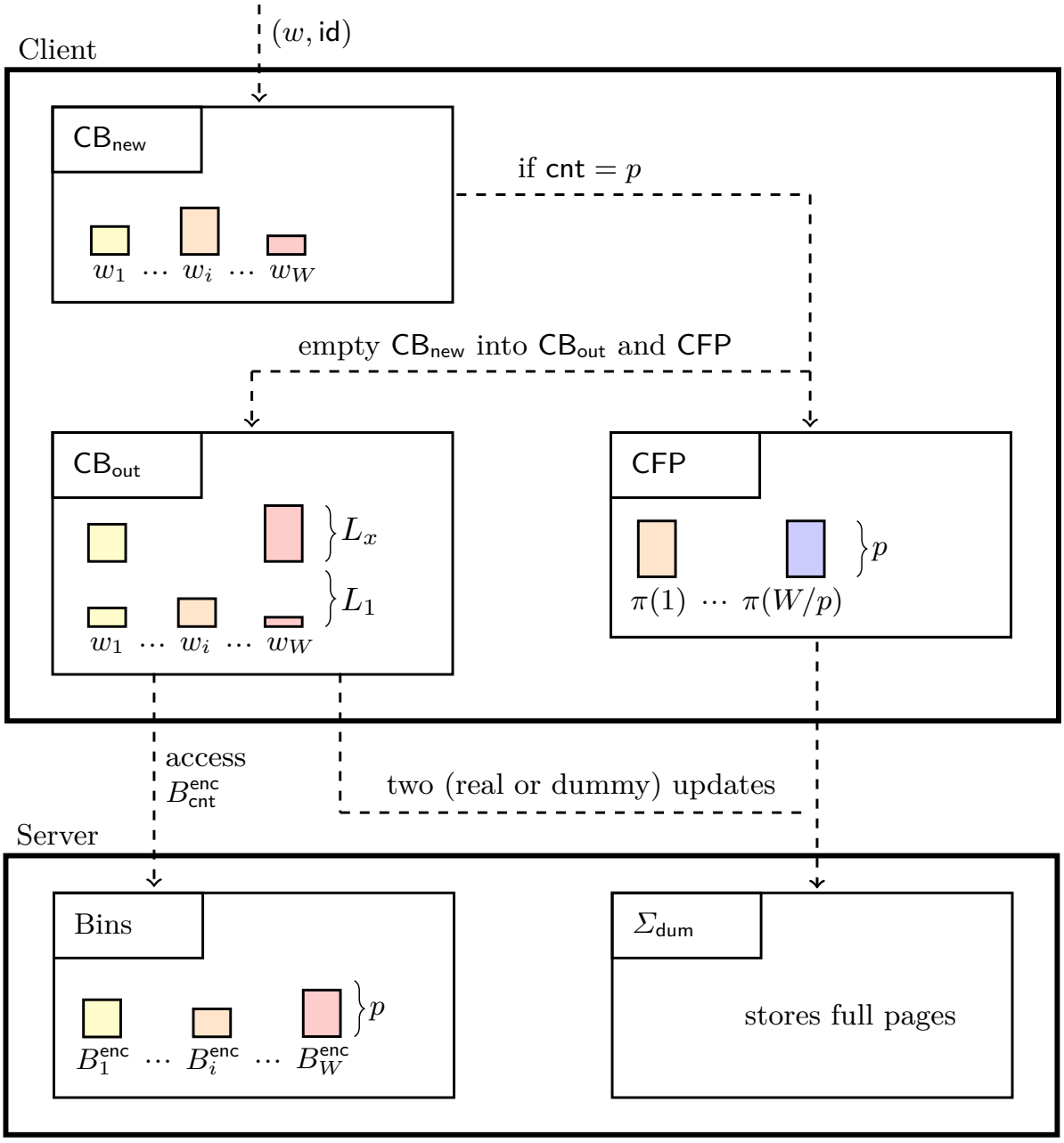

Figure 6.1: Sketch of the update schedule of $\mathsf{BigHermes}_1$. (The data structure T_{len} are omitted for clarity.) Each update, the variable cnt is incremented and the dotted lines are executed. See Algorithm 10 for pseudo-code.

Algorithm 10 $\mathsf{BigHermes}_1$

$\mathsf{BigHermes}_1.\mathsf{Setup}(\mathsf{K}, N, W, \mathsf{DB})$

Initialize bins $B_{w_1}, \ldots, B_{w_W}$ of capacity p
Initialize empty database $\mathsf{DB}_{\mathsf{dum}}$
for all keywords w **do**
    Split $\mathsf{DB}(w)$ into x lists L_i such that L_x has size at most $p-1$, and the remaining lists have size p
    Insert pairs $\{(w, L_i)\}_{i=1}^{x-1}$ into $\mathsf{DB}_{\mathsf{dum}}$
    Insert list L_x into B_w
    $T_{\mathsf{len}}[w] \leftarrow |\mathsf{DB}(w)|$
$\mathsf{cnt} \leftarrow 0$
$(N_{\mathsf{dum}}, D_{\mathsf{dum}}, W_{\mathsf{dum}}) = (N/p, 2N, W)$
$\mathsf{EDB}_{\Sigma_{\mathsf{dum}}} \leftarrow \Sigma_{\mathsf{dum}}.\mathsf{Setup}(\mathsf{K}_{\Sigma_{\mathsf{dum}}}, N_{\mathsf{dum}}, D_{\mathsf{dum}}, W_{\mathsf{dum}}, \mathsf{DB}_{\mathsf{dum}})$
$\mathsf{EDB} \leftarrow (\mathsf{EDB}_{\Sigma_{\mathsf{dum}}}, \{\mathsf{Enc}_{\mathsf{K}_{\mathsf{Enc}}}(B_{w_i})\}_{i=1}^{W})$
return EDB

$\mathsf{BigHermes}_1.\mathsf{KeyGen}(1^\lambda)$

Set $\mathsf{K}_{\Sigma_{\mathsf{dum}}} \leftarrow \Sigma_{\mathsf{dum}}.\mathsf{KeyGen}(1^\lambda)$
Sample key $\mathsf{K}_{\mathsf{Enc}}$ for Enc
return $\mathsf{K} = (\mathsf{K}_{\mathsf{Enc}}, \mathsf{K}_{\Sigma_{\mathsf{dum}}})$

$\mathsf{BigHermes}_1.\mathsf{Search}(\mathsf{K}, w; \mathsf{EDB})$

Client:

Perform $\Sigma_{\mathsf{dum}}.\mathsf{Search}(\mathsf{K}_{\Sigma_{\mathsf{dum}}}, w; \mathsf{EDB})$
send w

Server:

send B_w^{enc}

$\mathsf{BigHermes}_1.\mathsf{Update}(\mathsf{K}, (w, \mathsf{id}), \mathsf{add}; \mathsf{EDB})$

Client:

if $\mathsf{cnt} = p$ **then**
    $\pi \leftarrow$ uniformly random permuation of $[1, W/p]$
    Initialize empty set S of full pages
    for all keywords w **do**
        $L \leftarrow \mathsf{CB}_{\mathsf{new}}[w]$
        $r \leftarrow T_{\mathsf{len}}[w] \bmod p$
        $T_{\mathsf{len}}[w] = T_{\mathsf{len}}[w] + |L|$
        Split L into x lists L_i such that L_1 has size at most $p - r$ (exactly size $p - r$ if $x > 1$), L_x has size at most p, and the other lists all have size p
        $\mathsf{CB}_{\mathsf{out}}[w] \leftarrow \mathsf{CB}_{\mathsf{out}}[w] \cup \{(L_1, L_x)\}$
        $S \leftarrow S \cup \{(w, L_2), \ldots, (w, L_{x-1})\}$
    $\mathsf{CFP}[\pi(i)] \leftarrow S[i]$ for $1 \le i \le |S|$
    Empty $\mathsf{CB}_{\mathsf{new}}$ and set $\mathsf{cnt} = 0$
$\mathsf{CB}_{\mathsf{new}}[w] \leftarrow \mathsf{CB}_{\mathsf{new}}[w] \cup \{\mathsf{id}\}$
$\mathsf{cnt} \leftarrow \mathsf{cnt} + 1$
send cnt

Server:

send $B_{\mathsf{cnt}}^{\mathsf{enc}}$

(continue description of update)

Client:

Set $B_{\mathsf{cnt}} = \mathsf{Dec}_{\mathsf{K}_{\mathsf{Enc}}}(B_{\mathsf{cnt}}^{\mathsf{enc}})$
Retrieve list L of identifiers from B_{cnt}
$(L_1, L_x) \leftarrow \mathsf{CB}_{\mathsf{out}}[w_{\mathsf{cnt}}]$
$\mathsf{CB}_{\mathsf{out}}[w_{\mathsf{cnt}}] \leftarrow \perp$
if $L_1 \neq L_x$ **then** ▷ $x > 1$
    Run $\Sigma_{\mathsf{dum}}.\mathsf{Update}(\mathsf{K}, (w_{\mathsf{cnt}}, L_1 \cup L), \mathsf{add}; \mathsf{EDB})$
    $B_{\mathsf{cnt}} \leftarrow L_x$
else ▷ $x = 1$
    Run $\Sigma_{\mathsf{dum}}.\mathsf{DummyUpdate}(\mathsf{K}; \mathsf{EDB})$
    $B_{\mathsf{cnt}} \leftarrow L \cup L_1$
if $\mathsf{cnt} \le W/p$ **then**
    if $\mathsf{CFP}[\mathsf{cnt}] \neq \perp$ **then**
        $(w, L_S) \leftarrow \mathsf{CFP}[\mathsf{cnt}]$
        Run $\Sigma_{\mathsf{dum}}.\mathsf{Update}(\mathsf{K}, (w, L_S), \mathsf{add}; \mathsf{EDB})$
        $\mathsf{CFP}[\mathsf{cnt}] \leftarrow \perp$
    else
        Run $\Sigma_{\mathsf{dum}}.\mathsf{DummyUpdate}(\mathsf{K}; \mathsf{EDB})$
send $B_{\mathsf{cnt}}^{\mathsf{enc}} \leftarrow \mathsf{Enc}_{\mathsf{K}_{\mathsf{Enc}}}(B_{\mathsf{cnt}})$

Server:

Update $B_{\mathsf{cnt}}^{\mathsf{enc}}$

accessed, and the client searches for all full lists on the server in Σ_{dum}. All identifiers that are not retrieved are contained in a buffer on the client. Thus, correctness follows immediately from the correctness of Σ_{dum}.

The setup only leaks W and N to the server, as the bins are encrypted, and the security of Σ_{dum} guarantees that $\mathsf{EDB}_{\Sigma_{\mathsf{dum}}}$ leaks no other information. Further, updates leak no information which follows from two facts: (1) exactly one bin is accessed each update via a fixed schedule known in advance; (2) dummy updates and real updates leak no information, and are indistinguishable in the view of the server, owing to the security of Σ_{dum}.

It remains to consider searches. Each search on keyword w only leaks the query pattern qp, and the length ℓ of the identifier list for w. (Recall from Section 3.6 that the query pattern qp is equal to the search pattern sp and update pattern up.) To establish this, we need to show that the view of the server can be simulated using only qp and ℓ. Based on the search pattern and the fact that bins are encrypted with IND-CPA encryption, simulating access to the bin B_w is straightforward. Thus, security reduces to the simulation of Σ_{dum}. Because Σ_{dum} is secure, we know that its behavior can be simulated as long as we can compute the query pattern $\mathsf{qp}_{\Sigma_{\mathsf{dum}}}$ and answer length $\ell_{\Sigma_{\mathsf{dum}}}$ for Σ_{dum}. To see that this is the case, first observe that the number of Σ_{dum}-updates on w and the load of B_w per epoch can be recomputed given only up and ℓ. From there, the simulator can compute the number ℓ_i of full lists for keyword w that were pushed to Σ_{dum} during a given epoch i. Clearly, $\sum_i \ell_i = \ell_{\Sigma_{\mathsf{dum}}}$. To deduce the update pattern $\mathsf{up}_{\Sigma_{\mathsf{dum}}}$ for Σ_{dum}, it remains to determine when each update was performed during the epoch. Intuitively, because updates to Σ_{dum} occurring in a given epoch are permuted uniformly at random, it suffices to choose x updates to Σ_{dum} uniformly at random among updates issued during the epoch (excluding updates already chosen for the same purpose on a different keyword). This yields $\mathsf{up}_{\Sigma_{\mathsf{dum}}}$. On the other hand, each search query to $\mathsf{BigHermes}$ triggers exactly one search query to Σ_{dum}: the search pattern of Σ_{dum} matches the search pattern of $\mathsf{BigHermes}$. Thus, the simulator can compute $\mathsf{qp}_{\Sigma_{\mathsf{dum}}}$, and we are done.

We are now ready to prove Theorem 6.6. A formal proof of correctness is omitted as it is straight-forward.

Proof. We show that $\mathsf{BigHermes}$ is secure with leakage $\mathcal{L}^{\mathsf{fs}}$. Let Sim denote the simulator and $\mathcal{A}$ an arbitrary honest-but-curious PPT adversary. Further, let Sim_Σ be a simulator of Σ.

Initially, Sim receives $\mathcal{L}^{\mathsf{fs}}_{\mathsf{Stp}}(\mathsf{DB}, W, N) = (W, N)$ and later, an adaptive series of search and update queries with input $\mathcal{L}^{\mathsf{fs}}_{\mathsf{Srch}}(w_i) = (\mathsf{qp}, \ell_i)$ and $\mathcal{L}^{\mathsf{fs}}_{\mathsf{Updt}}(op_i, w_i, \mathsf{id}_i) = \perp$ respectively. First, Sim generates an encryption key $\mathsf{K}'_{\mathsf{Enc}}$ and initializes W bins $B_1, \ldots, B_W$ zeroed out up to size p. Then, Sim sets $\mathsf{EDB}'_\Sigma \leftarrow \mathsf{Sim}_\Sigma(W, N)$ and outputs $\mathsf{EDB}' = (\mathsf{Enc}_{\mathsf{K}'_{\mathsf{Enc}}}(B_1), \ldots, \mathsf{Enc}_{\mathsf{K}'_{\mathsf{Enc}}}(B_W), \mathsf{EDB}'_\Sigma)$. Further, Sim initializes a counter $\mathsf{cnt} = 0$ of the number of updates. Next, Sim simulates the search and update queries.

For search queries, Sim receives query pattern $\mathsf{qp} = (\mathsf{sp}, \mathsf{up})$ and the total number ℓ of identifiers matching the searched keyword. If the search pattern sp indicates that the keyword was already searched, Sim outputs the keyword w' from the previous query. Otherwise, Sim outputs a new uniformly random keyword w'. Also, Sim associates an index $\mu \in [1, W]$ to w' at random but distinct from indices of other keywords. Next, Sim simulates a search query of Σ. Recall that up is a bit vector that indicates for each update query whether it was an update on the searched keyword or not. As the updates on $\mathsf{SmallHermes}$ induce an altered update pattern on Σ, the simulator Sim needs to reconstruct the update pattern up_Σ of Σ from up. For this, she proceeds as follows for the k-th epoch:

She initializes an all-zero vector $\mathsf{up}_\Sigma = (0, \ldots, 0)$ of length $2\mathsf{cnt}$. If the coordinates of up_Σ were already computed for the k-th epoch during a previous search query for the current keyword, she sets up_Σ as before at these positions. If otherwise no search query was issued on the keyword after the k-th epoch, the corresponding coordinates of up_Σ were not yet set. Recall that during the last epoch, the updates on Σ depend on the updates during the previous epoch, and during

the first epoch, only dummy updates are performed. She will set the coordinates as follows for $k > 1$.

Sim recomputes the number of added sublists x in the $(k-1)$-th epoch for the searched keyword. (Note based on up and ℓ, Sim can recompute the number of identifiers in the bin of the searched keyword at the beginning of the $(k-1)$-th epoch and the number of identifiers added during the $(k-1)$-th epoch. These values determine x. Then, $x-1$ sublists are then written to Σ during the k-th epoch due to preprocessing.) Then, if $x \geq 2$, she sets $\mathsf{up}_\Sigma[2(k+\mu)-1] \leftarrow 1$, where μ is the index of w'. Further, for all $i \in [1, x-2]$, she chooses some $j \in [1, W/p]$ that has not been chosen during the $(k-1)$-th epoch at random and sets $\mathsf{up}_\Sigma[2(k+j)] \leftarrow 1$. (If $2(k+\mu)-1$ or $2(k+j)$ are larger than $2\mathsf{cnt}$, then the client remembers the choice until the update happened and sets up accordingly for subsequent searches.) Note that for the first epoch, up_Σ remains zeroed out. Similarly, Sim computes the number n_{client} of identifiers matching the searched keyword that are still buffered on the client via ℓ and up, and sets sets $\ell_\Sigma \leftarrow \lfloor (\ell - n_{\mathsf{client}})/p \rfloor$. Finally, Sim invokes Sim_Σ with input qp_Σ and ℓ_Σ to simulate the search protocol of Σ.

In order to simulate an updates, Sim simply invokes Sim_Σ twice (with input $\perp$) to simulate the update protocol of Σ and increments cnt.

We now show that the real game is indistinguishable from the ideal game. For this, we define four hybrid games.

- Hybrid 0 is identical to the real game.
- Hybrid 1 is the same as Hybrid 0, except the simulated keywords w' are output during search. As we assume that w is the output of a PRF and thus indistinguishable from random, Hybrid 0 and Hybrid 1 are (computationally) indistinguishable.
- Hybrid 2 is the same as Hybrid 1, except the updates on Σ are induced by up_Σ. That is, updates for a given keyword w' on Σ are performed each u-th update operation, where $\mathsf{up}_\Sigma[u] = 1$ (but with the real identifiers). Note that by construction, for each such update on Σ, there is a full list of identifiers that was inserted during the previous epoch. Hybrid 2 is identically distributed to Hybrid 1, as the π is a random permutation.
- Hybrid 2 is the same as Hybird 1, except Σ is simulated. That is, EDB_Σ is replaced with the simulated EDB'_Σ, the search queries are simulated with input ℓ_Σ and up_Σ, and the updates on Σ are simulated with input $\perp$. Hybrid 2 and Hybrid 1 are indistinguishable based on the security of Σ.
- Hybrid 3 is the same as Hybrid 2, except the bins are zeroed out. That is, EDB is replaced fully with EDB' and received bins are only reencrypted during updates. Since Enc is $\mathsf{IND\text{-}CPA}$ secure, Hybrid 3 and Hybrid 2 are indistinguishable.
- Hybrid 4 is identical to the ideal game. Hybrid 4 and Hybrid 3 are identically distributed and thus, indistinguishable. □

6.3.5 Efficiency

We now analyze the efficiency of $\mathsf{BigHermes}$, when Σ_{dum} is efficient (in the sense of Section 6.2.3). Each client-side data structure, including identifier buffers and tables, has size $\mathcal{O}(W)$. Since there is a constant number of such structures, overall client storage is $\mathcal{O}(W)$. On the server side, Σ_{dum} has storage efficiency $\mathcal{O}(1)$, and the bins require $pW = \mathcal{O}(N)$ storage, hence overall storage efficiency is $\mathcal{O}(1)$. We turn to page efficiency. During each update, one bin of size p is read, and two updates on Σ_{dum} are performed. Since Σ_{dum} only stores full pages, of which there can be at most N/p, $\Sigma_{\mathsf{dum}}.\mathsf{Update}$ has page efficiency $\widetilde{\mathcal{O}}(\log\log(N/p))$. Similarly, a search on a keyword w with ℓ matching identifiers induces one bin access and (at most) $\lfloor \ell/p \rfloor$ accesses of $\widetilde{\mathcal{O}}(\log\log(N/p))$ pages each. We conclude that $\mathsf{BigHermes}$ has page efficiency $\widetilde{\mathcal{O}}(\log\log(N/p))$.

6.4 SmallHermes: the Small Database Regime

Recall that p is the page size, N is an upper bound on the total number of identifiers and W is an upper bound on the total number of distinct keywords. In this section, we assume $N \leq pW$. We detail our scheme SmallHermes. Our construction builds on the page-efficient dynamic SSE scheme LayeredSSE from Section 5.4. While LayeredSSE is not forward secure, we show that with the techniques developed in this chapter, we can construct an oblivious update algorithm with $\mathcal{O}(W)$ client memory. Note that our technique can only be applied if $N \leq pW$ in an efficient manner. Our construction preserves the efficiency properties of LayeredSSE, namely $\widetilde{\mathcal{O}}\left(\log\log\frac{N}{p}\right)$ page efficiency and constant storage efficiency, and uses only $\mathcal{O}(W)$ client storage.

We present the results as follows. First, we recall how LayeredSSE works and outline a simple amortized update procedure that it forward secure. This intermediate construction $\mathsf{SmallHermes}_0$ is presented in Section 6.4.1. Next, we present $\mathsf{SmallHermes}_1$ version, which has deamortized communication. This version is also given in pseudo-code in Algorithm 11 and the used data structures are presented in Table 6.3. Then, we show how to deamortize both communication and computation via the construction $\mathsf{SmallHermes}_2$. Finally, we analyze the security and efficiency.

Table 6.3: Overview of the data structures used in SmallHermes (see Algorithm 11).

Data Structure	Comments
Bins $B_1, \ldots, B_m$	Each bin stores up to $p \cdot b_{N,p}$ identifiers as in LayeredSSE with $b_{N,p} = \widetilde{\mathcal{O}}\left(\log\log(N/p)\right)$
$T_{\mathsf{len}} : w \mapsto \ell$	Table that stores for each keyword w the number ℓ of matching identifiers
$\mathsf{CB}_{\mathsf{new}} : w \mapsto L$	Table that buffers for each keyword w fresh identifiers L matching w (up to W identifiers in total)
$\mathsf{CB}_{\mathsf{out}} : \gamma \mapsto L$	Table that buffers for the γ-th bin the identifiers (potentially matching different keywords) to be added to B_γ
$T_{\mathsf{load}} : (\gamma, \kappa) \mapsto n_{\gamma,\kappa}$	Stores for bin γ the number $n_{\gamma,\kappa}$ of sublists in layer κ

6.4.1 $\mathsf{SmallHermes}_0$: Amortized SmallHermes

Recall that LayeredSSE stores identifiers in $m = o(N/p)$ encrypted bins according to L2C. That is, each list L of (at most p) identifiers matching keyword w is mapped to two bins B_α, B_β with $\log\log\frac{N}{p}$ conceptual layers and capacity $\widetilde{\mathcal{O}}\left(p\log\log\frac{N}{p}\right)$. Then, L is stored in either B_α or B_β depending on the load of each bin at layer κ, where κ depends on the size of L. For a search on keyword w, the bins B_α and B_β are requested from the server. The client can read the matching identifiers from the bins after decryption. These bins are also retrieved for each update on keyword w, and the new identifier is inserted in one of the bins according to L2C. When a list has more than p identifiers, it is split into sublists of size p which are treated independently as described above. This results in $\widetilde{\mathcal{O}}\left(\log\log\frac{N}{p}\right)$ page efficiency and $\mathcal{O}(1)$ storage efficiency. See Section 5.4 for more details.

The setup and search of $\mathsf{SmallHermes}_0$ are identical to setup and search of LayeredSSE. As updates of LayeredSSE are not forward secure, we now show how to adapt the update procedure such that it has *no* leakage. With $\mathcal{O}(W)$ client storage, the client can buffer $\mathcal{O}(W)$ fresh keyword-identifier pairs in a buffer $\mathsf{CB}_{\mathsf{new}}$. When $\mathsf{CB}_{\mathsf{new}}$ is full, she can download the entire encrypted database EDB from the server and perform the updates locally with amortized $\mathcal{O}(1)$ page efficiency, as $N \leq pW$. For this, she has to perform up to W insertions according to L2C operations. In the following, we show how to deamortize this simple approach. As the final variant $\mathsf{SmallHermes}_2$ is technically involved, we first present an intermediate variant $\mathsf{SmallHermes}_1$.

6.4.2 SmallHermes$_1$: SmallHermes with Deamortized Communication

SmallHermes$_1$ deamortizes the communication at the cost of $\mathcal{O}(W \log\log N)$ client storage. Note that the overhead can be avoided heuristically via the use weighted 2C or via more pre-computation (details follow later). Since each bin is of size $\widetilde{\mathcal{O}}\left(p \log\log \frac{N}{p}\right)$, we aim for $\widetilde{\mathcal{O}}\left(\log\log \frac{N}{p}\right)$ page efficiency and constant storage efficiency. We proceed as follows, assuming that per keyword, there are at most p matching identifiers.

The client accesses each bin in a fixed schedule and inserts new identifiers into the required bin, based on locally computed load information. The client stores an additional table T_{load} which maps a bin B_γ and layer κ to the number $n_{\gamma,\kappa} \leftarrow T_{\mathsf{load}}(\gamma, \kappa)$ of lists stored in B_γ at layer κ. [2] Whenever the client decides to store an identifier list in a bin, she also updates T_{load} accordingly. With T_{load}, the client can now directly decide locally where to insert each new identifier. Note that the client cannot download the corresponding bin directly to insert the identifier, as this would break forward security. But we can still leverage the load information of T_{load} with an additional identifier buffer $\mathsf{CB}_{\mathsf{out}}$.

Each update, the client inserts the new keyword-identifier pair into $\mathsf{CB}_{\mathsf{new}}$. After W updates $\mathsf{CB}_{\mathsf{new}}$ is filled and the client pre-computes the location of the new identifiers. That is, for all keywords w, the client pre-computes the index γ of bin B_γ in which to insert the list L_{new} of new identifiers matching w (that are buffered in $\mathsf{CB}_{\mathsf{new}}$). The list L_{new} is then moved into $\mathsf{CB}_{\mathsf{out}}[\gamma]$. Note that the index γ can be computed via T_{load}, if T_{load} is continuously updated throughout the pre-computation.

After this pre-computation, for each bin B_γ, the buffer $\mathsf{CB}_{\mathsf{out}}[\gamma]$ contains all identifiers from $\mathsf{CB}_{\mathsf{new}}$ to be inserted into bin B_γ. $\mathsf{CB}_{\mathsf{new}}$ can be emptied subsequently. During the next epoch, the client can download each bin B_γ via a fixed schedule and insert $\mathsf{CB}_{\mathsf{out}}[i]$ into B_i. For this, SmallHermes$_1$ downloads bin B_i the i-th update operation of the epoch.

Note that a fresh identifier is written to the server after at most $2W$ update operations and inserted into the bin B_γ chosen according to L2C. Consequently, no bin overflows with overwhelming probability due the correctness of L2C. Also, during each update operation, at most one bin of size $p\mathcal{O}(\log\log \frac{N}{p})$ is downloaded. As in LayeredSSE, exactly two bins are accessed during a Search. Thus, SmallHermes$_1$ has $\mathcal{O}(\log\log \frac{N}{p})$ page efficiency and $\mathcal{O}(1)$ storage efficiency.

Handling arbitrary list lengths. If there are more than p identifiers per keyword, we split the lists L of identifiers matching keyword w into full lists and (at most) one underfull list. Sublists with exactly p identifiers are called *full*, whereas sublists with less than p identifiers are referred to as *underfull*. In order to compute the correct bin for the new identifiers, the client also stores the length of L for each keyword in a table T_{len}. During an update, the client computes the size $r = T_{\mathsf{len}}[w] \bmod p$ of the underfull list on the server. L_{new} (defined as above) is split into $x = \lceil \frac{r+|L_{\mathsf{new}}|}{p} \rceil$ sublists as follows:

- L_1, the sublist that fills the underfull list on the server (if possible).
- $L_2, \ldots, L_{x-1}$, the full sublists of size p.
- L_x, the remaining underfull sublist (if any).

For each sublist L_i, we again compute the bin B_γ in which we need to insert L_i via T_{load} (according to L2C) and insert L_i into $\mathsf{CB}_{\mathsf{out}}[\gamma]$. Note that during this process, we interpret L_1 as a list of size $|L_1| + r$, as it will complete the underfull sublist on the server, if possible. Also, T_{len} and T_{load} are updated accordingly throughout the pre-computation.

[2] The table T_{load} is the reason $\mathcal{O}(W \log\log N)$ client storage is required. Using some additional pre-computation, the client can compute T_{load} at only $\mathcal{O}(W)$ required positions. This retains $\mathcal{O}(W)$ client storage (see section 6.4.3).

Algorithm 11 $\mathsf{SmallHermes}_1$

$\mathsf{SmallHermes}_1.\mathsf{Setup}(\mathsf{K}, N, \mathsf{DB})$

1: Set $\mathsf{in} \leftarrow \{(w_i, \mathsf{DB}(w_i))\}_i$
2: Set $(B_1, ..., B_m) \leftarrow \mathsf{L2C.Setup}(\mathsf{in}, N/p)$
3: Set $B_i^{\mathsf{enc}} \leftarrow \mathsf{Enc}_{\mathsf{K}_{\mathsf{Enc}}}(B_i)$ for $i \in [1, m]$
4: $T_{\mathsf{len}}[w] \leftarrow \ell_i$ for all keywords w
5: Setup T_{load} according to load of $(B_1, ..., B_m)$
6: Set $\mathsf{cnt} \leftarrow 0$
7: **return** $\mathsf{EDB} = (B_1^{\mathsf{enc}}, ..., B_m^{\mathsf{enc}})$

$\mathsf{SmallHermes}_1.\mathsf{KeyGen}(1^\lambda)$

1: Sample $\mathsf{K}_{\mathsf{Enc}}$ for Enc
2: **return** $K = \mathsf{K}_{\mathsf{Enc}}$

$\mathsf{SmallHermes}_1.\mathsf{Search}(\mathsf{K}, w; \mathsf{EDB})$

Client:

1: Set $\ell \leftarrow T_{\mathsf{len}}(w)$ and $x = \lceil \ell/p \rceil$
2: **send** w, x

Server:

1: Set $\alpha_i, \beta_i \leftarrow \mathsf{H}(w \mid\mid i)$ for $i \in [1, x]$
2: **send** $\{B_{\alpha_i}^{\mathsf{enc}}, B_{\beta_i}^{\mathsf{enc}}\}_{i=1}^{x}$

$\mathsf{SmallHermes}_1.\mathsf{Update}(\mathsf{K}, w, \mathsf{id}, \mathsf{add}; \mathsf{EDB})$

Client:

1: **if** $\mathsf{cnt} = p$ **then**
2: **for all** keywords w **do**
3: Set $L \leftarrow \mathsf{CB}_{\mathsf{new}}[w]$
4: Set $r \leftarrow T_{\mathsf{len}}[w] \bmod p$
5: Set $x \leftarrow \lceil (r + |L|)/p \rceil$
6: Split L into x lists L_i such that L_1 has size at most $p - r$, L_x has size at most p, and the other lists $L_2, ..., L_{x-1}$ have size p
7: Precompte bin index γ_i of L_i via load information in T_{load} for all $i \in [1, x]$
8: Set $\mathsf{CB}_{\mathsf{out}}[\gamma_i] \leftarrow \mathsf{CB}_{\mathsf{out}}[\gamma_i] \cup \{L_i\}$ for all $i \in [1, x]$
9: Update load information in T_{load} accordingly
10: Set $T_{\mathsf{len}} \leftarrow T_{\mathsf{len}}[w] + |L|$
11: Empty $\mathsf{CB}_{\mathsf{new}}$ and set $\mathsf{cnt} = 0$
12: $\mathsf{CB}_{\mathsf{new}}[w] \leftarrow \mathsf{CB}_{\mathsf{new}}[w] \cup \{\mathsf{id}\}$
13: $\mathsf{cnt} \leftarrow \mathsf{cnt} + 1$
14: **if** $\mathsf{cnt} \leq m$ **then**
15: **send** cnt

Server:

1: **send** $B_{\mathsf{cnt}}^{\mathsf{enc}}$

Client:

1: Set $B_{\mathsf{cnt}} \leftarrow \mathsf{Dec}_{\mathsf{K}_{\mathsf{Enc}}}(B_{\mathsf{cnt}}^{\mathsf{enc}})$
2: Insert $\mathsf{CB}_{\mathsf{out}}[\gamma_i]$ into B_{cnt}
3: **send** $B_{\mathsf{cnt}}^{\mathsf{enc}} \leftarrow \mathsf{Enc}_{\mathsf{K}_{\mathsf{Enc}}}(B_{\mathsf{cnt}})$

Server:

1: Update $B_{\mathsf{cnt}}^{\mathsf{enc}}$

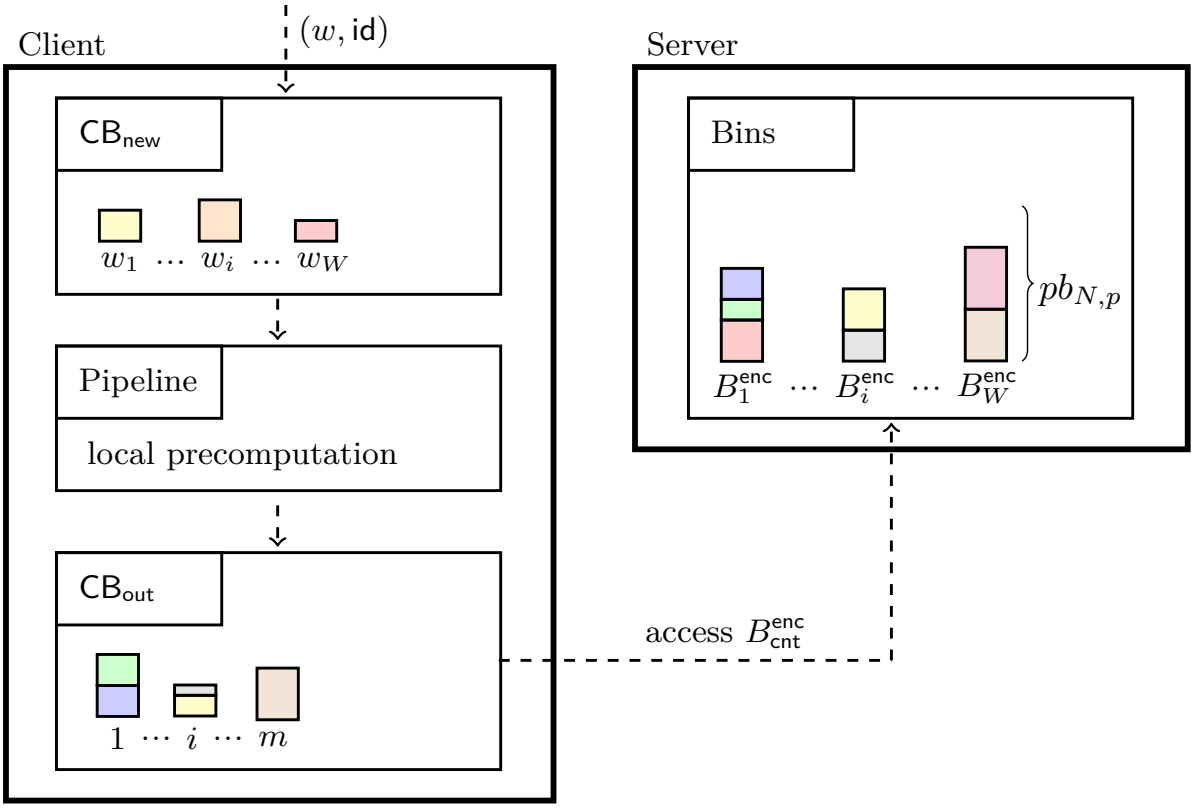

Figure 6.2: Sketch of the update schedule of $\mathsf{SmallHermes}_1$. (The data structures T_{len} and T_{load} are omitted for clarity.) Each update, the variable cnt is incremented and added keyword-identifier pairs are moved along the dotted lines. See Algorithm 11 for pseudo-code. Note that the $\mathsf{SmallHermes}_2$ shares the same structure, though the pipeline that pre-computes the final bin location of added identifiers is much more sophisticated.

6.4.3 $\mathsf{SmallHermes}_2$: Fully Deamortized SmallHermes

Here, we remove the requirement of $\mathcal{O}(W \log\log \frac{N}{p})$ client storage and deamortize both the communication and computation of $\mathsf{SmallHermes}$. This optimization is possible, because we observe that for inserting the $\mathcal{O}(W)$ new identifiers of $\mathsf{CB}_{\mathsf{new}}$, we do not require the entire load information of $\mathsf{L2C}$. Only the load of bin layers in which a new identifier is inserted is required. Per epoch, each new identifier requires the load of some bin B_γ at some layer κ. There at most $2W$ such pairs (γ, κ). The load information of these pairs can be precomputed in a pipeline with three steps, where each step is performed during $\mathcal{O}(W)$ updates. In an additional fourth step, the client can finally fill $\mathsf{CB}_{\mathsf{out}}$ as before using the load information. During the next W updates, the content of the filled buffer $\mathsf{CB}_{\mathsf{out}}$ is pushed to the server as before. This approach naturally yields an update operation with at most $\mathcal{O}(p \log\log \frac{N}{p})$ communication *and* computation.

Before we present each pipeline step, we fix some convention as all steps rely on the results of previous steps. We assume that there are independent copies of client data structures $\mathsf{CB}_{\mathsf{new}}, \mathsf{CB}_{\mathsf{out}}, T_{\mathsf{load}}$ and T_{hash}, per pipeline step. We index a data structure ds from step i via $\mathsf{ds}^{(i)}$. Note that T_{len} is a table shared by all steps. The pipeline steps are executed in reverse order, i.e.step $i+1$ is executed before step i. We implicitly assume that the content of $\mathsf{ds}^{(i)}$ is copied to $\mathsf{ds}^{(i+1)}$ before execution of pipeline step i. The contents of all data structures of step i are emptied after the step is performed and the content was copied to step $i+1$. The 5 pipeline steps are as follows:

1. In the first step, the client simply buffers the identifiers id in $\mathsf{CB}_{\mathsf{new}}^{(1)}$ as before, i.e.for each

new keyword-identifier pair (w, id), the client adds id to the list $\mathsf{CB}^{(1)}_{\mathsf{new}}[w]$.

2. In the second step, the client splits for each keyword w the list of $L \leftarrow \mathsf{CB}^{(2)}_{\mathsf{new}}[w]$ of new identifiers matching w into sublists L_i (as explained above) and computes for each of these sublists the two bins it could be stored in. The result (along with auxiliary information) is stored in T_{hash}. Further, all required layers are marked in T_{load} with ∞ (and filled with the correct value in the following step).

   In more detail, during the k-th update operation, the client takes some keyword w for which $\mathsf{CB}^{(2)}_{\mathsf{new}}$ is not empty. She sets $r \leftarrow T_{\mathsf{len}}[w] \bmod p$ if w was considered for the first time this step and set $r \leftarrow 0$ otherwise [3]. Also, she sets $d \leftarrow \lfloor T_{\mathsf{len}}[w]/p \rfloor$. The client removes up to $p - r$ identifiers $L_{\mathsf{new}} = (\mathsf{id}_1, \ldots, \mathsf{id}_{\ell_{\mathsf{new}}})$ from $\mathsf{CB}^{(2)}_{\mathsf{new}}[w]$. Note that L_{new} corresponds to the i-th sublist L_i, if w was considered for the i-th time during this step. She sets $\kappa = \mathsf{layer}(\ell_{\mathsf{new}} + r)$. Then, she computes $\alpha, \beta \leftarrow \mathsf{H}(w \mid\mid (1 + d))$ and stores $T^{(2)}_{\mathsf{hash}}[\beta] \leftarrow T^{(2)}_{\mathsf{hash}}[\beta] \mid\mid (L_{\mathsf{new}}, \kappa, \alpha)$ and $T^{(2)}_{\mathsf{hash}}[\alpha] \leftarrow T^{(2)}_{\mathsf{hash}}[\alpha] \mid\mid (L_{\mathsf{new}}, \kappa, \beta)$. Also, she sets $T^{(2)}_{\mathsf{load}}[\alpha, \kappa] \leftarrow 0$. Finally, she updates $T_{\mathsf{len}}[w] \leftarrow \ell_{\mathsf{new}} + T_{\mathsf{len}}[w]$.

3. In the third step, the client fetches the load for all required bin-layer pairs (γ, κ) and stores it in $T^{(3)}_{\mathsf{load}}$, i.e.all keys of $T^{(3)}_{\mathsf{load}}$ that are mapped to an integer n. The load $n_{\gamma,\kappa}$ will either be 0, if it was marked in step 2, or equal to the load of the bin at the given layer, if it was updated in step 4 (see step 4 for more details). The client proceeds as follows.

   Whenever she fetches the bin B_{cnt} during an update operation, she retrieves the load $n_{\gamma,\kappa}$ of bin B_{cnt} at all layer κ. Then, *only* if $T^{(3)}_{\mathsf{load}}[\gamma, \kappa] = 0$, she stores the load in $T^{(3)}_{\mathsf{load}}[\gamma, \kappa] = n_{\gamma,\kappa}$.

4. In the fourth step, the client fills $\mathsf{CB}_{\mathsf{out}}$ with the new sublists. Also, she updates T_{load} from the previous level according to the updates in order to avoid inconsistencies. She proceeds as follows.

   During the k-th update operation, she retrieves and removes some list and auxiliary information $(w, L_{\mathsf{new}}, \beta, \kappa)$ from $T^{(4)}_{\mathsf{hash}}[\alpha]$ for some bin B_α. Then, she computes the bin γ in which to store L_{new} according to $\mathsf{L2C}$ using the load information from $T^{(4)}_{\mathsf{load}}[\gamma, \kappa]$ and inserts the new sublist into $\mathsf{CB}^{(4)}_{\mathsf{out}}[\gamma]$. Further, she increments $T^{(4)}_{\mathsf{load}}[\gamma, \kappa]$. Note that this changes the load of the bin but we do not push L_{new} to the server in this step. Thus, the load information of the subsequent step 4 would be computed wrong for layers of bins that were updated in this step. In order to avoid these inconsistencies, the client further updates the load information of step 3 accordingly, i.e.sets $T^{(3)}_{\mathsf{load}}[\gamma, \kappa] = T^{(4)}_{\mathsf{load}}[\gamma, \kappa] + 1$ directly.

5. In the fifth step, the client writes the content of $\mathsf{CB}_{\mathsf{out}}$ to the server as in Algorithm 11.

Careful inspection shows that the pipeline pre-computation retains correctness. Also, the view of the server remains unchanged, thus $\mathsf{SmallHermes}_2$ remains semantically secure with the same leakage. Notably, even the client computation is de-amortized.

6.4.4 Heuristic Variant via Weighted 2C

In SmallHermes, we can replace L2C with weighted 2C. That is, weighted balls are inserted in the least loaded bin of two bins chosen at random, independent of layers. While we know of no non-trivial upper bound for this variant, heuristically it performs similar to L2C (see Figure 6.3). With this adaption, the entire load information can be kept on the client with $\mathcal{O}(W)$ storage. Thus, the $\mathsf{SmallHermes}_1$ has $\mathcal{O}(W)$ client storage using weighted 2C. Further, each update, we can directly decide in which bin to insert the added identifier, independent of layers, and

[3] This can be decided using another table T_{flg} that matches a keyword w to a flag $b \in \{0, 1\}$, initialized with 0 for each keyword. When the keyword w is considered, the flag T_{flg} is set to 1.

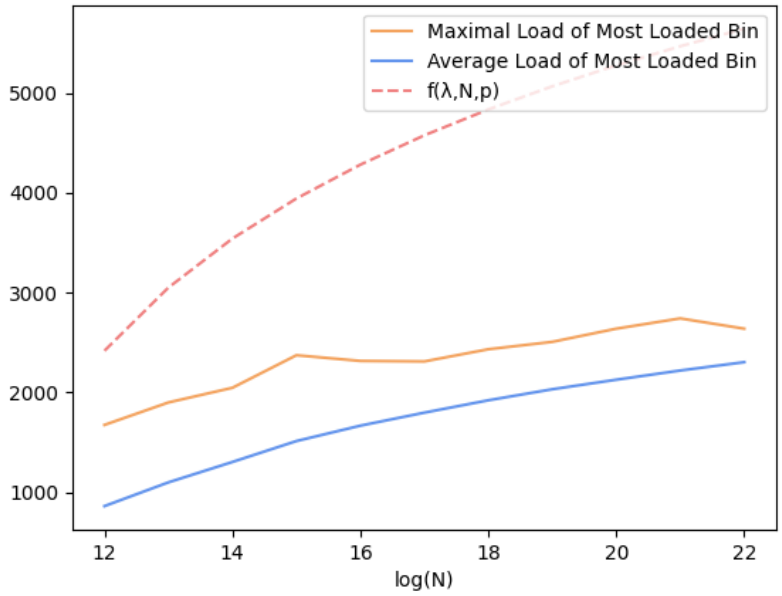

Figure 6.3: The loads of 2C for 10000 runs with balls of total weight N. The function $f(\lambda, N, p) = c \log\log\log(\lambda) \log\log(N/p)$ is the theoretical upper bound for the most loaded bin in L2C with constant $c = 2$ and $\lambda = 128$. The weights $\{w_i\}$ are chosen at random in $[1, 512]$.

update the load information accordingly. Recall that in $\mathsf{SmallHermes}_2$, we previously had to fetch *exactly* the right load information which resulted in complicated pipeline pre-computation. With weighted 2C, the client-side pipeline in $\mathsf{SmallHermes}_2$ can be heavily simplified, which reduces both the computation per update and the client storage. As this variant is only heuristically secure, we omit details.

6.4.5 Security

SmallHermes is forward secure with the same leakage as BigHermes.

Theorem 6.7. *Let N be an upper bound on the size of the database, W be an upper bound on the number of keywords and let $p \leq N^{1-1/\log\log\lambda}$ be the page size. We model $\mathsf{H} : \{0,1\}^* \mapsto \{1, \ldots, m\}$ as a random oracle. Let $N \leq pW$. Let* Enc *be a* IND-CPA *secure symmetric encryption scheme and* PRF *be a secure pseudorandom function (for the preprocessing of w).*

Let $\mathcal{L}^{\mathsf{fs}} = (\mathcal{L}^{\mathsf{fs}}_{\mathsf{Stp}}, \mathcal{L}^{\mathsf{fs}}_{\mathsf{Srch}}, \mathcal{L}^{\mathsf{fs}}_{\mathsf{Updt}})$, with $\mathcal{L}^{\mathsf{fs}}_{\mathsf{Stp}}(\mathsf{DB}, W, N) = (W, N)$, $\mathcal{L}^{\mathsf{fs}}_{\mathsf{Srch}}(w_i) = (\mathsf{qp}, \ell_i)$ where ℓ_i is the number of identifiers matching w_i, and $\mathcal{L}^{\mathsf{fs}}_{\mathsf{Updt}}(\mathsf{op}, w, \mathsf{id}) = \perp$. The scheme SmallHermes *is correct and $\mathcal{L}^{\mathsf{fs}}$-adaptively semantically secure.*

Before we give a detailed proof, we sketch the argument. The underlying L2C scheme is scaled such that up to N identifiers (organized into lists of size at most p) fit into the bins without overflowing (for any list distribution). SmallHermes inserts lists into the least loaded of two randomly chosen bins at the corresponding layer. Consequently, the bins are filled according to L2C. It follows from Theorem 4.4 that no bin overflows its capacity with overwhelming probability. Correctness follows immediately.

