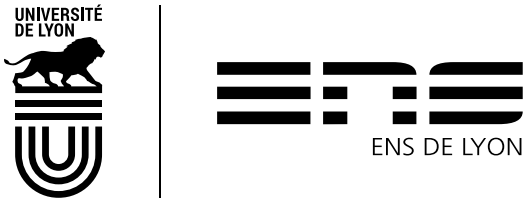
Constrained Pseudorandom Functions: New Constructions and Connections with Secure Computation

Mahshid Riahinia



THÈSE

en vue de l'obtention du grade de Docteur, délivré par
l'École Normale Supérieure de Lyon

École Doctorale N° 512
École Doctorale en Informatique et Mathématiques de Lyon

Discipline : Informatique

Soutenue publiquement le 08/07/2024, par
Mahshid RIAHINIA

Constrained Pseudorandom Functions: New Constructions and Connections with Secure Computation

Fonctions Pseudo-Aléatoires Contraintes:
Nouvelles Constructions et Liens avec Le Calcul Sécurisé

Devant le jury composé de :

Pointcheval David, Directeur de recherche, ENS	Rapporteur
Vergnaud Damien, Professeur des universités, Sorbonne Université	Rapporteur
Canard Sébastien, Professeur, Télécom Paris	Examinateur
Kohl Lisa, Personnalité scientifique, CWI	Examinatrice
Passelègue Alain, Chercheur, ENS de Lyon	Examinateur
Ràfols Carla, Professeur, Universitat Pompeu Fabra	Examinatrice
Villard Gilles, Directeur de recherche, CNRS et ENS de Lyon	Directeur de thèse

To Pouria, who taught me that it is okay to spend more than 5 minutes on a problem!

Abstract

Pseudorandom functions (PRFs) were introduced in 1986 by Goldreich, Goldwasser, and Micali as efficient means of generating randomness and serve as essential tools in cryptography. These functions use a master secret key to map different inputs to pseudorandom outputs. Constrained pseudorandom functions (CPRFs), introduced in 2013, extend PRFs by additionally allowing the delegation of *constrained keys* that enable the evaluation of the function *only* on specific subsets of inputs. Notably, given a constrained key that evaluates the function on a subset of inputs, the output of a CPRF should remain pseudorandom on inputs outside of this subset. In this thesis, we establish links between CPRFs and two other cryptographic tools which were introduced in the context of secure computation:

1. We show how CPRFs can be constructed from Homomorphic secret sharing (HSS) protocols. Homomorphic secret sharing protocols allow distributed computations over shares of a secret. We start by identifying two extensions of HSS protocols and show how they can be transformed into CPRFs generating constrained keys for subset of inputs that can be expressed via inner-product and NC^1 predicates. Next, we observe that HSS protocols that already exist in the literature can be adapted to these new extensions. This leads to the discovery of five new CPRF constructions based on various standard hardness assumptions.

2. We show how CPRFs can be used to construct pseudorandom correlation functions (PCFs) for oblivious transfer (OT) correlations. PCFs for OT correlations enable two parties to generate OT-correlated pairs that can be used in fast secure computation protocols. Next, we instantiate our transformation by applying a slight modification to the well-known PRF construction of Naor and Reingold. We finally present a method for the non-interactive generation of evaluation keys for the latter instantiation which results in an efficient *public-key* PCF for OT correlations from standard assumptions.

Résumé

Les fonctions pseudo-aléatoires (Pseudorandom Functions, alias PRFs) ont été introduites en 1986, par Goldreich, Goldwasser et Micali, comme moyen efficace de générer de l'aléa et servent depuis d'outils essentiels en cryptographie. Ces fonctions utilisent une clé secrète principale pour faire correspondre différentes entrées à des sorties pseudo-aléatoires. Les fonctions pseudo-aléatoires contraintes (Constrained Pseudorandom Functions, alias CPRFs), introduites en 2013, étendent les PRFs en autorisant la délégation des *clés contraintes* qui permettent l'évaluation de la fonction *uniquement* sur des sous-ensembles spécifiques d'entrées. Notamment, même avec cette évaluation partielle, la sortie d'une CPRF devrait rester pseudo-aléatoire sur les entrées en dehors de ces sous-ensembles. Dans cette thèse, nous établissons des liens entre les CPRFs et deux autres outils cryptographiques qui ont été introduits dans le contexte du calcul sécurisé :

1. Nous montrons comment les CPRFs peuvent être construites à partir de protocoles de partage de secrets homomorphes (Homomorphic Secret Sharing, alias HSS). Les protocoles de partage de secrets homomorphes permettent des calculs distribués sur des parties d'un secret. Nous commençons par identifier deux nouvelles versions des protocoles HSS et montrons comment elles peuvent être transformées en CPRFs générant des clés contraintes pour des sous-ensembles d'entrées qui peuvent être exprimés via des prédicats de produit scalaire ou de NC^1. Ensuite, nous observons que les constructions de protocoles HSS qui existent déjà dans la littérature peuvent être adaptées à ces nouvelles extensions. Cela conduit à la découverte de cinq nouvelles constructions CPRF basées sur diverses hypothèses de sécurité standards.

2. Nous montrons comment les CPRFs peuvent être utilisées pour construire des fonctions de corrélation pseudo-aléatoires (Pseudorandom Correlation Functions, alias PCFs) pour les corrélations de transfert inconscient (Oblivious Transfer, alias OT). Les PCFs pour les corrélations OT permettent à deux parties de générer des paires corrélées OT qui peuvent être utilisées dans des protocoles de calcul sécurisés rapides. Ensuite, nous détaillons l'instanciation de notre transformation en appliquant une légère modification à la construction PRF bien connue de Naor et Reingold. Enfin, nous présentons une méthode de génération non-interactive de clés d'évaluation pour cette dernière instanciation, qui permet d'obtenir un PCF *à clé publique* efficace pour les corrélations OT à partir d'hypothèses standards.

Contents

Chapter 1

Introduction

Many situations in our daily life necessitate a degree of *privacy*. Privacy, typically used as an umbrella term, captures a wide range of requirements that in essence involve the desire to hide certain information while maintaining the ability to use them for practical purposes. For instance when using a messaging application, we want our intended recipient to be able to recover our message (functionality) while ensuring that no one else has the ability of doing so (privacy). Cryptography addresses this goal by providing frameworks that mathematically model privacy across different contexts. Consequently, cryptographic protocols emerge as sets of algorithms designed to guarantee specific notion of privacy while providing certain functionality within given scenarios.

The most basic cryptographic protocols address scenarios where privacy can be provided through *encryption* and *authentication*. Examples of such protocols include secret-key and public-key encryption, message authentication codes, digital signatures, key exchange protocols, pseudorandom functions, and one-way functions. These protocols form the backbone of cryptography. They have numerous applications in real-life scenarios and serve as a basis for developing more advanced protocols.

As the field of cryptography expanded and our foundational tools became more developed, more *advanced* protocols emerged that required more than simple encryption and authentication. These protocols call for more involved security notions and more refined functionalities. Examples of such protocols include functional encryption, homomorphic encryption, homomorphic secret sharing protocols, and multi-party computation. These protocols tackle more complex scenarios and can be considered as being more tuned to address real-world use cases.

As the need for privacy arises across various scenarios in our world, many different protocols are defined in cryptography, each uniquely tailored for a specific task. As a result, zooming out, the landscape of cryptographic protocols contains innumerable abstractions, each with unique technical details, with very few connections identified among them. An impactful direction of research is therefore zooming in on these protocols and diving into their details in order to uncover their subtle relations, even though the connection might not be immediately apparent. In particular, it yields significant results to focus on protocols that are promising to imply multiple other ones. Identifying such relations not only leads to a deep understanding of the protocols but, more importantly, reveals opportunities for new designs. For instance, if protocol A can serve as a building block for protocol B, any new and improved construction of protocol A implies a correspondingly

enhanced construction for protocol B. In this thesis, we zoom in on a protocol that, while simple in definition, has impressive theoretical and practical implications: constrained pseudorandom functions.

Constrained pseudorandom functions can be viewed as advanced protocols that enable the generation of randomness. In the following section, we begin with a brief overview of randomness generation protocols, and then introduce constrained pseudorandom functions.

1

Randomness Generation Protocols

The problem of generating random numbers is a well-known question in computer science and plays a fundamental role in cryptography. Recall that the primary objective of cryptography is to provide solutions to hide certain information. Randomness plays a major role in achieving this goal, as it carries *unpredictability*. However, while flipping a coin in real life seems like an effortless task, it is quite the opposite for computers. Since tasks on computers are performed by deterministic algorithms, it is paradoxical to expect the numbers generated by an algorithm to be completely random, in particular, to have no relationship to each other. As a result the concept of *Pseudorandom Generators* (PRGs) was introduced by Blum and Micali [BM84]. PRGs are efficient deterministic functions that transform a short random string, called the *seed*, into a larger string that *appears to be random* (hence the prefix "pseudo"). While PRGs require an initial short randomness, they significantly facilitate the process of generating randomness; one can generate a short amount of randomness by using *physical number generators* that utilize natural sources of entropy like electrical noise in physical systems. A drawback of pseudorandom generators is that a PRG can expand a given seed, that should be itself random, *only once* to a pseudorandom string of a length that is fixed in advance.