Semantic security also follows from the following facts. (1) The bins are encrypted and thus, only the upper bound N is leaked during setup. (2) During search, $2 \cdot \lceil \ell/p \rceil$ bins are accessed, where ℓ is the number of identifiers matching the searched keyword w. These bins are re-accessed if w is searched again but look random random to the server for the first search (as bin indices

are the output of a hash function and w is random in the view of the server). Thus, a search leaks the search pattern and the number of sublists. (3) Updates are performed by accessing each bin via a fixed schedule (that solely depends on the number of updates).

We now prove Theorem 6.7. Again, we omit correctness as it is tedious but straight-forward.

Proof. We show that SmallHermes is secure with leakage $\mathcal{L}^{\mathsf{fs}}$. Let Sim denote the simulator and $\mathcal{A}$ an arbitrary honest-but-curious PPT adversary. Initially, Sim receives $\mathcal{L}^{\mathsf{fs}}_{\mathsf{Stp}}(\mathsf{DB}, W, N) = (W, N)$ and later, an series of search and update queries with input $\mathcal{L}^{\mathsf{fs}}_{\mathsf{Srch}}(w_i) = (\mathsf{qp}, \ell_i)$ and $\mathcal{L}^{\mathsf{fs}}_{\mathsf{Updt}}(op_i, w_i, \mathsf{id}_i) = \perp$ respectively. First, Sim generates an encryption key $\mathsf{K}'_{\mathsf{Enc}}$ and initializes $m = N/(p \cdot b_{N,p})$ bins $B_1, \ldots, B_m$ zeroed out up to size $p \cdot b_{N,p}$. Then, Sim outputs $\mathsf{EDB}' = (\mathsf{Enc}_{\mathsf{K}'_{\mathsf{Enc}}}(B_1), \ldots, \mathsf{Enc}_{\mathsf{K}'_{\mathsf{Enc}}}(B_m))$. Further, Sim initializes a counter $\mathsf{cnt} = 0$. Next, Sim simulates the search and update queries.

For search queries, Sim receives query pattern qp and the length ℓ of the searched identifier list. If the query pattern qp indicates that the keyword was already searched, Sim outputs the keyword w' from the previous query. Otherwise, Sim outputs a new uniformly random keyword w' that has not been output during a search query yet. In addition, Sim forwards $x = \lceil \ell/p \rceil$ to the $\mathcal{A}$.

For update queries, Sim receives no input. Sim simply increments cnt, sends cnt to the $\mathcal{A}$, re-encrypts the received bin and sends it back to to $\mathcal{A}$.

We now show that the real game is indistinguishable from the ideal game. For this, we define three hybrid games.

- Hybrid 0 is identical to the real game.
- Hybrid 1 is the same as Hybrid 0, except the simulated keywords w' are output. As we assume that w is the output of a PRF and thus indistinguishable from random, Hybrid 0 and Hybrid 1 are (computationally) indistinguishable.
- Hybrid 2 is the same as Hybrid 1, except the encrypted database EDB is replaced with EDB' and whenever the simulator receives a bin, she simply sends back re-encrypted bins. Since Enc is IND-CPA secure (and bins always have size $p \cdot b_{N,p}$), it follows that Hybrid 1 and Hybrid 2 are indistinguishable.
- Hybrid 3 is identical to the ideal game. Hybrid 3 and Hybrid 2 are identically distributed and thus, indistinguishable. □

6.4.6 Efficiency

Each data structure on the client side requires at most $\mathcal{O}(W)$ storage. As there are only a constant number of data structures and pipeline steps, the total client storage is $\mathcal{O}(W)$. At most one bin is downloaded each update and each search on keyword w, exactly $\lceil \ell/p \rceil$ bins are retrieved, where ℓ is the length of the list of identifiers matching keyword w. As each bin of size $\widetilde{\mathcal{O}}\left(p \log\log \frac{N}{p}\right)$, the page efficiency is $\widetilde{\mathcal{O}}\left(\log\log \frac{N}{p}\right)$ and the server storage is $\mathcal{O}(N)$.

6.5 The Hermes Scheme: Putting Everything Together

We have constructed BigHermes (Section 6.3), an I/O-efficient SSE scheme with forward security in the big database regime, i.e., $N \geq pW$. Similarly, we built SmallHermes (Section 6.4) in the regime $N \leq pW$. We combine them to construct Hermes.

6.5.1 Hermes

The Hermes scheme simply uses either BigHermes or SmallHermes, depending on which regime the global parameters N, W and p are in. That is, Setup, Search, and Update for Hermes behave exactly as in BigHermes, if $N \geq pW$, and as in SmallHermes otherwise. Clearly, Hermes has $\widetilde{\mathcal{O}}(\log\log(N/p))$ page efficiency and $\mathcal{O}(1)$ storage efficiency. Further, because both sub-schemes are forward-secure with leakage $\mathcal{L}^{\mathsf{fs}}$ (Theorems 6.6 and 6.7), the same holds for Hermes. This is formalized in Theorem 6.8.

Theorem 6.8. *Let N be an upper bound on the size of the database, and let W be an upper bound on the number of keywords. Let p be the page size. Assume $p \leq N^{1-1/\log\log\lambda}$, and $N/p \geq \lambda$. Let* Enc *be an* IND-CPA*-secure encryption scheme, and let* PRF *be a secure pseudo-random function.*

Let $\mathcal{L}^{\mathsf{fs}} = (\mathcal{L}^{\mathsf{fs}}_{\mathsf{Stp}}, \mathcal{L}^{\mathsf{fs}}_{\mathsf{Srch}}, \mathcal{L}^{\mathsf{fs}}_{\mathsf{Updt}})$, with $\mathcal{L}^{\mathsf{fs}}_{\mathsf{Stp}}(\mathsf{DB}, W, N) = (W, N)$, $\mathcal{L}^{\mathsf{fs}}_{\mathsf{Srch}}(w_i) = (\mathsf{qp}, \ell_i)$ where ℓ_i is the number of identifiers matching w_i, and $\mathcal{L}^{\mathsf{fs}}_{\mathsf{Updt}}(\mathsf{op}, w, \mathsf{id}) = \perp$. Then Hermes *is correct and $\mathcal{L}^{\mathsf{fs}}$-adaptively semantically secure.*

6.5.2 Optimizations and Trade-offs

For ease of exposition, several design choices in Hermes have been made in favor of simplicity. Depending on the deployment scenario, various tradeoffs and optimizations are possible. We sketch a few in this section. These tradeoffs apply to both sub-schemes of Hermes, BigHermes and SmallHermes.

Round Trip Time

Hermes requires 1 RTT for searches, which is optimal for response-hiding SSE. Using standard techniques (piggy backing), it is straightforward to make Hermes optimal in RTT for updates. That is, the client performs each update over the course of multiple queries by stashing the update responses, and resuming the operation on the next query (either update or search).

Full-fledged Database

A full-fledged encrypted database for documents can be constructed in a natural way using Hermes. The client prepares a separate encrypted documents database on the server, and uses Hermes to encrypt the inverse index. The document database can be a hash table mapping document identifiers to encrypted documents [4]. For a search, the client first queries Hermes to retrieve the matching identifiers, then she queries the document hash table to obtain the concrete documents. For an update, she adds the fresh encrypted document to the document database directly, and then updates the inverse index encrypted via Hermes accordingly. As Hermes is forward secure and as pushing a document leaks no information about matching keywords, the scheme remains forward secure. Note that we do not have to buffer documents on the client during updates; only the inverse index implemented with Hermes requires client buffering.

In a plaintext database, those documents would also be stored in such a manner (but in plaintext). As page efficiency is relative to a plaintext database, the page efficiency of the document database is optimal, and the inverse index managed by Hermes retains its sublogarithmic page efficiency. As usual, this solution further leaks the access pattern. Note that we can also use Pluto or LayeredSSE to construct a full-fledged encrypted database in the same manner.

[4] As usual, this solution leaks the access pattern. It is possible to replace the hash table with an ORAM to obtain an SSE scheme that hides the access pattern, at the expense of a logarithmic document access cost. Note that the inverse index remains efficient.

Dynamic Server Storage

First, as defined, the Σ_{dum} component of Hermes requires allocating all memory upfront. If the database is intended to grow up to N elements, then $\mathcal{O}(N)$ memory must be allocated at setup time. In some use cases, that behavior may be undesirable. If so, the scheme can be modified so that its memory usage scales dynamically with the size of the database. First, setup an instance of Hermes with capacity k for some small k. Once that instance reaches full capacity, create a new instance with capacity $2k$, initialized with the content of the original instance. The original instance is then deleted, and the pattern repeats. This basic technique has been studied in depth in the context of memory allocation algorithms, and can be further refined in various way, for example to deamortize the cost of building the new instance; see *e.g.* [FKL01]. We leave further optimizations along this line for future works.

Reducing Client Storage

Hermes requires $\mathcal{O}(W)$ client storage. This is optimal for efficient forward secure SSE schemes [BF19]. In most such schemes, for example [Bos16, BMO17], the $\mathcal{O}(W)$ comes from the need to store a counter for each keyword, similar to T_{len} in Hermes. To improve memory efficiency, Hermes additionally buffers $\mathcal{O}(W)$ keyword-identifier pairs on the client. The client memory required for this can be reduced by a (constant) factor c in exchange for increasing update page efficiency by factor c. We sketch how to proceed for BigHermes. (The tradeoff can be applied to SmallHermes in a similar manner.) Each update, the client performs c updates of BigHermes at once. That is, she buffers the new keyword-identifier pair on the client, and instead of retrieving a single bin and performing two (dummy) updates to Σ, read c bins and perform $2c$ (dummy) updates to Σ. Now, each epoch lasts W/c updates and thus, only $\mathcal{O}(W)/c$ pairs need to be buffered on the client. While c bins are fetched each update (instead of one), note that the page efficiency of read queries is not impacted, as the worst-case load of the bins is not impacted.

On Leakage

In line with most SSE literature, Hermes does not specify how full documents are fetched on the server, once their identifier is retrieved. In SSE folklore, it is typically assumed that the documents are simply encrypted, and then queried from the server. That explains why most SSE schemes make no attempt to hide access pattern leakage (that is, which documents match a given query): this information will implicitly be revealed at a later stage, when the client queries the full documents. This leakage can sometimes be exploited by attacks, although this depends on the use case (see [BKM20] for a detailed analysis). Some recent work has proposed to obfuscate access pattern leakage at the expense of efficiency [GPPW20]. While such techniques are out of scope, we note that Hermes naturally lends itself to such obfuscation techniques, in particular because it never requires the server to learn document identifiers in the clear (in fact, it is naturally response-hiding).

Finally, we note that Hermes realizes an encrypted multi-map. As such, beyond the direct application to keyword search, it can be used in any application that relies on encrypted multi-maps, such as the graph search algorithms from [CK10].

Chapter

7 Practical Round-Optimal Blind Signatures in the ROM

The focus of this chapter is on blind signatures. We present two frameworks to construct blind signatures in the ROM. We instantiate each framework to obtain efficient round-optimal schemes secure under standard pairing assumptions.

Chapter content

7.1 Introduction 101
7.1.1 Contributions 103
7.1.2 Technical Overview 104
7.2 Optimizing the Fischlin Blind Signature 111
7.2.1 Construction 111
7.2.2 Correctness and Security 112
7.3 Instantiation of the Generic Construction 118
7.3.1 Optimizations and Efficiency 124
7.4 Blind Signatures based on Boneh-Boyen Signature 124
7.4.1 Construction 124
7.4.2 Correctness and Security 125
7.5 Instantiation of the Framework based on Boneh-Boyen 129
7.6 Frameworks for Partially Blind Signatures 143
7.6.1 Partial Blindness of the Optimized Fischlin Transform 143
7.6.2 Partial Blindness of Blind Signature based on Boneh-Boyen 145

7.1 Introduction

Blind signature is an interactive signing protocol between a signer and a user with advanced privacy guarantees. At the end of the protocol, the user obtains a signature for his choice of message while the signer remains blind to the message she signed. To capture the standard notion of unforgeability, it is further required that a user interacting with the signer at most ℓ-times is not be able to produce valid signatures on more than ℓ distinct messages. The former and latter are coined as the *blindness* and *one-more unforgeability* properties, respectively.

Chaum introduced the notion of blind signatures [Cha82] and showed its application to e-cash [Cha82, CFN90, OO92]. Since then, it has been an important building block for other applications such as anonymous credentials [Bra94, CL01], e-voting [Cha88, FOO92], direct anonymous attenstation [BCC04], and in more recent years, it has seen a renewed interest due to new applications in blockchains [YL19, BDE+22] and privacy-preserving authentication tokens [VPN22, HIP+22].

Round-Optimality. One of the main performance measures for blind signatures is *round-optimality*, where the user and signer are required to only send one message each to complete the signing protocol. While this is an ideal feature for practical applications, unfortunately, there are a few impossibility results [Lin08, FS10, Pas11] on constructing round-optimal blind signatures in the plain model (i.e., without any trusted setup) from standard assumptions (e.g., non-interactive assumptions and polynomial hardness). To circumvent this, cryptographers design round-optimal blind signatures by making a minimal relaxation of relying on the random oracle model (ROM) or the trusted setup model. Considering that trusted setups are a large obstacle for real-world deployment, in this chapter we focus on round-optimal blind signatures in the ROM under standard assumption[1]. We refer the readers to Section 2.4 on round optimal blind signatures under non-standard assumptions (e.g., interactive or super polynomial hardness) or relying on stronger idealized models such as the generic group model.

Practical Round-Optimal Blind Signatures. Constructing a *practical* round-optimal blind signature has been an active area of research. In a seminal work, Fischlin [Fis06] proposed the first generic round-optimal blind signature from standard building blocks. While the construction is simple, an efficient instantiation remained elusive since it required a non-interactive zero-knowledge (NIZK) proof for a relatively complex language.

Recently, in the lattice-setting, del Pino and Katsumata [dK22] showed a new lattice-tailored technique to overcome the inefficiency of Fischlin's generic construction and proposed a round-optimal blind signature with signature and communication sizes 100 KB and 850 KB.

A different approach that has recently accumulated attention is based on the work by Pointcheval [Poi98] that bootstraps a specific class of blind signature schemes into a fully secure one (i.e., one-more unforgeable even if polynomially many concurrent signing sessions are started). This approach has been improved by Katz et al. [KLR21] and Chairattana-Apirom et al. [CAHL+22], and the very recent work by Hanzlik et al. [HLW23] optimized this approach leading to a round-optimal blind signature based on the CDH assumption in the asymmetric pairing setting. One of their parameter settings provides a short signature size of 5 KB with a communication size 72 KB.

Finally, we observe that there are two constructions in the pairing setting with a trusted setup which can be instantiated in the ROM under standard assumptions [BFPV13, AJOR18][2]. Blazy et al. [BFPV13] exploited the randomizability of Waters signature [Wat05] and constructed a blinded version of Waters signature consisting of mere 2 group elements, i.e.96 B. While it achieves the shortest signature size in the literature, since the user has to prove some relation to his message in a bit-by-bit manner, the communication scales linearly in the message length. For example for 256 bit messages, it requires more than 220 KB in communication.

Abe et al. [AJOR18] use structure-preserving signatures (SPS) and Groth-Ostrovsky-Sahai (GOS) proofs [GOS12] to instantiate the Fischlin blind signature with signatures of size 5.8 KB with around 1 KB of communication [3].

While round-optimal blind signatures in the ROM are coming close to the practical parameter regime, the signature and communication sizes are still orders of magnitude larger compared to those relying on non-standard assumptions or strong idealized models such as blind RSA [Cha82, BNPS03] or blind BLS [Bol03]. Thus, we continue the above line of research to answer the following question:

> *How efficient can round-optimal blind signatures in the ROM be under standard assumptions?*

[1] We note that all of our results favor well even when compared with schemes in the trusted setup model.

[2] Both [BFPV13, AJOR18] require a trusted setup for a common reference string crs consisting of random group elements. We can remove the trusted setup by using a random oracle to sample crs.

[3] Interestingly, this natural approach yields a scheme with with 0.4 KB communication and 2.7 KB signature size under standard assumptions in the ROM, if instantiated with Jutla-Roy signatures [JR17].

Table 7.1: Comparison of Round-Optimal Blind Signatures in the ROM

Reference	Signature size	Communication size	Assumption
del Pino et al. [dK22]	100 KB	850 KB	DSMR, MLWE, MSIS
Blazy et al. [BFPV13]	96 B	220 KB †	SXDH, CDH
Abe et al. [AJOR18]	5.5 KB	1 KB	SXDH
Hanzlik et al. [HLW23]‡	5 KB 9 KB	72 KB 36 KB	CDH
Ours: Section 7.2	447 B	303 B	SXDH
Ours: Section 7.4	96 B	2.2 KB	DDH, CDH

All group-based assumptions are in the asymmetric paring setting, and MLWE and MSIS denote the module version of the standard LWE and SIS, respectively. DSMR denotes the decisional small matrix ratio problem, which can be viewed as the module variant of the standard NTRU. (†): Communication of [BFPV13] scales linearly with the message size, and is given here for 256 bit messages. (‡): [HLW23] offers tradeoffs between signature and communication sizes.

7.1.1 Contributions

We present two round-optimal blind signatures based on standard group-based assumptions in the asymmetric pairing setting. The efficiency is summarized in Table 7.1, along with the assumptions we rely on. The first construction has signature and communication sizes 447 B and 303 B, respectively. It has the smallest communication size among all prior schemes and is the first construction where the sum of the signature and communication sizes fit below 1 KB. The second construction has signature and communication sizes 96 B and 2.2 KB, respectively. While it has a larger communication size compared to our first construction, the signature only consists of 2 group elements, matching the previously shortest by Blazy et al. [BFPV13] while simultaneously improving their communication size by around two orders of magnitude. Both constructions have efficient partially blind variants.

For our first construction, we revisit the generic blind signature construction by Fischlin [Fis05]. We progressively weaken the building blocks required by Fischlin and show that the blind signature can be instantiated much more efficiently in the ROM than previously thought by a careful choice of the building blocks. At a high level, we show that the generic construction remains secure even if we replace the public-key encryption scheme (PKE) and online-extractable NIZK[4] with respectively a commitment scheme and a rewinding-extractable NIZK such as those offered by the standard Fiat-Shamir transform [FS87, PS00, BN06]. While these modifications may seem insignificant on the surface, it accumulates in a large saving in the concrete signature and communication sizes. Moreover, our security proof requires overcoming new technical hurdles incurred by the rewinding-extraction and relies on a fined-grained analysis of a variant of the forking lemma.

For our second construction, we revisit the idea by Blazy et al. [BFPV13] relying on randomizable signatures. However, our technique is not a simple application of their idea as their construction relies on the specific structure of the Waters signature in a non-black-box manner. Our new insight is that a specific class of signature schemes with an *all-but-one* (ABO) reduction can be used in an almost black-box manner to construct round-optimal blind signatures, where ABO reductions are standard proof techniques to prove selective security of public-key primitives (see references in [Nis20] for examples). Interestingly, we can cast the recent blind signature by

[4]This is a type of NIZK where the extractor can extract a witness from the proofs output by the adversary in an *on-the-fly* manner.

del Pino and Katsumata [dK22] that stated to use lattice-tailored techniques as one instantiation of our methodology.

In the instantiation of our second construction, we use the Boneh-Boyen signature [BB04a] that comes with an ABO reduction along with an online-extractable NIZK obtained via the Fiat-Shamir transform applied to Bulletproofs [BBB+18] and a Σ-protocol for some ElGamal related statements. To the best of our knowledge, this is the first time an NIZK that internally uses Bulletproofs was proven to be online-extractable in the ROM. Prior works either showed the non-interactive version of Bulletproofs to achieve the weaker rewinding extractability [AFK22, AC20] or the stronger online simulation extractability by further assuming the algebraic group model [GOP+22]. We believe the analysis of our online extractability to be novel and may be of independent interest.

7.1.2 Technical Overview

We give an overview of our techniques.

Fischlin's Round-Optimal Blind Signature. We review the generic construction by Fischlin [Fis05] as it serves as a starting point for both of our constructions. The construction relies on a PKE, a signature scheme, and an NIZK. The blind signature's verification and signing keys $(\mathsf{bvk}, \mathsf{bsk})$ are identical to those of the underlying signature scheme $(\mathsf{vk}, \mathsf{sk})$. For simplicity, we assume a perfect correct PKE with uniform random encryption keys ek and that ek is provided to all the players as an output of the random oracle. The user first sends an encryption $c \leftarrow \mathsf{PKE}(\mathsf{ek}, m; r)$ of the message m. The signer then returns a signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, c)$ on the ciphertext c. The user then encrypts $\widehat{c} \leftarrow \mathsf{PKE}(\mathsf{ek}, c \| r \| \sigma; \widehat{r})$ and generates an NIZK proof π of the following fact where $(c, \sigma, r, \widehat{r})$ is the witness: $\widehat{c}$ encrypts (c, r, σ) under $\widehat{r}$; c encrypts the message m under r; and σ is a valid signature on c. The user outputs the blind signature $\sigma_{\mathsf{BS}} = (\widehat{c}, \pi)$.

It is not hard to see that the scheme is blind under the IND-CPA security of the PKE and the zero-knowledge property of the NIZK. The one-more unforgeability proof is also straight-forward: The reduction will use the adversary $\mathcal{A}$ against the one-more unforgeability game to break the EUF-CMA of the signature scheme. The reduction first programs the random oracle so that it knows the corresponding decryption key dk of the PKE. When $\mathcal{A}$ submits c to the blind signing oracle, the reduction relays this to its signing oracle and returns $\mathcal{A}$ the signature σ it obtains. Moreover, it makes a list L of decrypted messages $m \leftarrow \mathsf{Dec}(\mathsf{dk}, c)$. When $\mathcal{A}$ outputs the forgeries $(\sigma_{\mathsf{BS},\mathsf{i}} = (\widehat{c}_i, \pi_i), m_i)_{i \in [\ell+1]}$, it searches a m_i such that $m_i \notin L$, which is guaranteed to exist since there are at most ℓ signing queries. The reduction then decrypts $(c_i, r_i, \sigma_i) \leftarrow \mathsf{Dec}(\mathsf{dk}, \widehat{c}_i)$. Since the PKE is perfectly correct and due to the soundness of the NIZK, c_i could not have been queried by $\mathcal{A}$ as otherwise $m_i \in L$, and hence, (c_i, σ_i) breaks EUF-CMA security.

Source of Inefficiency. There are two sources of inefficiency when trying to instantiate this generic construction. One is the use of a *layered* encryption: the NIZK needs to prove that c is a valid encryption of m on top of proving $\widehat{c}$ is a valid encryption of (c, r, σ). This contrived structure was required to bootstrap a sound NIZK to be *online-extractable*.[5] Specifically, the one-more unforgeability proof relied on the reduction being able to extract the (partial) witness (c_i, r_i, σ_i) in an on-the-fly manner from the outer encryption $\widehat{c}_i$ explicitly included in the blind signature. The other inefficiency stems from the heavy reliance on PKEs. As far as the correctness is concerned, the PKE seems replaceable by a computationally binding commitment scheme. This would be ideal since commitment schemes tend to be more size efficient than PKEs since decryptability is not required. However, without a PKE, it is not clear how the above proof

[5] Constructing an online extractable NIZK by adding a PKE on top of a sound NIZK is a standard method.

would work.

First Construction

We explain our first construction, an optimized variant of Fischlin's generic construction.

Using Rewinding-Extractable NIZKs. The first step is to relax the online-extractable NIZK with a (single-proof) rewinding-extractable NIZK. Such an NIZK allows extracting a witness from a proof output by an adversary $\mathcal{A}$ by *rewinding* $\mathcal{A}$ on a fixed random tape. NIZKs obtained by compiling a Σ-protocol using the Fiat-Shamir transform is a representative example of an efficient rewinding-extractable NIZK. The net effect of this modification is that we can remove the layer of large encryption by $\widehat{c}$, thus making the statement simpler and allowing us to remove $\widehat{c}$ from σ_{BS}.

Let us check if this rewinding-extractable NIZK suffices in the above proof of one-more unforgeability. At first glance, the proof does not seem to work due to a subtle issue added by the rewinding extractor. Observe that the reduction now needs to simulate $\mathcal{A}$ in the *rewound execution* as well. In particular, after rewinding $\mathcal{A}$, $\mathcal{A}$ may submit a new c' to the blind signing oracle, which was not queried in the initial execution. The reduction relays this c' to its signing oracle as in the first execution to simulate the signature σ'. As before, we can argue that there exists a message m_i in the forgeries output by $\mathcal{A}$ in the *first* execution such that $m_i \notin L$, but we need to further argue that $m_i \notin L'$, where L' is the list of decrypted messages $\mathcal{A}$ submitted in the *rewound* execution. Namely, we need to argue that $m_i \notin L \cup L'$ for the reduction to break EUF-CMA security. However, a naive counting argument as done before no longer works because $|L \cup L'|$ can be large as 2ℓ, exceeding the number of forgeries output by $\mathcal{A}$, i.e., $\ell + 1$.

We can overcome this issue by taking a closer look at the internal of a particular class of rewinding-extractable NIZK. Specifically, throughout this paper, we focus on NIZKs constructed by applying the Fiat-Shamir transform on a Σ-protocol (or in more general a public-coin interactive protocol). A standard way to argue rewinding-extractability of a Fiat-Shamir NIZK is by relying on the forking lemma [PS00, BN06], which states (informally) that if an event E happened in the first run, then it will happen in the rewound round with non-negligible probability. In the above context, we define E to be the event that the i-th message in $\mathcal{A}$'s forgeries satisfy $m_i \notin L$, where i is sampled uniformly random by the reduction at the outset of the game. Here, note that E is well-defined since the reduction can prepare the list L by decrypting $\mathcal{A}$'s signing queries. The forking lemma then guarantees that we also have $m_i \notin L'$ in the rewound execution.[6] This slightly more fine-grained analysis allows us to replace the online-extractable NIZK with a rewinding-extractable NIZK.

Issue with Using Commitments. The next step is to relax the PKE by a (computationally binding) commitment scheme. While the correctness and blindness hold without any issue, the one-more unforgeability proof seems to require a major reworking. The main reason is that without the reduction being able to decrypt $\mathcal{A}$'s signing queries c, we won't be able to define the list L. In particular, we can no longer define the event E, and hence, cannot invoke the forking lemma. Thus, we are back to the situation where we cannot argue that the extracted witness (c_i, r_i, σ_i) from $\mathcal{A}$'s forgeries, is a valid forgery against the EUF-CMA security game. Even worse, $\mathcal{A}$ could potentially be breaking the computationally binding property of the commitment scheme by finding two message-randomness pairs (m_i, r_i) and (m'_i, r'_i) such that they both commit to c_i but $m_i \neq m'_i$. In such a case, extracting from a single proof does not seem sufficient since a

[6] For the keen readers, we note that we are guaranteed to have the same i-th message in both executions since these values are fixed at the forking point due to how the Fiat-Shamir transform works.

reduction would need at least two extracted witnesses to break the binding of the commitment scheme.

To cope with the latter issue first, we extend the one-more unforgeability proof to rely on a *multi-proof* rewinding-extractable NIZK. In general, multi-proof rewinding-extractors run in exponential time in the number of proofs that it needs to extract from [SG98, BFW15]. However, in our situation, with a careful argument, we can prove that our extractor runs in strict polynomial time since $\mathcal{A}$ provides all the proofs to the extractor only at the end of the game. This is in contrast to the settings considered in [SG98, BFW15] where $\mathcal{A}$ can adaptively submit multiple proofs to the extractor throughout the game.

We note that the assumption we require has not changed: a Σ-protocol for the same relation as in the single-proof setting compiled into an NIZK via the Fiat-Shamir transform. To prove multi-proof rewinding-extractability of this Fiat-Shamir NIZK, we can no longer rely on the now standard general forking lemma by Bellare and Neven [BN06] that divorces the probabilistic essence of the forking lemma from any particular application context. A naive extension of the general forking lemma to the multi-forking setting will incur an exponential loss in the success probability. To provide a meaningful bound, we must take into account the extra structure offered by the Fiat-Shamir transform, and thus our analysis is akin to the more traditional forking lemma analysis by Pointcheval and Stern [PS00] or by Micali and Reyzin [MR02]. To the best of our knowledge, we provide the first formal analysis of the multi-proof rewinding-extractability of an NIZK obtained by applying the Fiat-Shamir transform to a Σ-protocol. We believe this analysis to be of independent interest.

Final Idea to Finish the Proof. Getting back to the proof of one-more unforgeability, the reduction now executes the multi-proof rewinding-extractor to extract all the witnesses $(c_i, r_i, \sigma_i)_{i\in[\ell+1]}$ from the forgeries. Relying on the binding of the commitment scheme, we are guaranteed that all the commitment c_i's are distinct. Moreover, since $\mathcal{A}$ only makes ℓ blind signature queries *in the first execution*, we further have that there exists at least one c_i in the forgeries which $\mathcal{A}$ did not submit in the first execution.

However, we are still stuck since it's unclear how to argue that this particular c_i was never queried by $\mathcal{A}$ in any of the rewound executions. Our next idea is to slightly strengthen the NIZK so that the proof π is statistically binding to a portion of the witness that contains the commitments.[7] We note that this is still strictly weaker and more efficiently instantiable compared to an online-extractable NIZK required by Fischlin's construction since we do not require the full list of witnesses to be efficiently extractable from the proofs in an online manner. We use this property to implicitly fix the commitments $(c_i)_{i\in[\ell+1]}$ included in the forgeries after the end of the first execution of $\mathcal{A}$. This will be the key property to completing the proof.

The last idea is for the reduction to randomize what it queries to its signing oracle. For this, we further assume the commitment scheme is randomizable, where we emphasize that this is done for ease of explanation and we do not strictly require such an assumption (see remark 7.6). When $\mathcal{A}$ submits a commitment c to the blind signing oracle, the reduction randomizes c to c' using some randomness rand and instead sends c' to its signing oracle. It returns the signature σ *and* rand to $\mathcal{A}$. $\mathcal{A}$ checks if c becomes randomized to c' using rand and if σ is a valid signature on c'. It then uses c' instead of c to generate the blind signature as before. The key observation is that the reduction is invoking its signing oracle with randomness outside of $\mathcal{A}$'s control. Since the commitments $(c_i)_{i\in[\ell+1]}$ were implicitly fixed at the end of the first execution, any randomized c' sampled in the subsequent rewound execution is independent of these commitments. Hence, the probability that the reduction queries c_i to the signing oracle in any of the rewound execution is negligible, thus constituting a valid forgery against the `EUF-CMA` security game as desired.

[7] At the Σ-protocol abstraction, we call this new property f-unique extraction. It is a strictly weaker property than the *unique response* property considered in the literature [Fis05, Unr12].

Instantiation. We instantiate the framework in the asymmetric pairing setting, i.e.we have groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order p, some fixed generators $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, and a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$. For the commitment scheme, we choose Pedersen commitments ($\mathsf{C}_{\mathrm{Ped}}$) of the form $c = g_1^m \mathsf{pp}^r$, as $\mathsf{C}_{\mathrm{Ped}}$ is randomizable and consists of a single group element. Note that the public parameter $\mathsf{pp} \in \mathbb{G}_1$ is generated via a random oracle. We then need to choose an appropriate signature scheme that allows signing $\mathsf{C}_{\mathrm{Ped}}$ commitments. We choose SPS as all components of the scheme are group elements, in particular, the message space is $\mathbb{G}_1^\ell$, where ℓ is the message length. The most efficient choice in the standard model is [JR17] with signatures of size 335 Byte. Instead, we optimize KPW signatures [KPW15] to a signature size of 223 Byte (from originally 382 Byte). Our optimized variant $\mathsf{S}_{\mathrm{KPW}}$ is no longer structure-preserving, as it consists of one element τ in $\mathbb{Z}_p$, but suffices for our applications. We refer to section 7.3 for more details.

Note that $\mathsf{S}_{\mathrm{KPW}}$ would be an inefficient choice in the original Fischlin blind signature [Fis06], as it requires encrypting the signature τ over $\mathbb{Z}_p$ to instantiate the online-extractable NIZK. In the pairing setting, this incurs an overhead in proof size linear in the security parameter λ [8]. The benefit of using our framework with the weaker rewinding-extractable NIZK is that we now only need to prove knowledge of τ, and thus can get away without encrypting it. Such an NIZK is possible with a single element in $\mathbb{Z}_p$ based on a Schnorr-type Σ-protocol (compiled with Fiat-Shamir). In the Σ-protocol, we further commit to group elements $(w_i)_i \in \mathbb{G}_1^n$ in the witness via ElGamal commitments (C_{EG}) of the form $E_i = (w_i \cdot \mathsf{pp}^{r_i}, g_1^{r_i})$, which the prover sends to the verifier in the first flow. In particular, this ensures f-unique extraction, as E_i fixes the commitment $c \in \{w_i\}_i$ statistically. Naively, this approach requires $2n$ group elements, where n is the number of group elements in the witness. Instead, we share the randomness among all commitments under different public parameters pp_i generated via a random oracle. The commitments remain secure but require only $n+1$ group elements. In particular, we set $E_i = (w_i \cdot \mathsf{pp}_i^s)$ and fix s via $S = g_1^s$. Then, we can open *all* commitments E_i in zero-knowledge with a *single* element in $\mathbb{Z}_p$, as knowledge of s is sufficient to recover the witness w_i from all E_i. Then, we compile our Σ-protocol with Fiat-Shamir to obtain a rewinding-based NIZK. We apply a well-known optimization to avoid sending some of the first flow α, and include the hash value $\beta \leftarrow \mathsf{H}(x, \alpha)$ in the proof explicitly. In total, compared to sending the witness to the verifier in the *clear*, our NIZK only has an overhead of 1 group element in $\mathbb{G}_1$ and 3 elements in $\mathbb{Z}_p$. The additional group element is S. The three additional $\mathbb{Z}_p$ elements are the hash value β, and values in the third flow required for (i) showing knowledge of s and (ii) linearizing a quadratic equation in the signature verification.

The instantiation of our framework achieves communication size of 303 Byte and signature size of 447 Byte.

Second Construction

We explain our second construction relying on randomizable signatures with an ABO reduction.

Getting Rid of NIZKs in the Signature. While the previous construction provides a small sum of signature and communication sizes, one drawback is that the blind signature has inherently a larger signature than those of the underlying signature scheme. The source of this large blind signature stems from using an NIZK to hide the underlying signature provided by the signer.

A natural approach used in the literature is to rely on techniques used to construct *randomizable* signature schemes [BFPV13, FHS15, FHKS16, KSD19]. Informally, a randomizable signature scheme allows to publicly randomize the signature σ on a message m to a fresh signature σ'.

[8] For instance, with ElGamal, the message is encrypted in the exponent and decryption would require a discrete logarithm computation. Thus, the message is typically encrypted bit-wise which incurs an overhead of $\log_2(p)$.

Many standard group-based signature schemes (in the standard model and ROM) are known to satisfy this property, e.g., [BB04a, Wat05]. A failed attempt would be for the user to randomize the signature σ provided by the signer and output the randomized σ' as the blind signature. Clearly, this is not secure since the user is not hiding the message m, that is, σ and σ' are linkable through m thus breaking blindness. An idea to fix this would be to let the user send a commitment $c = \mathsf{Com}(m; r)$ to the signer and the signature signs the "message" c. However, unless the commitment c can be randomized consistently with σ, we would still need to rely on an NIZK to hide c. This calls for a signature scheme that is somehow compatible with commitments.

Signatures with All-But-One Reductions. Our main insight is that a specific class of signature schemes with an *all-but-one* (ABO) reduction is naturally compatible with blind signatures. An ABO reduction is a standard proof technique to prove selective security of public key primitives, e.g., [BB04b, SW05, GPSW06, ABB10], where a formal treatment can be found in [Nis20]. In the context of signature schemes, this is a proof technique that allows the reduction to embed the challenge message m^* (i.e., the signature for which the adversary forges) into the verification key. The reduction can simulate any signatures on $m \neq m^*$, and when the adversary outputs a forgery on m^*, then the reduction can break some hard problems.

Let us now specify the class of signature scheme. We assume an additive homomorphic commitment scheme, that is, $\mathsf{Com}(m; r) + \mathsf{Com}(m'; r') = \mathsf{Com}(m + m'; r + r')$. We then assume a signature scheme where the signing algorithm $\mathsf{Sig}(\mathsf{sk}, m)$ can be rewritten as $\widehat{\mathsf{Sig}}(\mathsf{sk}, \mathsf{Com}(m; 0)+u)$, where u is some fixed but random commitment included in the verification key. Namely, Sig first commits to the message m using no randomness, adds u to it, and proceeds with signing. Note that if $u = \mathsf{Com}(-m'; r')$ for some (m', r'), then $\mathsf{Com}(m; 0) + u = \mathsf{Com}(m - m'; r')$. While contrived at first glance, this property is naturally satisfied by many of the signature schemes that admit an ABO reduction; the ABO reduction inherently requires embedding the challenge message m^* into the verification key in an unnoticeable manner and further implicitly requires message m submitted to the signing query to interact with the "committed" m^*. Specifically, the former hints at a need for an (implicit) commitment scheme and the later hints at the need for some operation between the commitments. Finally, to be used in the security proof, we assume there is a simulated signing algorithm $\widehat{\mathsf{SimSig}}$ along with a trapdoor td such that $\widehat{\mathsf{SimSig}}(\mathsf{td}, \mathsf{Com}(m - m'; r'), m - m', r') = \widehat{\mathsf{Sig}}(\mathsf{sk}, \mathsf{Com}(m; 0) + u)$ if and only if $m \neq m'$, where recall $u = \mathsf{Com}(-m'; r')$. Specifically, $\widehat{\mathsf{SimSig}}$ can produce a valid signature if it knows the *non-zero* commitment message and randomness.

Let us explain the ABO reduction in slightly more detail. In the security proof, the reduction guesses (or the adversary $\mathcal{A}$ submits) a challenge message m^* that $\mathcal{A}$ will forge on. It then sets up the verification key while replacing the random commitment u to $u = \mathsf{Com}(-m^*; r^*)$ while also embedding a hard problem that it needs to solve. Due to the hiding property of the commitment scheme, this is unnoticeable from $\mathcal{A}$. Then, instead of using the real signing algorithm $\widehat{\mathsf{Sig}}$, the reduction uses the simulated signing algorithm $\widehat{\mathsf{SimSig}}$. As long as $m \neq m^*$, $\widehat{\mathsf{SimSig}}(\mathsf{td}, \mathsf{Com}(m - m^*; r^*), m - m^*, r^*)$ outputs a valid signature, and hence, can be used to simulate the signing oracle. Finally, given a forgery on m^*, the reduction is set up so that it can break a hard problem.

Turning it into a Blind Signature. To turn this into a blind signature, the key observation is that $\widehat{\mathsf{Sig}}$ is agnostic to the committed message and randomness of $\mathsf{Com}(m; 0) + u$ — these are only used during the security proof when running $\widehat{\mathsf{SimSig}}$. Concretely, a user of a blind signature can generate a valid commitment $\mathsf{Com}(m; r)$, send it to the signer, and the signer can simply return $\sigma_r \leftarrow \widehat{\mathsf{Sig}}(\mathsf{sk}, \mathsf{Com}(m; r) + u)$. If the signature admits a way to map σ_r back to a normal signature σ for m, then we can further rely on the randomizability of the signature scheme to obtain a fresh signature σ' on the message m.

The proof of one-more unforgeability of this abstract blind signature construction is almost identical to the original ABO reduction with one exception. For the reduction to invoke the simulated $\widehat{\mathsf{SimSig}}$, recall it needs to know the message and randomness of the commitment $\mathsf{Com}(m;r)+u$. Hence, we modify the user to add an *online-extractable* NIZK to prove the correctness of the commitment $\mathsf{Com}(m;r)$ so that the reduction can extract (m,r). Here, we require online-extractability rather than rewinding-extractability since otherwise, the reduction will run exponentially in the number of singing queries [SG98, BFW15]. Also, this is why the communication size becomes larger compared with our first construction. Finally, when the adversary outputs a forgery including m^*, the reduction can break a hard problem as before. Here, we note that we can simply hash the messages m with a random oracle to obtain an adaptively secure scheme using the ABO reduction.

Interestingly, while the recent lattice-based blind signature by del Pino and Katsumata [dK22] stated to use lattice-tailored techniques to optimize Fischlin's generic construction, the construction and the proof of one-more unforgeability follows our above template, where they use the Agrawal-Boneh-Boyen signature [ABB10] admitting an ABO reduction. The only difference is that since lattices do not have nice randomizable signatures, they still had to rely on an NIZK for the final signature. While we focused on ABO reductions where only one challenge message m^* can be embedded in the verification key, the same idea naturally extends to all-but-*many* reductions. The blind signature by Blazy et al. [BFPV13] relying on the Waters signature can be viewed as one such instantiation. Finally, while we believe we can make the above approach formal using the ABO reduction terminology defined in [Nis20], we focus on one class of instantiation in the main body for better readability. Nonetheless, we believe the above abstract construction will be useful when constructing round-optimal blind signatures from other assumptions.

Instantiation. We instantiate the above framework with the Boneh-Boyen signature scheme S_{BB} [BB04a, BB08]. Recall that signatures of S_{BB} on a message $m \in \mathbb{Z}_p$ are of the form $\sigma = (\mathsf{sk}\cdot(u_1^m \cdot h_1)^r, g_1^r)$, where $u_1, h_1 \in \mathbb{G}_1$ are part of the verification key, sk is the secret key and $r \leftarrow \mathbb{Z}_p$ is sampled at random. We observe that S_{BB} is compatible with the Pedersen commitment scheme $\mathsf{C}_{\mathrm{Ped}}$ with generators u_1 and g_1. Roughly, the user commits to the message m via $c = u_1^m \cdot g_1^s$[9], where $s \leftarrow \mathbb{Z}_p$ blinds the message, proves that she committed to m honestly with a proof π generated via an appropriate online-extractable NIZK Π, and sends (c,π) to the signer. The signer checks π and signs c via $(\mu_0,\mu_1) \leftarrow (\mathsf{sk}\cdot(c\cdot h_1)^r, g_1^r)$. Note that as c shares the structure u_1^m with S_{BB} signatures on message m, the user can recompute a valid signature on m via $\sigma \leftarrow (\mu_0 \cdot \mu_1^{-s}, \mu_1)$. Before presenting σ to a verifier, the user rerandomizes σ to ensure blindness. We refer to section 7.4 for more details.

The main challenge is constructing an efficient *online-extractable* NIZK Π for the relation $\mathsf{R}_{\mathsf{bb}} = \{(x,w) : c = u_1^m \cdot g_1^s\}$, where $x = (c, u_1, g_1)$ and $w = (m,s)$. As we require online-extraction, a simple Σ-protocol showing $c = u_1^m \cdot g_1^s$ compiled via Fiat-Shamir is no longer sufficient as in our prior instantiation, as the extractor needs to rewind the adversary in order to extract (m,s). For example, we could instantiate Π with the (online-extractable) GOS proofs but such a proof has a size of around 400 KB. Another well-known approach is to additionally encrypt the witness (m,s) via a PKE and include the ciphertext into the relation; recall this method was used when explaining the Fischlin blind signature. The extractor can then use the secret key to decrypt the witnesses *online*. While a common choice for the PKE would be ElGamal encryption, this is insufficient since the extractor can only decrypt group elements g_1^m and g_1^s and not the witness in $\mathbb{Z}_p$ as required. To circumvent this, a common technique is to instead encrypt the binary decompositions $(m_i, s_i)_{i\in[\ell_2]}$ of m, s, respectively, with ElGamal, where $\ell_2 = \log_2(p)$. It then proves with a (non-online extractable) NIZK that $m = \sum_{i=1}^{\ell_2} m_i 2^{i-1}$ and $s = \sum_{i=1}^{\ell_2} s_i 2^{i-1}$ are valid openings of c, while also proving that m_i, s_i encrypted in the ElGamal ciphertexts are elements

[9] In the actual construction, we further hash m by a random oracle; this effectively makes S_{BB} *adaptively* secure.

in $\{0,1\}$, where the latter can be done via the equivalent identity $x \cdot (1-x) = 0$. The extractor can now decrypt the ElGamal encryptions of m_i to $g_1^{m_i} \in \{g_1, 1_{\mathbb{G}_1}\}$ and efficiently decide whether m_i is 0 or 1. Similarly, it can recover the decomposition s_i. Unfortunately, this approach requires at least $2\ell_2$ ElGamal ciphertexts which amount to 32 KB alone. In fact, the bit-by-bit encryption of the witness is also the efficiency bottleneck of GOS proofs for $\mathbb{Z}_p$ witnesses.

We refine the above approach in multiple ways to obtain concretely efficient online-extractable NIZKs. Instead of using the binary decomposition, we observe that the extractor can still recover x from g_1^x if $x \in [0, B-1]$ is short, i.e., $B = \mathsf{poly}(\lambda)$. Thus, we let the prover encrypt the B-ary decompositions $(m_i, s_i)_{i\in[\ell]}$ of m and s, where $\ell = \log_B(p)$. For example, setting $B = 2^{32}$ allows the extractor to recover m_i via a brute-force calculation of the discrete logarithm, and the number of encryptions is reduced by a factor of 32. Concretely, we modify the prover to prove that an ElGamal ciphertext encrypts $(m_i, s_i)_{i\in[\ell]}$ such that (i) each m_i and s_i are in $[0, B-1]$, and (ii) $m = \sum_{i=1}^{\ell} m_i B^{i-1}$, $s = \sum_{i=1}^{\ell} s_i B^{i-1}$, and $c = u_1^m \cdot g_1^s$.

To instantiate our approach, we glue two different (non-online extractable) NIZKs Π_{rp} and Π_{ped} together, each being suitable to show relations (i) and (ii), respectively. For the range relation (i), we appeal to the batched variant of Bulletproofs [AC20] and turn it non-interactive with Fiat-Shamir. For the linear relation (ii), we use a standard NIZK with an appropriate Σ-protocol compiled with Fiat-Shamir. We further apply three optimizations to make this composition of NIZKs more efficient:

1. While Bulletproofs require committing to the decompositions $(m_i, s_i)_{i\in[\ell]}$ in Pedersen commitments, we use the shared structure of ElGamal ciphertexts and Pedersen commitments to avoid sending additional Pedersen commitments. This also makes the relation simpler since we do not have to prove consistency between the committed components in the ElGamal ciphertext and Pedersen commitment.

2. We use a more efficient discrete logarithm algorithm during extraction with runtime $\mathcal{O}(\sqrt{B})$, which allows us to choose more efficient parameters for the same level of security. This further reduces the number of encryptions by a factor 2.

3. We perform most of the proof in a more efficient elliptic curve $\widehat{\mathbb{G}}$ of same order p without pairing structure. As both the NIZKs Π_{rp} and Π_{ped} are not reliant on pairings, this reduces the size and efficiency of the NIZK considerably.

Proof of Instantiation. Finally, we analyze the security of the optimized online-extractable NIZK Π obtained by gluing Π_{rp} and Π_{ped} together. Correctness and zero-knowledge are straightforward. Also, online-extraction seems immediate on first sight. The extractor decrypts the decomposition, reconstructs the witness (m, s), and checks whether $c = u_1^m g_1^s$. To show why it works, we rely on the soundness of the range proof Π_{rp} to guarantee that the committed values are short. This allows the extractor to decrypt efficiently. Moreover, we rely on the soundness of Π_{ped} to guarantee that the decrypted values form a proper B-ary decompositions of an opening (m, s) of c. However, this high-level idea misses many subtle issues.

First, Bulletproofs are not well-established in the non-interactive setting in the ROM. While Attema et al. [AFK22] show that special sound multi-round proof systems are knowledge sound (or rewinding-extractable) when compiled via Fiat-Shamir, Bulletproofs are only *computationally* special sound under the DLOG assumption. An easy fix for this is to relax the relation of the extracted witness. That is we use two different relations: one to be used by the prover and the other to be used by the extractor. We define an extracted witness w to be in the relaxed relation if either w is in the original relation *or* w is a DLOG solution with respect to (part of) the statement. With this relaxation, the interactive Bulletproofs becomes special sound for the relaxed relation since we can count the extracted DLOG solution as a valid witness.

Observing that the result of [AFK22] naturally translates to relaxed relations, we can conclude the non-interactive Bulletproofs to be rewinding-extractable in the ROM.

The second subtlety is more technical. For the formal proof, when the adversary submits a proof such that the online-extraction of Π fails, we must show that the adversary is breaking either the soundness of the underlying NIZKs Π_{rp} or Π_{ped}. Recall that Π_{rp} and Π_{ped} are glued together via the ElGamal ciphertext (cf. item 1). Specifically, each witness $w \in (m_i, s_i)_{i\in[\ell]}$ are encrypted as $c = (c_0, c_1) = (g^w \mathsf{pp}^r, g^r)$ with randomness $r \leftarrow \mathbb{Z}_p$, and Π_{rp} uses the partial "Pedersen part" c_0, while Π_{ped} uses the entire "ElGamal part" c. Thus one possibility for the online-extraction of Π failing is when the adversary breaks the tie between the two NIZKs by breaking the binding property of the Pedersen commitment. That is, if the adversary finds the DLOG between (g, pp), it can break the consistency between the two NIZKs in such a way that online-extraction of Π fails.

Put differently, to show that no adversary can trigger a proof for which the online-extraction of Π fails, we must show (at the minimum) that we can use such an adversary to extract a DLOG solution between (g, pp). This in particular implies that we have to *simultaneously* extract the witness w_0 of Π_{rp} containing one opening of c_0 and the witness w_1 of Π_{ped} containing the other opening of c_0 in order to break DLOG with respect to (g, pp), or equivalently to break the binding property of the Pedersen commitment. The issue with this is that we cannot conclude that both extractions succeed at the same time even if Π_{rp} and Π_{ped} *individually* satisfy the standard notion of rewinding-extractability. For instance, using the standard notion of rewinding-extractability, we cannot exclude the case where the adversary sets up the proofs π_0, π_1 of $\Pi_{\mathsf{rp}}, \Pi_{\mathsf{ped}}$, respectively, in such a way that if the extractor of Π_{rp} succeeds, then the extractor of Π_{ped} fails. We thus show in a careful non-black box analysis that the extraction of both proofs succeeds at the same time with non-negligible probability. To the best of our knowledge, this is the first time an NIZK that internally uses Bulletproofs is proven to be online-extractable in the ROM. We believe that our new analysis is of independent interest.

7.2 Optimizing the Fischlin Blind Signature

In this section, we provide an optimized generic construction of blind signatures compared with the Fischlin blind signature [Fis06]. In particular, we relax the extractable (and perfect binding) commitment and multi-online extractable NIZK used as the central building block for the Fischlin blind signature by a computationally binding commitment and a standard rewinding-based NIZK built from a Σ-protocol satisfying f-unique extraction. As we show in Section 7.3, this relaxation allows us to minimize the sum of the communication and signature size. We construct a natural partially blind variant in Section 7.6.

7.2.1 Construction

Our generic construction is based on the building blocks $(\mathsf{C}, \mathsf{S}, \Sigma)$ that satisfy some specific requirements. If $(\mathsf{C}, \mathsf{S}, \Sigma)$ satisfies these requirements, then we call it $\mathsf{BS}_{\mathsf{Rnd}}$-*suitable*.

Definition 7.1 ($\mathsf{BS}_{\mathsf{Rnd}}$-Suitable $(\mathsf{C}, \mathsf{S}, \Sigma)$)**.** The tuple of schemes $(\mathsf{C}, \mathsf{S}, \Sigma)$ are called $\mathsf{BS}_{\mathsf{Rnd}}$-*suitable*, if it holds that

- C is a correct and hiding *rerandomizable* commitment scheme with public parameter, message, randomness, and commitment spaces $\{0,1\}^{\ell_{\mathsf{C}}}, \mathcal{C}_{\mathsf{msg}}, \mathcal{C}_{\mathsf{rnd}}$, and $\mathcal{C}_{\mathsf{com}}$, respectively, such that $\mathcal{C}_{\mathsf{msg}}$ is efficiently sampleable and $1/|\mathcal{C}_{\mathsf{msg}}| = \mathsf{negl}(\lambda)$,
- S is a correct and EUF-CMA secure *deterministic* signature scheme with message space $\mathcal{S}_{\mathsf{msg}}$ that contains $\mathcal{C}_{\mathsf{com}}$, i.e., $\mathcal{C}_{\mathsf{com}} \subseteq \mathcal{S}_{\mathsf{msg}}$ and we assume elements in $\mathcal{S}_{\mathsf{msg}}$ are efficiently checkable,

– Σ is a correct, HVZK, 2-special sound Σ-protocol with high min-entropy, and challenge space $\mathcal{CH}$ with $1/|\mathcal{CH}| = \mathsf{negl}(\lambda)$ for the relation

$$\mathsf{R}_{\mathsf{rnd}} := \{x = (\mathsf{pp}, \mathsf{vk}, \overline{m}), w = (\mu, c, r) \mid \mathsf{C.Commit}(\mathsf{pp}, \overline{m}; r) = (c, r) \ \wedge \ \mathsf{S.Verify}(\mathsf{vk}, \mu, c) = 1\}.$$

We also require Σ to be f-unique extraction where $f(w) = c$, i.e., f outputs c and ignores (μ, r).

Overview. Let $(\mathsf{C}, \mathsf{S}, \Sigma)$ be $\mathsf{BS}_{\mathsf{Rnd}}$-suitable. Let $\mathsf{H}_{\mathsf{par}}, \mathsf{H}_{\mathsf{M}}, \mathsf{H}_{\beta}$ be a random oracles from $\{0,1\}^*$ into $\{0,1\}^{\ell_{\mathsf{C}}}, \mathcal{C}_{\mathsf{msg}}, \mathcal{CH}$, respectively. We now describe our framework $\mathsf{BS}_{\mathsf{Rnd}}[\mathsf{C}, \mathsf{S}, \Sigma]$, or $\mathsf{BS}_{\mathsf{Rnd}}$ for short.

For key generation, the signer samples $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{S.KeyGen}(1^\lambda)$ and publishes $\mathsf{bvk} = \mathsf{vk}$ and stores $\mathsf{bsk} = \mathsf{sk}$. Further, the verification key bvk (or rather the hash functions) *implicitly* specifies the public parameter pp for C via $\mathsf{pp} = \mathsf{H}_{\mathsf{par}}(0)$.

To sign a message m, the user first commits to $\mathsf{H}_{\mathsf{M}}(m)$ via C and sends the commitment c to the signer. The signer then rerandomizes the commitment c to c' via sampling a rerandomization randomness Δr, and signs c' via S. It then sends the signature μ to the user along with Δr. The user checks by recomputing c' from c and Δr, and checks if μ is a valid signature on c'. Finally, the final blind signature is a proof π for relation $\mathsf{R}_{\mathsf{ext}}$, generated via Σ using Fiat-Shamir. Note that π is a non-interactive proof of knowledge of a signature μ on a commitment c' to $\mathsf{H}_{\mathsf{M}}(m)$.

Description. In more detail, we have the following, where we assume pp is provided to all of the algorithms for readability.

– $\mathsf{BS}_{\mathsf{Rnd}}.\mathsf{KeyGen}(1^\lambda)$: samples $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{S.KeyGen}(1^\lambda)$ and outputs verification key $\mathsf{bvk} = \mathsf{vk}$ and signing key $\mathsf{bsk} = \mathsf{sk}$.

– $\mathsf{BS}_{\mathsf{Rnd}}.\mathsf{User}(\mathsf{bvk}, m)$: sets $\overline{m} \leftarrow \mathsf{H}_{\mathsf{M}}(m)$ and outputs the commitment $c \in \mathcal{C}_{\mathsf{com}}$ generated via $(c, r) \leftarrow \mathsf{C.Commit}(\mathsf{pp}, \overline{m})$ as the first message and stores the randomness $\mathsf{st} = r \in \mathcal{C}_{\mathsf{rnd}}$.

– $\mathsf{BS}_{\mathsf{Rnd}}.\mathsf{Signer}(\mathsf{bsk}, c)$: checks if $c \in \mathcal{C}_{\mathsf{com}}$, samples a rerandomization randomness $\Delta r \leftarrow \mathcal{C}_{\mathsf{rnd}}$, rerandomizes the commitment c via $c' = \mathsf{C.RerandCom}(\mathsf{pp}, c, \Delta r)$, signs $\mu \leftarrow \mathsf{S.Sign}(\mathsf{sk}, c')$, and finally outputs the second message $\rho = (\mu, \Delta r)$.

– $\mathsf{BS}_{\mathsf{Rnd}}.\mathsf{Derive}(\mathsf{st}, \rho)$: parse $\mathsf{st} = r$, $\rho = (\mu, \Delta r)$ and checks $\Delta r \in \mathcal{C}_{\mathsf{rnd}}$. It then computes the randomized commitment $c'' = \mathsf{C.RerandCom}(\mathsf{pp}, c, \Delta r)$ and randomized randomness $r' \leftarrow \mathsf{C.RerandRand}(\mathsf{pp}, c, \overline{m}, r, \Delta r)$, and checks $\mathsf{S.Verify}(\mathsf{vk}, c'', \mu) = 1$ and $c'' = \mathsf{C.Commit}(\mathsf{pp}, \overline{m}; r')$. Finally, it outputs a signature $\sigma = \pi$, where $(\alpha, \mathsf{st}') \leftarrow \Sigma.\mathsf{Init}(x, w)$, $\beta \leftarrow \mathsf{H}_{\beta}(x, \alpha), \gamma \leftarrow \Sigma.\mathsf{Resp}(x, \mathsf{st}', \beta)$, $\pi = (\alpha, \beta, \gamma)$ with $x = (\mathsf{pp}, \mathsf{vk}, \overline{m})$, $w = (\mu, c'', r')$.

– $\mathsf{BS}_{\mathsf{Rnd}}.\mathsf{Verify}(\mathsf{bvk}, m, \sigma)$: parses $\sigma = \pi$ and $\pi = (\alpha, \beta, \gamma)$, sets $\overline{m} = \mathsf{H}_{\mathsf{M}}(m)$ and $x = (\mathsf{pp}, \mathsf{vk}, \overline{m})$, and outputs 1 if $\beta = \mathsf{H}_{\beta}(x, \alpha)$, $\Sigma.\mathsf{Verify}(x, \alpha, \beta, \gamma) = 1$, and otherwise outputs 0.

7.2.2 Correctness and Security

We prove correctness, blindness, and one-more unforgeability. The correctness of $\mathsf{BS}_{\mathsf{Rnd}}$ follows directly from the correctness of the underlying schemes $(\mathsf{C}, \mathsf{S}, \Sigma)$. Blindness follows mainly from the HVZK property of Σ and the hiding property of C. The only thing to be aware of is that the user needs to check the validity of the rerandomized commitment c'' by computing a rerandomized randomness using the randomness r used to compute the original commitment c. In order to

invoke the hiding property of C on c, we rely on the correctness of the randomization property so that the reduction no longer needs to check the validity of c''.

The main technical challenge is the proof of one-more unforgeability. Here, we argue that we can use a successful attacker on the one-more unforgeability of $\mathsf{BS}_{\mathsf{Rnd}}$ to extract some forgery σ for S. The crux is to ensure that the extracted commitment was never signed during $\mathsf{Signer}(\mathsf{bsk}, \cdot)$ queries, even if we rewind the adversary in order to extract the witness from the forgery. We ensure this with two ideas: We first use f-unique extractionof Σ to argue that the forgeries implicitly fix the to-be-extracted commitments c_i's in the first execution of the adversary. We then use the fact that the reduction adds new randomness in the rewound executions outside of the adversary's control, that is, rerandomize the commitment c submitted to the signing oracle, to argue that the c_i's cannot appear in the list of rerandomized commitments. Finally, we note that we extract from all the proofs included in the forgeries since one extraction is not enough: since we're only using a computationally binding commitment, the adversary may be breaking the binding property, in which case, the reduction needs at least two witnesses. To this end, we perform a more fine-grained analysis of the standard forking lemma. More details can be found in section 7.1.2.

Theorem 7.2 (Correctness)**.** *The scheme* $\mathsf{BS}_{\mathsf{Rnd}}$ *is correct.*

Proof. Let $\mathsf{pp} \leftarrow \mathsf{H}_{\mathsf{par}}(0)$, $m \in \{0,1\}^*$ and $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{S.KeyGen}(1^\lambda)$. The user first computes $\overline{m} \leftarrow \mathsf{H}_{\mathsf{M}}(m)$ and $(c, r) \leftarrow \mathsf{C.Commit}(\mathsf{pp}, \overline{m}; r)$ for some $r \leftarrow \mathcal{C}_{\mathsf{rnd}}$. Note that $c \in \mathcal{C}_{\mathsf{com}}$ as $\overline{m} \in \mathcal{C}_{\mathsf{msg}}$. The signer computes $\Delta r \leftarrow \mathcal{C}_{\mathsf{rnd}}$, $c' = \mathsf{C.RerandCom}(\mathsf{pp}, c, \Delta r)$, and $\mu \leftarrow \mathsf{S.Sign}(\mathsf{sk}, c')$, where $c' \in \mathcal{C}_{\mathsf{com}} \subseteq \mathcal{S}_{\mathsf{msg}}$. The user computes the randomized commitment $c'' = \mathsf{C.RerandCom}(\mathsf{pp}, c, \Delta r)$ and randomized randomness $r' \leftarrow \mathsf{C.RerandRand}(\mathsf{pp}, c, \overline{m}, r, \Delta r)$, and checks $\mathsf{S.Verify}(\mathsf{vk}, c'', \mu) = 1$ and $c'' = \mathsf{C.Commit}(\mathsf{pp}, \overline{m}; r')$. This holds since $\mathsf{RerandCom}$ is deterministic, and by the correctness of the *rerandomized* commitment (see definition 3.16) and of S. It then computes $\sigma = \pi$, where π is a proof generated using $x = (\mathsf{pp}, \mathsf{vk}, \overline{m}), w = (\sigma, c', r')$ and we have $(x, w) \in \mathsf{R}_{\mathsf{rnd}}$ due to the previous check. As $\overline{m} = \mathsf{H}_{\mathsf{M}}(m)$ and Σ is correct, we have $\Sigma.\mathsf{Verify}(x, \alpha, \beta, \gamma) = 1$ as desired. □

Theorem 7.3 (Blindness)**.** *The scheme* $\mathsf{BS}_{\mathsf{Rnd}}$ *is blind under malicious keys under the hiding and rerandomization properties of* C *and the high min-entropy and HVZK properties of* Σ*.*

Proof. Let $\mathcal{A}$ be a PPT adversary against blindness and Q_H denote the number of H_β queries. We define the following hybrids and denote by $\mathsf{Adv}^{\mathrm{H}_i}_{\mathcal{A}}(\lambda)$ the advantage of $\mathcal{A}$ in Hybrid i.

- Hybrid 0 is identical to the real game.
- Hybrid 1 is the same as Hybrid 0, except the challenger aborts if the random oracle H_β was already queried on (x_b, α_b) before generating $\pi_b = (\alpha_b, \beta_b, \gamma_b)$ for $b \in \{0,1\}$. In more detail, the challenger runs $(\alpha_b, \mathsf{st}'_b) \leftarrow \Sigma.\mathsf{Init}(x_b, w_b)$, where x_b and w_b are defined as in the protocol, samples $\beta_b \leftarrow \mathcal{CH}$, runs $\gamma_b \leftarrow \Sigma.\mathsf{Resp}(x, \alpha_b, \mathsf{st}'_b, \beta_b)$. Then if (x_b, α_b) for either $b = 0$ or 1 were queried to H_β, the challenger aborts the game. Otherwise, the challenger programs $\mathsf{H}_\beta(x_b, \alpha_b) \leftarrow \beta_b$.

  Hybrids 0 and 1 differ only when the game aborts. Due to the high min-entropy of Σ, the probability that the random oracle is already defined on input (x_b, α_b) is bounded by $Q_H \cdot \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)$ for each $b \in \{0,1\}$. Taking the union bound on $b \in \{0,1\}$, we have that $|\mathsf{Adv}^{\mathrm{H}_0}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathrm{H}_1}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$.
- Hybrid 2 is the same as Hybrid 1, except we omit the check on the rerandomized commitments. That is, when the challenger receives a second message $\rho_b = (\mu_b, \Delta r_b)$ from the adversary, it only computes the randomized commitment $c''_b = \mathsf{C.RerandCom}(\mathsf{pp}, c_b, \Delta r_b)$ and checks $\mathsf{S.Verify}(\mathsf{vk}, c''_b, \mu_b) = 1$. Recall in the previous hybrid, the challenger further computed the randomized randomness $r'_b \leftarrow \mathsf{C.RerandRand}(\mathsf{pp}, c_b, \overline{m}_b, r_b, \Delta r_b)$ and checked $c''_b = \mathsf{C.Commit}(\mathsf{pp}, \overline{m}_b; r'_b)$.

Hybrids 1 and 2 differ only when $c_b'' \neq \mathsf{C.Commit}(\mathsf{pp}, \overline{m}_b; r_b')$. However, this can never occur due to the correctness of the rerandomized commitment (see definition 3.16). Hence, $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_1}(\lambda) = \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_2}(\lambda)$.

– Hybrid 3 is the same as Hybrid 2, except the proofs π_b are simulated without the witness w_b. That is, the challenger generates a simulated transcript $(\alpha_b, \beta_b, \gamma_b)$ by sampling $\beta_b \leftarrow \mathcal{CH}$ and running $(\alpha_b, \gamma_b) \leftarrow \Sigma.\mathsf{Sim}(x_b, \beta_b)$.

  We can construct an adversary $\mathcal{B}_\Sigma$ such that $|\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_2}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_3}(\lambda)| \leq 2 \cdot \mathsf{Adv}_{\mathcal{B}_\Sigma}^{\mathrm{hvzk}}(\lambda)$. Essentially, $\mathcal{B}_\Sigma$ challenges $\mathcal{A}$ and uses the provided oracles to generate proofs and answer H_β queries. If the oracle outputs simulated proofs, the game is distributed identically to Hybrid 3. Else, the oracle outputs real proofs and behaves as in Hybrid 2.

– Hybrid 4 is the same as Hybrid 3, except the commitment-randomness pair $(c_b, r_b) \leftarrow \mathsf{C.Commit}(\mathsf{pp}, \mathsf{H}_\mathsf{M}(m_b))$ are instead computed as commitments to $\mathsf{H}_\mathsf{M}(0)$.

  It is straightforward to construct an adversary $\mathcal{B}_\mathsf{C}$ on the hiding property of C with $|\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_3}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_4}(\lambda)| \leq 2 \cdot \mathsf{Adv}_{\mathcal{B}_\mathsf{C}}^{\mathrm{hide}}(\lambda)$ by noticing that the challenger no longer requires the commitment randomness r_b to simulate $\mathcal{A}$ due to the modification we made in Hybrids 2 and 3. We note that the public parameters pp obtained from the hiding challenger can be programmed into $\mathsf{H}_{\mathsf{par}}(0)$.

In Hybrid 4, the value of coin is information-theoretically hidden from $\mathcal{A}$, as the commitments c_b and the proofs π_b are identically distributed for $b \in \{0, 1\}$. Consequently, $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_4}(\lambda) = 0$. Also, the running time of the adversaries $\mathcal{B}_\mathsf{C}$ and $\mathcal{B}_\Sigma$ are roughly that of $\mathcal{A}$. Combining the inequalities yields the statement. □

Theorem 7.4 (One-More Unforgeability)**.** *The scheme* $\mathsf{BS}_{\mathsf{Rnd}}$ *is one-more unforgeable under the binding and rerandomizability properties of* C*,* ***EUF-CMA*** *security of* S*, and the* 2*-special soundness and* f*-unique extraction properties of* Σ*.*

Proof. Let $\mathcal{A}$ be a PPT adversary against one-more unforgeability. Denote by Q_S the number of signing queries, by Q_M the number of H_M queries, and by Q_H the number of H_β queries. Recall that we model $\mathsf{H}_{\mathsf{par}}, \mathsf{H}_\mathsf{M}$, and H_β as random oracles, where we assume without loss of generality that $\mathcal{A}$ never repeats queries. In the end of the interaction with $\mathcal{A}$, that is after Q_S signing queries, $\mathcal{A}$ outputs $Q_S + 1$ forgeries $\{(m_i, \sigma_i)\}_{i \in [Q_S+1]}$. We write $\sigma_i = \pi_i$ and denote by c_i the Q_S first message queries to $\mathsf{BS}_{\mathsf{Rnd}}.\mathsf{Signer}(\mathsf{bsk}, \cdot)$ issued by $\mathcal{A}$. Note that if $\mathcal{A}$ is successful, then we have $\Sigma.\mathsf{Verify}(x_i, \alpha_i, \beta_i, \gamma_i) = 1$ and $\beta_i = \mathsf{H}_\beta(x_i, \alpha_i)$ for $\overline{m}_i = \mathsf{H}_\mathsf{M}(m_i)$, $x_i = (\mathsf{pp}, \mathsf{vk}, \overline{m}_i)$, and $\pi_i = (\alpha_i, \beta_i, \gamma_i)$. We first slightly alter the real game and remove subtle conditions to make the later proofs easier. We denote by $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_i}(\lambda)$ the advantage of $\mathcal{A}$ in Hybrid i for $i \in \{0, 1\}$.

– Hybrid 0 is identical to the real game.

– Hybrid 1 is the same as Hybrid 0, except it aborts if there is a collision in H_M or H_β, or there is some (x_i, α_i) for $i \in [Q_S + 1]$ that was never queried to H_β.

  It suffices to upper bound the abort probability. A collision in H_M (resp. H_β) happens with probability at most $Q_M^2/|\mathcal{C}_{\mathsf{msg}}|$ (resp. $Q_H^2/|\mathcal{CH}|$) (which follows for example from a union bound). Moreover, the probability that some fixed β_i of $\mathcal{A}$'s output equals to $\mathsf{H}_\beta(x_i, \alpha_i)$ is exactly $1/|\mathcal{CH}|$, if (x_i, α_i) was never queried to H_β. Thus, it follows that $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_0}(\lambda) \leq \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_1}(\lambda) + \frac{Q_M^2}{|\mathcal{C}_{\mathsf{msg}}|} + \frac{Q_H^2+1}{|\mathcal{CH}|} = \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_1}(\lambda) + \mathsf{negl}(\lambda)$.

Description of Wrapper Algorithm $\mathcal{B}$. We now present a wrapper algorithm $\mathcal{B}$ that simulates the interaction between the challenger $\mathcal{G}$ and $\mathcal{A}$ in Hybrid 1. Looking ahead we apply a generalization of the standard forking lemma on $\mathcal{B}$ to extract the witnesses from all the proof (i.e.forgery) output by $\mathcal{A}$.

Notice that $\mathcal{G}$ is deterministic once the keys $(\mathsf{vk}, \mathsf{sk})$ of the (deterministic) signature scheme S, the Q_S rerandomization randomness in $\mathcal{C}_{\mathsf{rnd}}$, and the outputs of the random oracles $\mathsf{H}_{\mathsf{par}}, \mathsf{H}_{\mathsf{M}}, \mathsf{H}_\beta$ are determined. Since $\mathsf{H}_{\mathsf{par}}$ is only used to generate the public parameter pp of the commitment scheme, we assume without loss of generality that only pp is given to $\mathcal{A}$ rather than access to $\mathsf{H}_{\mathsf{par}}$. We use coin to denote all the Q_M outputs of H_{M} and the random coins used by $\mathcal{A}$. We use $\vec{h} = (\widehat{\beta}_i, \Delta r_i)_{i\in[Q_H+Q_S]} \in (\mathcal{CH} \times \mathcal{C}_{\mathsf{rnd}})^{Q_H+Q_S}$ to explicitly denote the list that will be used to simulate the outputs of H_β and rerandomziation randomness sampled by $\mathcal{G}$. Here, we note that $\vec{h}$ is deliberately defined redundantly since $\mathcal{G}$ only needs Q_H hash outputs and Q_S rerandomziation randomness, rather than $Q_H + Q_S$ of them each. We also use $\widehat{\beta} \in \mathcal{CH}$ to denote the output of H_β to distinguish between the hash value β included in $\mathcal{A}$'s forgeries. We then define $\mathcal{B}$ as an algorithm that has oracle access to $\mathsf{S.Sign}(\mathsf{sk}, \cdot)$ as follows:

$\mathcal{B}^{\mathsf{S.Sign}(\mathsf{sk},\cdot)}(\mathsf{pp}, \mathsf{vk}, \vec{h}; \mathsf{coin})$: On input pp, vk, and $\vec{h} \in (\mathcal{CH} \times \mathcal{C}_{\mathsf{rnd}})^{Q_H+Q_S}$, $\mathcal{B}$ simulates the interaction between the challenger $\mathcal{G}$ and $\mathcal{A}$ in Hybrid 1. $\mathcal{B}$ invokes $\mathcal{A}$ on the randomness included in coin and simulates $\mathcal{G}$, where it runs the same code as $\mathcal{G}$ except for the following differences:

- It uses the provided pp and vk rather than generating it on its own;
- All Q_M random oracle queries to H_{M} are answered using the hash values include in coin;
- On the i-th ($i \in [Q_H]$) random oracle query to H_β, it retrieves an unused $(\widehat{\beta}_k, \Delta r_k)$ with the smallest index $k \in [Q_H + Q_S]$ and outputs $\widehat{\beta}_k$ and discards Δr_k;
- On the i-th ($i \in [Q_S]$) first message $c_i \in \mathcal{C}_{\mathsf{com}}$ from $\mathcal{A}$, it retrieves an unused $(\widehat{\beta}_k, \Delta r_k)$ with the smallest index $k \in [Q_H + Q_S]$ and discards $\widehat{\beta}_k$. It then computes $c'_i = \mathsf{C.RerandCom}(\mathsf{pp}, c_i, \Delta r_k)$, queries the signing oracle on c'_i, obtains $\mu_i \leftarrow \mathsf{S.Sign}(\mathsf{sk}, c'_i)$, and returns the second message $\rho_i = (\mu_i, \Delta r_k)$.

At the end of the game when $\mathcal{A}$ outputs the forgeries, $\mathcal{B}$ checks if the forgeries are valid and the added condition in Hybrid 1. If the check does not pass, then $\mathcal{B}$ outputs $((0)_{i\in[Q_S+1]}, \perp)$, i.e., $Q_S + 1$ zeros followed by a $\perp$. Otherwise, $\mathcal{B}$ finds the indices $I_i \in [Q_H + Q_S]$ such that $\mathsf{H}_\beta(x_i, \alpha_i) = \beta_i = \widehat{\beta}_{I_i}$ for $i \in [Q_S + 1]$, which are guaranteed to exist uniquely due to the modification we made in Hybrid 1. It then sets $\Lambda = (x_i, \alpha_i, \beta_i, \gamma_i)_{i\in[Q_S+1]}$ and outputs $((I_i)_{i\in[Q_S+1]}, \Lambda)$. It can be checked that $\mathcal{B}$ perfectly simulates the view of the challenger $\mathcal{G}$ in Hybrid 1. Therefore, $\mathcal{B}$ outputs $\Lambda \neq \perp$ with probability $\mathsf{Adv}^{\mathrm{H}_1}_{\mathcal{A}}(\lambda)$.

Description of Forking Algorithm $\mathsf{F}_{\mathcal{B}}$. We now define a generalization of the standard forking algorithm F so that F keeps on rewinding $\mathcal{B}$ until some condition is satisfied. Concretely, F takes as input $(\mathsf{pp}, \mathsf{vk})$, has oracle access to $\mathsf{S.Sign}(\mathsf{sk}, \cdot)$, and invokes $\mathcal{B}$ internally as depicted in algorithm 12, where the number of repetition T is defined below.

We show that if $\mathcal{A}$ succeeds in breaking one-more unforgeability in Hybrid 1 with non-negligible probability, then we can set a specific number of repetition T so that the forking algorithm $\mathsf{F}_{\mathcal{B}}$ terminates in polynomial time and succeeds in outputting a non-$\perp$ with non-negligible probability. Formally, we have the following lemma.

Lemma 7.5. *Let $\epsilon = \mathsf{Adv}^{H_1}_{\mathcal{A}}(\lambda)$. Then, if we set $T = \left(\frac{\epsilon}{(Q_H+Q_S)(Q_S+2)^2}\right)^{-1} \cdot \log(2Q_S + 2)$, $\mathsf{F}_{\mathcal{B}}$ outputs a non-$\perp$ with probability at least $\frac{\epsilon}{2(Q_S+2)^2}$.*

Algorithm 12 Description of the forking algorithm $\mathsf{F}_{\mathcal{B}}^{\mathsf{S.Sign}(\mathsf{sk},\cdot)}(\mathsf{pp},\mathsf{vk})$

Pick coin for $\mathcal{B}$ at random.
$\vec{h} \leftarrow (\mathcal{CH} \times \mathcal{C}_{\mathsf{rnd}})^{Q_H+Q_S}$
$((I_i)_{i\in[Q_S+1]}, \Lambda) \leftarrow \mathcal{B}^{\mathsf{S.Sign}(\mathsf{sk},\cdot)}(\mathsf{pp},\mathsf{vk},\vec{h};\mathsf{coin})$
if $\Lambda = \bot$ **then**
  return $\bot$ ▷ Return fail.
$D := ()$ ▷ Prepare empty list.
for $j \in [Q_S+1]$ **do**
  $(c, \mathsf{flag}) := (1, \bot)$
  while $c \in [T] \wedge \neg\mathsf{flag}$ **do**
    $\vec{h}^{(c)}_{j,\geq I_j} \leftarrow (\mathcal{CH} \times \mathcal{C}_{\mathsf{rnd}})^{Q_H+Q_S-I_j+1}$
    $\vec{h}^{(c)}_j := \vec{h}_{<I_j} \| \vec{h}^{(c)}_{j,\geq I_j}$
    $((I^{(c)}_{j,i})_{i\in[Q_S+1]}, \Lambda^{(c)}_j) \leftarrow \mathcal{B}^{\mathsf{S.Sign}(\mathsf{sk},\cdot)}(\mathsf{pp},\mathsf{vk},\vec{h}^{(c)}_j;\mathsf{coin})$
    if $I^{(c)}_{j,j} = I_j$ **then**
      $D = D \cup (j, I_j, \Lambda^{(c)}_j)$
      $\mathsf{flag} = \top$ ▷ Break from while loop.
    $c = c+1$
if $|D| < Q_S+1$ **then** ▷ Check if $\mathcal{B}$ succeeds in all Q_S+1 run.
  return $\bot$ ▷ Return fail.
return (Λ, D)

In particular, if ϵ is non-negligible, then $T = \mathsf{poly}(\lambda)$. Moreover, the running time of $\mathsf{F}_{\mathcal{B}}$ is at most (roughly) a factor $T \cdot (Q_S+1)+1$ more of $\mathcal{B}$ (or equivalently $\mathcal{A}$), so $\mathsf{F}_{\mathcal{B}}$ runs in polynomial time.

Proof. Assume $\mathcal{B}$ outputs a valid $\Lambda = (x_i, \alpha_i, \beta_i, \gamma_i)_{i\in[Q_S+1]}$ in the first execution and denote this event as E. For $i \in [Q_S+1]$, we denote the tuple $(x_i, \alpha_i, \beta_i, \gamma_i)$ as the i-th *forgery*. For any $(i,k) \in [Q_S+1] \times [Q_H+Q_S]$, we denote $\mathsf{E}_{i,k}$ as the event that forgery is associated to the k-th hash query, i.e., the k-th entry of $\vec{h} \in (\mathcal{CH} \times \mathcal{C}_{\mathsf{rnd}})^{Q_H+Q_S}$ includes β_i. Here, note that $\forall i \in [Q_S+1]$, we have $\sum_{k\in[Q_H+Q_S]} \Pr[\mathsf{E}_{i,k}] = 1$. We define the set P_i as

$$P_i = \left\{ k \;\middle|\; \Pr[\mathsf{E}_{i,k} \mid \mathsf{E}] \geq \frac{1}{(Q_H+Q_S)(Q_S+2)} \right\},$$

where for any $k \in P_i$, we have $\Pr[\mathsf{E}_{i,k}] \geq \frac{\epsilon}{(Q_H+Q_S)(Q_S+2)}$. Let us define $\mathsf{E}^{\mathsf{good}}_i = \bigvee_{k\in P_i} \mathsf{E}_{i,k}$. Then, we have $\Pr\left[\mathsf{E}^{\mathsf{good}}_i \;\middle|\; \mathsf{E}\right] \geq \frac{Q_S+1}{Q_S+2}$, since there are at most (Q_H+Q_S) possible values of k's not in P_i and they can only account to a probability at most $(Q_H+Q_S) \times \frac{1}{(Q_H+Q_S)(Q_S+2)} = \frac{1}{Q_S+2}$.

Next, for any $(i,k) \in [Q_S+1] \times P_i$, let us define $X_{i,k} = R_{\mathsf{coin}} \times (\mathcal{CH} \times \mathcal{C}_{\mathsf{rnd}})^{k-1}$ and $Y_{i,k} = (\mathcal{CH} \times \mathcal{C}_{\mathsf{rnd}})^{Q_H+Q_S-k+1}$, where R_{coin} denotes the randomness space of coin. Here, note that $(\mathsf{x}_i, \vec{h}_{\geq k}) \in X_{i,k} \times Y_{i,k}$ can be parsed appropriately to be $(\mathsf{coin}, \vec{h})$, and defines all the inputs of $\mathcal{B}$, where we assume a fixed $(\mathsf{pp},\mathsf{vk})$. We further define $A_{i,k} \subseteq X_{i,k} \times Y_{i,k}$ to be the set of inputs that triggers event $\mathsf{E}_{i,k}$. Then using the splitting lemma (cf. lemma 3.3) with $\alpha = \frac{Q_S+1}{Q_S+2} \cdot \frac{\epsilon}{(Q_H+Q_S)(Q_S+2)}$, there exists a set $B_{i,k} \subset X_{i,k} \times Y_{i,k}$ such that

$$B_{i,k} = \left\{ (\mathsf{x}_i, \vec{h}_{\geq k}) \in X_{i,k} \times Y_{i,k} \;\middle|\; \Pr_{\vec{h}'_{\geq k} \leftarrow Y_{i,k}} \left[(\mathsf{x}_i, \vec{h}'_{\geq k}) \in A_{i,k} \right] \geq \frac{\epsilon}{(Q_H+Q_S)(Q_S+2)^2} \right\}, \quad (7.1)$$

and

$$\Pr_{(\mathsf{x}_i,\vec{h'}_{\geq k})\leftarrow X_{i,k}\times Y_{i,k}}\left[(\mathsf{x}_i,\vec{h}_{\geq k})\in B_{i,k}\;\middle|\;(\mathsf{x},\vec{h}_{\geq k})\in A_{i,k}\right]\geq\frac{Q_S+1}{Q_S+2}. \tag{7.2}$$

We are now ready to evaluate the success probability of the forking algorithm $\mathsf{F}_{\mathcal{B}}$. With probability ϵ, $\mathcal{B}$ outputs $((I_i)_{i\in[Q_S+1]},\Lambda)$ in the first execution on input $(\mathsf{coin},\vec{h})\in R_{\mathsf{coin}}\times(\mathcal{CH}\times\mathcal{C}_{\mathsf{rnd}})^{Q_H+Q_S}$. Then the probability that event $\mathsf{E}_i^{\mathsf{good}}$ occurrs for all $i\in[Q_S+1]$ is at least

$$\Pr\left[\forall i\in[Q_S+1],\;\mathsf{E}_i^{\mathsf{good}}\middle|\mathsf{E}\right]\geq 1-\sum_{i\in[Q_S+1]}\Pr\left[\neg\mathsf{E}_i^{\mathsf{good}}\middle|\mathsf{E}\right]\geq\frac{1}{Q_S+2},$$

where the first inequality follows from the union bound and the second inequality follows from $\Pr\left[\mathsf{E}_i^{\mathsf{good}}\;\middle|\;\mathsf{E}\right]\geq\frac{Q_S+1}{Q_S+2}$.

Then, from eq. (7.2) and following the same union bound argument, $\mathsf{F}_{\mathcal{B}}$ samples a good input such that $(\mathsf{coin},\vec{h})\in B_{i,I_i}$ for all $i\in[Q_S+1]$ conditioned on $\mathsf{E}_i^{\mathsf{good}}$ for all $i\in[Q_S+1]$ with probability at least $\frac{1}{(Q_S+2)}$. Therefore, by eq. (7.1), if $\mathsf{F}_{\mathcal{B}}$ resamples $\vec{h}_{i,\geq I_i}\in Y_{i,I_i}=(\mathcal{CH}\times\mathcal{C}_{\mathsf{rnd}})^{Q_H+Q_S-I_i+1}$ conditioned on the set B_{i,I_i}, $\mathcal{B}$ succeeds on input $(\mathsf{coin},\vec{h}_{i,<I_i}\|\vec{h}_{i,\geq I_i})$ with probability at least $\frac{\epsilon}{(Q_H+Q_S)(Q_S+2)^2}$. Conditioning on sampling an input $(\mathsf{coin},\vec{h})\in B_{i,I_i}$ for all $i\in[Q_S+1]$ and noting the independence of each rewinding, the probability that $\mathcal{B}$ succeeds in all j-th rewinding for $j\in[Q_S+1]$ is at least

$$\begin{aligned}\left(1-\left(1-\frac{\epsilon}{(Q_H+Q_S)(Q_S+2)^2}\right)^T\right)^{Q_S+1}&\geq\left(1-\frac{1}{e^{\log(2Q_S+2)}}\right)^{Q_S+1}\\&=\left(1-\frac{1}{2(Q_S+1)}\right)^{Q_S+1}\geq\frac{1}{2}.\end{aligned}$$

Collecting all the bounds, we conclude that $\mathsf{F}_{\mathcal{B}}$ succeeds with probability at least $\frac{\epsilon}{2(Q_S+2)^2}$ as desired. Moreover, the running time of $\mathsf{F}_{\mathcal{B}}$ is roughly the same as running $\mathcal{B}$ for at most $T\cdot(Q_S+1)+1$ times, where the runtime of $\mathcal{B}$ is roughly the same as the runtime of $\mathcal{A}$.

□

Using $\mathsf{F}_{\mathcal{B}}$ to Break Binding of C or `EUF-CMA` of S. We are now ready to finish the proof. Assume $\epsilon=\mathsf{Adv}^{\mathrm{H}_1}_{\mathcal{A}}(\lambda)$ is non-negligible. We use $\mathsf{F}_{\mathcal{B}}$ to extract the witnesses from the proofs output by $\mathcal{A}$ with non-negligible probability and show that such witnesses can be used to break either the binding of C or the `EUF-CMA` security of S. Thus establishing that $\epsilon=\mathsf{negl}(\lambda)$ by contradiction.

We define adversary $\mathcal{A}_{\mathsf{C},\mathsf{S}}$ on both the binding property of C and the `EUF-CMA` property of S as follows. Initially, $\mathcal{A}_{\mathsf{C},\mathsf{S}}$ obtains pp from the binding challenger. Further, she receives vk and oracle access to a signing oracle $\mathsf{S.Sign}(\mathsf{sk},\cdot)$ from the `EUF-CMA` challenger. Then, she runs the forking algorithm $R\leftarrow\mathsf{F}_{\mathcal{B}}^{\mathsf{S.Sign}(\mathsf{sk},\cdot)}(\mathsf{pp},\mathsf{vk})$. She checks $R\neq\perp$, and parses $R=(\Lambda,D)$, where $\Lambda=(x_i,\alpha_i,\beta_i,\gamma_i)_{i\in[Q_S+1]}$ and $D=(j,I_j,\Lambda_j)_{j\in[Q_S+1]}$. Due to lemma 7.5, $\mathsf{F}_{\mathcal{B}}$ runs in polynomial time and has non-negligible success probability. Below, we describe the second part of $\mathcal{A}_{\mathsf{C},\mathsf{S}}$ and analyze its success probability conditioned on $\mathsf{F}_{\mathcal{B}}$ succeeding. (If $R=\perp$, then $\mathcal{A}_{\mathsf{C},\mathsf{S}}$ outputs $\perp$ and aborts.)

For $j\in[Q_S+1]$, we denote by $(x'_j,\alpha'_j,\beta'_j,\gamma'_j)$ the j-th element of the tuple Λ_j. Moreover, note that the same coin and values $(\widehat{\beta}_1,\Delta r_1),\ldots(\widehat{\beta}_{I_j-1},\Delta r_{I_j-1})$ are used for the initial run of $\mathcal{B}$ and the run of $\mathcal{B}$ where $\mathcal{B}$ outputs Λ_j. Thus, we have for all $j\in[Q_S+1]$ that $(x_i,\alpha_i)=(x'_i,\alpha'_i)$. Moreover, we have $\widehat{\beta}_{I_j}\neq\widehat{\beta}_{j,I_j}$, or equivalently $\beta_j\neq\beta'_j$ for all $j\in[Q_S+1]$ with probability at least $1-\frac{Q_S+1}{|\mathcal{CH}|}=1-\mathsf{negl}(\lambda)$ since each hash outputs are sampled uniformly and independently at

random. This allows $\mathcal{A}_{\mathsf{C},\mathsf{S}}$ to invoke 2-special soundness of Σ with overwhelming probability. For all $i \in [Q_S + 1]$, she runs Ext on $(x_i, (\alpha_i, \beta_i, \gamma_i), (\alpha_i, \beta_i', \gamma_i'))$ to extract a witness $w_i = (\mu_i, c_i, r_i)$ such that $\mathsf{C.Commit}(\mathsf{pp}, \overline{m}_i; r_i) = c_i \wedge \mathsf{S.Verify}(\mathsf{vk}, \mu_i, c_i) = 1$, where $x_i = (\mathsf{pp}, \mathsf{vk}, \overline{m}_i)$.

If there exists distinct $i, j \in [Q_S + 1]$ with $c_i = c_j$, $\mathcal{A}_{\mathsf{C},\mathsf{S}}$ sends $(\overline{m}_i, \overline{m}_j, r_i, r_j)$ to the binding security game of C. Note that due to the check in Hybrid 1, the $(\overline{m}_i)_{i\in[Q_S+1]}$ are pairwise distinct, in particular $\overline{m}_i \neq \overline{m}_j$ but $\mathsf{C.Commit}(\mathsf{pp}, \overline{m}_i; r_i) = \mathsf{C.Commit}(\mathsf{pp}, \overline{m}_j; r_j)$. However, due to the binding property of C, this can happen with only negligible probability. Thus, the extracted commitments $(c_i)_{i\in[Q_S+1]}$ must be distinct with overwhelming probability.

In such a case, there must be at least one $i^* \in [Q_S + 1]$ such that c_i^* was never queried to the signing oracle $\mathsf{S.Sign}(\mathsf{sk}, \cdot)$ *in the first execution* of $\mathcal{B}$ or equivalently of $\mathcal{A}$. This is because due to the one-more unforgeability game, $\mathcal{A}$ only queries the signing oracle Q_S times. Thus, $\mathcal{A}_{\mathsf{C},\mathsf{S}}$ finds such i^* with the smallest index and outputs (μ_{i^*}, c_i^*) as a forgery against the $\mathtt{EUF\text{-}CMA}$ security of S.

It remains to show that what $\mathcal{A}_{\mathsf{C},\mathsf{S}}$ output is a valid forgery, i.e., $\mathcal{B}$ never queried c_i^* to the signing oracle in any of the *rewound executions*. To argue this, we first show that all the extracted commitments $(c_i)_{i\in[Q_S+1]}$ are fixed *after the first execution ends* due to f-unique extraction (cf. definition 3.26). For any $(x_i, \tau_i := (\alpha_i, \beta_i, \gamma_i)) \in \Lambda$ defined in the first execution of $\mathcal{B}$, conditioning on $\mathsf{F}_{\mathcal{B}}$ succeeding, another valid transcript $(x_i, \tau_i' := (\alpha_i, \beta_i', \gamma_i')) \in \Lambda_i$ with $\beta_i \neq \beta_i'$ is guaranteed to exist with overwhelming probability. Due to f-unique extraction, for any such valid transcript the value $f(\mathsf{Ext}(x_i, \tau_i, \tau_i')) = c_i$ is identical, where recall f simply outputs the commitment included in the witness. Put differently, conditioning on $\mathsf{F}_{\mathcal{B}}$ succeeding, (x_i, τ_i) uniquely defines c_i with overwhelming probability. We emphasize that c_i does not need to be efficiently computable given only (x_i, τ_i); we only care if c_i is determined by (x_i, τ_i) in a statistical sense.

Now, assume $\mathcal{B}$ queried c_i^* to the signing oracle in one of the rewound executions. This means $\mathcal{A}$ outputs some c^* to $\mathcal{B}$ (or equivalently the simulated challenger $\mathcal{G}$ in Hybrid 1) and $\mathcal{B}$ computed $c_i^* = \mathsf{C.RerandCom}(\mathsf{pp}, c^*, \Delta r^*)$, where Δr^* is a fresh randomness sampled by $\mathsf{F}_{\mathcal{B}}$ to be used in the rewound execution. However, this cannot happen with all but negligible probability due to the rerandomizability of C since we have established above that Δr^* is sampled independently from c_i^*. Since there are at most $T \cdot (Q_S + 1)$ rewound executions, the probability that $\mathcal{B}$ queries c_i^* to the signing oracle during in one of the rewound execution is bounded by $T \cdot (Q_S + 1) \cdot \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)$, where we use $T = \mathsf{poly}(\lambda)$ due to lemma 7.5.