Pseudorandom Functions (PRFs) were subsequently introduced by Goldreich, Goldwasser and Micali [GGM86] as generalizations of PRGs. While PRGs generate pseudorandom strings, PRFs provide pseudorandom *functions*. More specifically, a PRF is a family of deterministic function $\{F_k : \mathcal{X} \to \mathcal{Y}\}_{k\in\mathcal{K}}$ with domain $\mathcal{X}$ and range $\mathcal{Y}$, where each F_k uses a master secret key $k \in \mathcal{K}$ to map different inputs to pseudorandom outputs. More formally, it is required for a *secure* PRF that any randomly selected function F_k should look like a truly random function with the same domain and range from the point of view of anyone with limited computational power who does not have access to the master secret key.

Since their introduction, pseudorandom functions have found many applications in theory and practice of cryptography. As their most basic application, PRFs can be used to construct simple *secret-key encryption* (SKE) protocols. Secret-key encryption allows two parties, that hold a common secret key, to privately communicate over a public channel. This communication involves the sender encrypting the message using the secret key and sending the ciphertext to the receiver, who then decrypts it using the same key to recover the message. SKE can be constructed using PRFs as follows: a secret key in this scheme consists of a random master secret key k of a PRF F. To encrypt a message m, one adds the output of a PRF F_k on a random string r to the message m and outputs a ciphertext $(r, m \oplus F_k(r))$. Decryption proceeds by using the secret key k to compute $F_k(r)$, and removing this value from the second part of the ciphertext to recover m. Such a ciphertext hides the underlying message m from any third party that does not have access to the secret key k. This is because m is masked by the value $F_k(r)$ which looks

random, and therefore cannot be guessed or removed without knowing k.

From a more theoretical perspective, seminal works of [GGM86], [GL89], and [HILL99] showed the equivalency of PRFs and PRGs to *one-way functions* (OWFs). One-way functions are efficiently-computable functions that are hard to invert and count as fundamental tools in cryptography. The equivalency of PRFs to one-way functions in particular implies that PRFs are necessary and sufficient for all *private-key* cryptography.

Note that PRFs contribute not only to the security of protocols, but also to their efficiency. More precisely, instead of storing a large number of random elements, one can store only a short master secret key of a PRF and generate randomness, on the fly, by evaluating the PRF on different inputs. Interestingly, another form of randomness, called *correlated randomness*, which consists of pairs of random elements that jointly belong to a certain distribution, has remarkable effects on the efficiency of protocols. Particularly, correlated randomness contributes to decreasing the communication in *secure computation* protocols, where two or more parties aim to evaluate a function over their private inputs without revealing more information than what can be derived from the output value. The seminal work of Beaver [Bea92] showed that, in such scenarios, if the parties have access to triples of additive shares of some random elements a, b, c such that $c = ab$, they can evaluate any function while communicating only 2 field elements per multiplication gate.[1] Several studies show that, in general, protocol performance improves significantly when the parties have access to correlated randomness [DPSZ12, NNOB12, FKOS15, LPSY15, WRK17, HSS20]. The notion of *pseudorandom correlation functions* (PCFs) was subsequently introduced by Boyle, Couteau, Gilboa, Ishai, Kohl and Scholl [BCG+20] as efficient means of generating correlated randomness on demand. PCFs allow two parties, that have two short correlated keys, to locally map different inputs to pseudorandom correlated strings. We discuss further the notion of PCFs later in this section.

Constrained PRFs. Over a decade ago, a new variant of PRFs was introduced concurrently in [BW13, KPTZ13, BGI14] under the name *constrained pseudorandom functions* (CPRFs). Constrained PRFs are PRFs that additionally allow the owner of a master secret key to delegate partial keys for different subsets of inputs. These partial keys can be used to locally evaluate the function on these subsets while the output of the function still appears random on inputs outside of these subsets. More formally, a CPRF is a family of pseudorandom functions $\{F_k : \mathcal{X} \to \mathcal{Y}\}_{k\in\mathcal{K}}$, where having access to a master secret key $k \in \mathcal{K}$, one can generate a *constrained key* ck_S for a (possibly superpolynomially-large) subset $S \subset \mathcal{X}$. Given a constrained key ck_S, one can locally compute the value of $F_k(x)$ on all inputs $x \in \mathcal{X}$ *iff* $x \in S$. More importantly, given a constrained key ck_S but not the master secret key k, the restriction of F_k on inputs $x \notin S$ should still be a pseudorandom function.