Thus, with overwhelming probability, what $\mathcal{A}_{\mathsf{C},\mathsf{S}}$ output is a valid forgery against the $\mathtt{EUF\text{-}CMA}$ security of S. However, due to the hardness of $\mathtt{EUF\text{-}CMA}$ security of S, this cannot happen with all but negligible probability. Combining all the arguments, we conclude that $\epsilon = \mathsf{Adv}^{\mathrm{H}_1}_{\mathcal{A}}(\lambda)$ is negligible. This completes the proof.

□

Remark 7.6 (Removing the Rerandomizability Property)**.** As briefly noted in our technical overview, an alternative approach to using rerandomizable commitment is to let the signer (i.e., $\mathsf{BS}_{\mathsf{Rnd}}.\mathsf{Signer}$) sample a random string rand and run $\mu \leftarrow \mathsf{S.Sign}(\mathsf{sk}, c \| \mathsf{rand})$ instead of $\mu \leftarrow \mathsf{S.Sign}(\mathsf{sk}, c')$, where $c' = \mathsf{C.RerandCom}(\mathsf{pp}, c, \Delta r)$ is the rerandomized commitment. The signer then sends $\rho = (\mu, \mathsf{rand})$ as the second message instead of $\rho = (\mu, \Delta r)$. By observing that rand has an identical effect as Δr in the security proof, it can be checked that the same proof can be used to show blindness and one-more unforgeability of this modified protocol. While this approach works for any commitment scheme, we chose not to since it requires a slightly larger NIZK proof due to the enlarged signing space of the underlying signature scheme S.

7.3 Instantiation of the Generic Construction

In this section, we instantiate the $\mathsf{BS}_{\mathsf{Rnd}}$-suitable schemes $(\mathsf{C}, \mathsf{S}, \Sigma)$ required for the construction of $\mathsf{BS}_{\mathsf{Rnd}}$ in section 7.2.

Commitments

We instantiate the commitment scheme C with Pedersen commitments. We also use a variant of ElGamal commitments in our instantiation of Σ.

Rerandomizable Commitment Scheme. We recall Pedersen commitments C_{Ped} [Ped92] with public parameters $\mathsf{pp} \in \mathbb{G}$ over a group $\mathbb{G}$ with generator g:

- $\mathsf{C}_{\text{Ped}}.\mathsf{Commit}(\mathsf{pp}, m; r)$: outputs commitment $c = g^m \cdot \mathsf{pp}^r$ and opening $r \leftarrow \mathbb{Z}_p$,

Pedersen commitments are correct, computationally binding under the DLOG assumption and statistically hiding. Further, note that it is rerandomizable via:

- $\mathsf{C}_{\text{Ped}}.\mathsf{RerandCom}(\mathsf{pp}, c, \Delta r)$: outputs $c' \leftarrow c \cdot \mathsf{pp}^{\Delta r}$
- $\mathsf{C}_{\text{Ped}}.\mathsf{RerandRand}(\mathsf{pp}, c, m, r, \Delta r)$: outputs $r' \leftarrow \Delta r + r$

ElGamal. We recall ElGamal commitments C_{EG} [ElG84] with public parameters $\mathsf{pp} \in \mathbb{G}$ over a group $\mathbb{G}$ with generator g:

- $\mathsf{C}_{\text{EG}}.\mathsf{Commit}(\mathsf{pp}, m; r)$: outputs commitment $c = (c_1, c_2) = (m \cdot \mathsf{pp}^r, g^r)$ and opening $r \leftarrow \mathbb{Z}_p$,

ElGamal commitments are correct, perfectly binding and computationally hiding under DDH. Also, it is extractable with the following simulator and extractor:

- $\mathsf{C}_{\text{EG}}.\mathsf{SimSetup}(1^\lambda)$: samples $x \leftarrow \mathbb{Z}_p$, and outputs public parameters $\mathsf{pp} = g^x$ and trapdoor $\mathsf{td} = x$.
- $\mathsf{C}_{\text{EG}}.\mathsf{Ext}(\mathsf{pp}, \mathsf{td}, c)$: extracts $m \leftarrow c_1 / c_2^{\mathsf{td}}$.

Note that the second part $c_2 = g^r$ of the commitment can be reused across multiple commitments, if each commitment uses different public parameters, i.e.$\mathsf{pp}_i \leftarrow \mathbb{G}$, as below.

Remark 7.7 (ElGamal with Message Space $\mathbb{G}^n$)**.** Let $n \in \mathbb{N}$ and $\mathsf{pp}_i \leftarrow \mathbb{G}$ for $i \in [n]$. We can commit to messages $(m_1, \ldots, m_n) \in \mathbb{G}^n$ via $R \leftarrow g^r$ and $c_i = m_i \mathsf{pp}_i^r$. The (non-compact) vector commitment $(R, c_1, \ldots, c_2)$ remains correct, perfectly binding with message space $\mathbb{G}^n$ and hiding under DDH [BBM00]. Extraction is possible as before with the trapdoor $\mathsf{td} = (x_i)_{i \in [n]}$ with $g^{x_i} = \mathsf{pp}_i$ via $m_i \leftarrow c_i / R^{x_i}$.

Remark 7.8 (ElGamal with Message Space $\mathbb{Z}_p$)**.** We can commit to a message $m \in \mathbb{Z}_p$ via $c = (g^m \mathsf{pp}_i^r, g^r)$. This variant remains perfectly binding with message space $\mathbb{Z}_p$ and hiding under DDH. Note that this variant is not extractable for message space $\mathbb{Z}_p$. If the message m is of polynomial size, i.e.$|m| \leq B$ for some bound $B = \mathsf{poly}(\lambda)$, extraction is possible via $g \leftarrow c_1 / c_2^{\mathsf{td}}$ and then calculating the discrete logarithm m of g. This can be done in polynomial time by trying all $m' \in \mathbb{Z}_p$ with $|m'| \leq B$ with brute-force.

Signature Scheme

For the signature scheme S, we use a variant of the Kiltz-Pan-Wee (KPW) structure-preserving signature (SPS) scheme [KPW15] in the asymmetric pairing setting. The message space of KPW is $\mathbb{G}_1^\ell$, where $\ell \in \mathbb{N}$ is the message length.

Any SPS must contain at least three group elements, and at least one in each $\mathbb{G}_2$ and in $\mathbb{G}_1$ [AGHO11]. But as the bit size of elements in $\mathbb{G}_2$ is larger than the bit size of elements in $\mathbb{G}_1$ and $\mathbb{Z}_p$, removing elements in $\mathbb{G}_2$ in the signature is desirable. For $\mathsf{BS}_{\mathsf{Rnd}}$, we do not require the

full structure-preserving property of KPW, as we can design efficient Σ-protocols for signature verification, even if the signature contains elements in $\mathbb{Z}_p$.

Indeed, KPW signatures contain an element σ_4 in $\mathbb{G}_2$. We observe that we can safely replace σ_4 with its discrete logarithm τ. Further, we can omit two more elements in $\mathbb{G}_1$ for free, as they can be recomputed via τ and the remaining signature elements.

Our optimized variant is given below.

- $\mathsf{S}_{\mathrm{KPW}}.\mathsf{KeyGen}(1^\lambda)$: samples $a,\ b \leftarrow \mathbb{Z}_p$ and sets $\boldsymbol{A} \leftarrow (1,a)^\top$ and $\boldsymbol{B} \leftarrow (1,b)^\top$. It samples $\boldsymbol{K} \leftarrow \mathbb{Z}_p^{(\ell+1)\times 2}$, $\boldsymbol{K}_0,\ \boldsymbol{K}_1 \leftarrow \mathbb{Z}_p^{2\times 2}$ and sets $\boldsymbol{C} \leftarrow \boldsymbol{K}\boldsymbol{A}$. It sets $(\boldsymbol{C}_0, \boldsymbol{C}_1) \leftarrow (\boldsymbol{K}_0\boldsymbol{A}, \boldsymbol{K}_1\boldsymbol{A})$, $(\boldsymbol{P}_0, \boldsymbol{P}_1) \leftarrow (\boldsymbol{B}^\top\boldsymbol{K}_0, \boldsymbol{B}^\top\boldsymbol{K}_1)$, $\mathsf{vk} \leftarrow ([\boldsymbol{C}_0]_2, [\boldsymbol{C}_1]_2, [\boldsymbol{C}]_2, [\boldsymbol{A}]_2)$, and $\mathsf{sk} \leftarrow (\boldsymbol{K}, [\boldsymbol{P}_0]_1, [\boldsymbol{P}_1]_1, [\boldsymbol{B}]_1)$. It outputs $(\mathsf{vk}, \mathsf{sk})$.

- $\mathsf{S}_{\mathrm{KPW}}.\mathsf{Sign}(\mathsf{sk}, [\boldsymbol{m}]_1)$: samples $r, \tau \leftarrow \mathbb{Z}_p$ and sets $\sigma_1 \leftarrow [(1, \boldsymbol{m}^\top)\boldsymbol{K} + r(\boldsymbol{P}_0 + \tau\boldsymbol{P}_1)]_1 \in \mathbb{G}_1^2$, $\sigma_2 \leftarrow [r\boldsymbol{B}^\top]_1 \in \mathbb{G}_1^2$, and $\sigma_3 \leftarrow \tau \in \mathbb{Z}_p$. It outputs $(\sigma_1, \sigma_2, \sigma_3)$.

- $\mathsf{S}_{\mathrm{KPW}}.\mathsf{Verify}(\mathsf{vk}, [\boldsymbol{m}]_1, (\sigma_1, \sigma_2, \sigma_3))$: checks $e(\sigma_1, [\boldsymbol{A}]_2) = e([(1, \boldsymbol{m}^\top)]_1, [\boldsymbol{C}]_2) \cdot e(\sigma_2, [\boldsymbol{C}_0]_2 \cdot \tau[\boldsymbol{C}_1]_2)$.

We show that $\mathsf{S}_{\mathrm{KPW}}$ is $\mathtt{EUF\text{-}CMA}$ under the SXDH assumption in Theorem 7.9. The proof relies on the computational core lemma of [KW15]. $\mathsf{S}_{\mathrm{KPW}}$ can be made deterministic via a pseudorandom function.

Theorem 7.9. *The scheme $\mathsf{S}_{\mathrm{KPW}}$ is correct and $\mathtt{EUF\text{-}CMA}$ secure under the SXDH assumption.*

Proof. Correctness follows from a simple calculation. For $\mathtt{EUF\text{-}CMA}$ security, we first recall the computational core lemma of [KW15].

Lemma 7.10. *For all adversaries $\mathcal{A}$, there exists an adversary $\mathcal{B}$ with almost the same running time satisfying*

$$\Pr\left[b = b' : \begin{array}{l} a,\ b \leftarrow \mathbb{Z}_p,\ \boldsymbol{A} \leftarrow (1,a)^\top,\ \boldsymbol{B} \leftarrow (1,b)^\top \\ \boldsymbol{K}_0,\ \boldsymbol{K}_1 \leftarrow \mathbb{Z}_p^{2\times 2} \\ (\boldsymbol{P}_0, \boldsymbol{P}_1) \leftarrow (\boldsymbol{B}^\top\boldsymbol{K}_0, \boldsymbol{B}^\top\boldsymbol{K}_1) \\ \mathsf{pk} \leftarrow ([\boldsymbol{P}_0]_1, [\boldsymbol{P}_1]_1, [\boldsymbol{B}]_1, \boldsymbol{K}_0\boldsymbol{A}, \boldsymbol{K}_1\boldsymbol{A}, \boldsymbol{A}) \\ b \leftarrow \{0,1\},\ b' \leftarrow \mathcal{A}^{\mathcal{O}_b, \mathcal{O}^*}(\mathsf{pk}) \end{array}\right] \leq \frac{1}{2} + 2Q\mathsf{Adv}^{\mathrm{sxdh}}_{\mathcal{B}}(\lambda) + Q/p$$

where Q is the number of queries $\mathcal{A}$ makes to $\mathcal{O}_b$, $\mathcal{A}$ is not allowed to make the same query to both $\mathcal{O}_b$ and $\mathcal{O}^*$, and

- *$\mathcal{O}_b(\tau)$: on a query $\tau \in \mathbb{Z}_p$ returns $([b\mu\boldsymbol{a}^\perp + r(\boldsymbol{P}_0 + \tau\boldsymbol{P}_1)]_1, [r\boldsymbol{B}^\top]_1) \in (\mathbb{G}_1^2)^2$ with $\mu \leftarrow \mathbb{Z}_p$, $r \leftarrow \mathbb{Z}_p$, where $\boldsymbol{a}^\perp \in \mathbb{Z}_p^2$ is a non-zero vector that satisfies $\boldsymbol{a}^\perp\boldsymbol{A} = \boldsymbol{0}$,*

- *$\mathcal{O}^*(\tau^*)$: on a query $\tau^* \in \mathbb{Z}_p$ returns $[\boldsymbol{K}_0 + \tau^*\boldsymbol{K}_1]_2$ with only a single query to $\mathcal{O}^*$ allowed for $\mathcal{A}$,*

Now let $\mathcal{A}$ be an adversary against the $\mathtt{EUF\text{-}CMA}$ security of the scheme $\mathsf{S}_{\mathrm{KPW}}$. After Q signing queries, $\mathcal{A}$ outputs a forgery $([\boldsymbol{m}^*]_1, \sigma^*)$. We follow the proof structure of [KPW15], but adapt it to our optimizations using lemma 7.10. We define the following hybrids and denote by $\mathsf{Adv}^{\mathrm{H}_i}_{\mathcal{A}}(\lambda)$ the advantage of $\mathcal{A}$ in Hybrid i.

- Hybrid 0 is the real game.

– Hybrid 1 is the same as Hybrid 0, except the verification check (to verify the final output of $\mathcal{A}$), is replaced with $\mathsf{S}_{\mathrm{KPW}}.\mathsf{Verify}^*$, defined as follows. On input of verification key vk, message $[\boldsymbol{m}]_1$, and signature $\sigma = (\sigma_1, \sigma_2, \sigma_3)$, $\mathsf{S}_{\mathrm{KPW}}.\mathsf{Verify}^*$ checks $e(\sigma_1, [\mathbf{1}]_2) = e([(1, \boldsymbol{m}^\top)]_1, [\boldsymbol{K}]_2) \cdot e(\sigma_2, [\boldsymbol{K}_0]_2 \cdot \sigma_3[\boldsymbol{K}_1]_2)$.

  Note that for any $([\boldsymbol{m}]_1, \sigma)$, we have

$$\begin{aligned} & e(\sigma_1, [\boldsymbol{A}]_2) = e([(1, \boldsymbol{m}^\top)]_1, [\boldsymbol{C}]_2) \cdot e(\sigma_2, [\boldsymbol{C}_0]_2 \cdot \tau[\boldsymbol{C}_1]_2) \\ \iff & e(\sigma_1, [\boldsymbol{A}]_2) = e([(1, \boldsymbol{m}^\top)]_1, [\boldsymbol{K}\boldsymbol{A}]_2) \cdot e(\sigma_2, [\boldsymbol{K}_0\boldsymbol{A}]_2 \cdot \tau[\boldsymbol{K}_1\boldsymbol{A}]_2) \\ \impliedby & e(\sigma_1, [1]_2) = e([(1, \boldsymbol{m}^\top)]_1, [\boldsymbol{K}]_2) \cdot e(\sigma_2, [\boldsymbol{K}_0 + \tau\boldsymbol{K}_1]_2) \end{aligned}$$

  Thus, if $([\boldsymbol{m}]_1, \sigma)$ passes $\mathsf{S}_{\mathrm{KPW}}.\mathsf{Verify}$ but not $\mathsf{S}_{\mathrm{KPW}}.\mathsf{Verify}^*$, we have that $\sigma_1 - ([(1, m^\top)\boldsymbol{K}]_1 + \sigma_2(\boldsymbol{K}_0 + \tau\boldsymbol{K}_1)) \in \mathbb{G}_1^2$ is a non-zero vector in the kernel of $\boldsymbol{A}$. Finding such a vector is hard under the SXDH assumption, and we can construct an adversary $\mathcal{B}_{\mathrm{dh}}$ such that $|\mathsf{Adv}^{\mathrm{H}_0}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathrm{H}_1}_{\mathcal{A}}(\lambda)| \leq \mathsf{Adv}^{\mathrm{sxdh}}_{\mathcal{B}_{\mathrm{dh}}}(\lambda)$.

– Hybrid 2 is the same as Hybrid 1, except if the chosen tags $\tau_1, \ldots, \tau_Q$ during the signing queries are not all distinct.

  A simple union bound implies that $|\mathsf{Adv}^{\mathrm{H}_1}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathrm{H}_2}_{\mathcal{A}}(\lambda)| \leq Q^2/p$.

– Hybrid 3 is the same as Hybrid 2, except it samples $i^* \leftarrow [Q+1]$ and aborts if i^* is not the smallest index such that $\tau_i = \sigma_3^*$, where $\tau_{Q+1} = \sigma_3^*$ is the tag from the forgery.

  As there are Q queries, we have $\mathsf{Adv}^{\mathrm{H}_3}_{\mathcal{A}}(\lambda) \geq 1/(Q+1)\mathsf{Adv}^{\mathrm{H}_2}_{\mathcal{A}}(\lambda)$.

– Hybrid 4 is the same as Hybrid 3, except the signature queries are answered as follows. On input of $[\boldsymbol{m}]_1$ in the i-th signing query, samples $r, \tau, \mu \leftarrow \mathbb{Z}_p$, and set $\mu \leftarrow 0$ if $\tau = \tau^*$. Then, set $\sigma_1 \leftarrow [(1, \boldsymbol{m}^\top)\boldsymbol{K} + \mu\boldsymbol{a}^\perp + r(\boldsymbol{P}_0 + \tau\boldsymbol{P}_1)]_1$, $\sigma_2 \leftarrow [r\boldsymbol{B}^\top]_1$, and $\sigma_3 \leftarrow \tau \in \mathbb{Z}_p$, where $\boldsymbol{a}^\perp$ is a non-zero vector in the kernel of $\boldsymbol{A}$ such that $\boldsymbol{a}^\perp\boldsymbol{A} = \mathbf{0}$. Finally, outputs $(\sigma_1, \sigma_2, \sigma_3)$.

  Hybrid 3 and Hybrid 4 are indistinguishable which we can show by constructing an adversary $\mathcal{B}_4$ against lemma 7.10. $\mathcal{B}_4$ first samples $\boldsymbol{K}$ and receives $([\boldsymbol{P}_0]_1, [\boldsymbol{P}_1]_1, [\boldsymbol{B}]_1, \boldsymbol{K}_0\boldsymbol{A}, \boldsymbol{K}_1\boldsymbol{A}, \boldsymbol{A})$ with which she sets up the verification key vk for $\mathcal{A}$. $\mathcal{B}_4$ answers the i-th signing query for $i \neq i^*$ via $(\sigma_1, \sigma_2, \sigma_3) \leftarrow ([(1, \boldsymbol{m}^\top)\boldsymbol{K}]_1 \cdot \sigma_1', \sigma_2)$, where $(\sigma_1', \sigma_2) \leftarrow \mathcal{O}_b(\tau)$. The i^*-th signing query is answered honestly. A quick calculation shows that for $b = 0$ (resp. $b = 1$) the queries are answered as in Hybrid 3 (resp. 4). Note that $\mathsf{S}_{\mathrm{KPW}}.\mathsf{Verify}^*$ can be simulated via a query to $\mathcal{O}^*(\tau^*)$. Thus, we have $|\mathsf{Adv}^{\mathrm{H}_3}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathrm{H}_4}_{\mathcal{A}}(\lambda)| \leq 2Q\mathsf{Adv}^{\mathrm{sxdh}}_{\mathcal{B}}(\lambda) + Q/p$, where $\mathcal{B}$ is the adversary from lemma 7.10.

– Hybrid 5 is the same as Hybrid 4, except $\boldsymbol{K}$ is computed as $\boldsymbol{K} \leftarrow \boldsymbol{K'} + \boldsymbol{u}\boldsymbol{a}^\perp$ for $\boldsymbol{K'} \leftarrow \mathbb{Z}_p^{(\ell+1)\times 2}, \boldsymbol{u} \leftarrow \mathbb{Z}_q^{\ell+1}$. Clearly, Hybrid 5 and Hybrid 4 are identically distributed and we have $\mathsf{Adv}^{\mathrm{H}_4}_{\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathrm{H}_5}_{\mathcal{A}}(\lambda)$.

Finally, observe that as in [KPW15], the verification key vk and signing queries for $\tau_i \neq \tau^*$ leak no information about $\boldsymbol{u}$, as $\boldsymbol{C} = (\boldsymbol{K'} + \boldsymbol{u}\boldsymbol{a}^\perp)\boldsymbol{A} = \boldsymbol{K'}\boldsymbol{A}$, and $(1, \boldsymbol{m}^\top)(\boldsymbol{K'} + \boldsymbol{u}\boldsymbol{a}^\perp) + \mu\boldsymbol{a}^\perp$ is identically distributed to $(1, \boldsymbol{m}^\top)\boldsymbol{K'} + \mu'\boldsymbol{a}^\perp$. The i^*-th signing query leaks $(1, \boldsymbol{m}^\top)(\boldsymbol{K'} + \boldsymbol{u}\boldsymbol{a}^\perp)$, captured by $(1, \boldsymbol{m}^\top)\boldsymbol{u}$. To provide a valid forgery, $\mathcal{A}$ needs to compute

$$\boldsymbol{v} \coloneqq (1, \boldsymbol{m}^{*\top})(\boldsymbol{K'} + \boldsymbol{u}\boldsymbol{a}^\perp).$$

Given $(1, \boldsymbol{m}^\top)\boldsymbol{u}$, for any adaptively chosen $\boldsymbol{m}^* \neq \boldsymbol{m}$, we have that $(1, \boldsymbol{m}^{*\top})\boldsymbol{u}$ is uniformly random over $\mathbb{Z}_p$ in the view of $\mathcal{A}$. Thus, $\boldsymbol{v}$ is also uniformly random over $\mathbb{Z}_p$ and we have $\mathsf{Adv}^{\mathrm{H}_5}_{\mathcal{A}}(\lambda) \leq 1/p$. □

Σ-protocol

Now, we instantiate the Σ-protocol Σ for relation $\mathsf{R}_{\mathsf{rnd}}$, where C is instantiated with $\mathsf{C}_{\mathrm{Ped}}$ over $\mathbb{G}_1$ and S instantiated with $\mathsf{S}_{\mathrm{KPW}}$. In this context, the relation $\mathsf{R}_{\mathsf{rnd}}$ can be written as

$$\begin{aligned}\mathsf{R}_{\mathsf{rnd}} = \{(x,w) : c = g_1^{\overline{m}}\mathsf{pp}^r \wedge \\ e(\boldsymbol{\mu_1},[\boldsymbol{A}]_2) = e((g_1,c),[\boldsymbol{C}]_2)\cdot e(\boldsymbol{\mu_2},[\boldsymbol{C}_0]_2\cdot\mu_3[\boldsymbol{C}_1]_2)\},\end{aligned}$$

where $x = (\mathsf{pp},\mathsf{vk},\overline{m}), w = (\mu, c, r)$ for $\mathsf{vk} = ([\boldsymbol{C}_0]_2,[\boldsymbol{C}_1]_2,[\boldsymbol{C}]_2,[\boldsymbol{A}]_2)$ and $\mu = (\boldsymbol{\mu_1},\boldsymbol{\mu_2},\mu_3)$.

We now provide the Σ-protocol, denoted Σ_{rnd}. For readability, we further assume that random generators $(\mathsf{pp}_1,\ldots,\mathsf{pp}_5)$ are setup without trapdoors via a random oracle for simplicity. (In the construction, we let these values be output by the hash functions H_{pp} in addition to pp.) On a high level, the prover commits to the witness in ElGamal commitments under different public parameters pp_i and shared randomness $S = g_1^s$, and then shows that the committed values satisfy the required relations. Note that the verification equation of μ is quadratic, as both $\boldsymbol{\mu_2}$ and μ_3 are witnesses. For this part, we introduce a new witness $\omega = s\cdot\mu_3$.

- $\Sigma_{\mathsf{rnd}}.\mathsf{Init}(\mathsf{crs},x,w)$: for (crs,x,w) defined as above, denote $e_1 = c, e_2 = \mu_{1,1}, e_3 = \mu_{1,2}, e_4 = \mu_{2,1}, e_5 = \mu_{2,2}$ and $\tau = \mu_3, \omega = s\cdot\tau$. First, sets $S = g_1^s$ for $s \leftarrow \mathbb{Z}_p$ and commits to e_i with shared randomness via $E_i = e_i\mathsf{pp}_i^s$. Samples additive masks $\widetilde{s},\widetilde{r},\widetilde{\omega},\widetilde{\tau} \leftarrow \mathbb{Z}_p$ and sets

$$\begin{aligned}&D_{\overline{m}} = \mathsf{pp}_1^{-\widetilde{s}}\cdot\mathsf{pp}^{-\widetilde{r}}, \qquad D_s = g_1^{-\widetilde{s}}, \qquad D_\omega = S^{\widetilde{\tau}}g_1^{-\widetilde{\omega}}\\ &\boldsymbol{D_\mu} = e((\mathsf{pp}_2^{-\widetilde{s}},\mathsf{pp}_3^{-\widetilde{s}}),[\boldsymbol{A}]_2)^{-1}\cdot e((1_{\mathbb{G}_1},\mathsf{pp}_1^{-\widetilde{s}}),[\boldsymbol{C}]_2)\\ &\qquad\cdot e((\mathsf{pp}_4^{-\widetilde{s}},\mathsf{pp}_5^{-\widetilde{s}}),[\boldsymbol{C_0}]_2)\cdot e((E_4^{\widetilde{\tau}}\mathsf{pp}_4^{-\widetilde{\omega}},E_5^{\widetilde{\tau}}\mathsf{pp}_5^{-\widetilde{\omega}}),[\boldsymbol{C_1}]_2).\end{aligned}$$

  Outputs $\alpha = (S,E_1,\ldots,E_5,D_{\overline{m}},D_s,D_\omega,\boldsymbol{D_\mu})$ and stores $(\widetilde{s},\widetilde{r},\widetilde{\omega},\widetilde{\tau})$ in st,

- $\Sigma_{\mathsf{ext}}.\mathsf{Chall}()$: samples a challenge $\beta \leftarrow \mathbb{Z}_p$,

- $\Sigma_{\mathsf{ext}}.\mathsf{Resp}(\mathsf{st},\beta)$: sets $\gamma_r = \beta\cdot r+\widetilde{r}, \gamma_s = \beta\cdot s+\widetilde{s}, \gamma_\tau = \beta\cdot\tau+\widetilde{\tau}, \gamma_\omega = \beta\cdot\omega+\widetilde{\omega}$, and outputs the response $\gamma = (\gamma_r,\gamma_s,\gamma_\tau,\gamma_\omega)$,

- $\Sigma_{\mathsf{ext}}.\mathsf{Verify}(\mathsf{crs},x,\alpha,\beta,\gamma)$: checks the following equations

$$\begin{aligned}&D_{\overline{m}} = E_1^\beta\cdot\mathsf{pp}_1^{-\gamma_s}g_1^{-\beta\cdot\overline{m}}\mathsf{pp}^{-\gamma_r}, \qquad D_s = S^\beta g_1^{-\gamma_s}, \qquad D_\omega = S^{\gamma_\tau}\cdot g_1^{-\gamma_\omega},\\ &\boldsymbol{D_\mu} = e(\boldsymbol{F_1},[\boldsymbol{A}]_2)^{-1}\cdot e(\boldsymbol{F_m},[\boldsymbol{C}]_2)\cdot e(\boldsymbol{F_2},[\boldsymbol{C_0}]_2)\cdot e(\boldsymbol{F_3},[\boldsymbol{C_1}]_2),\end{aligned}$$

  where $\boldsymbol{F_1} = (E_2^\beta\cdot\mathsf{pp}_2^{-\gamma_s},E_3^\beta\cdot\mathsf{pp}_3^{-\gamma_s}), \boldsymbol{F_m} = (g_1^\beta,E_1^\beta\cdot\mathsf{pp}_1^{-\gamma_s}), \boldsymbol{F_2} = (E_4^\beta\cdot\mathsf{pp}_4^{-\gamma_s},E_5^\beta\cdot\mathsf{pp}_5^{-\gamma_s}), \boldsymbol{F_3} = (E_4^{-\gamma_\tau}\cdot\mathsf{pp}_4^{-\gamma_\omega},E_5^{-\gamma_\tau}\cdot\mathsf{pp}_5^{-\gamma_\omega})$, and outputs 1 iff all checks succeed. Note that the first equation checks that the commitment c committed in E_1 is a valid $\mathsf{C}_{\mathrm{Ped}}$ commitments to $\overline{m}$, the second equation fixes s (and thus the values committed in E_i) and the third equation fixes $\omega = s\cdot\tau$ which allows to open the commitments E_4^τ and E_5^τ in zero-knowledge. Finally the last equation checks whether the committed signature μ is valid. For this, we rewrite $e(\boldsymbol{\mu_2},[\boldsymbol{C}_0]_2\cdot\mu_3[\boldsymbol{C}_1]_2) = e(\boldsymbol{\mu_2},[\boldsymbol{C}_0]_2)\cdot e(\boldsymbol{\mu_2}^{\mu_3},[\boldsymbol{C}_1]_2)$, and use that (E_5^τ,E_6^τ) commits to $\boldsymbol{\mu_2}^{\mu_3}$.

Theorem 7.11 (Correctness)**.** *The scheme Σ_{rnd} is correct.*

Proof. Note that we have $E_i^\beta\mathsf{pp}_i^{-\gamma_s} = e_i^\beta\cdot\mathsf{pp}_i^{-\widetilde{s}}$ and $E_i^{-\gamma_\tau}\mathsf{pp}_i^{-\gamma_\omega} = e_i^{\beta\cdot\tau}\cdot E_i^{-\widetilde{\tau}}\cdot\mathsf{pp}_i^{-\widetilde{\omega}}$. With these identities, the identities in $\Sigma_{\mathsf{rnd}}.\mathsf{Verify}$ follow from a straightforward calculation, and we omit details. □

Theorem 7.12 (HVZK)**.** *The scheme Σ_{rnd} is HVZK under the* DDH *assumption.*

Proof. Let $x = (\mathsf{pp}, \mathsf{vk}, \overline{m})$ and $\beta \leftarrow \mathbb{Z}_p$. We define the simulator Sim as follows. On input of (x, β), samples $s \leftarrow \mathbb{Z}_p$, and sets $S = g_1^s, E_i \leftarrow \mathsf{pp}_i^s$ for $i \in [5]$. Then, samples $\gamma = (\gamma_r, \gamma_s, \gamma_\tau, \gamma_\omega) \leftarrow \mathbb{Z}_p^4$ and computes $D_{\overline{m}}, D_s, D_\omega, \boldsymbol{D_\mu}$ via the identities in $\Sigma_{\mathsf{ext}}.\mathsf{Verify}$. Finally, sets $\alpha = (S, E_1, ..., E_5, D_{\overline{m}}, D_s, D_\omega, \boldsymbol{D_\mu})$ and outputs the transcript (α, β, γ).

To show that Sim outputs transcripts that are indistinguishable from real transcripts, we define the following hybrids and denote by $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_i}(\lambda)$ the advantage of $\mathcal{A}$ in Hybrid i.

- Hybrid 0 outputs honestly generated transcripts.
- Hybrid 1 is the same as Hybrid 0, except the elements $(D_r, D_s, D_\omega, \boldsymbol{D_\mu})$ are generated as in Sim. A quick calculation shows that Hybrid 0 and Hybrid 1 are identically distributed, and we have $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_0}(\lambda) = \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_1}(\lambda)$.
- Hybrid 2 is the same as Hybrid 1, except γ is computed as in Sim. Again, it is easy to check that $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_1}(\lambda) = \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_2}(\lambda)$.
- Hybrid 3 is the same as Hybrid 2, except the commitments $(S, E_1, ..., E_5)$ are commitments to $1_{\mathbb{G}_1}$ instead. If $\mathcal{A}$ can distinguish between Hybrid 2 and Hybrid 3, we can construct an adversary $\mathcal{B}$ against the DDH assumption, as $(S, E_1, ..., E_5)$ is hiding under DDH (see remark 7.7). Here, we use the fact that the public parameters $(\mathsf{pp}_1, ..., \mathsf{pp}_5)$ are output by a random oracle which is programmed with the challenge pp_i appropriately.

As Hybrid 3 outputs simulated transcripts, the statement follows. □

Theorem 7.13 (Special Soundness)**.** *The scheme Σ_{ext} is 2-special sound.*

Proof. Let $x = (\mathsf{pp}, \mathsf{vk}, \overline{m})$. We define the extractor Ext as follows. First, Ext receives as input two valid transcripts (α, β, γ) and $(\alpha, \beta', \gamma')$ with $\beta \neq \beta'$. Next, Ext parses $\alpha = (S, E_1, ..., E_5, D_{\overline{m}}, D_s, D_\omega, \boldsymbol{D_\mu})$ and $\gamma = (\gamma_r, \gamma_s, \gamma_\tau, \gamma_\omega), \gamma' = (\gamma'_r, \gamma'_s, \gamma'_\tau, \gamma'_\omega)$. From the identities in $\Sigma_{\mathsf{ext}}.\mathsf{Verify}$, we obtain the identities

$$
\begin{aligned}
S^{\beta'} \cdot g_1^{-\gamma'_s} &= S^{\beta} \cdot g_1^{-\gamma_s}, \\
E_1^{\beta'} \cdot \mathsf{pp}_1^{-\gamma'_s} g_1^{-\beta' \cdot \overline{m}} \mathsf{pp}^{-\gamma'_r} &= E_1^{\beta} \cdot \mathsf{pp}_1^{-\gamma_s} g_1^{-\beta \cdot \overline{m}} \mathsf{pp}^{-\gamma_r} \\
S^{\gamma'_\tau} \cdot g_1^{-\gamma'_\omega} &= S^{\gamma_\tau} \cdot g_1^{-\gamma_\omega}, \\
e(\boldsymbol{F_1'}, [\boldsymbol{A}]_2)^{-1} \cdot e(\boldsymbol{F_m'}, [\boldsymbol{C}]_2) &= e(\boldsymbol{F_1}, [\boldsymbol{A}]_2)^{-1} \cdot e(\boldsymbol{F_m}, [\boldsymbol{C}]_2) \\
\cdot e(\boldsymbol{F_2'}, [\boldsymbol{C_0}]_2) \cdot e(\boldsymbol{F_3'}, [\boldsymbol{C_1}]_2) \quad &\cdot e(\boldsymbol{F_2}, [\boldsymbol{C_0}]_2) \cdot e(\boldsymbol{F_3}, [\boldsymbol{C_1}]_2),
\end{aligned}
$$

where $\boldsymbol{F_1} = (E_2^{\beta} \cdot \mathsf{pp}_2^{-\gamma_s}, E_3^{\beta} \cdot \mathsf{pp}_3^{-\gamma_s})$, $\boldsymbol{F_m} = (g_1^{\beta}, E_1^{\beta} \cdot \mathsf{pp}_1^{-\gamma_s})$, $\boldsymbol{F_2} = (E_4^{\beta} \cdot \mathsf{pp}_4^{-\gamma_s}, E_5^{\beta} \cdot \mathsf{pp}_5^{-\gamma_s})$, $\boldsymbol{F_3} = (E_4^{-\gamma_\tau} \cdot \mathsf{pp}_4^{-\gamma_\omega}, E_5^{-\gamma_\tau} \cdot \mathsf{pp}_5^{-\gamma_\omega})$ and similarly for $\boldsymbol{F_1'}$, $\boldsymbol{F_m'}$, $\boldsymbol{F_2'}$ and $\boldsymbol{F_3'}$. We denote $\Delta x = (\gamma_x - \gamma'_x)$ for $x \in \{r, s, \tau, \omega\}$ and $\Delta\beta = (\beta - \beta')$. Note that $\Delta\beta \neq 0$. We further denote $s = \Delta\gamma_s / \Delta\beta, r = \Delta\gamma_r / \Delta\beta, \tau = \Delta\gamma_\tau / \Delta\beta$ and $\omega = \Delta\gamma_\omega / \Delta\beta$.

From the first equation, we obtain $S^{\Delta\beta} \cdot g_1^{-\Delta\gamma_s} = 1_{\mathbb{G}_1}$. Taking both sides to the power of $1/\Delta\beta$ yields $S = g_1^{\Delta\gamma_s / \Delta\beta} = g_1^s$. Similarly, we obtain from the second equation that $E_1^{\Delta\beta} \cdot \mathsf{pp}_1^{-\Delta\gamma_s} g_1^{-\Delta\beta\overline{m}} \cdot \mathsf{pp}^{-\Delta\gamma_r} = 1_{\mathbb{G}_1}$, and thus $E_1 = g_1^{\overline{m}} \cdot \mathsf{pp}^r \cdot \mathsf{pp}_1^s$. Consequently, we have $c = g_1^{\overline{m}} \cdot \mathsf{pp}^r$, for $c = E_1 \cdot \mathsf{pp}_1^{-s}$ As above, the third equation yields $S^\tau = g_1^\omega$, so $\omega = s \cdot \tau$.

We can now recompute the value e_i committed in E_i via $e_i \leftarrow E_i \mathsf{pp}_i^{-s}$. Note that that $\boldsymbol{F_1}(\boldsymbol{F_1'})^{-1} = (E_2^{\Delta\beta} \cdot \mathsf{pp}_2^{-\Delta\gamma_s}, E_3^{\Delta\beta} \cdot \mathsf{pp}_3^{-\Delta\gamma_s})$ and thus $(\boldsymbol{F_1}(\boldsymbol{F_1'})^{-1})^{1/\Delta\beta} = (E_2 \cdot \mathsf{pp}_3^{-s}, E_3 \cdot \mathsf{pp}_4^{-s}) = (e_3, e_4)$. Similarly, we have $(\boldsymbol{F_m}(\boldsymbol{F_m'})^{-1})^{1/\Delta\beta} = (g_1, e_1)$, $(\boldsymbol{F_2}(\boldsymbol{F_2'})^{-1})^{1/\Delta\beta} = (e_4, e_5)$ and $(\boldsymbol{F_3}(\boldsymbol{F_3'})^{-1})^{1/\Delta\beta} = (e_4^\tau, e_5^\tau)$. Finally, the last identity implies:

$$
e((e_2, e_3), [\boldsymbol{A}]_2) = e((g_1, e_1), [\boldsymbol{C}]_2) \cdot e((e_4, e_5), [\boldsymbol{C_0}]_2) \cdot e((e_4^\tau, e_5^\tau), [\boldsymbol{C_1}]_2)
$$

As $e((e_4^\tau, e_5^\tau), [\boldsymbol{C_1}]_2) = e((e_4, e_5), \tau[\boldsymbol{C_1}]_2)$, it follows that $(\boldsymbol{\mu_1} = (e_2, e_3), \boldsymbol{\mu_2} = (e_4, e_5), \mu_3 = \tau)$ is a valid signature on message $e_1 = c$. Also, we know $c = g_1^{\overline{m}} \cdot \mathsf{pp}^r$, as desired. □

Theorem 7.14. *The Σ-protocol Σ_{ext} has high min-entropy.*

Proof. Observe that, for example, D_s is distributed uniformly random in $\mathbb{G}_1$. It follows that the advantage of any adversary in the min-entropy game is at most $1/p = \mathsf{negl}(\lambda)$. □

Theorem 7.15. *The Σ-protocol Σ_{ext} has f-unique extraction, where $f(\mu, c, r) = c$.*

Proof. As c is committed via an ElGamal commitment, even the first flow α of a transcript (α, β, γ) fixes c perfectly. The statement follows. □

7.3.1 Optimizations and Efficiency

We analyze the efficiency of $\mathsf{BS}_{\mathsf{Rnd}}$ when instantiated with the above schemes.

Optimizations. Note that in the construction, we apply Fiat-Shamir to the Σ_{rnd}. It is well-known that the values $(D_{\overline{m}}, D_s, D_\omega, \boldsymbol{D_\mu})$ can be omitted from the proof, as the identities can be recomputed as in $\Sigma_{\mathsf{rnd}}.\mathsf{Verify}$ and checked via β.

Efficiency. The scheme $\mathsf{BS}_{\mathsf{Rnd}}$ is secure under SXDH. The user sends 1 element in $\mathbb{G}_1$ and 1 element in $\mathbb{Z}_p$, the signer sends 4 elements in $\mathbb{G}_1$ and 1 element in $\mathbb{Z}_p$ and the final signature contains 6 elements in $\mathbb{G}_1$ and 5 elements in $\mathbb{Z}_p$. Consequently, the total communication is 303 Byte and signatures are of size 447 Byte for $\lambda = 128$.

7.4 Blind Signatures based on Boneh-Boyen Signature

In this section, we provide a blind signature based on randomizable signatures. Compared to the optimized generic construction of the Fischlin blind signature in section 7.2, the resulting signature size is much smaller since it only consists of one signature of the underlying randomizable signature scheme. The construction also relies on an online-extractable NIZK, which we show in section 7.5 that can be instantiated efficiently by carefully combining Bulletproofs and another NIZK for an ElGamal commitment. In Section 7.6.2 we show how to adapt the scheme for a partially blind variant, where we modify the Boneh-Boyen signature [BB04a, BB08] in order to embed the common message into the verification key.

7.4.1 Construction

We focus on the asymmetric pairing setting. We note that there is also a natural variant of this scheme in the symmetric setting and we omit details. First, we recall the Boneh-Boyen signatures [BB04a, BB08] in the asymmetric setting. While this is implicit in our proof, we note the following is known to be selectively secure in the standard model under the CDH assumption:

- $\mathsf{S}_{\mathsf{BB}}.\mathsf{KeyGen}(1^\lambda)$: samples $\alpha, \beta, \gamma \leftarrow \mathbb{Z}_p$, and sets $u_1 = g_1^\alpha, u_2 = g_2^\alpha, h_1 = g_1^\gamma, h_2 = g_2^\gamma, v = e(g_1, g_2)^{\alpha\beta}$, and outputs $\mathsf{vk} = (u_1, u_2, h_1, h_2, v)$ and $\mathsf{sk} = g_1^{\alpha\beta}$,
- $\mathsf{S}_{\mathsf{BB}}.\mathsf{Sign}(\mathsf{sk}, m)$: samples $r \in \mathbb{Z}_p$ and outputs $(\sigma_1, \sigma_2) = (\mathsf{sk} \cdot (u_1^m h_1)^r, g_1^r) \in \mathbb{G}_1^2$,
- $\mathsf{S}_{\mathsf{BB}}.\mathsf{Verify}(\mathsf{vk}, m, (\sigma_1, \sigma_2))$: outputs 1 if $e(\sigma_1, g_2) = v \cdot e(\sigma_2, u_2^m h_2)$, and otherwise outputs 0.

Overview. We present our framework for blind signatures based on S_{BB}. Let Π be an online-extractable NIZK proof system, with random oracle $\mathsf{H}_{\mathsf{zk}} : \{0,1\}^* \mapsto \{0,1\}^{\ell_{\mathsf{zk}}}$ and common reference string crs of length ℓ_{crs} for the relation

$$\mathsf{R}_{\mathsf{bb}} := \{x = (c, u_1, g_1), w = (\overline{m}, s) \mid c = u_1^{\overline{m}} \cdot g_1^s\}.$$

Let $\mathsf{H}_{\mathsf{M}}, \mathsf{H}_{\mathsf{crs}}$ be a random oracles mapping into $\mathbb{Z}_p, \{0,1\}^{\ell_{\mathsf{crs}}}$ respectively. The framework $\mathsf{BS}_{\mathsf{BB}}[\Pi]$, or $\mathsf{BS}_{\mathsf{BB}}$ for short, broadly works as follows. The verification and signing keys are identical to the ones of S_{BB}, that is $\mathsf{vk} = (u_1, u_2, h_1, h_2, v)$ and $\mathsf{sk} = g_1^{\alpha\beta}$. Additionally, the oracle $\mathsf{H}_{\mathsf{crs}}$ implicitly defines a common random string $\mathsf{crs} = \mathsf{H}_{\mathsf{crs}}(0)$ for Π. In order to obtain a signature on message m, the user first commits to $\overline{m} = \mathsf{H}_{\mathsf{M}}(m)$ in a Pedersen commitment $c \leftarrow u_1^{\overline{m}} \cdot g_1^s \in \mathbb{G}_1$, where $s \leftarrow \mathbb{Z}_p$. Then, computes a proof π via Π showing that c was computed honestly, and sends $\rho_1 = (c, \pi)$ to the signer. The signer then checks the proof and sends $\rho_2 = (\rho_{2,0}, \rho_{2,1}) \leftarrow (\mathsf{sk} \cdot (c \cdot h_1)^r, g_1^r)$ to the user, where $r \leftarrow \mathbb{Z}_p$. Finally, the user checks that the ρ_2 is valid with respect to c and that $(\rho_{2,0}, \rho_{2,1})$ are consistent, and then derives a re-randomized S_{BB} signature on $\overline{m} = \mathsf{H}_{\mathsf{M}}(m)$ via $\sigma = (\rho_{2,0}/\rho_{2,1}^s \cdot (u_1^{\overline{m}} h_1)^{r'}, \rho_{2,1} \cdot g_1^{r'})$.

Description. In more detail, we have the following, where we assume that crs is provided to all of the algorithms for readability.

- $\mathsf{BS}_{\mathsf{BB}}.\mathsf{KeyGen}(1^\lambda)$: outputs $(\mathsf{bvk}, \mathsf{bsk}) \leftarrow \mathsf{S}_{\mathsf{BB}}.\mathsf{KeyGen}(1^\lambda)$, where $\mathsf{bvk} = (u_1, u_2, h_1, h_2, v)$ with $u_1 = g_1^\alpha, u_2 = g_2^\alpha, h_1 = g_1^\gamma, h_2 = g_2^\gamma, v = e(g_1, g_2)^{\alpha\beta}$ and $\mathsf{bsk} = g_1^{\alpha\beta}$.
- $\mathsf{BS}_{\mathsf{BB}}.\mathsf{User}(\mathsf{bvk}, m)$: checks validity of the verification key bvk via $e(u_1, g_2) = e(g_1, u_2)$ and $e(h_1, g_2) = e(g_1, h_2)$, sets $\overline{m} \leftarrow \mathsf{H}_{\mathsf{M}}(m)$ and computes a Pedersen commitment $c = u_1^{\overline{m}} g_1^s \in \mathbb{G}_1$ to $\overline{m}$ and a proof $\pi \leftarrow \Pi.\mathsf{Prove}^{\mathsf{H}_{\mathsf{zk}}}(\mathsf{crs}, x, w)$, where $s \leftarrow \mathbb{Z}_p$, $x = (c, u_1, g_1)$, and $w = (\overline{m}, s)$. It outputs the first message $\rho_1 = (c, \pi)$ and stores the randomness $\mathsf{st} = s$.
- $\mathsf{BS}_{\mathsf{BB}}.\mathsf{Signer}(\mathsf{bsk}, \rho_1)$: parses $\rho_1 = (c, \pi)$, checks $\Pi.\mathsf{Verify}^{\mathsf{H}_{\mathsf{zk}}}(\mathsf{crs}, x, \pi) = 1$ and outputs the second message $\rho_2 = (\rho_{2,0}, \rho_{2,1}) \leftarrow (\mathsf{sk} \cdot (c \cdot h_1)^r, g_1^r) \in \mathbb{G}_1^2$, where $r \leftarrow \mathbb{Z}_p$.
- $\mathsf{BS}_{\mathsf{BB}}.\mathsf{Derive}(\mathsf{st}, \rho_2)$: parses $\mathsf{st} = s$ and $\rho_2 = (\rho_{2,0}, \rho_{2,1})$, checks $e(\rho_{2,0}, g_2) = v \cdot e(\rho_{2,1}, u_2^{\overline{m}} g_2^s \cdot h_2)$, and outputs the signature $\sigma = (\rho_{2,0}/\rho_{2,1}^s \cdot (u_1^{\overline{m}} h_1)^{r'}, \rho_{2,1} \cdot g_1^{r'}) \in \mathbb{G}_1^2$ for $r' \leftarrow \mathbb{Z}_p$.
- $\mathsf{BS}_{\mathsf{BB}}.\mathsf{Verify}(\mathsf{bvk}, m, \sigma)$: sets $\overline{m} \leftarrow \mathsf{H}_{\mathsf{M}}(m)$ and outputs $b \leftarrow \mathsf{S}_{\mathsf{BB}}.\mathsf{Verify}(\mathsf{bvk}, \overline{m}, \sigma)$.

7.4.2 Correctness and Security

We prove correctness, blindness and one-more unforgeability. Correctness follows from a simple calculation. Blindness follows from the zero-knowledge property of Π, and as c statistically hides the message and σ is re-randomized. The proof follows a similar all-but-one reduction as the underlying Boneh-Boyen signature. The only difference is that we modify the Boneh-Boeyn signature which is selectively secure in the standard model, to be adaptively secure in the ROM, and to use the (multi)-online extractor to extract randomness of c submitted by the adversary. Concretely, the reduction first guesses a query $\overline{m}^* = \mathsf{H}_{\mathsf{M}}(m^*)$ and embeds a CDH challenge into vk such that it can sign all values in $\mathbb{Z}_p \setminus \{\overline{m}^*\}$. For each signing query, the reduction extracts the randomness of c from the proof π, simulates the signing of m as in the original `EUF-CMA` proof of S_{BB}, and finally reapplies the randomness of c to the intermediate signature. If the extracted message is $\overline{m}^*$, the reduction aborts. Here, we crucially require that Π is online-extractable. In the end, the reduction hopes to receive a valid signature on $\overline{m}^*$ with which it can solve CDH. More details can be found in section 7.1.2.

Theorem 7.16 (Correctness)**.** *The scheme* $\mathsf{BS}_{\mathsf{BB}}$ *is correct.*

Proof. Let $(\mathsf{bvk}, \mathsf{bsk}) \leftarrow \mathsf{BS}_{\mathsf{BB}}.\mathsf{KeyGen}(1^\lambda)$, where $\mathsf{bvk} = (u_1, u_2, h_1, h_2, v)$ and $\mathsf{sk} = g_1^{\alpha\beta}$ with $u_1 = g_1^\alpha, u_2 = g_2^\alpha, h_1 = g_1^\gamma, h_2 = g_2^\gamma, v = e(g_1, g_2)^{\alpha\beta}$ and $\mathsf{bsk} = g_1^{\alpha\beta}$. Let $\rho_1 = (c = (u_1^{\overline{m}} g_1^s), \pi) \leftarrow \mathsf{BS}_{\mathsf{BB}}.\mathsf{User}(\mathsf{bvk}, m)$ for any message m and $\overline{m} \leftarrow \mathsf{H}_{\mathsf{M}}(m)$, where note that the check on the verification key performed by the user passes by construction. Under the correctness of Π, the proof π verifies, and we have $\rho_2 = (\rho_{2,0}, \rho_{2,1}) = (\mathsf{sk} \cdot (c \cdot h_1)^r, g_1^r) \leftarrow \mathsf{BS}_{\mathsf{BB}}.\mathsf{Signer}(\mathsf{bsk}, \rho_1)$. Note that the check in $\mathsf{BS}_{\mathsf{BB}}.\mathsf{Derive}(\mathsf{st}, \rho_2)$ passes as

$$\begin{aligned} e(\rho_{2,0}, g_2) &= e(\mathsf{sk} \cdot (c \cdot h_1)^r, g_2) \\ &= e(\mathsf{sk}, g_2) \cdot e(c \cdot h_1, g_2)^r \\ &= v \cdot e(u_1^{\overline{m}} g_1^s \cdot h_1, g_2)^r \\ &= v \cdot e(g_1^r, u_2^{\overline{m}} g_2^s \cdot h_2) = v \cdot e(\rho_{2,1}, u_2^{\overline{m}} g_2^s \cdot h_2). \end{aligned}$$

Then, we can verify that $\sigma = (\sigma_0, \sigma_1) \leftarrow \mathsf{BS}_{\mathsf{BB}}.\mathsf{Derive}(\mathsf{st}, \rho)$ indeed outputs a valid S_{BB} signature:

$$\begin{aligned} (\sigma_0, \sigma_1) &= ((\mathsf{sk} \cdot (u_1^{\overline{m}} g_1^s \cdot h_1)^r) / g_1^{rs} \cdot (u_1^{\overline{m}} h_1)^{r'}, & g_1^r \cdot g_1^{r'}) \\ &= (\mathsf{sk} \cdot (u_1^{\overline{m}} h_1)^r \cdot (u_1^{\overline{m}} h_1)^{r'}, & g_1^{r+r'}) \\ &= (\mathsf{sk} \cdot (u_1^{\overline{m}} h_1)^{r+r'}, & g_1^{r+r'}). \end{aligned}$$

□

Theorem 7.17 (Blindness)**.** *The scheme* $\mathsf{BS}_{\mathsf{BB}}$ *is blind under malicious keys under the zero-knowledge property of* Π*.*

Proof. Let $\mathcal{A}$ be a PPT adversary against blindness. We define the following hybrids and denote by $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_i}(\lambda)$ the advantage of $\mathcal{A}$ in Hybrid i.

- Hybrid 0 is identical to the real game.
- Hybrid 1 is the same as Hybrid 0, except the H_{zk} queries and the proofs (π_0, π_1) are instead simulated via $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{H}_{\mathsf{zk}}}, \mathsf{Sim}_\pi)$ of Π. In more detail, for every query q to the random oracle H_{zk}, it outputs $\mathsf{Sim}_{\mathsf{H}_{\mathsf{zk}}}(q)$. After receiving (bvk, m_0, m_1) from $\mathcal{A}$, it checks the validity of the verification key bvk and sets $\pi_b \leftarrow \mathsf{Sim}_\pi(\mathsf{crs}, x_b)$ for $x_b = (c_b, u_1, g_1)$, where $c_b = u_1^{\overline{m}_b} g_1^{s_b}$ and $\overline{m}_b = \mathsf{H}_{\mathsf{M}}(m_b)$

  We can construct an adversary $\mathcal{B}_\Pi$ against the zero-knowledge property of Π with advantage $\mathsf{Adv}_{\mathcal{B}_\Pi}^{\mathrm{zk}}(\lambda) \geq |\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_0}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_1}(\lambda)|$. $\mathcal{B}_\Pi$ simulates the view to $\mathcal{A}$ by programming the received crs into H_{zk}. It then uses the provided oracles to answer H_{zk} queries and to generate proofs (π_0, π_1). Finally, $\mathcal{B}_\Pi$ forwards the guess of $\mathcal{A}$ to its challenger. If the oracle outputs simulated proofs, the game is distributed identically to Hybrid 1. Else, the oracle outputs real proofs and behaves as in Hybrid 0. Thus, we have

  $$|\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_0}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_1}(\lambda)| \leq \mathsf{Adv}_{\mathcal{B}_\Pi}^{\mathrm{zk}}(\lambda).$$

- Hybrid 2 is the same as Hybrid 1, except that the (inefficient) challenger recovers the signing key bsk, and prepares the signatures (σ_0, σ_1) on its own. In more detail, the challenger brute force searches the exponent $\alpha, \beta, \gamma \in \mathbb{Z}_p$ such that $u_1 = g_1^\alpha$, $h_1 = g_1^\gamma$, and $v = e(g_1, g_2)^{\alpha\beta}$ in the verification key bvk output by $\mathcal{A}$. When $\mathcal{A}$ returns the second message $\rho_{b,2} = (\rho_{b,2,0}, \rho_{b,2,1})$, it first checks if $e(\rho_{b,2,0}, g_2) = v \cdot e(\rho_{b,2,1}, u_2^{\overline{m}_b} g_2^{s_b} \cdot h_2)$ as in the previous Hybrid 1. If so, it runs $\sigma_b \leftarrow \mathsf{S}_{\mathsf{BB}}.\mathsf{Sign}(\mathsf{sk}, \overline{m}_b)$, where $\mathsf{sk} = g_1^{\alpha\beta}$ and $\overline{m}_b = \mathsf{H}_{\mathsf{M}}(m_b)$. Otherwise, it is the same as the previous Hybrid 1.

We show that Hybrids 1 and 2 are perfectly indistinguishable. Since the check performed by the challenger passes, we have the following for both Hybrids:

$$\begin{aligned} & e(\rho_{b,2,0}, g_2) = v \cdot e(\rho_{b,2,1}, u_2^{\overline{m}_b} g_2^{s_b} \cdot h_2) \\ \Leftrightarrow \quad & e(\rho_{b,2,0}, g_2) = e(g_1, g_2)^{\alpha\beta} \cdot e(\rho_{b,2,1}, g_2)^{\alpha\overline{m}_b + s_b + \gamma}. \end{aligned}$$

If we set $\rho_{b,2,1} = g_1^{r_b^*}$, then we have $\rho_{b,2,0} = g_1^{\alpha\beta + r_b^* \cdot (\alpha\overline{m}_b + s_b + \gamma)\cdot} = \mathsf{sk} \cdot (c_b \cdot h_1)^{r_b^*}$, where $\mathsf{sk} = g_1^{\alpha\beta}$. In Hybrid 1, the challenger then outputs the signature

$$\begin{aligned} \sigma_b &= (\rho_{2,b,0} / \rho_{2,b,1}^{s_b} \cdot (u_1^{\overline{m}_b} h_1)^{r'_b}, \rho_{2,b,1} \cdot g_1^{r'_b}) \\ &= (\mathsf{sk} \cdot (u_1^{\overline{m}_b} g_1^{s_b} \cdot h_1)^{r_b^*} \cdot g_1^{-r_b^* s_b} \cdot (u_1^{\overline{m}_b} h_1)^{r'_b}, g_1^{r_b^* + r'_b}) \\ &= (\mathsf{sk} \cdot (u_1^{\overline{m}_b} h_1)^{r_b^* + r'_b}, g_1^{r_b^* + r'_b}), \end{aligned}$$

where $r'_b \leftarrow \mathbb{Z}_p$. Since r'_b is information-theoretically hidden from $\mathcal{A}$, σ_b is identically distributed as a signature output by $\mathsf{S}_{\mathsf{BB}}.\mathsf{Sign}(\mathsf{sk}, \overline{m}_b)$ in Hybrid 2. Hence, we have $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_1}(\lambda) = \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_2}(\lambda)$.

– Hybrid 3 is the same as Hybrid 2, except that the challenger samples a random $c_b \leftarrow \mathbb{G}_1$, simulates a proof $\pi_b \leftarrow \mathsf{Sim}_\pi(\mathsf{crs}, x_b)$ for $x_b = (c_b, u_1, g_1)$, and outputs $\rho_{b,1} = (c_b, \pi_b)$ as the first message. It can be checked that Hybrids 2 and 3 are perfectly indistinguishable by noticing that $s_b \leftarrow \mathbb{Z}_p$ is information-theoretically hidden from $\mathcal{A}$ due to the modifications we made in Hybrids 1 and 2. Namely, in Hybrid 2, we have $c_b = u_1^{\overline{m}_b} g_1^{s_b}$ where s_b is uniform over $\mathbb{Z}_p$ from the view of $\mathcal{A}$. Hence, sampling c_b uniform over $\mathbb{G}$ results in the same distribution. Hence, we have $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_2}(\lambda) = \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_3}(\lambda)$.

In Hybrid 3, the value of coin is information-theoretically hidden from $\mathcal{A}$, as the commitments c_b and the proofs π_b are identically distributed for $b \in \{0, 1\}$. Consequently, $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_3}(\lambda) = 0$. Also, the running time of the adversaries $\mathcal{B}_\Pi$ is roughly that of $\mathcal{A}$. Combining the inequalities yields the statement. □

Theorem 7.18 (Unforgeability)**.** *The scheme* $\mathsf{BS}_{\mathsf{BB}}$ *is one-more unforgeable under the* CDH *assumption and the online-extractability of* Π*.*

Proof. Let $\mathcal{A}$ be a PPT adversary against one-more unforgeability of $\mathsf{BS}_{\mathsf{BB}}$. Let Ext be the extractor and $\mathsf{Sim}_{\mathsf{crs}}$ the simulator of Π from definition 3.31. We denote by Q_S the number of signing queries, by Q_M the number of H_{M} queries, and by Q_H the number of H_{zk} queries. Recall that we model $\mathsf{H}_{\mathsf{crs}}, \mathsf{H}_{\mathsf{zk}}$ and H_{M} as random oracles, where we assume without loss of generality that $\mathcal{A}$ never repeats queries. We denote by q_j the j-th query to H_{M} for $j \in [Q_M]$. After Q_S signing queries, $\mathcal{A}$ outputs $Q_S + 1$ forgeries $\{(m_i, \sigma_i)\}_{i=1}^{Q_S+1}$. We write $\sigma_i = (\sigma_{i,1}, \sigma_{i,2})$, and denote by $\rho_{1,i} = (c_i, \pi_i)$ the Q_S first message queries to $\mathsf{BS}_{\mathsf{BB}}.\mathsf{Signer}(\mathsf{bsk}, \cdot)$ issued by $\mathcal{A}$. We define the following hybrids and denote by $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_i}(\lambda)$ the advantage of $\mathcal{A}$ in Hybrid i.

– Hybrid 0 is identical the real game.

– Hybrid 1 is the same as Hybrid 0, except it samples $(\overline{\mathsf{crs}}, \tau) \leftarrow \mathsf{Sim}_{\mathsf{crs}}(1^\lambda)$ and programs $\overline{\mathsf{crs}}$ into the random oracle $\mathsf{H}_{\mathsf{crs}}$ via $\mathsf{H}_{\mathsf{crs}}(0) \leftarrow \overline{\mathsf{crs}}$. It is easy to construct an adversary $\mathcal{B}_{\mathsf{crs}}$ against the CRS indistinguishability of Π such that $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_1}(\lambda) \geq \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_0}(\lambda) - \mathsf{Adv}_{\mathcal{B}_{\mathsf{crs}}}^{\mathrm{crs}}(\lambda)$.

– Hybrid 2 is the same as Hybrid 1, except $(\overline{m}_i, s_i)$ is extracted from all the proofs $\{\pi_i\}_{i \in [Q_S]}$ using Ext_Π. Specifically, when $\mathcal{A}$ provides the signing query $\rho_{1,i} = (c_i, \pi_i)$, the challenger runs $w_i \leftarrow \mathsf{Ext}(\mathsf{crs}, \mathsf{td}, x_i, \pi_i)$, where $x_i = (c_i, u_1, g_1)$. It parses $w_i = (\overline{m}_i, s_i)$ and aborts if $u_1^{\overline{m}_i} \cdot g_1^{s_i} \neq c_i$. Otherwise, it is defined identically to the previous Hybrid 1.

We can construct an adversary $\mathcal{B}_{\mathsf{Ext}}$ against the multi-online extractability of Π with $\mathsf{Adv}^{\mathrm{H}_2}_{\mathcal{A}}(\lambda) \geq \frac{\mathsf{Adv}^{\mathrm{H}_1}_{\mathcal{A}}(\lambda)-\mathsf{negl}(\lambda)}{p_{\mathsf{P}}(\lambda,Q_H)}$ with additional runtime overhead $p_{\mathsf{T}}(\lambda, Q_H) \cdot \mathsf{Time}(\mathcal{A})$, where p_{P} and p_{T} are polynomials as defined in definition 3.31. In more detail, let us first consider $\mathcal{B}'_{\mathsf{Ext}}$ that receives $\overline{\mathsf{crs}}$ from its challenger, and simulates the challenger of Hybrid 2 for $\mathcal{A}$, after programming $\overline{\mathsf{crs}}$ into $\mathsf{H}_{\mathsf{crs}}$ and by answering H_{zk} queries via its provided oracle. At the end of the game, $\mathcal{B}'_{\mathsf{Ext}}$ outputs the pairs $\{(x_i, \pi_i)\}_{i\in[Q_S]}$, where $x_i = (c_i, u_1, h_1)$. Because $\mathcal{A}$ succeeds with probability $\mathsf{Adv}^{\mathrm{H}_1}_{\lambda}$, the pre-condition of definition 3.31 holds, and the above bound follows.

Observe that the output w_i of Ext is not used anywhere in Hybrid 2. Also, aborting in case of extraction failure and the runtime of Ext does not impact the success probability of $\mathcal{A}$ in Hybrid 2. Therefore, we can equally define an adversary $\mathcal{B}_{\mathsf{Ext}}$ that runs Ext during the game instead of at the end. Specifically, this is identical to the description of the Hybrid 2 challenger.

– Hybrid 3 is the same as Hybrid 2, except it aborts if there is a collision in H_{M} or if there is some message m_i in $\mathcal{A}$'s output that was never queried to H_{M}.

  The abort probability is upper bounded by $\frac{Q_M^2+1}{p}$, as a collision in H_{M} happens with probability at most Q_M^2/p and the probability that some S_{BB} signature σ_i is valid for a random message $\overline{m} \in \mathbb{Z}_p$ is $1/p$. It follows that $\mathsf{Adv}^{\mathrm{H}_3}_{\mathcal{A}}(\lambda) \geq \mathsf{Adv}^{\mathrm{H}_2}_{\mathcal{A}}(\lambda) - \frac{Q_M^2+1}{p}$.

– Hybrid 4 is the same as Hybrid 3, except it guesses a query index j^* of H_{M} for which it outputs some random $\overline{m}^* \in \mathbb{Z}_p$, and aborts if either some sign query contains $\overline{m}^*$ or if $\mathcal{A}$ provides no forgery for some message that hashes to $\overline{m}^*$. More concretely, before Hybrid 3 interacts with $\mathcal{A}$, it samples $\overline{m}^* \leftarrow \mathbb{Z}_p$ and $j^* \leftarrow [Q_M]$. For the j^*-th query to H_{M}, it sets $\mathsf{H}_{\mathsf{M}}(q_{j^*}) \leftarrow \overline{m}^*$. At the i-th signing query, the signer aborts if the extracted witness is $(\overline{m}^*, s_i)$, else proceeds as usual. Also, Hybrid 3 aborts if $\overline{m}^* \notin \{\mathsf{H}_{\mathsf{M}}(m_i) \mid i \in [Q_S + 1]\}$, where m_i are the messages from the forgeries output of $\mathcal{A}$.

  A simple calculation yields $\mathsf{Adv}^{\mathrm{H}_4}_{\mathcal{A}}(\lambda) \geq \frac{1}{Q_M}\mathsf{Adv}^{\mathrm{H}_3}_{\mathcal{A}}(\lambda)$, where E_3 denotes the event that $\mathcal{A}$ is successful in Hybrid 3.

$$\begin{aligned}
\mathsf{Adv}^{\mathrm{H}_4}_{\mathcal{A}}(\lambda) &= \Pr[\overline{m}^* \notin \{\mathsf{Ext}(\mathsf{crs}, \mathsf{td}, x_i, \pi_i)\}_{i\in[Q_S]} \wedge \overline{m}^* \in \{\mathsf{H}_{\mathsf{M}}(m_i)\}_{i=\in[Q_S+1]} \mid E_3] \cdot \Pr[E_3] \\
&= \Pr[\overline{m}^* \in \{\mathsf{H}_{\mathsf{M}}(m_i)\}_{i\in[Q_S+1]} \setminus \{\mathsf{Ext}(\mathsf{crs}, \mathsf{td}, x_i, \pi_i)\}_{i\in[Q_S]} \mid E_3] \cdot \Pr[E_3] \\
&\geq \Pr[\overline{m}^* = \overline{m}^\star \mid E_3] \cdot \Pr[E_3] \\
&= 1/Q_M \cdot \mathsf{Adv}^{\mathrm{H}_3}_{\mathcal{A}}(\lambda)
\end{aligned}$$

  Here, we use for the inequality that there is no collision in H_{M}, and thus there exists at least one $\overline{m}^\star \in \{\mathsf{H}_{\mathsf{M}}(m_i)\}_{i\in[Q_S+1]} \setminus \{\mathsf{Ext}(\mathsf{crs}, \mathsf{td}, x_i, \pi_i)\}_{i\in[Q_S]}$. In the last equality, we use that j^* is uniform in $\mathcal{A}$'s view.

– Hybrid 5 is the same as Hybrid 4, except it sets up a punctured verification key and simulates signing without knowing the full sk. Specifically, it samples $g_1 \leftarrow \mathbb{G}_1, g_2 \leftarrow \mathbb{G}_2$ and sets $A_1 \leftarrow g_1^\alpha, A_2 \leftarrow g_2^\alpha, B_1 \leftarrow g_1^\beta, B_2 \leftarrow g_2^\beta, u_1 = A_1, u_2 = A_2, h_1 = u_1^{-\overline{m}^*} \cdot g_1^\delta, h_2 = u_2^{-\overline{m}^*} \cdot g_2^\delta, v = e(A_1, B_2)$, where $\overline{m}^*$ is the j^*-th H_{M} output and $\alpha, \beta, \delta \leftarrow \mathbb{Z}_p$, and sends $\mathsf{bvk} = (g_1, g_2, u_1, u_2, h_1, h_2, v)$ to $\mathcal{A}$. Note that implicitly $\gamma = -\overline{m}^* \cdot \alpha + \delta \in \mathbb{Z}_p$. For each signing query $\rho_{i,1} = (c_i, \pi_i)$ with extracted witness $w_i = (\overline{m}_i, s_i)$, the challenger samples some $\tilde{r}_i \leftarrow \mathbb{Z}_p$, and sets $\rho_{i,2,1} = g_1^{\tilde{r}_i} \cdot B_1^{-1/(\overline{m}_i - \overline{m}^*)}$ and $\rho_{i,2,0} = A^{(\overline{m}_i-\overline{m}^*)\tilde{r}} \cdot g_1^{(s_i+\delta)\tilde{r}_i} \cdot B_1^{-(s_i+\delta)/(\overline{m}_i-\overline{m}^*)}$. It then returns $\mathcal{A}$ the second message as $\rho_{i,2} = (\rho_{i,2,0}, \rho_{i,2,1})$. Otherwise, it is defined identically as in the previous Hybrid 4.

  Clearly, bvk is distributed identically in Hybrids 4 and 5. Also, $\rho_{i,2,0} = g_1^{\tilde{r}_i - \beta/(\overline{m}_i - \overline{m}^*)}$ is distributed identically in Hybrids 4 and 5, as $r_i = \tilde{r}_i - \beta/(\overline{m}_i - \overline{m}^*)$ is distributed uniformly

over $\mathbb{Z}_p$ for a uniform $\tilde{r_i}$. The same holds for $\rho_{i,2,1}$, as $\overline{m}_i \neq \overline{m}^*$ due to the abort condition and

$$\begin{aligned} g_1^{\alpha\beta}(c_i \cdot h_1)^{r_i} &= g_1^{\alpha\beta}(A^{\overline{m}_i} \cdot g_1^{s_i} \cdot A^{-\overline{m}^*} \cdot g_1^{\delta})^{\tilde{r_i}-\beta/(\overline{m}_i-\overline{m}^*)} \\ &= g_1^{\alpha\beta}(A^{\overline{m}_i-\overline{m}^*} \cdot g_1^{s_i+\delta})^{\tilde{r_i}-\beta/(\overline{m}_i-\overline{m}^*)} \\ &= g_1^{\alpha\beta}A^{-\beta} \cdot A^{(\overline{m}_i-\overline{m}^*)\tilde{r}} \cdot g_1^{(s_i+\delta)\tilde{r_i}} \cdot B^{-(s_i+\delta)/(\overline{m}_i-\overline{m}^*)} \\ &= A^{(\overline{m}_i-\overline{m}^*)\tilde{r}} \cdot g_1^{(s_i+\delta)\tilde{r_i}} \cdot B^{-(s_i+\delta)/(\overline{m}_i-\overline{m}^*)} = \rho_{i,2,1}. \end{aligned}$$

Thus, $\mathsf{Adv}^{\mathrm{H}_5}_{\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathrm{H}_4}_{\mathcal{A}}(\lambda)$.

We now show that we construct an adversary $\mathcal{B}_{\mathsf{CDH}}$ with $\mathsf{Adv}^{\mathrm{CDH}}_{\mathcal{A}_{\mathsf{CDH}}}(\lambda) = \mathsf{Adv}^{\mathrm{H}_5}_{\mathcal{A}}(\lambda)$. First, $\mathcal{B}_{\mathsf{CDH}}$ receives CDH-tuple $(g_1, g_2, A_1, A_2, B_1, B_2)$, and uses these values to simulate the challenger of Hybrid 5 to $\mathcal{A}$. After $\mathcal{A}$ outputs the forgeries $\{(m_i, \sigma_i)\}_{i=1}^{Q_S+1}$, $\mathcal{B}_{\mathsf{CDH}}$ outputs $\sigma_{i^*,1}/\sigma_{i^*,2}^{\delta}$ to its challenger, where i^* such that $\mathsf{H}_{\mathsf{M}}(m_{i*}) = \overline{m}^*$ and $\sigma_{i^*} = (\sigma_{i^*,1}, \sigma_{i^*,2})$.

Note that due to the abort conditions in Hybrid 5, the probability that $\mathcal{B}_{\mathsf{CDH}}$ outputs such a value $\sigma_{i^*,1}/\sigma_{i^*,2}^{\delta}$ with $e(\sigma_{i^*,1}, g_2) = v \cdot e(\sigma_{i^*,2}, u_2^{\overline{m}_{i^*}} \cdot h_2)$ is at least $\mathsf{Adv}^{\mathrm{H}_5}_{\mathcal{A}}(\lambda)$. In that case, we have

$$\begin{aligned} & e(\sigma_{i^*,1}, g_2) = v \cdot e(\sigma_{i^*,2}, u_2^{\overline{m}_{i^*}} \cdot h_2) \\ \Longrightarrow\ & e(\sigma_{i^*,1}, g_2) = v \cdot e(\sigma_{i^*,2}, A_2^{\overline{m}^*} \cdot A_2^{-\overline{m}^*} g_2^{\delta}) \\ \Longrightarrow\ & e(\sigma_{i^*,1}, g_2) = v \cdot e(\sigma_{i^*,2}, g_2^{\delta}) \\ \Longrightarrow\ & e(\sigma_{i^*,1}/\sigma_{i^*,2}^{\delta}, g_2) = v \end{aligned}$$

Thus, $\sigma_{i^*,1}/\sigma_{i^*,2}^{\delta} = g_1^{\alpha\beta}$ as desired. The statement follows after collecting all the above bounds on the success probability and runtime. □

7.5 Instantiation of the Framework based on Boneh-Boyen

Here, we instantiate the online-extractable NIZK Π and analyze the efficiency of the blind signature $\mathsf{BS}_{\mathsf{BB}}[\Pi]$. First, we present our instantiation of Π. Π is a NIZK for showing knowledge of an opening of a Pedersen commitment $c = u_1^m g_1^s$, where $u_1, g_1 \in \mathbb{G}_1$ and $m, s \in \mathbb{Z}_p$,[10] i.e.for the relation

$$\mathsf{R}_{\mathsf{bb}} := \{x = (c, u_1, g_1), w = (m, s) \mid c = u_1^m g_1^s\}.$$

On a high level, we follow the well-known paradigm of combining an extractable commitment scheme (or equivalently a PKE) with a rewinding-based (non-online-extractable) NIZK Π' to construct an online-extractable NIZK Π. In the paradigm, the prover commits to the witness (m, s) using the extractable commitments, and adds a proof π via Π' to ensure that she indeed committed to openings of c, i.e.that $c = u_1^m g_1^s$ and (m, s) are committed in the extractable commitments. The online extractor can recover (m, s) from π by extracting the commitments via an appropriate trapdoor. Note that the soundness of the NIZK Π' guarantees consistency of the committed values and the openings of c. Indeed, in case the online extraction fails with non-negligible probability, we obtain a contradiction to the soundness of the underlying NIZK Π' via rewinding.

In the group setting, a common choice is ElGamal commitments. Note that we require the variant of ElGamal with message space $\mathbb{Z}_p$, i.e.$c_m = (g_1^m \mathsf{pp}^{r_m}, g_1^{r_m})$ and $(c_s = g_1^s \mathsf{pp}^{r_s}, g_1^{r_s})$ (see remark 7.8). Unfortunately, we can only extract (m, s) if the values are short, i.e.$m, s \in [0, B-1]$ for $B = \mathsf{poly}(\lambda)$. A common technique to circumvent this issue is to instead commit to the binary

[10] While we used $\overline{m}$ in the previous section, we omit the bar and simply denote m for readability.

decomposition $(m_i)_i$ and $(s_i)_i$, where $m = \sum_{i=1}^{\lceil \log_2 p \rceil} m_i 2^{i-1}$ and $s = \sum_{i=1}^{\lceil \log_2 p \rceil} s_i 2^{i-1}$, and show that the committed values $(m_i)_i$ and $(s_i)_i$ are bits and form valid decompositions of m and s, respectively. Unfortunately, this approach requires $2 \cdot \lceil \log_2(p) \rceil$ ElGamal commitments, and thus at least 1024 group elements for $\lambda = 256$. These elements already amount to more than 32 KB alone.

We instead commit to the B-ary decomposition $(m_i)_{i\in[\ell]}$ and $(s_i)_{i\in[\ell]}$ of $m = \sum_{i=1}^{\ell} m_i B^{i-1}$ and $s = \sum_{i=1}^{\ell} s_i B^{i-1}$, respectively, where $\ell = \lceil \log_B p \rceil$. As before, we show consistency of the committed values with the openings of c via a NIZK that ensures $c = u_1^m g_1^s$, $m = \sum_{i=1}^{\ell} m_i B^{i-1}$ and $s = \sum_{i=1}^{\ell} s_i B^{i-1}$, and a range proof which ensures that all ElGamal committed m_i and s_i lie in the range $[0, B-1]$. This approach improves efficiency considerably. For example for $B = 2^{32}$, we have $\ell = 8$ and require only 32 group elements for the ElGamal commitments, instead of 1024.

We instantiate the NIZK Π by composing two NIZKs together: one for proving the ElGamal commitments (Π_{ped}) and the other for the range proof (Π_{rp}). For the first NIZK Π_{ped}, we use the Fiat-Shamir transformation on an appropriate Σ-protocol Σ_{ped}. For the second NIZK Π_{rp} for the range proof, we would like to use the Fiat-Shamir transformation on a variant of Bulletproofs [BBB+18] that shows range membership for multiple Pedersen commitments ([AC20], Appendix F.2). However, Bulletproofs are not well-established in the non-interactive setting. Recently, [GOP+22] shows that non-interactive Bulletproofs are sound in the AGM and ROM, but as the proof relies on the AGM, it is not sufficient for our purpose. Attema, Fehr and Klooß show in another recent result [AFK22] that the Fiat-Shamir transformation is sound for multi-round interactive proof systems Σ_{int}, if Σ_{int} is standard *special* sound, i.e.given a valid *transcript tree*, it is possible to recover the witness unconditionally. Unfortunately, Bulletproofs and its variant [AC20] satisfy only *computational* special soundness which is insufficient to apply the result of [AFK22] directly. To this end, we show that for an appropriate *relaxed* special soundness relation, we can nonetheless apply the Fiat-Shamir transform on Bulletproofs and its variant [AC20] using the result of [AFK22]. While the resulting NIZK Π_{rp} for the range proof satisfies a relaxed notion of special soundness, this is sufficient for our purpose.

Equipped with the above tools, we can instantiate Π. We apply three further optimizations:

1. The (interactive) range proof of [AC20] requires the witness $\boldsymbol{e} = (m_1, \ldots, m_\ell, s_1, \ldots, s_\ell)$ to be committed in the Pedersen commitments. Note that we can reuse the ElGamal commitments E_i as Pedersen commitments for the range proof, where $(E_i = g_1^{e_i} \mathsf{pp}^{r_i}, R_i = g_1^{r_i})_{i\in[2\ell]}$ are already required for online extraction. However, as the extractor knows the trapdoor td such that $g_1^{\mathsf{td}} = \mathsf{pp}$, we need to be careful in the security analysis, as td also allows to equivocate Pedersen commitments (and thus potentially break soundness of the range proof). This subtlety is reflected in the relaxed soundness relation. Fortunately, we can analyze the extraction probability without knowing td and the proof goes through. We provide more details in section 7.5.

2. During extraction, we use a more efficient algorithm to compute the discrete logarithm in $\mathcal{O}(\sqrt{B})$. This allows us to choose better parameters, as a larger bound B improves efficiency but impacts the runtime of the extractor.

3. Observe that pairing groups are generally larger and slower than simple prime-order groups. Thus, we move the parts that are not required to be in $\mathbb{G}_1$ into a group $\widehat{\mathbb{G}}$ of the same order p as $\mathbb{G}_1$ (to maintain algebraic consistency). As both NIZKs Π_{ped} and Π_{rp} are not reliant on pairings, we can perform both almost exclusively in $\widehat{\mathbb{G}}$.

In the following, we first present the Σ-protocol Σ_{ped} (which we will later transform into a NIZK Π_{ped} via Fiat-Shamir) and the NIZK Π_{rp} for the range proof. We then combine both proof systems into an online-extractable NIZK Π for the relation R_{bb} and analyze its security. In more detail, when online-extraction of Π fails, it is easy to show that we can extract a witness from *at*

least one of the proofs generated by Π_{ped} and Π_{rp} by relying on the rewinding-extraction of Π_{ped} and the adaptive knowledge soundness of Π_{rp} (cf. definition 3.29) in an independent manner. What is non-trivial is to show that both extractions succeed *simultaneously*. Such simultaneous extraction is necessary since the two NIZKs are glued together by the Pedersen commitment $E_i = g_1^{e_i}\mathsf{pp}^{r_i}$: each NIZK may be using a different opening to construct the proof, in which case we need to extract from both proofs to break binding. For instance, using the standard notion of rewinding-extractability, we cannot exclude the case where the adversary sets up the proofs π_0, π_1 of $\Pi_{\mathsf{rp}}, \Pi_{\mathsf{ped}}$, respectively, in such a way that if the extractor of Π_{rp} succeeds, then the extractor of Π_{ped} fails. We show through a careful non-black analysis of the underlying NIZKs that there is a non-negligible probability of the rewinding-extraction of Π_{ped} and the adaptive knowledge soundness of Π_{rp} succeeding at the same time. Finally, we evaluate the efficiency of $\mathsf{BS}_{\mathsf{BB}}[\Pi]$. For readability, we mark elements $\widehat{g}$ in $\widehat{\mathbb{G}}$ with a hat.

Σ-protocol Σ_{ped} for the Decomposition

We first present a Σ-protocol Σ_{ped} for the relation $\mathsf{R}_{\mathsf{ped}}$ where

$$\mathsf{R}_{\mathsf{ped}} = \{(x,w) : c = u_1^m g_1^s, E_i = \widehat{g}^{e_i}\widehat{\mathsf{pp}}^{r_i}, R_i = \widehat{g}^{r_i}, \prod_{i\in[\ell]} E_i^{B^{i-1}} = \widehat{g}^m \cdot \widehat{\mathsf{pp}}^{t_m}, \prod_{i\in[\ell]} E_{i+\ell}^{B^{i-1}} = \widehat{g}^s \cdot \widehat{\mathsf{pp}}^{t_s}\},$$

where $x = (c, u_1, g_1, \widehat{g}, \widehat{\mathsf{pp}}, (E_i, R_i)_{i\in[2\ell]}, B)$ and $w = (m, s, (e_i, r_i)_{i\in[2\ell]}, t_m, t_s)$. Note that the relation shows that $m = \sum_{i=1}^{\ell} e_i B^{i-1}$ and $s = \sum_{i=1}^{\ell} e_{i+\ell} B^{i-1}$ under the DLOG assumption. The protocol is given below.

- $\Sigma_{\mathsf{ped}}.\mathsf{Init}(x,w)$: for (x,w) as above, samples additive masks $\widetilde{m}, \widetilde{s}, \widetilde{e}_i, \widetilde{r}_i, \widetilde{t}_m, \widetilde{t}_s \leftarrow \mathbb{Z}_p$, sets

$$D_c = u_1^{\widetilde{m}} g_1^{\widetilde{s}}, D_{e_i} = \widehat{g}^{\widetilde{e}_i}\widehat{\mathsf{pp}}^{\widetilde{r}_i}, D_{r_i} = \widehat{g}^{\widetilde{r}_i},$$
$$D_m = \widehat{g}^{\widetilde{m}}\widehat{\mathsf{pp}}^{\widetilde{t_m}}, D_s = \widehat{g}^{\widetilde{s}}\widehat{\mathsf{pp}}^{\widetilde{t_s}},$$

  where $i \in [2\ell]$, outputs $\alpha = (D_c, (D_{e_i}, D_{r_i})_{i\in[2\ell]}, D_m, D_s)$, and stores $(\widetilde{m}, \widetilde{s}, (\widetilde{e}_i, \widetilde{r}_i)_{i\in[2\ell]}, \widetilde{t}_m, \widetilde{t}_s)$ in st.

- $\Sigma_{\mathsf{ped}}.\mathsf{Chall}()$: samples a challenge $\beta \leftarrow \mathbb{Z}_p$,

- $\Sigma_{\mathsf{ped}}.\mathsf{Resp}(\mathsf{st}, \beta)$: sets $\gamma_k = \beta \cdot k + \widetilde{k}$ for $k \in \{m, s, e_1, ..., e_{2\ell}, r_1, ..., r_{2\ell}, t_m, t_s\}$, and outputs the response $\gamma = (\gamma_m, \gamma_s, (\gamma_{e_i}, \gamma_{e_i})_{i\in[2\ell]}, \gamma_{t_m}, \gamma_{t_s})$,

- $\Sigma_{\mathsf{ped}}.\mathsf{Verify}(x, \alpha, \beta, \gamma)$: checks the following equations

$$D_c = u_1^{\gamma_m} \cdot g_1^{\gamma_s} \cdot c^{-\beta}, D_{e_i} = \widehat{g}^{\gamma_{e_i}}\widehat{\mathsf{pp}}^{\gamma_{r_i}} \cdot E_i^{-\beta}, D_{r_i} = \widehat{g}^{\gamma_{r_i}} \cdot R_i^{-\beta},$$
$$D_m = \widehat{g}^{\gamma_m}\widehat{\mathsf{pp}}^{\gamma_{t_m}} \cdot (\prod_{i=1}^{\ell} E_i^{B^{i-1}})^{-\beta}, D_s = \widehat{g}^{\gamma_s}\widehat{\mathsf{pp}}^{\gamma_{t_s}} \cdot (\prod_{i=1}^{\ell} E_{i+\ell}^{B^{i-1}})^{-\beta},$$

  and outputs 1 if and only if all checks succeed. Note that the first three equations open the commitments c and (E_i, R_i), and the last two equations show the products hold. Also, observe that the equations are well-defined, as both $\widehat{\mathbb{G}}$ and $\mathbb{G}_1$ have order p.

We now show that Σ_{ped} is correct, HVZK, 2-special sound and has high min-entropy.

Theorem 7.19. *The Σ-protocol Σ_{ped} is correct.*

Proof. Let (α, β, γ) be a honest transcript for (x, w), where $x = (c, u_1, g_1, \widehat{g}, \widehat{\mathsf{pp}}, (E_i, R_i)_{i\in[2\ell]}, B)$ and $w = (m, s, (e_i, r_i)_{i\in[2\ell]}, t_m, t_s)$. We use the notation from above. We show that the first and fourth check pass, the remaining identities follow similarly.

$$u_1^{\gamma_m} \cdot g_1^{\gamma_s} \cdot c^{-\beta} = u_1^{\beta\cdot m+\widetilde{m}} \cdot g_1^{\beta\cdot s+\widetilde{s}} \cdot c^{-\beta} =$$
$$(u_1^m \cdot g_1^s)^\beta \cdot u_1^{\widetilde{m}} g_1^{\widetilde{s}} \cdot c^{-\beta} = c^\beta \cdot u_1^{\widetilde{m}} g_1^{\widetilde{s}} \cdot c^{-\beta} = D_c$$

$$\widehat{g}^{\gamma_m}\widehat{\mathsf{pp}}^{\gamma_{t_m}} \cdot (\prod_{i=1}^{\ell} E_i^{B^{i-1}})^{-\beta} = \widehat{g}^{\beta\cdot m+\widetilde{m}}\widehat{\mathsf{pp}}^{\beta\cdot t_m+\widetilde{t_m}} \cdot (\prod_{i=1}^{\ell} E_i^{B^{i-1}})^{-\beta} =$$
$$(\widehat{g}^m \cdot \widehat{\mathsf{pp}}^{t_m})^\beta \cdot \widehat{g}^{\widetilde{m}}\widehat{\mathsf{pp}}^{\widetilde{t_m}} \cdot (\prod_{i=1}^{\ell} E_i^{B^{i-1}})^{-\beta} = \widehat{g}^{\widetilde{m}}\widehat{\mathsf{pp}}^{\widetilde{t_m}} = D_m$$

□

Theorem 7.20. *The Σ-protocol Σ_{ped} is HVZK.*

Proof. Let $x = (c, u_1, g_1, \widehat{g}, \widehat{\mathsf{pp}}, (E_i, R_i)_{i\in[2\ell]}, B)$ and $\beta \in \mathbb{Z}_p$. We define the simulator Sim as follows. On input (x, β), samples $\gamma = (\gamma_m, \gamma_s, (\gamma_{e_i}, \gamma_{e_i})_{i\in[2\ell]}, \gamma_{t_m}, \gamma_{t_s}) \leftarrow \mathbb{Z}_p^{4+4\ell}$ and computes $(D_c, (D_{e_i}, D_{r_i})_{i\in[2\ell]}, D_m, D_s)$ via the identities in $\Sigma_{\mathsf{ped}}.\mathsf{Verify}$. Finally, sets $\alpha = (D_c, (D_{e_i}, D_{r_i})_{i\in[2\ell]}, D_m, D_s)$ and outputs the transcript (α, β, γ).

To show that Sim outputs transcripts that are indistinguishable from real transcripts, we define the following hybrids and denote by $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_i}(\lambda)$ the advantage of $\mathcal{A}$ in Hybrid i.

- Hybrid 0 outputs honestly generated transcripts.
- Hybrid 1 is the same as Hybrid 0, except the elements $(D_c, (D_{e_i}, D_{r_i})_{i\in[2\ell]}, D_m, D_s)$ are generated as in Sim. It is easy to check that Hybrid 0 and Hybrid 1 are identically distributed, and we have $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_0}(\lambda) = \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_1}(\lambda)$.
- Hybrid 2 is the same as Hybrid 1, except γ is computed as in Sim. As the values $\widetilde{k}$ serves as one-time pad for $\beta \cdot k$, where $k \in \{m, s, e_1, ..., e_{2\ell}, r_1, ..., r_{2\ell}, t_m, t_s\}$, it follows that Hybrid 1 and Hybrid 2 are identically distributed. Thus, we have $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_1}(\lambda) = \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_2}(\lambda)$.

As Hybrid 2 outputs simulated transcripts, the statement follows. □

Theorem 7.21. *The Σ-protocol Σ_{ped} is 2-special sound.*

Proof. Let $x = (c, u_1, g_1, \widehat{g}, \widehat{\mathsf{pp}}, (E_i, R_i)_{i\in[2\ell]}, B)$. We define the extractor Ext as follows. On input valid transcripts (α, β, γ) and $(\alpha, \beta', \gamma')$ with $\beta \neq \beta'$. Ext parses $\alpha = (D_c, (D_{e_i}, D_{r_i})_{i\in[2\ell]}, D_m, D_s)$ and $\gamma = (\gamma_m, \gamma_s, (\gamma_{e_i}, \gamma_{e_i})_{i\in[2\ell]}, \gamma_{t_m}, \gamma_{t_s}), \gamma' = (\gamma'_m, \gamma'_s, (\gamma'_{e_i}, \gamma'_{e_i})_{i\in[2\ell]}, \gamma'_{t_m}, \gamma'_{t_s})$. As both transcripts are valid, we obtain the following identities via the verification identities

$$u_1^{\gamma_m} \cdot g_1^{\gamma_s} \cdot c^{-\beta} = u_1^{\gamma'_m} \cdot g_1^{\gamma'_s} \cdot c^{-\beta'},$$
$$\widehat{g}^{\gamma_{e_i}}\widehat{\mathsf{pp}}^{\gamma_{r_i}} \cdot E_i^{-\beta} = \widehat{g}^{\gamma'_{e_i}}\widehat{\mathsf{pp}}^{\gamma'_{r_i}} \cdot E_i^{-\beta'},$$
$$\widehat{g}^{\gamma_{r_i}} \cdot R_i^{-\beta} = \widehat{g}^{\gamma'_{r_i}} \cdot R_i^{-\beta'},$$
$$\widehat{g}^{\gamma_m}\widehat{\mathsf{pp}}^{\gamma_{t_m}} \cdot (\prod_{i=1}^{\ell} E_i^{B^{i-1}})^{-\beta} = \widehat{g}^{\gamma'_m}\widehat{\mathsf{pp}}^{\gamma'_{t_m}} \cdot (\prod_{i=1}^{\ell} E_i^{B^{i-1}})^{-\beta'},$$
$$\widehat{g}^{\gamma_s}\widehat{\mathsf{pp}}^{\gamma_{t_s}} \cdot (\prod_{i=1}^{\ell} E_{i+\ell}^{B^{i-1}})^{-\beta} = \widehat{g}^{\gamma'_s}\widehat{\mathsf{pp}}^{\gamma'_{t_s}} \cdot (\prod_{i=1}^{\ell} E_{i+\ell}^{B^{i-1}})^{-\beta'},$$

We denote $\Delta k = (\gamma_k - \gamma'_k)$ and $k = \Delta k/\Delta\beta$ for $k \in \{m, s, e_1, ..., e_{2\ell}, r_1, ..., r_{2\ell}, t_m, t_s\}$ and $\Delta\beta = (\beta - \beta')$. Note that $\Delta\beta \neq 0$. The extractor finally outputs $(m, s, e_1, ..., e_{2\ell}, r_1, ..., r_{2\ell}, t_m, t_s)$.

From the first equation, we obtain $u_1^{\gamma_m - \gamma'_m} \cdot g_1^{\gamma_s - \gamma'_s} \cdot c^{-(\beta-\beta')} = 1_{\mathbb{G}_1}$. Taking both sides to the power of $1/\Delta\beta$ yields $u_1^{\Delta m/\Delta\beta} \cdot g_1^{\Delta s/\Delta\beta} = c$. By definition of m and s, we obtain $c = u_1^m g_1^s$ as desired. Similarly, we obtain from the second and third equation that $\widehat{g}^{e_i}\widehat{\mathsf{pp}}^{r_i} = E_i$ and $\widehat{g}^{r_i} = R_i$, respectively. The fourth equation yields that $\widehat{g}^{\gamma_m - \gamma'_m}\widehat{\mathsf{pp}}^{\gamma_{t_m} - \gamma'_{t_m}} \cdot (\prod_{i=1}^{\ell} E_i^{B^{i-1}})^{-(\beta-\beta')} = 1_{\widehat{\mathbb{G}}}$. Again, multiplying with $1/\Delta\beta$ in the exponent yields $\prod_{i\in[\ell]} E_i^{B^{i-1}} = \widehat{g}^m \cdot \widehat{\mathsf{pp}}^{t_m}$, and similarly the fifth equation yields $\prod_{i\in[\ell]} E_{i+\ell}^{B^{i-1}} = \widehat{g}^s \cdot \widehat{\mathsf{pp}}^{t_s}$. This concludes the proof. □

Theorem 7.22. *The Σ-protocol Σ_{ped} has high min-entropy.*

Proof. Observe that all D_{r_i} are distributed uniformly random in $\widehat{\mathbb{G}}$. It follows that the advantage of any adversary in the min-entropy game is at most $1/p^{\ell} = \mathsf{negl}(\lambda)$. □

Range Proof Π_{rp} for the Decomposition

We now describe the NIZK Π_{rp} for the range proof for the vectors committed in E_i. We start with an appropriate multi-round interactive proof system and obtain Π_{rp} via the Fiat-Shamir transformation. We follow the definitions for multi-round interactive proof systems and the notion of $(\boldsymbol{k}, \boldsymbol{N})$-special soundness with vectors $\boldsymbol{k}$ and $\boldsymbol{N}$ of [AFK22]. We use the definitions of correctness, HVZK of [AC20]. Since we can rely on the result of [AFK22, AC20] in a black-box manner, we refer the readers to [AFK22, AC20] for the formal definitions.

Let H_{rp} be a random oracle mapping into $\mathbb{Z}_p$ [11]. Π_{rp} is a NIZK with random oracle H_{rp} for the relation

$$\mathsf{R}_{\mathsf{rp}} = \{(x, w) : \ E_i = \widehat{g}^{e_i} \cdot \widehat{\mathsf{pp}}^{r_i}, e_i \in [0, B-1] \text{ for } i \in [2\ell]\},$$

with $x = (B, (E_i)_{i\in[2\ell]})$ and $w = ((e_i, r_i)_{i\in[2\ell]})$, where B is a power of two. We obtain Π_{rp} by applying the Fiat-Shamir transformation as described in [AFK22] to the multi-round interactive proof system $\Sigma_{\mathsf{rp}}^{2\ell}$ with $\mathsf{crs} = (\widehat{g}, \widehat{\mathsf{pp}}, (\widehat{g}_i)_{i\in[\ell_{\mathsf{rp}}]})$ from [AC20] (Appendix F.2), for appropriate $\ell_{\mathsf{rp}} \in \mathbb{N}$.

Denote with $\mathsf{R}_{\mathsf{dlog}} = \{(\mathsf{crs}, w^*)\}$ the relation that contains all non-trivial DLOG relations w^* for crs, i.e.computing w^* for random crs allows to solve the DLOG assumption (see [AC20] for more details). Via Theorem 14 of [AC20], we can show that Π_{rp} is correct and zero-knowledge. Moreover, we can show adaptive knowledge soundness for the relaxed relation

$$\mathsf{R}_{\mathsf{lax}} := \{(x, w) : \ (x, w) \in \mathsf{R}_{\mathsf{rp}} \text{ or } (\mathsf{crs}, w) \in \mathsf{R}_{\mathsf{dlog}}\},$$

using Theorem 4 of [AFK22], which is sufficient for our purpose. We sketch the proof below.

Theorem 7.23. *The NIZK Π_{rp} for relation R_{rp} is correct, zero-knowledge and adaptively knowledge sound for the relaxed relation $\mathsf{R}_{\mathsf{lax}} \supseteq \mathsf{R}_{\mathsf{rp}}$.*

Sketch. Correctness follows directly, as $\Sigma_{\mathsf{rp}}^{2\ell}$ is correct (Theorem 14, [AC20]) and the Fiat-Shamir transformation retains correctness of the interactive protocol. For showing zero-knowledge, observe that the intermediate prover outputs in $\Sigma_{\mathsf{rp}}^{2\ell}$ have high min-entropy. Thus, with all but negligible probability, these outputs were never queried to the random oracle. Consequently, the simulator can simulate a proof π by first simulating a transcript with challenge vector $\boldsymbol{\beta}$ using the HVZK property of $\Sigma_{\mathsf{rp}}^{2\ell}$, and then programming the random oracle with $\boldsymbol{\beta}$ accordingly.

[11] Note that technically, Π_{rp} requires a tuple of hash function $(\mathsf{H}_i)_{i\in[5+\mu]}$ mapping into $\mathbb{Z}_p$, where $\mu = \lceil \log_2(4\log_2(B)\ell + 4)\rceil - 1$. With sufficient input separation, we view $(\mathsf{H}_i)_{i\in[5+\mu]}$ as a single random oracle H_{rp}, for example if we query $\mathsf{H}_{\mathsf{rp}}(i, q)$ instead of $\mathsf{H}_i(q)$.

Note that [AFK22] shows *computational* special soundness of $\Sigma_{\mathsf{rp}}^{2\ell}$ for the relation R_{rp}. Here, computational special soundness means that either some w such that $(x, w) \in \mathsf{R}_{\mathsf{rp}}$ or w^* such that $(w^*, \mathsf{crs}) \in \mathsf{R}_{\mathsf{dlog}}$ is extracted from a transcript tree. The latter happens with negligible probability under the DLOG assumption, so w is extracted with overwhelming probability. However, to apply Theorem 4 from [AFK22] to Π_{rp}, we require standard special soundness of $\Sigma_{\mathsf{rp}}^{2\ell}$, and thus we make the witness w^* explicit in the relation $\mathsf{R}_{\mathsf{lax}}$. For the relation $\mathsf{R}_{\mathsf{lax}}$, the interactive proof system $\Sigma_{\mathsf{rp}}^{2\ell}$ is $(\boldsymbol{k}, \boldsymbol{N})$-special sound with vectors $\boldsymbol{k} = (2\ell+1, 4n\ell+1, 2n\ell+3, 2, 2, n_1, \ldots, n_\mu)$ and $\boldsymbol{N} = (N_i)_{i=1}^{5+\mu}$, where $n = \log_2(B), \mu = \lceil \log_2(4n\ell+4) \rceil - 1, n_i = 3$ and $N_i = p$. As [AFK22] never requires correctness or HVZK of the proof system, it is fine that R_{rp} and $\mathsf{R}_{\mathsf{lax}}$ are different. Thus, we can apply Theorem 4 in [AFK22] to show adaptive knowledge soundness [12]. Note that the knowledge error $\mathsf{Er}(\boldsymbol{k}, \boldsymbol{N})$ is negligible in λ, following the notation of [AFK22], because:

$$\begin{aligned}\mathsf{Er}(\boldsymbol{k}, \boldsymbol{N}) &= 1 - \prod_{i=1}^{5+\mu} (1 - \frac{k_i - 1}{N_i}) \\ &\leq 1 - \prod_{i=1}^{5+\mu} (1 - \frac{4n\ell}{p}) \\ &= \mathsf{negl}(\lambda)\end{aligned}$$

As we also have $\prod_{i=1}^{5+\mu} k_i \leq (5n\ell)^5 \cdot 3^{\log_2(8n\ell)} = \mathcal{O}((n\ell)^6) = \mathsf{poly}(\lambda)$, the extractor runs in time $\mathsf{poly}(\lambda)$ as desired. The statement follows. □

Online-extractable NIZK Π for R_{bb}

We are now ready to instantiate the online-extractable NIZK Π for the relation R_{bb} with $\mathsf{crs} = (\widehat{g}, \widehat{\mathsf{pp}}, (\widehat{g}_i)_{i \in [\ell_{\mathsf{rp}}]})$. Let H_{rp} be the random oracle of Π_{rp} and H_β be a random oracle mapping into $\mathbb{Z}_p$. We denote by $\mathsf{H}_{\mathsf{bb}} = (\mathsf{H}_{\mathsf{rp}}, \mathsf{H}_\beta)$ the random oracle of Π [13]. Let $B = \mathsf{poly}(\lambda)$ be a power of two. The scheme is given below.

– $\Pi.\mathsf{Prove}^{\mathsf{H}_{\mathsf{bb}}}(\mathsf{crs}, x, w)$: on input crs, $x = (c, u_1, g_1)$ and $w = (m, s)$, decomposes $m = \sum_{i=1}^{\ell} m_i B^{i-1}, s = \sum_{i=1}^{\ell} s_i B^{i-1}$, and computes $R_i = \widehat{g}^{r_i}, E_i = \widehat{g}^{e_i} \widehat{\mathsf{pp}}_i^{r_i}$ for $i \in [2\ell]$, where $\boldsymbol{e} = (m_1, \ldots, m_\ell, s_1, \ldots, s_\ell)$ and $r_i \leftarrow \mathbb{Z}_p$. Then, sets $t_m \leftarrow \sum_{i=1}^{\ell} r_i B^{i-1}$ and $t_s \leftarrow \sum_{i=1}^{\ell} r_{i+\ell} B^{i-1}$, and computes

$$\pi_0 \leftarrow \Pi_{\mathsf{rp}}.\mathsf{Prove}^{\mathsf{H}_{\mathsf{rp}}}(\mathsf{crs}, x_0, w_0),$$

for statement $x_0 = (B, (E_i)_{i \in [2\ell]})$ and witness $w_0 = ((e_i, r_i)_{i \in [2\ell]})$, and

$$\begin{aligned}&(\alpha, \mathsf{st}) \leftarrow \Sigma_{\mathsf{ped}}.\mathsf{Init}(x_1, w_1), \\ &\beta \leftarrow \mathsf{H}_\beta(x_1, \alpha), \\ &\gamma \leftarrow \Sigma_{\mathsf{ped}}.\mathsf{Resp}(x_1, \mathsf{st}, \beta), \\ &\pi_1 \leftarrow (\alpha, \beta, \gamma),\end{aligned}$$

for statement $x_1 = (c, u_1, g_1, \widehat{g}, \widehat{\mathsf{pp}}, (E_i, R_i)_{i \in [2\ell]}, B)$ and witness $w_1 = (m, s, (e_i, r_i)_{i \in [2\ell]}, t_m, t_s)$. Outputs $\pi = (\pi_0, \pi_1, (E_i, R_i)_{i \in [2\ell]})$.

[12] Technically, our definition of adaptive knowledge soundness (cf. definition 3.29) differs slightly from the definition in [AFK22]. Ours allows us to prove online-extractability for our construction Π later, and it is easy to check that the extractor of [AFK22] suffices for our definition (cf. remark 3, [AFK22]).

[13] Note that as in section 7.5, we can see H_{bb} as a single random oracle mapping into $\mathbb{Z}_p$ (to fit our definition). For readability, we allow H_{bb} to be a tuple of random oracles in the security proof.

– $\Pi.\mathsf{Verify}^{\mathsf{H}_{\mathsf{bb}}}(\mathsf{crs}, x, \pi)$: on input crs, $x = (c, u_1, g_1)$ and $\pi = (\pi_0, \pi_1, (E_i, R_i)_{i\in[2\ell]})$, checks

$$\begin{aligned}
&\Pi_{\mathsf{rp}}.\mathsf{Verify}^{\mathsf{H}_{\mathsf{rp}}}(\mathsf{crs}, x_0, \pi_0) = 1,\\
&\mathsf{H}_\beta(x_0, \alpha) = \beta' \ \wedge \ \beta = \beta',\\
&\Sigma_{\mathsf{ped}}.\mathsf{Verify}(x_1, \alpha, \beta, \gamma) = 1,
\end{aligned}$$

where $\pi_1 = (\alpha, \beta, \gamma)$ and x_0, x_1 are defined as above, and outputs 1 iff all checks succeed.

We show that Π is correct, zero-knowledge under the DDH assumption and online-extractable under the DLOG assumption. Correctness follows immediately from the correctness of Π_{rp} and Σ_{ped}. Also, zero-knowledge is easy to show via the hiding property of ElGamal commitments, the zero-knowledge property of Π_{rp} (cf. theorem 7.23) and the HVZK and high min-entropy property of Σ_{ped} (cf. theorems 7.20 and 7.22).

The proof for multi-proof extractability is more intricate. Roughly, the extractor embeds a trapdoor td for the commitment scheme in the crs. Then, given a statement-proof pair (x, π) with $x = (c, u_1, g_1)$ and $\pi = (\pi_0, \pi_1, (E_i, R_i)_{i\in[2\ell]})$, it decrypts the witnesses $(e_i)_i$ from the ElGamal commitment $(E_i, R_i)_i$ and tries to check if the extracted witness reconstructs to a witness in the relation R_{bb}. We expect that this is possible, as the range proof guarantees that the committed values are short and Σ_{ped} proves the linear relations in the exponents.

For the sake of exposition, below we only consider extracting from a single pair $(x, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ generated by some adversary $\mathcal{A}$. The argument generalizes to Q_S pairs in a straightforward manner. Note that (x, π) defines statement-proof pairs (x_0, π_1) for Π_{rp} and (x_1, π_1) for Σ_{ped} as in $\Pi.\mathsf{Verify}$.

For the sake of contradiction, let us assume that extraction fails. We first try to extract a witness $w_0 = (e_i', r_i')_i$ from π_0 via the knowledge extractor of Π_{rp}, and a witness $w_1 = (m, s, (e_i, r_i)_i)$ from π_1 from two related transcripts obtained via rewinding $\mathcal{A}$. Here, it is important that $\mathcal{A}$ is run with the same random tape $\mathsf{coin}_{\mathcal{A}}$ for both extractions to guarantee that the statements x_0 and x_1 share the commitments $(E_i)_i$. For now, let us assume that both extractions succeed, i.e.$(x_0, w_0) \in \mathsf{R}_{\mathsf{rp}}$ and $(x_1, w_1) \in \mathsf{R}_{\mathsf{ped}}$. Assuming the soundness of Π_{rp}, we have $e_i' \in [0, B-1]$. Moreover, assuming the soundness of the non-interactive Σ_{ped}, the extracted $(e_i)_i$ form the B-ary decomposition of a valid opening of c. Then, under the assumption that extraction fails, we must have $e_i' \neq e_i$ for some i. However, this breaks the binding property of the Pedersen commitment implicitly defined by the ElGamal commitments. In particular, we found a DLOG relation for the tuple $(\widehat{g}, \widehat{\mathsf{pp}}_i)$. Note that while the extracted DLOG relation is a trapdoor information td the extractor uses to extract the witnesses, this will not be an issue since we do not need td to analyze the success probability of the adversary.

It remains to show that extraction of w_0 and w_1 succeeds. Recall that we assumed that the extraction of w_0 and w_1 succeeds simultaneously, even if we initially run $\mathcal{A}$ on a shared random coin. But these events are dependent, and applying adaptive knowledge soundness of Π_{rp} and a general forking lemma independently is not sufficient. Instead, we first extract w_0 with the extractor of Π_{rp}. This step has a high success probability due to knowledge soundness of Π_{rp}. Then, we define a specialized forking algorithm that first runs $\mathcal{A}$ on the same randomness (and same initial random oracle choices), and then rewinds $\mathcal{A}$ to obtain related transcripts. Finally, a careful non-black box analysis of the forking algorithm, similar to [PS00], allows us to conclude that the algorithm succeeds in finding two related transcripts.

Theorem 7.24. *The* NIZK Π *is correct.*

Proof. By construction, it holds that $(x_0, w_0) \in \mathsf{R}_{\mathsf{rp}}$. Similarly, we have $(x_1, w_1) \in \mathsf{R}_{\mathsf{ped}}$, as

$$\begin{aligned}\prod_{i\in[\ell]} E_i^{B^{i-1}} &= \prod_{i\in[\ell]} (\widehat{g}^{e_i}\widehat{\mathsf{pp}}_i^{r_i})^{B^{i-1}} \\ &= \widehat{g}^{\sum_{i=1}^{\ell} m_i B^{i-1}} \cdot \widehat{\mathsf{pp}}^{\sum_{i=1}^{\ell} r_i B^{i-1}} \\ &= \widehat{g}^{m} \cdot \widehat{\mathsf{pp}}^{t_m},\end{aligned}$$

and similarly $\prod_{i\in[\ell]} E_{i+\ell}^{B^{i-1}} = \widehat{g}^{s} \cdot \widehat{\mathsf{pp}}^{t_s}$. □

Theorem 7.25. *The* NIZK Π *is zero-knowledge under the zero-knowledge property of* Π_{rp}*, the HVZK and high min-entropy property of* Σ_{ped}*, and under the* DDH *assumption in* $\widehat{\mathbb{G}}$*.*

Proof. Denote by $\mathsf{Sim}_0 = (\mathsf{Sim}_{\mathsf{H}_{\mathsf{rp}}}, \mathsf{Sim}_{\pi_0})$ the simulator of Π_{rp} and by $\Sigma_{\mathsf{ped}}.\mathsf{Sim}_1$ the simulator of Σ_{ped}. We define the simulator $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{H}_{\mathsf{bb}}}, \mathsf{Sim}_{\pi})$ of Π as follows. $\mathsf{Sim}_{\mathsf{H}_{\mathsf{bb}}}$ prepares an empty list L. For every new query q to the random oracle H_β, it returns a random element $\beta \leftarrow \mathbb{Z}_p$ and stores (q, β) in L, and answers old queries consistently via L. For every query q to the random oracle H_{rp}, it returns $\mathsf{Sim}_{\mathsf{H}_{\mathsf{rp}}}(q)$. Now, for each proof query $(x, w) \in \mathsf{R}_{\mathsf{bb}}$, Sim_π sets $(E_i, R_i) \leftarrow (\widehat{\mathsf{pp}}^{r_i}, \widehat{g}^{r_i})$ for random $r_i \leftarrow \mathbb{Z}_p$ and $i \in [2\ell]$, and prepares the two statements x_0, x_1 as in the real protocol. It then simulates $\pi_0 \leftarrow \mathsf{Sim}_{\pi_0}(\mathsf{crs}_0, x_0)$ and $\pi_0 = (\alpha, \beta, \gamma) \leftarrow \Sigma_{\mathsf{ped}}.\mathsf{Sim}_1(x_1, \beta)$, where $\beta \leftarrow \mathbb{Z}_p$. If H_β was already queried on input (x_1, α), then Sim_π outputs $\perp$. Otherwise, Sim_π outputs the simulated proof $\pi = (\pi_0, \pi_1, (E_i, R_i)_{i\in[2\ell]})$, and $\mathsf{Sim}_{\mathsf{H}_{\mathsf{bb}}}$ stores $((x_1, \alpha), \beta_1)$ in the list L.

Let $\mathcal{A}$ be PPT adversary on the zero-knowledge property of Π and let $Q_{\mathsf{H}_{\mathsf{rp}}}, Q_{\mathsf{H}_\beta}, Q_S$ denote the number of $\mathsf{H}_{\mathsf{rp}}, \mathsf{H}_\beta, \mathsf{Sim}_\pi$ queries, respectively. Without loss of generality, we assume that $\mathcal{A}$ never queries H_{rp} and H_β twice on the same input. We define the following hybrids and denote by $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_i}(\lambda)$ the probability that $\mathcal{A}$ outputs 1 in Hybrid i.

- Hybrid 0 is identical to real game, where proofs are honestly generated. Specifically, the proof oracle outputs on input (x, w) the value $\perp$ if $(x, w) \notin \mathsf{R}_{\mathsf{bb}}$, and else the value $\Pi.\mathsf{Prove}^{\mathsf{H}_{\mathsf{bb}}}(\mathsf{crs}, x, w)$. By definition, $\mathcal{A}$ outputs 1 with probability $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_0}(\lambda)$.

- Hybrid 1 is identical to Hybrid 0, except we simulate the proofs π_0 using the simulator $\mathsf{Sim}_0 = (\mathsf{Sim}_{\mathsf{H}_{\mathsf{rp}}}, \mathsf{Sim}_{\pi_0})$ of Π_{rp}. In more detail, for every query $(x, w) \in \mathsf{R}_{\mathsf{bb}}$, the challenger computes $(E_i, R_i)_{i\in[2\ell]}$ as before, but sets $\pi_0 \leftarrow \mathsf{Sim}_{\pi_0}(\mathsf{crs}, x_0)$ for $x_0 = (B, (E_i)_{i\in[2\ell]})$. The simulator still generates the proof π_1 honestly, and outputs $\pi = (\pi_0, \pi_1, (E_i, R_i)_{i\in[2\ell]})$. Similarly, for every query q to H_{rp}, outputs $\mathsf{Sim}_{\mathsf{H}_{\mathsf{rp}}}(q)$.

  We can construct an adversary $\mathcal{B}_{\Pi_{\mathsf{rp}}}$ against the zero-knowledge property of Π_{rp} with advantage $\mathsf{Adv}^{\mathrm{zk}}_{\mathcal{B}_{\Pi_{\mathsf{rp}}}} \geq |\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_0}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_1}(\lambda)|$. Concretely, $\mathcal{B}_{\Pi_{\mathsf{rp}}}$ challenges $\mathcal{A}$ and uses the provided oracles to generate the proofs π_0 and answer the H_{rp} queries. In the end, $\mathcal{B}_{\Pi_{\mathsf{rp}}}$ outputs the bit b received from $\mathcal{A}$. If the provided oracle generate real proofs, then $\mathcal{B}_{\Pi_{\mathsf{rp}}}$ simulates Hybrid 0 to $\mathcal{A}$, else it simulates Hybrid 1, and thus we have

  $$|\mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_0}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathrm{H}_1}(\lambda)| \leq \mathsf{Adv}^{\mathrm{zk}}_{\mathcal{B}_{\Pi_{\mathsf{rp}}}}.$$

- Hybrid 2 is the same as Hybrid 1, except for every query $(x, w) \in \mathsf{R}_{\mathsf{bb}}$, the simulator aborts if the random oracle H_β was already queried on its input when generating π_1. Concretely, after generating π_0 as in Hybrid 1, the simulator sets $(\alpha, \mathsf{st}) \leftarrow \Sigma_{\mathsf{ped}}.\mathsf{Init}(x_1, w_1)$ for (x_1, w_1) defined as before. It then draws $\beta \leftarrow \mathbb{Z}_p$ and finishes the generation of $\pi_1 = (\alpha, \beta, \gamma)$ via $\gamma \leftarrow \Sigma_{\mathsf{ped}}.\mathsf{Resp}(x_1, \mathsf{st}, \beta)$. Finally, it checks if (x_1, α) was already queried to H_β and aborts if so. Otherwise, it programs $\mathsf{H}_\beta(x_1, \alpha) \leftarrow \beta$.

Hybrids 1 and 2 differ only when the game aborts. Due to the high min-entropy of Σ_{ped}, the probability that the random oracle is already defined on input (x_1, α) is bounded by $Q_{\mathsf{H}_\beta} \cdot \mathsf{negl}(\lambda)$. Further, there are at most Q_S queries to Sim_π, and because $Q_S Q_{\mathsf{H}_\beta} \cdot \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)$, we have

$$|\mathsf{Adv}^{\mathrm{H}_1}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathrm{H}_2}_{\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda).$$

– Hybrid 3 is the same as Hybrid 2, except for every query $(x, w) \in \mathsf{R}_{\mathsf{bb}}$, the proofs π_1 are simulated without the witness w_1. That is, the simulator generates a simulated proof $\pi_1 = (\alpha, \beta, \gamma)$ by setting $\beta \leftarrow \mathbb{Z}_p$ and running $(\alpha, \gamma) \leftarrow \Sigma_{\mathsf{ped}}.\mathsf{Sim}(x_1, \beta)$, and programs H_β accordingly.

  We can construct an adversary $\mathcal{B}_{\Sigma_{\mathsf{ped}}}$ against the HVZK property of Σ_{ped}. Concretely, for every $(x, w) \in \mathsf{R}_{\mathsf{bb}}$, $\mathcal{B}_{\Sigma_{\mathsf{ped}}}$ obtains the transcript (α, β, γ). If (x_1, α) was already queried to H_β, then it aborts as in the previous game. Otherwise, it uses $\pi_1 = (\alpha, \beta, \gamma)$ to generate π as in Hybrid 2. If $\mathcal{B}_{\Sigma_{\mathsf{ped}}}$ receives simulated proofs, the game is distributed as in Hybrid 3, else it is distributed as in Hybrid 2. Consequently, we have

$$|\mathsf{Adv}^{\mathrm{H}_2}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathrm{H}_3}_{\mathcal{A}}(\lambda)| \leq Q_S \cdot \mathsf{Adv}^{\mathrm{hvzk}}_{\mathcal{B}_{\Sigma_{\mathsf{ped}}}}(\lambda).$$

– Hybrid 4 is the same as Hybrid 3, except for every query $(x, w) \in \mathsf{R}_{\mathsf{bb}}$, the commitments $(E_i, R_i)_{i \in [2\ell]}$ are commitments to 0.

  As the openings of $(E_i, R_i)_{i \in [2\ell]}$ are not required anymore for generating π_0 and π_1, we can construct an adversary $\mathcal{B}_{\mathsf{DDH}}$ against the hiding property of the ElGamal commitments (which holds under DDH). We thus have

$$|\mathsf{Adv}^{\mathrm{H}_3}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathrm{H}_4}_{\mathcal{A}}(\lambda)| \leq 2\ell \cdot \mathsf{Adv}^{\mathrm{ddh}}_{\mathcal{B}_{\mathsf{DDH}}}(\lambda)$$

Note that the description of the simulator in Hybrid 4 is identical to $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{H}_{\mathsf{bb}}}, \mathsf{Sim}_\pi)$. Collecting the above bounds yields the statement. □

Theorem 7.26. *The* NIZK Π *is multi-online extractable under adaptive knowledge soundness of* Π_{rp}*, the 2-special soundness property of* Σ_{ped}*, and under the* DLOG *assumption in* $\widehat{\mathbb{G}}$*.*

Proof. The simulator and extractor are given below. Roughly, the extractor extracts the B-ary decomposition of (m, s) from the commitments $(E_i, R_i)_{i \in [2\ell]}$, then recomputes and outputs (m, s) if $c = u_1^m g_1^s$. If the reconstruction of (m, s) fails or the supplied proof is invalid, it outputs $\perp$.

– $\mathsf{SimCRS}(1^\lambda)$: sets $\overline{\mathsf{crs}} = (\widehat{g}, \widehat{\mathsf{pp}}, (\widehat{g}_i)_{i \in [\ell_{\mathsf{rp}}]})$ with $\widehat{g}, \widehat{g}_i \leftarrow \widehat{\mathbb{G}}, \mathsf{td} \leftarrow \mathbb{Z}_p$ and $\widehat{\mathsf{pp}} \leftarrow \widehat{g}^{\mathsf{td}}$, and outputs $(\overline{\mathsf{crs}}, \mathsf{td})$,

– $\mathsf{Ext}(\overline{\mathsf{crs}}, \mathsf{td}, x, \pi)$: on input $\overline{\mathsf{crs}}$, trapdoor td, and proof $\pi = (\pi_0, \pi_1, (E_i, R_i)_{i \in [2\ell]})$ for statement $x = (c, u_1, g_1)$, the extractor first checks if π is valid via $\Pi.\mathsf{Verify}^{\mathsf{H}_{\mathsf{bb}}}(\overline{\mathsf{crs}}, x, \pi) = 1$. Then, it sets $F_i \leftarrow E_i \cdot R_i^{-\mathsf{td}}$ and checks that there is some $e_i \in [0, B-1]$ such that $F_i = \widehat{g}^{e_i}$. If so, it sets $m = \sum_{i=1}^{\ell} e_i B^{i-1}$ and $s = \sum_{i=1}^{\ell} e_{i+\ell} B^{i-1}$, checks $c = u_1^m \cdot g_1^s$, and finally outputs $w = (m, s)$. If any of the above checks fails, it outputs $\perp$.

Note that both SimCRS and Ext are PPT because the exponent of F_i can be brute-forced in polynomial time, as $B = \mathsf{poly}(\lambda)$. Also, if Ext does not output $\perp$, Ext is guaranteed to output w such that $(x, w) \in \mathsf{R}_{\mathsf{bb}}$ by construction.

We show that Π has CRS indistinguishability with the simulator SimCRS. Observe that $\mathsf{SimCRS}(1^\lambda)$ and the random variable $\mathsf{crs} \leftarrow (\widehat{g}, \widehat{\mathsf{pp}}, (\widehat{g}_i)_{i \in [\ell_{\mathsf{rp}}]})$ with $\widehat{g}, \widehat{\mathsf{pp}}, \widehat{g}_i \leftarrow \widehat{\mathbb{G}}$ are identically distributed. It follows that for any PPT adversary $\mathcal{A}$, we have $\mathsf{Adv}^{\mathrm{crs}}_{\mathcal{A}}(\lambda) = 0$.

It remains to show that Π is (multi-)online-extractable with the extractor Ext. We denote by H_0 the random oracle H_{rp} and by H_1 the random oracle H_β. Let $\mathcal{A}$ be an adversary making at most

Q_0, Q_1 queries to $\mathsf{H}_0, \mathsf{H}_1$ respectively. For $(\overline{\mathsf{crs}}, \mathsf{td}) \leftarrow \mathsf{SimCRS}(1^\lambda), \{(x_i, \pi_i)\}_{i\in[Q_S]} \leftarrow \mathcal{A}^{\mathsf{H}_{\mathsf{bb}}}(\overline{\mathsf{crs}})$, we denote by $w_i \leftarrow \mathsf{Ext}(\overline{\mathsf{crs}}, \mathsf{td}, x_i, \pi_i)$ the extracted witness from the i-th proof. We define the following events.

- Ver is the event that all statement-proof pairs verify correctly, i.e.for all $i \in [Q_S]$ it holds that $\Pi.\mathsf{Verify}^{\mathsf{H}_{\mathsf{bb}}}(\overline{\mathsf{crs}}, x_i, \pi_i) = 1$.
- Fail_i is the event that the extractor fails for input $(\overline{\mathsf{crs}}, \mathsf{td}, x_i, \pi_i)$ on a valid proof, i.e.we have $w_i = \bot$ but $\Pi.\mathsf{Verify}^{\mathsf{H}_{\mathsf{bb}}}(\overline{\mathsf{crs}}, x_i, \pi_i) = 1$.
- Fail is the event that the extractor fails for some valid proof (x_i, π_i), i.e.there exists some $i \in [Q_S]$ such that the event Fail_i occurs.

With this notation, let us assume that $\Pr[\mathsf{Ver}] \geq \mu(\lambda)$ for some $\mu(\lambda)$. Under the DLOG assumption in $\widehat{\mathbb{G}}$, we show in Lemma 7.27 that $\Pr[\mathsf{Fail}_i] = \mathsf{negl}(\lambda)$. Thus:

$$\Pr[\mathsf{Fail}] = \Pr[\exists i \in [Q_S], \mathsf{Fail}_i] \leq \sum_{i\in[Q_S]} \Pr[\mathsf{Fail}_i] = \mathsf{negl}(\lambda),$$

Thus, we have as desired:

$$\begin{aligned} &\Pr[\mathsf{Ver} \wedge \forall i \in [Q_S], (x_i, w_i) \in \mathsf{R}_{\mathsf{bb}}] = \Pr[\mathsf{Ver} \wedge \neg\mathsf{Fail}] \\ &= \Pr[\mathsf{Ver}] - \Pr[\mathsf{Ver} \wedge \mathsf{Fail}] \geq \mu(\lambda) - \Pr[\mathsf{Fail}] \\ &\geq \mu(\lambda) - \mathsf{negl}(\lambda). \end{aligned}$$

It remains to show Lemma 7.27. Recall that for all PPT adversaries $\mathcal{A}_{\mathsf{dl}}$, it holds that

$$\Pr[w^* \leftarrow \mathcal{A}_{\mathsf{dl}}(\mathsf{crs}) : (\mathsf{crs}, w^*) \in \mathsf{R}_{\mathsf{dlog}}] = \mathsf{negl}(\lambda) \tag{7.3}$$

under the DLOG assumption, where the probability is over the randomness of crs and the random coins of $\mathcal{A}_{\mathsf{dl}}$. Recall that the simulated $\overline{\mathsf{crs}}$ is distributed identically to a random crs, and as the ElGamal trapdoor td is never provided to $\mathcal{A}$, $\mathcal{A}$'s output is identically distributed on input crs and $\overline{\mathsf{crs}}$. Thus, we only need to analyze the probability of the event Fail_i for a random crs (without known trapdoor td). This is important because td itself provides a non-trivial DLOG relation, but this is the hard problem we want to solve using $\mathcal{A}$.

Lemma 7.27. *For some fixed $i \in [Q_S]$, we have $\Pr[\mathsf{Fail}_i] = \mathsf{negl}(\lambda)$ under the DLOG assumption.*

Proof. Assume that $\Pr[\mathsf{Fail}_i]$ is non-negligible. We construct an adversary $\mathcal{A}_{\mathsf{dl}}$ that on input a random crs finds a DLOG relation w^* in crs with non-negligible probability in polynomial time. When $\mathcal{A}$ on input crs outputs statement-proof pairs $\{(x'_j, \pi'_j)\}_{j\in[Q_S]}$ for NIZK Π, the i-th pair (x'_i, π'_i) contains a range proof π_0 for Π_{rp} (resp. a Fiat-Shamir proof π_1 for Σ_{ped}), and their statements x_0 (resp. x_1) can be recomputed given (x'_i, π'_i) as in $\Pi.\mathsf{Verify}$. In the analysis, we are interested in these proof-statement pairs and wish to extract the witness for both statements x_0 and x_1 simultaneously. We proceed as follows.

First, we define two wrapper algorithms $\mathcal{B}_0, \mathcal{B}_1$ of $\mathcal{A}$ that output the i-th statement-proof pair $(x_0, \pi_0), (x_1, \pi_1)$ output by $\mathcal{A}$ for $\Pi_{\mathsf{rp}}, \Sigma_{\mathsf{ped}}$, respectively. $\mathcal{B}_0$ and $\mathcal{B}_1$ are defined to run $\mathcal{A}$ with identical random $\mathsf{coin}_{\mathcal{A}}$ and random oracle outputs to ensure the statements x_0 and x_1 are consistent, i.e., contain the same commitments $(E_j)_j$. These algorithms will later be used to define our DLOG solver $\mathcal{A}_{\mathsf{dl}}$.

Description of Wrapper Algorithm $\mathcal{B}_b$. We denote by $\mathsf{coin}_{\mathcal{A}}$ the random coin of $\mathcal{A}$, and by $\vec{h}_0 = (\widehat{\beta}_{0,j})_{j\in[Q_0]} \in \mathbb{Z}_p^{Q_0}, \vec{h}_1 = (\widehat{\beta}_{1,j})_{j\in[Q_1]} \in \mathbb{Z}_p^{Q_1}$ the outputs of $\mathsf{H}_0, \mathsf{H}_1$, respectively. Note that for fixed $(\mathsf{crs}, \mathsf{coin}_{\mathcal{A}}, \vec{h}_0, \vec{h}_1)$, calling $\mathcal{A}$ is deterministic and the statement-proof pairs $\{(x'_j, \pi'_j)\}_{j\in[Q_S]} = \mathcal{A}^{\mathsf{H}_{\mathsf{bb}}}(\mathsf{crs}; \mathsf{coin}_{\mathcal{A}})$ are uniquely defined, where H_{bb} queries are answered via $\vec{h}_0$ and $\vec{h}_1$. We then define $\mathcal{B}_b$ as an algorithm that has oracle access to H_b (and a variant with fixed H_b outputs $\vec{h}_b$) as follows:

$\mathcal{B}_b^{\mathsf{H}_b}(\mathsf{crs}; \mathsf{coin}_b)$: On input $\mathsf{crs} = (\widehat{g}, \widehat{\mathsf{pp}}, (\widehat{g}_i)_{i\in[\ell_{\mathsf{rp}}]})$ and $\mathsf{coin}_b = (\mathsf{coin}_{\mathcal{A}}, \vec{h}_{1-b})$, $\mathcal{B}_b$ runs $\mathcal{A}$ on input crs with fixed randomness $\mathsf{coin}_{\mathcal{A}}$, where for the j-th query to H_{1-b}, it outputs the j-th value $\widehat{\beta}_{1-b,j}$ in $\vec{h}_{1-b}$, and it simulates H_b with the provided oracle. After obtaining $\{(x'_j, \pi'_j)\}_{j\in[Q_S]} = \mathcal{A}(\mathsf{crs}, \mathsf{coin}_{\mathcal{A}})$ from $\mathcal{A}$, it checks if $\Pi.\mathsf{Verify}^{\mathsf{H}_{\mathsf{bb}}}(\mathsf{crs}, x'_i, \pi'_i) = 1$, i.e., the i-th proof verifies correctly. Then, parses $x'_i = (c, u_1, g_1)$ and $\pi'_i = (\pi_0, \pi_1, (E_j, R_j)_{j\in[2\ell]})$, and sets $x_0 = (B, (E_j)_{j\in[2\ell]})$ and $x_1 = (c, u_1, g_1, \widehat{g}, \widehat{\mathsf{pp}}, (E_j, R_j)_{j\in[2\ell]}, B)$. If any check fails, outputs $(\bot, \bot)$. Otherwise, for $b = 0$, outputs (x_0, π_0). For $b = 1$, parses $\pi_1 = (\alpha, \beta, \gamma)$, and looks for the index I such that $\beta = \widehat{\beta}_{1,I} = \mathsf{H}_1(x_1, \alpha)$, and finally outputs (I, Λ) with $\Lambda = (x_1, \alpha, \beta, \gamma)$. Note that without loss of generality, the index I is well defined, as guessing β correctly without querying H_1 on input (x_1, α) happens with probability at most $1/p = \mathsf{negl}(\lambda)$.

$\mathcal{B}_b(\mathsf{crs}, \vec{h}_b; \mathsf{coin}_b)$: runs $\mathcal{B}_b^{\mathsf{H}_b}(\mathsf{crs}; \mathsf{coin}_b)$, where the j-th query to H_b is answered with the j-th value $\widehat{\beta}_{b,j}$ of $\vec{h}_b$. Note that $\mathcal{B}_b$ is deterministic on input $(\mathsf{crs}, \vec{h}_b; \mathsf{coin}_b)$.

Description of Forking Algorithm $\mathsf{F}_{\mathcal{B}_1}$. We now define a variant $\mathsf{F}_{\mathcal{B}_1}$ of the standard forking algorithm that rewinds $\mathcal{B}_1$ until a related transcript is found. Concretely, $\mathsf{F}_{\mathcal{B}_1}$ takes as input $(\mathsf{crs}, \mathsf{coin}_1, \vec{h}_1)$, and invokes $\mathcal{B}_1$ internally as depicted in algorithm 13. Note that the standard forking algorithm chooses coin_1 and initial hash values $\vec{h}_1$ at random, whereas $\mathsf{F}_{\mathcal{B}_1}$ receives some fixed initial choice of coin_1 and $\vec{h}_1$. In $\mathcal{A}_{\mathsf{dl}}$, we will initialize the input of $\mathsf{F}_{\mathcal{B}_1}$ with the choices made by the extractor Ext_0 of Π_{rp}.

We expect that after at most $T = \mathsf{poly}(\lambda)$ calls to $\mathcal{B}_1$, the forking algorithm is successful with non-negligible probability, i.e., it outputs some non-$\bot$. Note that for inputs drawn independently and uniformly at random, the classical analysis of the forking algorithm yields the desired result. But here, the inputs are conditioned on the event that the extractor Ext_0 is successful on input (x_0, π_0), and we cannot apply the classical result directly. Thus, we analyze the success probability of $\mathsf{F}_{\mathcal{B}_1}$ later. Nevertheless, if $\mathsf{F}_{\mathcal{B}_1}$ is successful, it outputs related transcripts for the statement x_1, which is fixed via the initial run of $\mathcal{A}$ in $\mathcal{B}_1$.

Algorithm 13 Description of the forking algorithm $\mathsf{F}_{\mathcal{B}_1}(\mathsf{crs}, \mathsf{coin}_1, \vec{h}_1)$

```
(I, Λ) ← B_1(crs, h_1; coin_1)
if Λ = ⊥ then
    return ⊥                                   ▷ Return fail.
for c ∈ [T] do
    h_{1,≥I}^{(c)} ← Z_p^{Q_1 − I + 1}
    h_1^{(c)} := h_{1,<I} ‖ h_{1,≥I}^{(c)}
    (I^{(c)}, Λ^{(c)}) ← B_1(crs, h_1^{(c)}; coin_1)
    if I^{(c)} = I then                        ▷ Found related transcript.
        return (Λ, Λ^{(c)})
return ⊥                                       ▷ Return fail.
```

Description of the DLOG *Adversary* $\mathcal{A}_{\mathsf{dl}}$. We are now ready to define $\mathcal{A}_{\mathsf{dl}}$. Roughly, $\mathcal{A}_{\mathsf{dl}}$ samples inputs (incl. randomness) for both wrapping algorithms $(\mathcal{B}_0, \mathcal{B}_1)$ such that they output proofs

(π_0, π_1) for statements (x_0, x_1) with shared commitments $(E_j)_{j\in[2\ell]}$. Then, it invokes the extractor Ext_0 of Π_{rp} on (x_0, π_0) to extract a witness w_0 for the i-th proof output by $\mathcal{A}$. Next, it rewinds $\mathcal{B}_0$ with matching initial inputs via the forking algorithm $\mathsf{F}_{\mathcal{B}_1}$ to obtain a related transcript for (x_1, π_1), and extracts witness w_1 via 2-special soundness. If both extractions were successful, $\mathcal{A}_{\mathsf{dl}}$ can compute a DLOG relation in crs conditioned on event Fail_i, as x_0 and x_1 share the commitment $(E_j)_{j\in[2\ell]}$ and the parameter B by construction. We denote by Ext_0 the extractor of Π_{rp} (cf. Definition 3.29) and by Ext_1 the extractor of Σ_{ped} (cf. Definition 3.25). We now describe $\mathcal{A}_{\mathsf{dl}}$.

$\mathcal{A}_{\mathsf{dl}}(\mathsf{crs})$: On input crs, $\mathcal{A}_{\mathsf{dl}}$ prepares a list $\vec{h}_b \leftarrow \mathbb{Z}_p^{Q_b}$ of initial responses for H_b queries, where $b \in \{0,1\}$. Then, she draws some random $\mathsf{coin}_{\mathcal{A}}$ for $\mathcal{A}$, and initializes for $b \in \{0,1\}$ the randomness of $\mathcal{B}_b$ via $\mathsf{coin}_b = (\mathsf{coin}_{\mathcal{A}}, \vec{h}_{1-b})$. Then, $\mathcal{A}_{\mathsf{dl}}$ runs $(x_0, \pi_0) \leftarrow \mathcal{B}_0(\mathsf{crs}, \vec{h}_0; \mathsf{coin}_0)$, extracts a witness $w_0 \leftarrow \mathsf{Ext}_0(\mathsf{crs}, x_0, \pi_0, \mathsf{coin}_0, \vec{h}_0)$, and checks $(x_0, w_0) \in \mathsf{R}_{\mathsf{lax}}$. Next, she runs $R \leftarrow \mathsf{F}_{\mathcal{B}_1}(\mathsf{crs}, \mathsf{coin}_1, \vec{h}_1)$, checks $R \neq \bot$, and parses $R = \{(x_1, \pi_1), (x_1, \pi_2)\}$ with $\pi_1 = (\alpha, \beta, \gamma), \pi_2 = (\alpha, \beta', \gamma')$ and $\beta \neq \beta$. Then, she extracts $w_1 \leftarrow \mathsf{Ext}_1(x_1, \pi_1, \pi_2)$. Note that by construction, if $R \neq \bot$, we have $(x_1, w_1) \in \mathsf{R}_{\mathsf{ped}}$ under 2-special soundness of Σ_{ped} (see Lemma 7.29 for more details). Next, it parses the statements $x_0 = (B', (E'_j)_{j\in[2\ell]})$ and $x_1 = (c, u_1, g_1, \hat{g}, \widehat{\mathsf{pp}}, (E_j, R_j)_{j\in[2\ell]}, B)$. She outputs $\bot$ if any check fails. As the initial run of $\mathcal{A}$ in $\mathcal{B}_0(\mathsf{crs}, \vec{h}_0; \mathsf{coin}_0)$ and $\mathsf{F}_{\mathcal{B}_1}(\mathsf{crs}, \mathsf{coin}_1, \vec{h}_1)$ are identical, we have $B = B'$ and $E'_j = E_j$ for all $j \in [2\ell]$. Next, $\mathcal{A}_{\mathsf{dl}}$ computes a DLOG relation w^* as follows:

First, if $(\mathsf{crs}, w_0) \in \mathsf{R}_{\mathsf{dlog}}$, sets $w^* = w_0$, else parses $w_0 = ((e'_j, r'_j)_{j\in[2\ell]})$ and $w_1 = (m, s, (e_j, r_j)_{j\in[2\ell]}, t_m, t_s)$. Next, if there is some $j \in [2\ell]$ with $e'_j \neq e_j$, the binding property of the Pedersen commitment E_j is broken, and $w^* \leftarrow (e_j - e'_j)/(r'_j - r_j) \bmod p$ yields a non-trivial DLOG relation in crs, as $\widehat{\mathsf{pp}} = \hat{g}^{w^*}$. Further, if $m \neq \sum_{i=1}^{\ell} e_i B^{i-1}$, then $w^* \leftarrow (m - \sum_{i=1}^{\ell} m_i B^{i-1})/(\sum_{i=1}^{\ell} r_i B^{i-1} - t_m)$ yields a DLOG relation due to the following:

$$
\begin{aligned}
&\prod_{i\in[\ell]} E_i^{B^{i-1}} = \hat{g}^m \cdot \widehat{\mathsf{pp}}^{t_m} \\
\implies &\prod_{i\in[\ell]} (\hat{g}^{e_i} \widehat{\mathsf{pp}}^{r_i})^{B^{i-1}} = \hat{g}^m \cdot \widehat{\mathsf{pp}}^{t_m} \\
\implies &\hat{g}^{\sum_{i\in[\ell]} e_i B^{i-1}} \widehat{\mathsf{pp}}^{\sum_{i\in[\ell]} r_i B^{i-1}} = \hat{g}^m \cdot \widehat{\mathsf{pp}}^{t_m} \\
\implies &\hat{g}^{\sum_{i\in[\ell]} e_i B^{i-1} - m} = \widehat{\mathsf{pp}}^{t_m - \sum_{i\in[\ell]} r_i B^{i-1}} \\
\implies &\hat{g}^{(\sum_{i\in[\ell]} e_i B^{i-1} - m)/(t_m - \sum_{i\in[\ell]} r_i B^{i-1})} = \widehat{\mathsf{pp}},
\end{aligned}
$$

where the first equality is due to $(x_1, w_1) \in \mathsf{R}_{\mathsf{ped}}$. A similar calculation shows that $\mathcal{A}_{\mathsf{dl}}$ can compute a DLOG relation w^* if $s \neq \sum_{i=1}^{\ell} e_{i+\ell} B^{i-1}$.

In summary, $\mathcal{A}_{\mathsf{dl}}$ succeeds extracting a DLOG relation w^* if the extraction of both $(x_b, \pi_b)_{b\in\{0,1\}}$ succeeds and the extracted witness reconstructs to a witness *not* in R_{bb}. Otherwise, if either extraction fails or the extracted witness reconstructs to a witness in R_{bb}, she outputs $\bot$.

Analysis of the Success Probability of $\mathcal{A}_{\mathsf{dl}}$. We finally analyze the probability that $\mathcal{A}_{\mathsf{dl}}$ outputs a DLOG relation in crs conditioned on event Fail_i. If the probability is non-negligible, we conclude $\Pr[\mathsf{Fail}_i] = \mathsf{negl}(\lambda)$ as desired under the hardness of DLOG. Below, for simplicity, we omit the subscript and use Fail for Fail_i.

First, notice that conditioned on the event Fail, $\mathcal{A}_{\mathsf{dl}}$ cannot extract a witness that reconstructs to a witness in R_{bb}. Therefore, $\mathcal{A}_{\mathsf{dl}}$ outputs $\bot$ if and only if extraction fails, and on the other hand, when it outputs a non-$\bot$, then this always results in a DLOG relation in crs as desired. Thus, we only need to prove that $\mathcal{A}_{\mathsf{dl}}$ outputs a non-$\bot$ with non-negligible probability — or

equivalently, succeeds extracting from both (x_0, π_0) and (x_1, π_1) with non-negligible probability in polynomial time — to conclude the proof.

We first prove the following lemma which states that if event Fail occurs then $\mathcal{A}_{\mathsf{dl}}$ succeeds in extracting a witness from (x_0, π_0) with non-negligible probability. We later analyze the probability that $\mathcal{A}_{\mathsf{dl}}$ further succeeds in extracting a witness from (x_1, π_1).

Lemma 7.28. *We have* $\Pr[(x_0, w_0) \in \mathsf{R}_{\mathsf{lax}} \wedge \mathsf{Fail}] \geq \varepsilon$ *under adaptive knowledge soundness of* Π_{rp}*, where* $\varepsilon = (\Pr[\mathsf{Fail}] - \mathsf{negl}(\lambda))/p_{\mathsf{P}}(\lambda, Q_0)$ *and* p_{P} *is the polynomial in definition 3.29. Here, the probability is taken over the randomness of* crs *and those used by* $\mathcal{A}_{\mathsf{dl}}$*.*

Proof. The statement follows from adaptive knowledge soundness of Π_{rp} if we restrict $\mathcal{B}_0$ to only output proofs if extraction of (x_i', π_i') fails. As checking this condition requires knowledge of the trapdoor td of crs, we analyze the probability for some $\overline{\mathsf{crs}}$ with known td. The statement follows as crs and $\overline{\mathsf{crs}}$ are identically distributed.

In more detail, we define a wrapper algorithm $\mathcal{B}$ of $\mathcal{B}_0$. $\mathcal{B}$ samples $(\overline{\mathsf{crs}}, \mathsf{td}) \leftarrow \mathsf{SimCRS}(1^\lambda)$ and initializes coin_b and $\vec{h}_0$ as in $\mathcal{A}_{\mathsf{dl}}$. Then, it runs $(x_{\mathcal{B}}, \pi_{\mathcal{B}}) \leftarrow \mathcal{B}_0(\overline{\mathsf{crs}}, \vec{h}_0; \mathsf{coin}_0)$, and outputs $(x_{\mathcal{B}}, \pi_{\mathcal{B}})$ if $\bot = \mathsf{Ext}(\overline{\mathsf{crs}}, \mathsf{td}, x_i', \pi_i')$, where (x_i', π_i') is the i-th statement-proof pair output by $\mathcal{A}$ in $\mathcal{B}_0$. Note that by definition of $\mathcal{B}_0$, if $(x_{\mathcal{B}}, \pi_{\mathcal{B}}) \neq (\bot, \bot)$, then we also have $\Pi_{\mathsf{rp}}.\mathsf{Verify}^{\mathsf{H}_0}(\overline{\mathsf{crs}}, x_{\mathcal{B}}, \pi_{\mathcal{B}}) = 1$. Thus, under adaptive knowledge soundness of Π_{rp}, we have

$$\Pr[(x_{\mathcal{B}}, w_{\mathcal{B}}) \in \mathsf{R}_{\mathsf{lax}} \wedge \bot = \mathsf{Ext}(\overline{\mathsf{crs}}, \mathsf{td}, x_i', \pi_i')] \geq \varepsilon$$

for $w_{\mathcal{B}} \leftarrow \mathsf{Ext}_0(\overline{\mathsf{crs}}, x_{\mathcal{B}}, \pi_{\mathcal{B}}, \mathsf{coin}_0, \vec{h}_0)$. As crs and $\overline{\mathsf{crs}}$ are identically distributed, the pair (x_0, w_0) in $\mathcal{A}_{\mathsf{dl}}$ is identically distributed to the output of $\mathcal{B}$ conditioned on the event Fail. Thus, a quick calculation yields as desired

$$\Pr[(x_0, w_0) \in \mathsf{R}_{\mathsf{lax}} \wedge \mathsf{Fail}] \geq \Pr[(x_{\mathcal{B}}, w_{\mathcal{B}}) \in \mathsf{R}_{\mathsf{lax}} \wedge \bot = \mathsf{Ext}(\overline{\mathsf{crs}}, \mathsf{td}, x_i', \pi_i')] \geq \varepsilon.$$

This completes the proof. □

It remains to analyze the probability that $\mathcal{A}_{\mathsf{dl}}$ extracts $(x_1, w_1) \in \mathsf{R}_{\mathsf{ped}}$ with non-negligible probability, conditioned on $(x_0, w_0) \in \mathsf{R}_{\mathsf{lax}}$ and Fail. We stress that the two extractions are not independent, as the same random $\mathsf{coin}_{\mathcal{A}}$ and initial hash values $\vec{h}_0, \vec{h}_1$ are used for both extractions, so a non-black-box analysis is required. In Lemma 7.29, we show that the events $(x_0, w_0) \in \mathsf{R}_{\mathsf{lax}}, (x_1, w_1) \in \mathsf{R}_{\mathsf{ped}}$ and Fail occur after a polynomial number of forking steps in $\mathsf{F}_{\mathcal{B}_1}$ with non-negligible probability. Combining this with lemma 7.28, we conclude that $\mathcal{A}_{\mathsf{dl}}$ succeeds extracting from both (x_0, π_0) and (x_1, π_1) with non-negligible probability in polynomial time.

Lemma 7.29. *For* $T = 4Q_1/\varepsilon$*, we have* $\Pr[(x_0, w_0) \in \mathsf{R}_{\mathsf{lax}} \wedge (x_1, w_1) \in \mathsf{R}_{\mathsf{ped}} \wedge \mathsf{Fail}] \geq \frac{\varepsilon}{8} - \mathsf{negl}(\lambda)$*, and the runtime of* $\mathsf{F}_{\mathcal{B}_1}$ *is at most* $(4Q_1/\varepsilon) \cdot \mathsf{Time}(\mathcal{A}) = \mathsf{poly}(\lambda)$*. Here, the probability is taken over the randomness of* crs *and those used by* $\mathcal{A}_{\mathsf{dl}}$*.*

Proof. Denote by E the event that $(x_0, w_0) \in \mathsf{R}_{\mathsf{lax}}$ and the event Fail occurs, i.e.the extraction of Π_{rp} succeeds but online-extraction of Π failed for the i-th proof. Note that $\Pr[\mathsf{E}] \geq \varepsilon$ is non-negligible (cf. Lemma 7.28). We show that even though the forking algorithm uses the same initial randomness, it also holds that $\Pr[\mathsf{E} \wedge R \neq \bot]$ with non-negligible probability, where $R \leftarrow \mathsf{F}_{\mathcal{B}_1}(\mathsf{crs}, \mathsf{coin}_1, \vec{h}_1)$ is the output of $\mathsf{F}_{\mathcal{B}_1}$ in $\mathcal{A}_{\mathsf{dl}}$. This directly yields $(x_1, w_1) \in \mathsf{R}_{\mathsf{ped}}$ as follows:

$\mathsf{F}_{\mathcal{B}_1}$ runs $\mathcal{B}_1$ with identical randomness until the I-th H_1 query and it outputs statement-transcript pairs (x_1, π_1) and (x_1', π_2) for Σ_{ped}, each associated to the I-th H_1 query. Thus, we have that $\alpha = \alpha'$ and $x_1 = x_1'$ for $\pi_1 = (\alpha, \beta, \gamma), \pi_2 = (\alpha', \beta', \gamma')$. Also, we have that $\beta' \neq \beta$ except with negligible probability, as hash outputs are sampled uniformly and independently at random, in which case the extractor Ext_1 succeeds, i.e.$(x_1, w_1) \in \mathsf{R}_{\mathsf{ped}}$.

We are now ready to analyze the probability that $\Pr[\mathsf{E} \wedge R \neq \bot]$. The argument follows the high level structure of the proof of the forking lemma in [PS00]. First, denote by E_k the event

that π_1 is associated to the k-th H_1 query, i.e.the k-th entry $\widehat{\beta}_{1,k}$ of $\vec{h}_1$ is equal to β. Next, we define the set P as

$$P = \left\{ k \;\middle|\; \Pr[\mathsf{E}_k \mid \mathsf{E}] \geq \frac{1}{2Q_1} \right\}.$$

Note that for any $k \in P$, we have $\Pr[\mathsf{E}_k] \geq \frac{\epsilon}{2Q_1}$. Further, for the event $\mathsf{E}_1^{\mathsf{good}} = \bigvee_{k \in P} \mathsf{E}_k$, we have

$$\Pr\left[\mathsf{E}_1^{\mathsf{good}} \;\middle|\; \mathsf{E}\right] = \sum_{k \in P} \Pr\left[\mathsf{E}_k \mid \mathsf{E}\right] \geq \frac{Q_1}{2Q_1} = \frac{1}{2}. \tag{7.4}$$

We define the set $X_k = (R_{\mathcal{A}} \times R_{\mathsf{Ext}_0} \times \mathbb{Z}_p^{Q_0}) \times \mathbb{Z}_p^{k-1}$ and $Y_k = \mathbb{Z}_p^{Q_1-k+1}$, where $R_{\mathcal{A}}$ (resp. R_{Ext_0}) denotes the randomness space of $\mathcal{A}$ (resp. Ext_0). Note that for fixed crs, the tuple $(x, y) \in X_k \times Y_k$ can be parsed to define all inputs of $\mathcal{A}_{\mathsf{dl}}$, including randomness except the random choices in $\mathsf{F}_{\mathcal{B}_1}$. In more detail, parse $x = (\mathsf{coin}_{\mathcal{A}}, \mathsf{coin}_{\mathsf{Ext}_0}, \vec{h}_0, \vec{h}_{1,<k})$ and $y = \vec{h}_{1,\geq k}$. Set $\vec{h}_1 = (\vec{h}_{1,<k} \| \vec{h}_{1,\geq k})$. Note that given $\mathsf{coin}_{\mathsf{Ext}_0}$, $\mathcal{A}_{\mathsf{dl}}$ runs $w_0 = \mathsf{Ext}_0(\mathsf{crs}, x_0, \pi_0, \mathsf{coin}_0, \vec{h}_0; \mathsf{coin}_{\mathsf{Ext}_0})$ with randomness $\mathsf{coin}_{\mathsf{Ext}_0}$. Then, the execution of $\mathcal{A}_{\mathsf{dl}}$ on input crs and randomness (x, y) is deterministic up to the run of $\mathsf{F}_{\mathcal{B}_1}$.

Further, note that given $(x, y) \in X_k \times Y_k$, it can be determined whether E_k occurred. We define $A_k \subseteq X_k \times Y_k$ as the set of such inputs, i.e.$(x, y) \in A_k$ iff (x, y) triggers E_k. Then, the splitting lemma (cf. Lemma 3.3) with $\alpha = \frac{\varepsilon}{4Q_1}$ yields that that for the set $B_k \subseteq X_k \times Y_k$ defined

$$B_k = \left\{ (x, y) \in X_k \times Y_k \;\middle|\; \Pr_{y' \leftarrow Y_k}\left[(x, y') \in A_k\right] \geq \frac{\varepsilon}{4Q_1} \right\}, \tag{7.5}$$

we have

$$\Pr_{(x,y) \leftarrow X_k \times Y_k}\left[(x, y) \in B_k \mid (x, y) \in A_k\right] \geq \frac{1}{2}. \tag{7.6}$$

We are now ready to evaluate the probability of $(x_0, w_0) \in \mathsf{R}_{\mathsf{rp}}, (x_1, w_1) \in \mathsf{R}_{\mathsf{ped}}$ and Fail, when $\mathcal{A}_{\mathsf{dl}}$ is run with initial randomness (x, y). As shown above, this event occurs if both E and $R \neq 0$, i.e.the the run of $\mathsf{F}_{\mathcal{B}_1}$ outputs some non-$\perp$.

With probability ε, we have that E occurs. As in that case, the i-th proof output by $\mathcal{A}$ verifies, we further have that the initial run of $\mathcal{B}_1$ in $\mathsf{F}_{\mathcal{B}_1}$ produces some $(I, \Lambda) \neq \perp$. Then, the probability that $\mathsf{E}_1^{\mathsf{good}}$ occurs is at least $\Pr\left[\mathsf{E}_1^{\mathsf{good}} \;\middle|\; \mathsf{E}\right] \geq \frac{1}{2}$ due to eq. (7.4). In that case, we have $(x, y) \in A_I$ by definition. Then, from eq. (7.6) we have that $(x, y) \in B_k$ with probability at least $\frac{1}{2}$. Thus, eq. (7.5) yields that the probability that $\mathsf{F}_{\mathcal{B}_1}$ resamples some $y' \in Y_I$ with $(x, y') \in A_I$ is at least $\frac{\varepsilon}{4Q_1}$ conditioned on $(x, y) \in B_k$.

In words, in the c-th iteration in $\mathsf{F}_{\mathcal{B}_1}$, the sampled $\vec{h}_{1,\geq I}^{(c)} \leftarrow \mathbb{Z}_p^{Q_1-I+1}$ triggers E_I, as $Y_I = \mathbb{Z}_p^{Q_1-I+1}$. This means that $\mathcal{A}_{\mathsf{dl}}$ computes some π_1 that is associated to I-th H_1 query on input $(x, \vec{h}_{1,\geq I}^{(c)})$. Equivalently, we have $I^{(c)} = I$ on original input (x, y) and thus $R \neq \perp$.

Note that all $\vec{h}_{1,\geq I}^{(c)} \leftarrow \mathbb{Z}_p^{Q_1-I+1}$ are sampled independently and uniformly at random in $\mathsf{F}_{\mathcal{B}_1}$ for $c \in [T]$, where $T = 4Q_1/\varepsilon$. Thus, conditioned on $\mathcal{A}_{\mathsf{dl}}$ sampling some $(x, y) \in B_k$, the probability that $R \neq \perp$ is at least

$$1 - \left(1 - \frac{\varepsilon}{4Q_1}\right)^{\frac{4Q_1}{\varepsilon}} \geq 1 - \frac{1}{e} \geq \frac{1}{2}$$

Collecting all the bounds, we conclude that $R \neq \perp$ with probability at least $\frac{\varepsilon}{8}$. Further, the runtime of $\mathsf{F}_{\mathcal{B}_1}$ is at most $(4Q_1/\varepsilon) \cdot \mathsf{Time}(\mathcal{A})$. □

□

□

Optimizations and Efficiency

We now analyze the efficiency of the blind signature $\mathsf{BS}_{\mathsf{BB}}[\Pi]$.

Efficiency of Π. First, we optimize the extractor Ext of Π which allows us to choose better parameters. Note that the runtime of Ext scales linearly with the size of the exponents $e_i \in [0, B-1]$, as it brute-forces the discrete logarithm of 2ℓ group elements $F_i = \widehat{g}^{e_i}$. At the same time, the proof size scales linearly with $\ell = \lceil \log_B(\ell) \rceil$. Thus, for practical efficiency, we would like to set B as large as possible, while keeping the extractor efficient for the security reduction. If we use a more efficient algorithm to compute discrete logarithms of elements (with exponents in interval $[0, B-1]$), we can choose a larger bound B without any loss in runtime of the extractor. A good choice is Pollard's kangaroo algorithm [Pol78] which has runtime $\mathcal{O}(\sqrt{B})$. This allows us to increase the bit size of B by a factor 2 for the same level of security.

Second, we can omit α_1 in π_1, as the identities in $\Sigma_{\mathsf{ped}}.\mathsf{Verify}$ can be recomputed and then verified via β_1 due to collision resistance.

With these optimizations, the proof π_1 contains $5 + 4\ell$ elements in $\mathbb{Z}_p$. For $n = \log_2(B)$, the batched range proof π_0 contains $2\lceil \log_2(2n\ell + \ell + 4) \rceil + 1$ elements in $\widehat{\mathbb{G}}$ and 7 elements in $\mathbb{Z}_p$. As a proof π of Π consists of $\pi = (\pi_0, \pi_1, (E_i, R_i)_{i \in [2\ell]})$, the total proof size is $12 + 4\ell$ elements in $\mathbb{Z}_p$ and $2\lceil \log_2(2n\ell + \ell + 4) \rceil + 4\ell + 1$ elements in $\widehat{\mathbb{G}}$.

Efficiency of $\mathsf{BS}_{\mathsf{BB}}[\Pi]$. When $\mathsf{BS}_{\mathsf{BB}}[\Pi]$ is instantiated with Π for $B = \mathsf{poly}(\lambda)$, the user sends 1 element in $\mathbb{G}_1$, $2\lceil \log_2(2n\ell + \ell + 4) \rceil + 4\ell + 1$ in $\widehat{\mathbb{G}}$, and $10 + 2\ell$ elements in $\mathbb{Z}_p$ to the signer. The signer sends 2 elements in $\mathbb{G}_1$, and the final signature contains 2 elements in $\mathbb{G}_1$. We set $B = 2^{64}$ in order to have an extractor that performs roughly $\ell \cdot 2^{32}$ group operations, where $\ell = \lceil \log_B p \rceil = 4$. The total communication is 2.2 KB and signatures are of size 96 Byte for $\lambda = 128$.

7.6 Frameworks for Partially Blind Signatures

In this section, we present variants of our constructions in sections 7.2 and 7.4 that achieve partial blindness.

7.6.1 Partial Blindness of the Optimized Fischlin Transform

We show how to adapt $\mathsf{BS}_{\mathsf{Rnd}}$ for partial blindness. Roughly, instead of signing only the commitment c, the signer signs the vector $(c, \mathsf{H}_{\mathsf{T}}(t))$ instead, where t is the common message and H_{T} is a random oracle.

The partially blind signature $\mathsf{PBS}_{\mathsf{Rnd}}$ is based on building blocks $(\mathsf{C}, \mathsf{S}, \Sigma)$ that are $\mathsf{BS}_{\mathsf{Rnd}}$-suitable (cf. definition 7.1), except the message space $\mathcal{S}_{\mathsf{msg}}$ of the signature scheme S contains all tuples $(c, \bar{t})$, where $\bar{t} = \mathsf{H}_{\mathsf{T}}(t)$.

Definition 7.30 ($\mathsf{PBS}_{\mathsf{Rnd}}$-Suitable $(\mathsf{C}, \mathsf{S}, \Sigma)$)**.** The tuple of schemes $(\mathsf{C}, \mathsf{S}, \Sigma)$ are called $\mathsf{PBS}_{\mathsf{Rnd}}$-*suitable*, if it holds that

- C is the same as a commitment scheme that is $\mathsf{BS}_{\mathsf{Rnd}}$-suitable.
- S is the same as a signature scheme that is $\mathsf{BS}_{\mathsf{Rnd}}$-suitable, except that the message space $\mathcal{S}_{\mathsf{msg}}$ encompasses $\mathcal{C}_{\mathsf{com}} \times \mathcal{TH}$, where $\mathcal{TH}$ is a set with $1/|\mathcal{TH}| = \mathsf{negl}(\lambda)$.
- Σ is the same as a Σ-protocol that is $\mathsf{BS}_{\mathsf{Rnd}}$-suitable except for the relation

$$\mathsf{R}_{\mathsf{p\text{-}rnd}} \coloneqq \{x = (\mathsf{pp}, \mathsf{vk}, \overline{m}, \bar{t}), w = (\mu, c, r) \mid \mathsf{C.Commit}(\mathsf{pp}, \overline{m}; r) = c \wedge \mathsf{S.Verify}(\mathsf{vk}, \mu, (c, \bar{t})) = 1\}.$$

Construction

We present the partially blind signature $\mathsf{PBS}_{\mathsf{Rnd}}$ below.

Overview. Let $(\mathsf{C}, \mathsf{S}, \Sigma)$ be $\mathsf{PBS}_{\mathsf{Rnd}}$-suitable. Again, let $\mathsf{H}_{\mathsf{par}}, \mathsf{H}_{\mathsf{M}}, \mathsf{H}_{\beta}$ be a random oracles from $\{0,1\}^*$ into $\{0,1\}^{\ell_{\mathsf{C}}}, \mathcal{C}_{\mathsf{msg}}, \mathcal{CH}$, respectively. Further, let H_{T} be a random oracle from $\{0,1\}^*$ into $\mathcal{TH}$. We now present the framework $\mathsf{PBS}_{\mathsf{Rnd}}[\mathsf{C}, \mathsf{S}, \Sigma]$ (or $\mathsf{PBS}_{\mathsf{Rnd}}$ for short) for partially blind signatures based on $\mathsf{BS}_{\mathsf{Rnd}}$.

Key generation is the same as before, i.e.it outputs $(\mathsf{bvk}, \mathsf{bsk}) \leftarrow \mathsf{S.KeyGen}(1^\lambda)$, and implicitly defines a public parameter pp for C via $\mathsf{pp} = \mathsf{H}_{\mathsf{par}}(0)$. For message m and common message t, the user commits to $\overline{m} \leftarrow \mathsf{H}_{\mathsf{M}}(m)$ with randomness r via C and sends the commitment c to the signer. As before, the signer rerandomizes c with some random Δr, but signs $(c', \bar{t})$ instead of c' via S, where $\bar{t} = \mathsf{H}_{\mathsf{T}}(t)$, and sends the pair $(\mu, \Delta r)$ to the user. The derived signature is a proof π for relation $\mathsf{R}_{\mathsf{p\text{-}rnd}}$ generated by Σ using the Fiat-Shamir transform. Note that the user can recompute c' and its randomness via Δr and r, as before.

Description. For completeness, we provide the full description below, where we assume pp is provided to all of the algorithms for readability. The changes are highlighted with a box.

- $\mathsf{PBS}_{\mathsf{Rnd}}.\mathsf{KeyGen}(1^\lambda)$: samples $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{S.KeyGen}(1^\lambda)$ and outputs verification key $\mathsf{bvk} = \mathsf{vk}$ and signing key $\mathsf{bsk} = \mathsf{sk}$.

- $\mathsf{PBS}_{\mathsf{Rnd}}.\mathsf{User}(\mathsf{bvk}, t, m)$: sets $\overline{m} \leftarrow \mathsf{H}_{\mathsf{M}}(m)$ and outputs the commitment $c \in \mathcal{C}_{\mathsf{com}}$ generated via $(c, r) \leftarrow \mathsf{C.Commit}(\mathsf{pp}, \overline{m})$ as the first message and stores the randomness $\mathsf{st} = r \in \mathcal{C}_{\mathsf{rnd}}$.

- $\mathsf{PBS}_{\mathsf{Rnd}}.\mathsf{Signer}(\mathsf{bsk}, t, c)$: checks if $c \in \mathcal{C}_{\mathsf{com}}$, samples a rerandomization randomness $\Delta r \leftarrow \mathcal{C}_{\mathsf{rnd}}$, rerandomizes the commitment c via $c' \leftarrow \mathsf{C.RerandCom}(\mathsf{pp}, c, \Delta r)$, signs $\mu \leftarrow \mathsf{S.Sign}(\mathsf{sk}, \boxed{(c', \bar{t})})$ for $\bar{t} \leftarrow \mathsf{H}_{\mathsf{T}}(t)$, and finally outputs the second message $\rho = (\mu, \Delta r)$.

- $\mathsf{PBS}_{\mathsf{Rnd}}.\mathsf{Derive}(\mathsf{st}, t, \rho)$: parses $\mathsf{st} = r$, $\rho = (\mu, \Delta r)$ and checks $\Delta r \in \mathcal{C}_{\mathsf{rnd}}$. It then computes the randomized commitment $c'' = \mathsf{C.RerandCom}(\mathsf{pp}, c, \Delta r)$ and randomized randomness $r' \leftarrow \mathsf{C.RerandRand}(\mathsf{pp}, c, \overline{m}, r, \Delta r)$, and checks $\mathsf{S.Verify}(\mathsf{vk}, \boxed{(c'', \bar{t})}, \mu) = 1$ and $c'' = \mathsf{C.Commit}(\mathsf{pp}, \overline{m}; r')$. It then outputs a signature $\sigma = \pi$ for common message $\bar{t}$, where $(\alpha, \mathsf{st}') \leftarrow \Sigma.\mathsf{Init}(x, w)$, $\beta \leftarrow \mathsf{H}_{\beta}(x, \alpha), \gamma \leftarrow \Sigma.\mathsf{Resp}(x, \mathsf{st}', \beta)$, $\pi = (\alpha, \beta, \gamma)$ with $x = (\mathsf{pp}, \mathsf{vk}, \overline{m}, \boxed{\bar{t}})$, $w = (\mu, c', r')$.

- $\mathsf{PBS}_{\mathsf{Rnd}}.\mathsf{Verify}(\mathsf{bvk}, t, m, \sigma)$: parses $\sigma = \pi$ and $\pi = (\alpha, \beta, \gamma)$, and sets $\overline{m} = \mathsf{H}_{\mathsf{M}}(m)$, $\bar{t} \leftarrow \mathsf{H}_{\mathsf{T}}(t)$, and $x = (\mathsf{pp}, \mathsf{vk}, \overline{m}, \boxed{\bar{t}})$, and outputs 1 if $\beta = \mathsf{H}_{\beta}(x, \alpha)$, $\Sigma.\mathsf{Verify}(x, \alpha, \beta, \gamma) = 1$, and otherwise outputs 0.

Correctness and Security

We have the following theorem. As the proof is similar to the security proof of $\mathsf{BS}_{\mathsf{Rnd}}$, we only provide a sketch.

Theorem 7.31. *The partially blind signature* $\mathsf{PBS}_{\mathsf{Rnd}}$ *is (i) correct, (ii) partially blind under malicious keys under the hiding and rerandomization properties of* C *and the high min-entropy and HVZK properties of* Σ*, and (iii) one-more unforgeable under the binding and rerandomizability properties of* C*,* $\mathtt{EUF\text{-}CMA}$ *security of* S *and the* 2*-special soundness and* f*-unique extraction properties of* Σ*.*

Sketch. Compared to $\mathsf{BS}_{\mathsf{Rnd}}$, the only change is that the signer signs the message $(c', \bar{t})$ instead of c'. Note that the relation $\mathsf{R}_{\mathsf{p\text{-}rnd}}$ of Σ is adapted appropriately. Thus, correctness and zero-knowledge follow as before. One-more unforgeability is almost identical to before.

Let $\mathcal{A}$ be a PPT adversary that performs Q_S signing queries for some (adaptively chosen) common message t^*. We define the game $\mathcal{G}$ as the real game with $\mathcal{A}$, except the challenger aborts if there is a collision in H_{M} or H_{T} or there is some (x_i, α_i) in the forgeries of $\mathcal{A}$ that was never queried to H_β. As before, we can show that the challenger never aborts except with negligible probability.

Then, the challenger interacts with $\mathcal{A}$ as in the real game, and obtains forgeries $(\pi_i = (\alpha_i, \beta_i, \gamma_i))_{i\in[Q_S+1]}$ from $\mathcal{A}$, for messages m_i and common message t^*. Denote by $x_i = (\mathsf{pp}, \mathsf{vk}, \overline{m}_i, \bar{t}^*)$ the corresponding statements.

Then, we rewind the adversary $\mathcal{A}$ as in theorem 7.4 to obtain the witnesses $w_i = (\mu_i, c_i, r)$. Under 2-special soundness of Σ, we have $(x_i, w_i) \in \mathsf{R}_{\mathsf{rnd}}$. That is, we have for all $i \in [Q_S + 1]$ that $c_i = \mathsf{C.Commit}(\mathsf{pp}, \overline{m}_i; r)$ and $\mathsf{S.Verify}(\mathsf{vk}, \mu_i, (c_i, \bar{t}^*)) = 1$, where $\overline{m}_i = \mathsf{H}_{\mathsf{M}}(m_i)$ and $\bar{t}^* = \mathsf{H}_{\mathsf{T}}(t^*)$. Recall that there are no collisions in H_{M} in $\mathcal{G}$, so we have $\overline{m}_i \neq \overline{m}_j$ for all distinct $i, j \in [Q_S + 1]$. Thus, under the binding property of C, there cannot exist two distinct indices $i, j \in [Q_S + 1]$ such that $c_i = c_j$, as both c_i and c_j open to distinct messages $\overline{m}_i \neq \overline{m}_j$.

Consequently, as at most Q_S signing sessions under common message t^* were performed, there must be at least one commitment c_{i^*} that was never signed in the initial run under common message t^*. As before, this initial run fixes c_{i^*} due to f-unique extraction of Σ, so we are guaranteed to never sign the tuple $(c_{i^*}, \mathsf{H}_{\mathsf{T}}(t^*))$ during rewinding under rerandomizability of C. But as μ_i is a valid signature for $(c_{i^*}, \mathsf{H}_{\mathsf{T}}(t^*))$, we break $\mathtt{EUF\text{-}CMA}$ security of S. Note μ_i is a valid forgery, even if c_{i^*} was signed under some other common message $t \neq t^*$, as there are no collisions in H_{T} in $\mathcal{G}$ and we have $(c_{i^*}, \mathsf{H}_{\mathsf{T}}(t)) \neq (c_{i^*}, \mathsf{H}_{\mathsf{T}}(t^*))$. □

Instantiation

We can set $\mathcal{TH} = \mathbb{G}_1$. Then, H_{T} maps into $\mathbb{G}_1$ and we can instantiate the scheme using $\mathsf{S}_{\mathrm{KPW}}$ signatures and $\mathsf{C}_{\mathrm{Ped}}$ commitments as in section 7.3. Note that the size of $\mathsf{S}_{\mathrm{KPW}}$ signatures is independent of the number of signed group elements. Also, it is simple to adapt the Σ-protocol Σ_{rnd} accordingly. Consequently, the total communication and signature size remains 303 and 447 Byte for $\lambda = 128$, respectively.

7.6.2 Partial Blindness of Blind Signature based on Boneh-Boyen

We now show how to adapt the framework $\mathsf{BS}_{\mathsf{BB}}$ to obtain a partially blind signature $\mathsf{PBS}_{\mathsf{BB}}$. Again, we focus on the asymmetric setting. Our idea is to construct part of the verification key $\mathsf{vk} = (u_1, u_2, h_1, h_2, v)$ (cf. section 7.4.1) using a hash of the common message t. However, since $(u_1, u_2, h_1, h_2) = (g_1^\alpha, g_2^\alpha, g_1^\gamma, g_2^\gamma)$ formed a valid DDH tuple, it is not clear how to do this. To this end, we adapt the Boneh-Boyen signature scheme S_{BB} such that u_2, h_2 no longer needs to be a part of the verification key. Instead, we include one additional element in the signature to compensate for this modification. Looking ahead, since the verification key is now $\mathsf{vk} = (u_1, h_1, v)$ where h_1 is a random group element, we are able to create this term by a hash of the common message t.[14] The resulting scheme $\overline{S}_{\mathsf{BB}}$ is defined below. We later show (implicitly) in the security proof of $\mathsf{PBS}_{\mathsf{BB}}$ that the variant $\overline{S}_{\mathsf{BB}}$ is selectively secure under CDH. We omit the subscript and simply use (u, h) below.

- $\overline{S}_{\mathsf{BB}}.\mathsf{KeyGen}(1^\lambda)$: samples $\alpha, \beta, \gamma \in \mathbb{Z}_p$, and sets $u = g_1^\alpha, h = g_1^\gamma, v = e(g_1, g_2)^{\alpha\beta}$, and outputs $\mathsf{vk} = (u, h, v)$ and $\mathsf{sk} = g_1^{\alpha\beta}$.

[14] Note in the symmetric setting, we can use standard the Boneh-Boyen signature scheme S_{BB} and let a random oracle output h_1, as there is no h_2. The signature size in the symmetric setting remains 2 group elements.

– $\overline{S}_{\mathsf{BB}}.\mathsf{Sign}(\mathsf{sk}, m)$: samples $r \in \mathbb{Z}_p$ and outputs $(\sigma_0, \sigma_1, \sigma_2) = (\mathsf{sk} \cdot (u^m h)^r, g_1^r, g_2^r)$.

– $\overline{S}_{\mathsf{BB}}.\mathsf{Verify}(\mathsf{vk}, m, (\sigma_0, \sigma_1, \sigma_2))$: verify that $e(\sigma_0, g_2) = v \cdot e(u^m h, \sigma_2)$ and $e(\sigma_1, g_2) = e(g_1, \sigma_2)$.

Construction. Now, we construct $\mathsf{PBS}_{\mathsf{BB}}$ and detail the changes required to the framework $\mathsf{BS}_{\mathsf{BB}}$ (cf. section 7.4). Again, let Π be an online-extractable NIZK proof system, with random oracle $\mathsf{H}_{\mathsf{zk}} : \{0,1\}^* \mapsto \{0,1\}^{\ell_{\mathsf{zk}}}$ and common reference string length ℓ_{crs} for the relation

$$\mathsf{R}_{\mathsf{bb}} := \{x = (c, u, g_1), w = (\overline{m}, s) \mid c = u^{\overline{m}} \cdot g_1^s\},$$

and let $\mathsf{H}_{\mathsf{M}}, \mathsf{H}_{\mathsf{crs}}$ be a random oracles mapping into $\mathbb{Z}_p, \{0,1\}^{\ell_{\mathsf{crs}}}$ respectively. Further, we rely on a random oracle $\mathsf{H}_{\mathbb{G}_1}$ mapping into $\mathbb{G}_1$. The framework $\mathsf{PBS}_{\mathsf{BB}}[\Pi]$, or $\mathsf{PBS}_{\mathsf{BB}}$ for short, is as $\mathsf{BS}_{\mathsf{BB}}$ except that the underlying signature is replaced with $\overline{S}_{\mathsf{BB}}$ and the value h in the verification key is sampled via $\mathsf{H}_{\mathbb{G}_1}$.

Construction. Below, we detail the construction and highlight the changes with respect to $\mathsf{BS}_{\mathsf{BB}}$ via a box. We assume that crs is provided to all of the algorithms for readability.

– $\mathsf{PBS}_{\mathsf{BB}}.\mathsf{KeyGen}(1^\lambda)$: samples $\alpha, \beta \in \mathbb{Z}_p$, and sets $u = g_1^\alpha, v = e(g_1, g_2)^{\alpha\beta}$, and outputs $\mathsf{bvk} = (u, v)$ and $\mathsf{bsk} = g_1^{\alpha\beta}$.

– $\mathsf{PBS}_{\mathsf{BB}}.\mathsf{User}(\mathsf{bvk}, t, m)$: sets $\overline{m} \leftarrow \mathsf{H}_{\mathsf{M}}(m)$ and computes a Pedersen commitment $c = u^{\overline{m}} g_1^s$ and a proof $\pi \leftarrow \Pi.\mathsf{Prove}^{\mathsf{H}_{\mathsf{zk}}}(\mathsf{crs}, x, w)$, where $s \leftarrow \mathbb{Z}_p$, $x = (c, u, g_1)$, and $w = (\overline{m}, s)$. It outputs the first message $\rho_1 = (c, \pi)$ and stores the randomness $\mathsf{st} = s$.

– $\mathsf{PBS}_{\mathsf{BB}}.\mathsf{Signer}(\mathsf{bsk}, t, \rho_1)$: parses $\rho_1 = (c, \pi)$ and checks $\Pi.\mathsf{Verify}^{\mathsf{H}_{\mathsf{zk}}}(\mathsf{crs}, x, \pi) = 1$. It then sets $\boxed{h_t \leftarrow \mathsf{H}_{\mathbb{G}_1}(t)}$ and outputs the second meesage $\rho_2 = (\rho_{2,0}, \rho_{2,1}, \rho_{2,2}) \leftarrow (\mathsf{sk} \cdot (c \cdot \boxed{h_t})^r, g_1^r, \boxed{g_2^r})$, where $r \leftarrow \mathbb{Z}_p$.

– $\mathsf{PBS}_{\mathsf{BB}}.\mathsf{Derive}(\mathsf{st}, t, \rho_2)$: parses $\mathsf{st} = s$ and $\rho_2 = (\rho_{2,0}, \rho_{2,1}, \rho_{2,2})$, checks $e(\rho_{2,0}, g_2) = v \cdot e(c \cdot \boxed{h_t}, \rho_{2,2})$ and $\boxed{e(\rho_{2,1}, g_2) = e(g_1, \rho_{2,2})}$, and outputs the signature $\sigma = (\sigma_0, \sigma_1, \sigma_2) = (\rho_{2,0}/\rho_{2,1}^s \cdot (u^{\overline{m}} \boxed{h_t})^{r'}, \rho_{2,1} \cdot g_1^{r'}, \boxed{\rho_{2,2} \cdot g_2^{r'}})$ for $r' \leftarrow \mathbb{Z}_p$.

– $\mathsf{BS}_{\mathsf{BB}}.\mathsf{Verify}(\mathsf{bvk}, t, m, \sigma)$: checks $e(\sigma_0, g_2) = v \cdot e(u^{\overline{m}} \boxed{h_t}, \sigma_2)$ and $\boxed{e(\sigma_1, g_2) = e(g_1, \sigma_2)}$, where $\overline{m}$ and h_t as above.

Correctness and Security

We can show that $\mathsf{PBS}_{\mathsf{BB}}$ is correct, blind, and one-more unforgeable.

Theorem 7.32. *The scheme $\mathsf{PBS}_{\mathsf{BB}}$ is correct, blind under malicious keys under the zero-knowledge property of Π, and one-more unforgeable under the CDH assumption and the online-extractability of Π.*

Proof. The proof of correctness and blindness follow almost as in theorems 7.16 and 7.17, and we omit details. In the following, we show that $\mathsf{PBS}_{\mathsf{BB}}$ is one-more unforgeable. Roughly, the reduction punctures the verification key (including h_t for all common messages t) depending on whether (i) t is the (adaptively chosen) common message t^* from the forgeries of $\mathcal{A}$ or (ii) t is some other common message. In case of (i), the reduction punctures h_t such that it can sign all but some random message $\overline{m}_t \leftarrow \mathbb{Z}_p$. As the message $\overline{m}_t$ is hidden from $\mathcal{A}$, she does not provide a signing request for (a commitment of) $\overline{m}_t$ with high probability, and the reduction can answer all signing queries for $t \neq t^*$. In case of (ii), for the common message t^* of the $\mathcal{A}$'s forgeries, the reduction punctures h_{t^*} on $\overline{m}^* \leftarrow \mathbb{Z}_p$. As in theorem 7.18, we show that by programming the

random oracle H_M, we can argue that $\mathcal{A}$ never asks for a signature on (a commitment of) $\overline{m}^*$ but provides a forgery for message m_{i^*} such that $\overline{m}^* = \mathsf{H}_\mathsf{M}(m_{i^*})$ with noticeable probability, in which case we can solve CDH. We formalize the above intuition below.

Let $\mathcal{A}$ be a PPT adversary against the one-more unforgeability of $\mathsf{PBS}_{\mathsf{BB}}$. Let Ext and $\mathsf{Sim}_{\mathsf{crs}}$ be the extractor and simulator of Π, respectively (cf. definition 3.31). Without loss of generality, let Q_S be the number of signing queries per common message, and Q_T the total number of common messages. We denote by Q_M the number of H_M queries, by Q_H the number of H_{zk} queries, and by Q_G the number of $\mathsf{H}_{\mathbb{G}_1}$ queries. We assume without loss of generality that $\mathcal{A}$'s queries to the random oracles are unique. Moreover, we assume $\mathcal{A}$ queries the common message t to $\mathsf{H}_{\mathbb{G}_1}$ before submitting a signing query with t. We denote by q_j (resp. q'_j) the j-th query to H_M for $j \in [Q_M]$ (resp., $\mathsf{H}_{\mathbb{G}_1}$ for $j \in [Q_G]$). After Q_S signing queries per common message, $\mathcal{A}$ outputs $Q_S + 1$ forgeries $\{(m_i, \sigma_i)\}_{i\in[Q_S+1]}$ for some common message t^*. We write $\sigma_i = (\sigma_{i,0}, \sigma_{i,1}, \sigma_{i,2})$, and denote by $\rho_{1,i} = (c_i, \pi_i)$ the signing queries issued by $\mathcal{A}$. We define the following hybrids and denote by $\mathsf{Adv}^{\mathrm{H}_i}_{\mathcal{A}}(\lambda)$ the advantage of $\mathcal{A}$ in Hybrid i. (Note that until Hybrid 3, the steps are identical to the steps in theorem 7.18).

- Hybrid 0 is identical to the real game.
- Hybrid 1 is the same as Hybrid 0, except it samples $(\overline{\mathsf{crs}}, \tau) \leftarrow \mathsf{Sim}_{\mathsf{crs}}(1^\lambda)$ and programs $\overline{\mathsf{crs}}$ into the random oracle $\mathsf{H}_{\mathsf{crs}}$ via $\mathsf{H}_{\mathsf{crs}}(0) \leftarrow \overline{\mathsf{crs}}$. We can construct an adversary $\mathcal{B}_{\mathsf{crs}}$ against the CRS indistinguishability from Π such that $\mathsf{Adv}^{\mathrm{H}_1}_{\mathcal{A}}(\lambda) \geq \mathsf{Adv}^{\mathrm{H}_0}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathrm{crs}}_{\mathcal{B}_{\mathsf{crs}}}(\lambda)$.
- Hybrid 2 is the same as Hybrid 1, except the witnesses $(\overline{m}_i, s_i)$ are extracted from all the proofs π_i for all $i \in [Q_S \cdot Q_T]$ using Ext. Specifically, when $\mathcal{A}$ provides the signing query (c_i, π_i), the challenger runs $w_i \leftarrow \mathsf{Ext}(\mathsf{crs}, \mathsf{td}, x_i, \pi_i)$, where $x_i = (c_i, u, g_1)$. It parses $w_i = (\overline{m}_i, s_i)$ and aborts if $u^{\overline{m}_i} \cdot g_1^{s_i} \neq c_i$. Then, proceeds as in Hybrid 1.

  We have $\mathsf{Adv}^{\mathrm{H}_2}_{\mathcal{A}}(\lambda) \geq \frac{\mathsf{Adv}^{\mathrm{H}_1}_{\mathcal{A}}(\lambda) - \mathsf{negl}(\lambda)}{p_\mathsf{P}(\lambda, Q_H)}$ under the online extractability of Π. Note that the challenger has an additional runtime overhead $p_\mathsf{T}(\lambda, Q_H) \cdot \mathsf{Time}(\mathcal{A})$, and that p_P and p_T are polynomials as defined in definition 3.31. We note that Ext does not need to be invoked at the end of the game as required by the definition of online-extractability. See theorem 7.18 for more detail.
- Hybrid 3 is the same as Hybrid 2, except it aborts if there is a collision in H_M or if there is some message m_i in $\mathcal{A}$'s output that was never queried to H_M.

  It holds that $\mathsf{Adv}^{\mathrm{H}_3}_{\mathcal{A}}(\lambda) \geq \mathsf{Adv}^{\mathrm{H}_2}_{\mathcal{A}}(\lambda) - \frac{Q_M^2+1}{p}$.
- Hybrid 4 is the same as Hybrid 3, except it aborts if there is a collision in $\mathsf{H}_{\mathbb{G}_1}$ or if the common message t^* of $\mathcal{A}$'s output was never queried to $\mathsf{H}_{\mathbb{G}_1}$.

  A given signature verifies with respect to some random h_t at most with probability $1/p$ and a collision happens with probability at most Q_G^2/p. Thus, it holds that $\mathsf{Adv}^{\mathrm{H}_4}_{\mathcal{A}}(\lambda) \geq \mathsf{Adv}^{\mathrm{H}_3}_{\mathcal{A}}(\lambda) - \frac{Q_G^2+1}{p}$.
- Hybrid 5 is the same as Hybrid 4, except it first samples $\overline{m}^* \in \mathbb{Z}_p$, $k^* \leftarrow [Q_G]$, and $\delta_{k^*} \leftarrow \mathbb{G}_1$. For the k^*-th query q'_{k^*} to $\mathsf{H}_{\mathbb{G}_1}$, it sets $\mathsf{H}_{\mathbb{G}_1}(q'_{k^*}) \leftarrow h_t^* := u^{-\overline{m}^*} \cdot g_1^{\delta_{k^*}}$. Hybrid 5 aborts if the forgeries output by $\mathcal{A}$ is not on a common message t^* such that $h_t^* = \mathsf{H}_{\mathbb{G}_1}(t^*)$.

  Due to the modification we made in Hybrid 4, the common message t^* is guaranteed to be queried to $\mathsf{H}_{\mathbb{G}_1}$. Since k^* is information-theoretically hidden to $\mathcal{A}$ and the distribution of h_t^* is uniform over $\mathbb{G}_1$ as in the previous hybrid, we have $\mathsf{Adv}^{\mathrm{H}_5}_{\mathcal{A}}(\lambda) \geq 1/Q_G \cdot \mathsf{Adv}^{\mathrm{H}_4}_{\mathcal{A}}(\lambda)$.
- Hybrid 6 is the same as Hybrid 5, except it guesses a query index j^* of H_M for which it outputs $\overline{m}^* \in \mathbb{Z}_p$, and aborts if either the challenger extracts $\overline{m}^*$ in some signing query

with common message t^* or if $\mathcal{A}$ provides no forgery for some message that hashes to $\overline{m}^*$. More concrete, Hybrid 6 further samples $j^* \leftarrow [Q_M]$ at the outset of the game. For the j^*-th query to $\mathsf{H_M}$, it sets $\mathsf{H_M}(q_{j^*}) \leftarrow \overline{m}^*$. At the i-th signing query, the signer aborts if the extracted witness is $(\overline{m}^*, s_i)$ and the common message is t^*, else proceeds as usual. Also, Hybrid 6 aborts if $m^* \notin \{\mathsf{H_M}(m_i)\}_{i\in[Q_S+1]}$, where m_i are the messages from the forgeries of the final output of $\mathcal{A}$.

Conditioned on no collision in $\mathsf{H_M}$, there are only Q_S signing queries but $Q_S + 1$ values $\overline{m}_i \leftarrow \mathsf{H_M}(m_i)$ from the forgeries, and with probability $1/Q_M$ the challenger guesses the right $\mathsf{H_M}$ query. Following the same calculation as in theorem 7.18, we have $\mathsf{Adv}^{\mathrm{H}_6}_{\mathcal{A}}(\lambda) \geq \frac{1}{Q_M}\mathsf{Adv}^{\mathrm{H}_5}_{\mathcal{A}}(\lambda)$.

– Hybrid 7 is the same as Hybrid 6, except it sets up a punctured verification key and simulates signing without knowing the full sk. Specifically, it sets $A_1 = g_1^{\alpha}, A_2 = g_2^{\alpha}, B = g_1^{\beta}, B_2 = g_2^{\beta}, u = A_1, v = e(A_1, B_2)$, for $\alpha, \beta \leftarrow \mathbb{Z}_p$. Then, sends $\mathsf{bvk} = (u, v)$ to $\mathcal{A}$. Also, Hybrid 7 initially samples $\overline{m}_{t,k} \leftarrow \mathbb{Z}_p$ for $k \in [Q_G]\backslash\{k^*\}$, and answers the k-th $\mathsf{H}_{\mathbb{G}_1}$ query with $h_{t,k} = u^{-\overline{m}_{t,k}} \cdot g_1^{\delta_k}$ if $k \neq k^*$. It answers the k^*-th query as in the previous Hybrid 5.

For the i-th signing query (c_i, π_i) with common message t_i and extracted witness $w_i = (\overline{m}_i, s_i)$, parses $\mathsf{H}_{\mathbb{G}_1}(t_i) = h_{t,k} = u^{-\overline{m}_{t,k}} \cdot g_1^{\delta_k}$ for an appropriate $k \in [Q_G]$, and aborts if $\overline{m}_i = \overline{m}_{t,k}$ and $k \neq k^*$. Note that such a k uniquely exists due to Hybrid 4. Moreover, due to the modification we made in Hybrid 6, it always abort when $(\overline{m}_i, t_i) = (\overline{m}^*, t^*)$ or equivalently $(\overline{m}_i, k) = (\overline{m}^*, k^*)$. If it didn't abort, it samples some $\tilde{r}_i \leftarrow \mathbb{Z}_p$, sets $\rho_{2,1} = g_1^{\tilde{r}_i} \cdot B_1^{-1/(\overline{m}_i - \overline{m}_{t,k})}, \rho_{2,2} = g_2^{\tilde{r}_i} \cdot B_2^{-1/(\overline{m}_i - \overline{m}_{t,k})}$ and $\rho_{2,0} = A_1^{(\overline{m}_i - \overline{m}_{t,k})\tilde{r}} \cdot g_1^{(s_i+\delta_k)\tilde{r}_i} \cdot B_1^{-(s_i+\delta_k)/(\overline{m}_i - \overline{m}_{t,k})}$.

It is not hard to check that Hybrid 7 and Hybrid 6 are identically distributed as long as $\mathcal{A}$ doesn't query a common message $t_i \neq t^*$ such that $\overline{m}_i = \overline{m}_{t,k}$ for $k \neq k^*$. Since $\overline{m}_{t,k}$ is information-theoretically hidden from $\mathcal{A}$, we conclude via a union bound that the abort probability is at most $(Q_S Q_T)/p$. Thus, $\mathsf{Adv}^{\mathrm{H}_7}_{\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathrm{H}_6}_{\mathcal{A}}(\lambda) - (Q_S Q_T)/p$.

We can now construct an adversary $\mathcal{B}_{\mathsf{CDH}}$ to solve CDH with $\mathsf{Adv}^{\mathrm{CDH}}_{\mathcal{A}_{\mathsf{CDH}}}(\lambda) = \mathsf{Adv}^{\mathrm{H}_7}_{\mathcal{A}}(\lambda)$. First, $\mathcal{B}_{\mathsf{CDH}}$ receives a CDH-tuple $(g_1, g_2, A_1, A_2, B_1, B_2)$, with which it simulates Hybrid 7 to $\mathcal{A}$. After $\mathcal{A}$ outputs the forgeries $\{(m_i, \sigma_i)\}_{i\in[Q_S+1]}$, $\mathcal{B}_{\mathsf{CDH}}$ outputs $\sigma_{i^*,0} \cdot \sigma_{i^*,1}^{-\delta_{k^*}}$ to its challenger, where i^* such that $\mathsf{H_M}(m_{i^*}) = \overline{m}^*$.

Due to the abort conditions in Hybrid 5 and Hybrid 6, the adversary is guaranteed to output an appropriate forgery that verifies correctly with probability at least $\mathsf{Adv}^{\mathrm{H}_7}_{\mathcal{A}}(\lambda)$. In that case, we show that $\sigma_{i^*,0}^{\delta_{k^*}}/\sigma_{i^*,1}$ is indeed of the form $g_1^{\alpha\beta}$, where α, β is the discrete logarithm of A_1, B_1, respectively. For readability, we write $(\sigma_0, \sigma_1, \sigma_2) = (\sigma_{i^*,0}, \sigma_{i^*,1}, \sigma_{i^*,2})$ in the following (with slight abuse of notation).

As the i^*-th forgery verifies, we have $e(\sigma_0, g_2) = v \cdot e(u^{\overline{m}^*} \cdot h_t^*, \sigma_2)$ and $e(\sigma_1, g_2) = e(g_1, \sigma_2)$. The latter guarantees that the discrete logarithms of σ_1 and σ_2 are identical, i.e.we have $\sigma_1 = g_1^{\rho}$ and $\sigma_2 = g_2^{\rho}$ for some appropriate ρ. Note that we have $v = e(g_1, g_2)^{\alpha\beta}$, $u = A_1$ and $h_t^* = u^{-\overline{m}^*} \cdot g_1^{\delta_{k^*}}$

due to the changes in Hybrid 5 and Hybrid 7. Consequently, we have

$$
\begin{aligned}
& e(\sigma_0, g_2) = v \cdot e(u^{\overline{m}^*} \cdot h_t^*, \sigma_2) \\
\Longrightarrow\ & e(\sigma_0, g_2) = e(g_1, g_2)^{\alpha\beta} \cdot e(u^{\overline{m}^*} \cdot u^{-\overline{m}^*} \cdot g_1^{\delta_{k^*}}, \sigma_2) \\
\Longrightarrow\ & e(\sigma_0, g_2) = e(g_1, g_2)^{\alpha\beta} \cdot e(g_1^{\delta_{k^*}}, \sigma_2) \\
\Longrightarrow\ & e(\sigma_0, g_2) = e(g_1, g_2)^{\alpha\beta} \cdot e(g_1, g_2)^{\delta_{k^*} \cdot \rho} \\
\Longrightarrow\ & e(\sigma_0 \cdot g_1^{-\delta_{k^*} \cdot \rho}, g_2) = e(g_1, g_2)^{\alpha\beta} \\
\Longrightarrow\ & e(\sigma_0 \cdot \sigma_1^{-\delta_{k^*}}, g_2) = e(g_1^{\alpha\beta}, g_2) \\
\Longrightarrow\ & \sigma_0 \cdot \sigma_1^{-\delta_{k^*}} = g_1^{\alpha\beta}
\end{aligned}
$$

The statement follows by collecting all the above bounds. □

Instantiation.

We can use the online extractable NIZK Π from section 7.5. Both communication and signature size increase by one element in $\mathbb{G}_2$.

Chapter

8 Conclusion and Open Questions

In this thesis, we presented weighted variants of classical hashing algorithms such as one-choice allocation, two-choice allocation and cuckoo hashing. Then, we introduced the notion of page efficiency that captures memory efficiency of SSE schemes on SSDs. Based on our hashing variants, we constructed a static SSE scheme Pluto with optimal page efficiency, and a dynamic SSE scheme LayeredSSE with sublogarithmic page efficiency. Next, we constructed the dynamic SSE scheme Hermes, an SSE scheme with forward security and sublogarithmic page efficiency. The above tradeoffs were not known in the realm of memory-efficient SSE, including local SSE. Finally, we presented two efficient constructions of round-optimal blind signatures. We proved security under standard assumptions in the ROM, and our constructions are the most efficient schemes in that setting. Below, we list some intriguing open problems in the context of our contributions.

Open Problems in Weighted Hashing. The most interesting open problem for weighted hashing is perhaps an upper bound on the most loaded bin in weighted 2C allocation. While we provide an upper bound for a modified variant L2C, an upper bound for the classical variant remains elusive. Also, our analysis of L2C does not allow for deletions, and an upper bound in the presence of deletes would be interesting.

Similarly, dynamic variant of weighted cuckoo hashing would be interesting. Note that while our weighted cuckoo hashing algorithm is static, it is simple to add balls via the execution of an additional max flow algorithm. The runtime will likely be less than the first setup, as the allocation is already optimized (except for the new insertion). Nevertheless, an algorithm with constant insertion time for balls as in classical cuckoo hashing is unknown.

Open Problems in SSE. In the direction of memory-efficient SSE, we consider the question whether there are lower bounds for dynamic SSE with standard leakage (and/or forward security) to be interesting. Notably, our static construction Pluto has optimal page efficiency, but while our dynamic constructions LayeredSSE and Hermes come close to optimal, both have a non-constant overhead. Is this inherent or can the constructions be optimized?

Also, while there are constructions of local dynamic SSE with sublogarithmic efficiency (cf. [MR22]), it is unclear whether it is possible to construct efficient local SSE with forward security.

Open Probelems in blind signatures. In the pairing setting, our constructions of blind signatures in the ROM come close to the efficiency of constructions based on non-interactive assumptions or generic groups. In the lattice setting, there are efficient construction with poly-concurrent security under standard assumptions in the ROM [dK22]. Outside of the above settings, such constructions are not known (unless computation and communication scales with the number of signatures [KLR21, CAHL+22]).

Appendix

A Notation

Mathematical Notations

$x \leftarrow S$	Uniform and independent sampling from a set S
$x \leftarrow D$	Sampling from a distribution D
$D_0 \overset{s}{\approx} D_1$	Distributions D_0 and D_1 are statistically indistinguishable
$D_0 \overset{c}{\approx} D_1$	Distributions D_0 and D_1 are computationally indistinguishable
$\Pr[E]$	Probability of event E
$\mathsf{Exp}[E]$	Expectation of event E
$\lvert S \rvert$	Cardinality of set S
$[n]$	The set of integers $\{1, ..., n\}$
$[a, b]$	The interval of integers $\{a, ..., b\}$
$[a, b]_{\mathbb{R}}$	The interval of reals $\{x \in \mathbb{R} : a \leq x \leq b\}$
$\vec{h} = (h_1, ..., h_q)$	A row vector $(h_1, ..., h_q)$
$\vec{h}_{<i}$	The vector $(h_1, ..., h_{i-1})$
$\vec{h}_{\geq i}$	The vector $(h_i, ..., h_q)$
$\vec{h} \| \vec{h'}$	The concatenation of two vectors $\vec{h}$ and $\vec{h'}$
$\mathbb{Z}_p$	Integers modulo p
$\mathbb{G}$	Some prime order group

Algorithms and Cryptography

$y \leftarrow A(x)$	Execution of (randomized) algorithm A with input x and output y
$y \leftarrow A(x; r)$	Execution of randomized algorithm A with input x, explicit randomness r and output y
$A^{\mathcal{O}}$	Algorithm with access to some oracle $\mathcal{O}$
st	The state of an algorithm
λ	The security paramter
$\mathsf{Time}(A)$	The runtime of algorithm A
$\mathsf{negl}(\lambda)$	Some negligible function in λ
$\mathsf{poly}(\lambda)$	Some polynomial function in λ
$\Theta(\cdot), \mathcal{O}(\cdot), \Omega(\cdot), o(\cdot), \omega(\cdot)$	Asymptotic notation

Searchable Encryption

DB	Database
$\mathsf{DB}(w)$	Identifier list matching w
N	Upper bound on the size of the database
W	Upper bound on the number of keywords
p	Page size
$\mathsf{prot} = (\mathsf{prot}_A, \mathsf{prot}_B)$	A protocol between two parties A and B
$\mathsf{prot_A}(\mathsf{in}_A) \longleftrightarrow \mathsf{prot_B}(\mathsf{in}_B)$	The interaction of protocol prot between parties A and B
$\mathsf{prot}(\mathsf{in}_A; \mathsf{in}_B)$	Short notation for the interaction of protocol prot between parties A and B

Appendix

B Abbreviations

Algorithms

1C	One-choice allocation
2C	Two-choice allocation
L2C	Layered two-choice allocation (cf. Section 4.3)
WCuckoo	Weighted cuckoo hashing (cf. Section 4.4)

Assumptions

DLOG	Discrete Logarithm
CDH	Computational Diffie-Hellman
DDH	Decisional Diffie-Hellman
SXDH	Symmetric External Diffie-Hellman

Cryptographic Notions

PPT	Probabilistic Polynomial-Time
SSE	Searchable Symmetric Encryption
PRF	Pseudorandom Function
NIZK	Non-interactive Zero-Knowledge
HVZK	Honest-Verifier Zero-Knowledge
ROM	Random Oracle Model
SPS	Structure-preserving Signatures

Cryptographic Schemes

$\mathsf{C}_{\mathrm{Ped}}$	Pedersen Commitments
C_{EG}	ElGamal Commitments
$\mathsf{S}_{\mathrm{KPW}}$	Kiltz-Pan-Wee Signatures
S_{BB}	Boneh-Boyen Signatures
Pluto	SSE scheme (cf. Section 5.3)
LayeredSSE	SSE scheme (cf. Section 5.4)
Hermes	SSE scheme (cf. Section 6.5)
$\mathsf{BS}_{\mathsf{Rnd}}$	Blind signature (cf. Section 7.2)
$\mathsf{BS}_{\mathsf{BB}}$	Blind signature (cf. Section 7.4)

Bibliography

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 553–572, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany. *Cited on pages 108 and 109.*

[Abe01] Masayuki Abe. A secure three-move blind signature scheme for polynomially many signatures. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 136–151, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany. *Cited on pages 7 and 18.*

[ABKU94] Yossi Azar, Andrei Z Broder, Anna R Karlin, and Eli Upfal. Balanced allocations. In *Proceedings of the twenty-sixth annual ACM symposium on theory of computing*, pages 593–602, 1994. *Cited on pages 6, 17, and 40.*

[AC20] Thomas Attema and Ronald Cramer. Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 513–543, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany. *Cited on pages 104, 110, 130, and 133.*

[ADW14] Martin Aumüller, Martin Dietzfelbinger, and Philipp Woelfel. Explicit and efficient hash families suffice for cuckoo hashing with a stash. *Algorithmica*, 70(3):428–456, 2014. *Cited on page 58.*

[AFG+10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 209–236, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany. *Cited on pages 8 and 19.*

[AFK22] Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-shamir transformation of multi-round interactive proofs. In Eike Kiltz and Vinod Vaikuntanathan, editors, *Theory of Cryptography - 20th International Conference, TCC 2022, Chicago, IL, USA, November 7-10, 2022, Proceedings, Part I*, volume 13747 of *Lecture Notes in Computer Science*, pages 113–142. Springer, 2022. *Cited on pages 104, 110, 111, 130, 133, and 134.*

[AGHO11] Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 649–666, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany. *Cited on page 119.*

[AJOR18] Masayuki Abe, Charanjit S. Jutla, Miyako Ohkubo, and Arnab Roy. Improved (almost) tightly-secure simulation-sound QA-NIZK with applications. In Thomas

Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 627–656, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany. *Cited on pages 8, 19, 102, and 103.*

[AKSY22] Shweta Agrawal, Elena Kirshanova, Damien Stehlé, and Anshu Yadav. Practical, round-optimal lattice-based blind signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 39–53, Los Angeles, CA, USA, November 7–11, 2022. ACM Press. *Cited on pages 8 and 19.*

[AM23] Léonard Assouline and Brice Minaud. Weighted oblivious RAM, with applications to searchable symmetric encryption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 426–455, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany. *Cited on pages 4 and 15.*

[ANSS16] Gilad Asharov, Moni Naor, Gil Segev, and Ido Shahaf. Searchable symmetric encryption: optimal locality in linear space via two-dimensional balanced allocations. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 1101–1114, Cambridge, MA, USA, June 18–21, 2016. ACM Press. *Cited on pages 5, 6, 17, 43, 62, 63, and 74.*

[AO00] Masayuki Abe and Tatsuaki Okamoto. Provably secure partially blind signatures. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 271–286, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Heidelberg, Germany. *Cited on pages 7 and 18.*

[ASS18] Gilad Asharov, Gil Segev, and Ido Shahaf. Tight tradeoffs in searchable symmetric encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 407–436, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. *Cited on page 62.*

[ASS21] Gilad Asharov, Gil Segev, and Ido Shahaf. Tight tradeoffs in searchable symmetric encryption. *Journal of Cryptology*, 34(2):9, April 2021. *Cited on pages 5, 17, and 74.*

[BB04a] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 223–238, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany. *Cited on pages 9, 20, 104, 108, 109, and 124.*

[BB04b] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 56–73, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany. *Cited on page 108.*

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008. *Cited on pages 109 and 124.*

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press. *Cited on pages 9, 20, 104, and 130.*

[BBF+21] Angèle Bossuat, Raphael Bost, Pierre-Alain Fouque, Brice Minaud, and Michael Reichle. SSE and SSD: Page-efficient searchable symmetric encryption. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 157–184, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany. *Cited on pages 10, 12, 20, 21, 22, 40, 63, 70, and 74.*

[BBM00] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 259–274, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany. *Cited on page 119.*

[BCC04] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, *ACM CCS 2004*, pages 132–145, Washington, DC, USA, October 25–29, 2004. ACM Press. *Cited on pages 7, 18, and 101.*

[BCI+10] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 237–254, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany. *Cited on page 27.*

[BCLO09] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 224–241, Cologne, Germany, April 26–30, 2009. Springer, Heidelberg, Germany. *Cited on pages 4 and 16.*

[BCSV06] Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: The heavily loaded case. *SIAM Journal on Computing*, 35(6):1350–1385, 2006. *Cited on page 40.*

[BDE+22] Maxime Buser, Rafael Dowsley, Muhammed F. Esgin, Clémentine Gritti, Shabnam Kasra Kermanshahi, Veronika Kuchta, Jason T. LeGrow, Joseph K. Liu, Raphaël C.-W. Phan, Amin Sakzad, Ron Steinfeld, and Jiangshan Yu. A survey on exotic signatures for post-quantum blockchain: Challenges & research directions. *ACM Comput. Surv.*, 2022. Just accepted. *Cited on pages 7, 18, and 101.*

[BF19] Raphael Bost and Pierre-Alain Fouque. Security-efficiency tradeoffs in searchable encryption. *PoPETs*, 2019(4):132–151, October 2019. *Cited on pages 76, 77, and 99.*

[BFHM08] Petra Berenbrink, Tom Friedetzky, Zengjian Hu, and Russell Martin. On weighted balls-into-bins games. *Theoretical Computer Science*, 409(3):511–520, 2008. *Cited on pages 6, 17, 39, 40, 41, 42, and 56.*

[BFPV13] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Short blind signatures. *Journal of computer security*, 21(5):627–661, 2013. *Cited on pages 8, 9, 19, 20, 102, 103, 107, and 109.*

[BFW15] David Bernhard, Marc Fischlin, and Bogdan Warinschi. Adaptive proofs of knowledge in the random oracle model. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 629–649, Gaithersburg, MD, USA, March 30 – April 1, 2015. Springer, Heidelberg, Germany. *Cited on pages 106 and 109.*

[BIM04] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. *Journal of Cryptology*, 17(2):125–151, March 2004. *Cited on pages 4 and 15.*

[BKM20] Laura Blackstone, Seny Kamara, and Tarik Moataz. Revisiting leakage abuse attacks. In *ISOC Network and Distributed System Security – NDSS 2020*, 2020. *Cited on page 99.*

[BL13] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 1087–1098, Berlin, Germany, November 4–8, 2013. ACM Press. *Cited on pages 7 and 18.*

[BLL+21] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 33–53, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. *Cited on pages 7 and 18.*

[BLS03] Paulo SLM Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In *Security in Communication Networks: Third International Conference, SCN 2002 Amalfi, Italy, September 11–13, 2002 Revised Papers 3*, pages 257–267. Springer, 2003. *Cited on page 26.*

[BMO17] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1465–1482, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. *Cited on pages 5, 16, 36, 62, 74, 83, and 99.*

[BMR23] Raphael Bost, Brice Minaud, and Michael Reichle. Mergeable searchable encryption and applications. In submission, 2023. *Cited on pages 11, 12, and 22.*

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press. *Cited on pages 103, 105, and 106.*

[BNPS03] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *J. Cryptol.*, 16(3):185–215, 2003. *Cited on pages 8, 19, and 102.*

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46, Miami, FL, USA, January 6–8, 2003. Springer, Heidelberg, Germany. *Cited on pages 8, 19, and 102.*

[Bos16] Raphael Bost. $\Sigma o \phi o \varsigma$: Forward secure searchable encryption. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1143–1154, Vienna, Austria, October 24–28, 2016. ACM Press. *Cited on pages 36, 62, 73, 74, 75, 76, 83, and 99.*

[Bow17] Sean Bowe. Bls12-381: New zk-snark elliptic curve construction. `https://electriccoin.co/blog/new-snark-curve/`, 2017. Accessed: 2023-02-02. *Cited on page 26.*

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press. *Cited on page 28.*

[Bra94] Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 302–318, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany. *Cited on pages 7, 18, and 101.*

[BS93] Eli Biham and Adi Shamir. Differential cryptanalysis of the full 16-round DES. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 487–496, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Heidelberg, Germany. *Cited on pages 2 and 13.*

[CAHL+22] Rutchathon Chairattana-Apirom, Lucjan Hanzlik, Julian Loss, Anna Lysyanskaya, and Benedikt Wagner. PI-cut-choo and friends: Compact blind signatures via parallel instance cut-and-choose and more. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 3–31, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany. *Cited on pages 7, 18, 102, and 151.*

[CFN90] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 319–327, Santa Barbara, CA, USA, August 21–25, 1990. Springer, Heidelberg, Germany. *Cited on pages 7, 18, and 101.*

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218, Dallas, TX, USA, May 23–26, 1998. ACM Press. *Cited on page 28.*

[CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 79–88, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press. *Cited on pages 35 and 62.*

[CGKR22] Geoffroy Couteau, Dahmun Goudarzi, Michael Klooß, and Michael Reichle. Sharp: Short relaxed range proofs. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 609–622, Los Angeles, CA, USA, November 7–11, 2022. ACM Press. *Cited on pages 10 and 21.*

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th FOCS*, pages 41–50, Milwaukee, Wisconsin, October 23–25, 1995. IEEE Computer Society Press. *Cited on pages 4 and 15.*

[CGLS17] T.-H. Hubert Chan, Yue Guo, Wei-Kai Lin, and Elaine Shi. Oblivious hashing revisited, and applications to asymptotically efficient ORAM and OPRAM. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 660–690, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany. *Cited on page 40.*

[CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 668–679, Denver, CO, USA, October 12–16, 2015. ACM Press. *Cited on pages 5 and 16.*

[Cha82] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203, Santa Barbara, CA, USA, 1982. Plenum Press, New York, USA. *Cited on pages 7, 8, 18, 19, 101, and 102.*

[Cha88] David Chaum. Elections with unconditionally-secret ballots and disruption equivalent to breaking RSA. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 177–182, Davos, Switzerland, May 25–27, 1988. Springer, Heidelberg, Germany. *Cited on pages 7, 18, and 101.*

[CJJ+13] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for Boolean queries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 353–373, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. *Cited on pages 5, 16, and 62.*

[CJJ+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS 2014*, San Diego, CA, USA, February 23–26, 2014. The Internet Society. *Cited on pages 62, 63, and 76.*

[CK10] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 577–594, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany. *Cited on page 99.*

[CK20] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 44–75, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. *Cited on pages 4 and 15.*

[CKLR21] Geoffroy Couteau, Michael Klooß, Huang Lin, and Michael Reichle. Efficient range proofs with transparent setup from bounded integer commitments. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 247–277, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. *Cited on pages 9 and 20.*

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany. *Cited on pages 7, 18, and 101.*

[CR19] Geoffroy Couteau and Michael Reichle. Non-interactive keyed-verification anonymous credentials. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part I*, volume 11442 of *LNCS*, pages 66–96, Beijing, China, April 14–17, 2019. Springer, Heidelberg, Germany. *Cited on pages 9 and 20.*

[CT14] David Cash and Stefano Tessaro. The locality of searchable symmetric encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 351–368, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany. *Cited on pages 5, 16, 17, 36, 62, 74, and 75.*

[DCPP20] Ioannis Demertzis, Javad Ghareh Chamani, Dimitrios Papadopoulos, and Charalampos Papamanthou. Dynamic searchable encryption with small client storage.

In *NDSS 2020*, San Diego, CA, USA, February 23–26, 2020. The Internet Society. *Cited on pages 4 and 15.*

[DCPP22] Ioannis Demertzis, Javad Ghareh Chamani, Dimitrios Papadopoulos, and Charalampos Papamanthou. Dynamic searchable encryption with small client storage. In *ISOC Network and Distributed System Security – NDSS 2022*, 2022. *Cited on pages 5, 16, 74, and 75.*

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. *Cited on pages 2 and 14.*

[dK22] Rafaël del Pino and Shuichi Katsumata. A new framework for more efficient round-optimal lattice-based (partially) blind signature via trapdoor sampling. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 306–336, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany. *Cited on pages 8, 19, 33, 102, 103, 104, 109, and 151.*

[DP17a] Ioannis Demertzis and Charalampos Papamanthou. Fast searchable encryption with tunable locality. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1053–1067, 2017. *Cited on pages 5, 17, 74, and 75.*

[DP17b] Ioannis Demertzis and Charalampos Papamanthou. Fast searchable encryption with tunable locality. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1053–1067. ACM, 2017. *Cited on page 62.*

[DPP18] Ioannis Demertzis, Dimitrios Papadopoulos, and Charalampos Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 371–406, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. *Cited on pages 5, 17, 62, and 74.*

[EKPE18] Mohammad Etemad, Alptekin Küpçü, Charalampos Papamanthou, and David Evans. Efficient dynamic searchable encryption with forward privacy. In *Proceedings on Privacy Enhancing Technologie – PoPETS 2018*, 2018. *Cited on pages 5, 16, and 74.*

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany. *Cited on page 119.*

[FF56] Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956. *Cited on page 26.*

[FHKS16] Georg Fuchsbauer, Christian Hanser, Chethan Kamath, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model from weaker assumptions. In Vassilis Zikas and Roberto De Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pages 391–408, Amalfi, Italy, August 31 – September 2, 2016. Springer, Heidelberg, Germany. *Cited on pages 8, 19, and 107.*

[FHS15] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 233–253, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany. *Cited on pages 8, 19, and 107.*

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany. *Cited on pages 9, 20, 32, 103, 104, and 106.*

[Fis06] Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 60–77, Santa Barbara, CA, USA, August 20–24, 2006. Springer, Heidelberg, Germany. *Cited on pages 8, 19, 102, 107, and 111.*

[FKL01] Scott Friedman, Anand Krishnan, and Nicholas Leidenfrost. Hash tables for embedded and real-time systems. *EEE Real-Time Embedded System Workshop*, 2001. *Cited on page 99.*

[FOO92] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *AUSCRYPT*, pages 244–251. Springer, 1992. *Cited on pages 7, 18, and 101.*

[FPS20] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed ElGamal encryption in the algebraic group model. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 63–95, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. *Cited on pages 7 and 18.*

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany. *Cited on page 103.*

[FS10] Marc Fischlin and Dominique Schröder. On the impossibility of three-move blind signature schemes. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 197–215, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany. *Cited on pages 8, 19, and 102.*

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press. *Cited on pages 4 and 15.*

[GG14] Sanjam Garg and Divya Gupta. Efficient round optimal blind signatures. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 477–495, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany. *Cited on pages 8 and 19.*

[Gha17] Essam Ghadafi. Efficient round-optimal blind signatures in the standard model. In Aggelos Kiayias, editor, *FC 2017*, volume 10322 of *LNCS*, pages 455–473, Sliema, Malta, April 3–7, 2017. Springer, Heidelberg, Germany. *Cited on pages 8 and 19.*

[GKM21] Marilyn George, Seny Kamara, and Tarik Moataz. Structured encryption and dynamic leakage suppression. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 370–396, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. *Cited on pages 5 and 16.*

[GLMP19] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks.

In *2019 IEEE Symposium on Security and Privacy*, pages 1067–1083, San Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press. *Cited on pages 4, 5, and 16.*

[GM11] Michael T. Goodrich and Michael Mitzenmacher. Privacy-preserving access of outsourced data via oblivious RAM simulation. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011, Part II*, volume 6756 of *LNCS*, pages 576–587, Zurich, Switzerland, July 4–8, 2011. Springer, Heidelberg, Germany. *Cited on page 58.*

[GMP16] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. TWORAM: Efficient oblivious RAM in two rounds with applications to searchable encryption. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 563–592, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. *Cited on pages 4 and 15.*

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996. *Cited on pages 4 and 15.*

[GOP+22] Chaya Ganesh, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 397–426, Trondheim, Norway, May 30 – June 3, 2022. Springer, Heidelberg, Germany. *Cited on pages 104 and 130.*

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for non-interactive zero-knowledge. *Journal of the ACM (JACM)*, 59(3):1–35, 2012. *Cited on page 102.*

[GPPW20] Zichen Gui, Kenneth G. Paterson, Sikhar Patranabis, and Bogdan Warinschi. SWiSSSE: System-wide security for searchable symmetric encryption. Cryptology ePrint Archive, Report 2020/1328, 2020. https://ia.cr/2020/1328. *Cited on page 99.*

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 89–98, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press. Available as Cryptology ePrint Archive Report 2006/309. *Cited on page 108.*

[GRS+11] Sanjam Garg, Vanishree Rao, Amit Sahai, Dominique Schröder, and Dominique Unruh. Round optimal blind signatures. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 630–648, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany. *Cited on pages 8 and 19.*

[HIP+22] Scott Hendrickson, Jana Iyengar, Tommy Pauly, Steven Valdez, and Christopher A. Wood. Private access tokens. internet-draft draft-private-access-tokens-01, April 2022. Work in Progress. *Cited on pages 7, 18, and 101.*

[HKL19] Eduard Hauck, Eike Kiltz, and Julian Loss. A modular treatment of blind signatures from identification schemes. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 345–375, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. *Cited on pages 7 and 18.*

[HLW23] Lucjan Hanzlik, Julian Loss, and Benedikt Wagner. Rai-choo! Evolving blind signatures to the next level. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 753–783, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany. *Cited on pages 7, 8, 19, 102, and 103.*

[JK77] Norman Lloyd Johnson and Samuel Kotz. *Urn models and their application; an approach to modern discrete probability theory.* New York, NY (USA) Wiley, 1977. *Cited on pages 6, 17, and 39.*

[JLO97] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 150–164, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Heidelberg, Germany. *Cited on page 30.*

[JR17] Charanjit S. Jutla and Arnab Roy. Improved structure preserving signatures under standard bilinear assumptions. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 183–209, Amsterdam, The Netherlands, March 28–31, 2017. Springer, Heidelberg, Germany. *Cited on pages 102 and 107.*

[Kat10] Jonathan Katz. Digital signatures: Background and definitions. In *Digital Signatures*. Springer, 2010. *Cited on page 30.*

[KLR21] Jonathan Katz, Julian Loss, and Michael Rosenberg. Boosting the security of blind signature schemes. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 468–492, Singapore, December 6–10, 2021. Springer, Heidelberg, Germany. *Cited on pages 7, 18, 102, and 151.*

[KLX22a] Julia Kastner, Julian Loss, and Jiayu Xu. The Abe-Okamoto partially blind signature scheme revisited. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part IV*, volume 13794 of *Lecture Notes in Computer Science*, pages 279–309. Springer, 2022. *Cited on pages 7 and 18.*

[KLX22b] Julia Kastner, Julian Loss, and Jiayu Xu. On pairing-free blind signature schemes in the algebraic group model. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part II*, volume 13178 of *Lecture Notes in Computer Science*, pages 468–497. Springer, 2022. *Cited on pages 7 and 18.*

[KM18] Seny Kamara and Tarik Moataz. SQL on structurally-encrypted databases. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 149–180, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany. *Cited on pages 5 and 16.*

[KMO18] Seny Kamara, Tarik Moataz, and Olga Ohrimenko. Structured encryption and leakage suppression. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 339–370, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. *Cited on pages 4 and 15.*

[KMPQ21] Seny Kamara, Tarik Moataz, Andrew Park, and Lucy Qin. A decentralized and encrypted national gun registry. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1520–1537. IEEE, 2021. *Cited on pages 5, 16, and 74.*

[KMW10] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM Journal on Computing*, 39(4):1543–1561, 2010. *Cited on pages 6, 17, 40, 55, and 58.*

[KNYY21] Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Round-optimal blind signatures in the plain model from classical and quantum standard assumptions. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 404–434, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. *Cited on pages 8 and 19.*

[KPW15] Eike Kiltz, Jiaxin Pan, and Hoeteck Wee. Structure-preserving signatures from standard assumptions, revisited. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 275–295, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany. *Cited on pages 107, 119, 120, and 121.*

[KRS23] Shuichi Katsumata, Michael Reichle, and Yusuke Sakai. Practical round-optimal blind signatures in the rom. To appear at Asiacrypt, 2023. *Cited on pages 11, 12, 21, and 22.*

[KSD19] Mojtaba Khalili, Daniel Slamanig, and Mohammad Dakhilalian. Structure-preserving signatures on equivalence classes from standard assumptions. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 63–93, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany. *Cited on pages 8, 19, and 107.*

[KW15] Eike Kiltz and Hoeteck Wee. Quasi-adaptive NIZK for linear subspaces revisited. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 101–128, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany. *Cited on page 120.*

[Lin08] Yehuda Lindell. Lower bounds and impossibility results for concurrent self composition. *Journal of Cryptology*, 21(2):200–249, April 2008. *Cited on pages 8, 19, and 102.*

[LMW22] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. Cryptology ePrint Archive, Report 2022/1703, 2022. https://eprint.iacr.org/2022/1703. *Cited on pages 4 and 16.*

[LN18] Kasper Green Larsen and Jesper Buus Nielsen. Yes, there is an oblivious RAM lower bound! In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 523–542, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. *Cited on pages 4, 15, and 85.*

[MM17] Ian Miers and Payman Mohassel. IO-DSSE: Scaling dynamic searchable encryption to millions of indexes by improving locality. In *NDSS 2017*, San Diego, CA, USA, February 26 – March 1, 2017. The Internet Society. *Cited on pages 4, 5, 15, 17, 62, 63, 74, 75, 76, 83, and 85.*

[Mon22] MongoDB. Queryable encryption. https://www.mongodb.com/products/queryable-encryption, 2022. *Cited on pages 3 and 15.*

[MR02] Silvio Micali and Leonid Reyzin. Improving the exact security of digital signature schemes. *Journal of Cryptology*, 15(1):1–18, January 2002. *Cited on page 106.*

[MR22] Brice Minaud and Michael Reichle. Dynamic local searchable symmetric encryption. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 91–120, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany. *Cited on pages 10, 12, 21, 22, and 151.*

[MR23] Brice Minaud and Michael Reichle. Hermes: I/o-efficient forward-secure searchable symmetric encryption. To appear at Asiacrypt, 2023. *Cited on pages 10, 12, 21, 22, and 74.*

[MSF10] Sarah Meiklejohn, Hovav Shacham, and David Mandell Freeman. Limitations on transformations from composite-order to prime-order groups: The case of round-optimal blind signatures. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 519–538, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany. *Cited on pages 8 and 19.*

[Nao03] Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 96–109, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany. *Cited on pages 8 and 19.*

[Nis20] Ryo Nishimaki. Equipping public-key cryptographic primitives with watermarking (or: A hole is to watermark). In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 179–209, Durham, NC, USA, November 16–19, 2020. Springer, Heidelberg, Germany. *Cited on pages 103, 108, and 109.*

[NKW15] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 644–655, Denver, CO, USA, October 12–16, 2015. ACM Press. *Cited on pages 4 and 16.*

[Oka93] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 31–53, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Heidelberg, Germany. *Cited on pages 7 and 18.*

[OO92] Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 324–337, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany. *Cited on pages 7, 18, and 101.*

[oSN77] National Institute of Standards and Technology (NIST). Data encryption standard (des). *FIPS PUB 46*, 1977. *Cited on pages 2 and 13.*

[oSN01] National Institute of Standards and Technology (NIST). Advanced encryption standard (aes). *FIPS PUB 197*, 2001. *Cited on pages 2 and 14.*

[Pas11] Rafael Pass. Limits of provable security from standard assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 109–118, San Jose, CA, USA, June 6–8, 2011. ACM Press. *Cited on pages 8, 19, and 102.*

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany. *Cited on page 119.*

[Pet15] T. Kyle Petersen. *Eulerian numbers*. Springer, 2015. *Cited on page 58.*

[PKV+14] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos D. Keromytis, and Steve Bellovin. Blind seer: A scalable private DBMS. In *2014 IEEE Symposium on Security and Privacy*, pages 359–374, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press. *Cited on pages 5 and 16.*

[PM21] Sikhar Patranabis and Debdeep Mukhopadhyay. Forward and backward private conjunctive searchable symmetric encryption. In *ISOC Network and Distributed System Security – NDSS 2021*, 2021. *Cited on pages 5, 16, and 74.*

[Poi98] David Pointcheval. Strengthened security for blind signatures. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 391–405, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany. *Cited on pages 7, 18, and 102.*

[Pol78] John M Pollard. Monte carlo methods for index computation (mod p). *Mathematics of computation*, 32(143):918–924, 1978. *Cited on page 143.*

[PPYY19] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 79–93, London, UK, November 11–15, 2019. ACM Press. *Cited on page 40.*

[PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004. *Cited on pages 6, 17, and 40.*

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000. *Cited on pages 7, 18, 24, 103, 105, 106, 135, and 141.*

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In Jaeyeon Jung and Thorsten Holz, editors, *USENIX Security 2015*, pages 515–530, Washington, DC, USA, August 12–14, 2015. USENIX Association. *Cited on page 40.*

[PSWW18] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 125–157, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany. *Cited on page 58.*

[PTW10] Yuval Peres, Kunal Talwar, and Udi Wieder. The (1+ β)-choice process and weighted balls-into-bins. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1613–1619. SIAM, 2010. *Cited on page 40.*

[RMS01] Andrea W Richa, M Mitzenmacher, and R Sitaraman. The power of two random choices: A survey of techniques and results. *Combinatorial Optimization*, 9:255–304, 2001. *Cited on page 40.*

[SC12] Jae Hong Seo and Jung Hee Cheon. Beyond the limitation of prime-order bilinear groups, and round optimal blind signatures. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 133–150, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany. *Cited on pages 8 and 19.*

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany. *Cited on pages 7 and 18.*

[Sch03] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003. *Cited on page 25.*

[SG98] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 1–16, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany. *Cited on pages 106 and 109.*

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949. *Cited on pages 1 and 13.*

[SPS14] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *NDSS 2014*, San Diego, CA, USA, February 23–26, 2014. The Internet Society. *Cited on pages 4, 16, 36, 74, and 75.*

[Sta21] Statista. Shipments of hard and solid state disk (hdd/ssd) drives worldwide from 2015 to 2021. https://www.statista.com/statistics/285474/hdds-and-ssds-in-pcs-global-shipments-2012-2017/, 2021. *Cited on page 62.*

[SW97] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *Journal of the ACM (JACM)*, 44(4):585–591, 1997. *Cited on page 26.*

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany. *Cited on page 108.*

[SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55, Oakland, CA, USA, May 2000. IEEE Computer Society Press. *Cited on pages 3, 4, 15, and 16.*

[TW07] Kunal Talwar and Udi Wieder. Balanced allocations: the weighted case. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 256–265, 2007. *Cited on pages 40 and 41.*

[TW14] Kunal Talwar and Udi Wieder. Balanced allocations: A simple proof for the heavily loaded case. In *International Colloquium on Automata, Languages, and Programming*, pages 979–990. Springer, 2014. *Cited on pages 40 and 41.*

[TZ22] Stefano Tessaro and Chenzhi Zhu. Short pairing-free blind signatures with exponential security. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 782–811, Trondheim, Norway, May 30 – June 3, 2022. Springer, Heidelberg, Germany. *Cited on pages 7 and 18.*

[Unr12] Dominique Unruh. Quantum proofs of knowledge. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 135–152, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany. *Cited on pages 32 and 106.*

[Vöc03] Berthold Vöcking. How asymmetry helps load balancing. *Journal of the ACM (JACM)*, 50(4):568–589, 2003. *Cited on page 46.*

[VPN22] Vpn by Google one, explained. https://one.google.com/about/vpn/howitworks, 2022. Accessed: 2023-02-02. *Cited on pages 7, 18, and 101.*

[Wat05] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 114–127, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany. *Cited on pages 8, 19, 102, and 108.*

[Wie17] Udi Wieder. Hashing, load balancing and multiple choice. *Foundations and Trends in Theoretical Computer Science*, 12(3–4):275–379, 2017. *Cited on pages 42 and 58.*

[Yeo23] Kevin Yeo. Lower bounds for (batch) PIR with private preprocessing. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 518–550, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany. *Cited on pages 4 and 15.*

[YL19] Xun Yi and Kwok-Yan Lam. A new blind ECDSA scheme for bitcoin transaction anonymity. In Steven D. Galbraith, Giovanni Russello, Willy Susilo, Dieter Gollmann, Engin Kirda, and Zhenkai Liang, editors, *ASIACCS 19*, pages 613–620, Auckland, New Zealand, July 9–12, 2019. ACM Press. *Cited on pages 7, 18, and 101.*

[ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 707–720, Austin, TX, USA, August 10–12, 2016. USENIX Association. *Cited on pages 4, 16, and 74.*

RÉSUMÉ

Dans cette thèse, nous proposons des constructions efficaces de primitives cryptographiques avec des fonctionnalités avancées. Nous nous concentrons sur des primitives qui permettent des applications préservant la confidentialité, telles que la recherche sur des données chiffrées ou le vote électronique, avec une sécurité prouvable dans le modèle de l'oracle aléatoire (ROM). En particulier, nous construisons des schémas de *chiffrement avec recherche* (SSE) et des schémas de *signature aveugle*.

SSE permet à un client d'effectuer des requêtes par mots-clés sur une base de données chiffrée stockée sur un serveur distant. Pour obtenir le résultat, le serveur effectue souvent un grand nombre d'accès aléatoires à la mémoire. En conséquence, le débit mémoire d'un schéma SSE est souvent le principal goulot d'étranglement. Pour nos constructions, nous proposons d'abord des variantes de schémas de hachage classiques qui permettent l'allocation d'éléments pondérés. Sur la base de ces variantes, nous construisons plusieurs schémas SSE avec une bonne efficacité mémoire sur les supports de stockage modernes. Cela inclut Pluto, un schéma SSE statique avec une efficacité mémoire optimale, et Hermes, un schéma SSE dynamique avec une efficacité mémoire sous-logarithmique et sécurité persistante.

Les signatures aveugles servent d'outil fondamental pour les applications préservant la confidentialité (par exemple, le vote électronique, les jetons confidentiels d'authentification, les chaînes de blocs). Nous présentons deux cadres optimisés pour construire des signatures aveugles dans le ROM. Nous instancions chaque cadre dans le contexte des couplages et obtenons des signatures aveugles efficaces avec un nombre optimal de tours sous des hypothèses standards. La première construction est une variante hautement optimisée de la construction générique de signature aveugle de Fischlin (CRYPTO'06). Notre deuxième construction est une construction semi-générique à partir d'une classe spécifique de schémas de signature randomisables qui admet une réduction *tous-sauf-un*.

MOTS CLÉS

Chiffrement avec Recherche ⋆ Hachage Pondéré ⋆ Signature Aveugle

ABSTRACT

In this thesis, we propose efficient constructions of cryptographic primitives with advanced functionalities. We focus on primitives that enable privacy-preserving applications, such as encrypted search or electronic voting, with provable security in the random oracle model (ROM). In particular, we construct *searchable symmetric encryption* (SSE) and *blind signature* schemes.

SSE allows a client to perform keyword queries on an encrypted database stored on a distant server. To obtain the result, the server often performs a large number of random memory accesses. In consequence, the memory throughput of an SSE scheme is often the main bottleneck. For our constructions, we first propose variants of classical hashing schemes that allow for allocation of weighted items. Based on these variants, we construct several SSE schemes with good memory efficiency on modern storage media. This includes Pluto, a static SSE scheme with optimal memory efficiency, and Hermes, a dynamic SSE scheme with sublogarithmic memory efficiency and forward security.

Blind signatures serve as a foundational tool for privacy-preserving applications (e.g., electronic voting, privacy-authentication tokens, blockchains). We present two optimized frameworks to construct blind signatures in the ROM. We instantiate each framework in the pairing setting and obtain efficient round-optimal blind signatures under standard assumptions. The first construction is a highly optimized variant of the generic blind signature construction by Fischlin (CRYPTO'06). Our second construction is a semi-generic construction from a specific class of randomizable signature schemes that admits an *all-but-one* reduction.

KEYWORDS

Searchable Symmetric Encryption ⋆ Weighted Hashing ⋆ Blind Signature