The simplest variants of constrained PRFs are called *puncturable PRFs* (PPRFs). These pseudorandom functions allow the generation of keys that enable evaluating the PRF on all inputs except for one, referred to as *the punctured point*. The works of [BW13, KPTZ13, BGI14] showed that the tree-based PRF construction of [GGM86] yields a simple puncturable PRF. We briefly recall this construction. Let G be a pseudorandom generator that maps λ-bit inputs to 2λ-bit strings, and for any $k \in \{0,1\}^\lambda$, let $G_0(k)$ and $G_1(k)$ be respectively the first and second half of the string $G(k)$. The GGM pseudorandom function is defined as $\{F_k : \{0,1\}^n \to \{0,1\}^\lambda\}_{k\in\{0,1\}^\lambda}$, where $F_k(x) = G_{x_n}(G_{x_{n-1}}(\cdots(G_{x_1}(k))))$

[1] Such triples are called *Beaver triples*.

for any bit-string $x = x_1 \cdots x_n \in \{0,1\}^n$. This construction essentially defines a binary tree, as illustrated in Figure 1.1 for 3-bit inputs, where the root corresponds to a randomly-selected PRF key k, and each internal node can be computed by applying the PRG G to its parent. Finally, each leaf represents the PRF output when evaluated on the label associated with that leaf. In Figure 1.1, we have $F_k(x) = k_x$. Regarding the security, if G outputs a pseudorandom string when evaluated on a random seed, we can start by marking the root that contains a random key, and repeatedly replace the two children of a marked node by random strings and mark them. Repeating this process n times until reaching the leaves proves that the outputs of this PRF look random.

This construction admits a straightforward method for generating a punctured key that can evaluate the function on all inputs but one. Such a punctured key simply blocks the path from the root to the punctured leaf, and includes the values of some nodes that are enough for computing all other leaves of the tree. For instance, for the 3-bit-input construction shown in Figure 1.1, a punctured key that evaluates the PRF on all inputs except for $x = 011$ simply consists of the values of the nodes k_{010}, k_{00} and k_1. Given the values of these nodes, one can compute the value of any leaf except for $x = 011$, by recursively applying G on the given nodes. It can be shown that in general, when considering n-bit inputs, a punctured key consists of values of n nodes. It is also interesting to note that a punctured key in this construction reveals the punctured point. For instance, in our example, given the nodes k_{010}, k_{00} and k_1, it is easy to determine that the punctured point is $x = 011$.

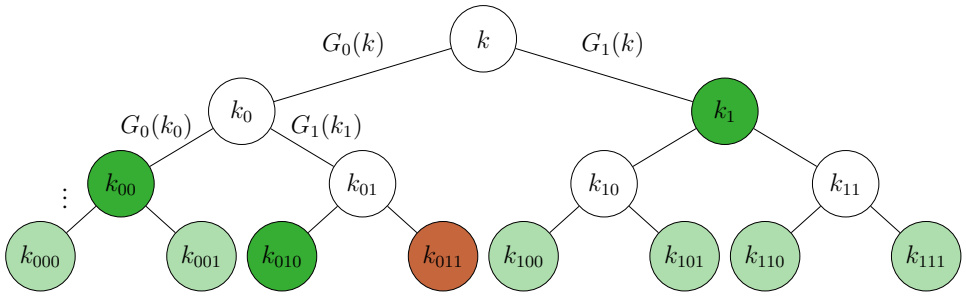


Figure 1.1 – Tree-based puncturable PRF construction of GGM for 3-bit inputs

A commonly considered variant of constrained PRFs in the literature is known as *circuit PRFs*. Put simply, these are constrained PRFs where a constrained key is associated with a circuit. For instance a constrained key ck_C for a predicate $C : \mathcal{X} \to \{0,1\}$ (called the constraint) enables the evaluation of the PRF on all inputs $x \in \mathcal{X}$ *iff* $C(x) = 0$. Throughout this thesis, we mainly discuss this variant.

The simple feature of partial key generation that CPRFs additionally provide compared to PRFs makes constrained PRFs considerably powerful tools with applications beyond standard PRFs. In particular, while PRFs can be used to construct basic private-key encryption schemes, in which a secret key can recover the plaintext in its entirety, CPRFs offer private-key encryption with more fine-grained decryption. More specifically, CPRFs can be straightforwardly used to construct secret-key *attribute-based encryption* (ABE) schemes, where a secret-key is associated with a policy and can decrypt a message associated with an attribute *iff* the policy of the key is satisfied by the attribute of the message. For instance, using an ABE scheme, one can tag their encrypted data with dates, and generate decryption keys that allows a user to recover the data within specific date ranges. Secret-key ABE can be constructed from PRFs as follows: a secret key consists

of a random master secret key k of a CPRF F. When encrypting a message m with an attribute a, one can output a ciphertext $(a, b = m \oplus F_k(a))$. Such a ciphertext hides the message m as long as the key k remains secret. To decrypt, anyone that holds a constrained key ck_C for a policy C such that $C(a) = 0$, can compute $F_k(a)$, remove this value from the ciphertext and recover m. Note that given a constrained key $\mathsf{ck}_{C'}$ for a policy C' such that $C'(a) \neq 0$, (and not the master secret key k), the value $F_k(a)$ remains pseudorandom, and therefore such a ciphertext still hides the underlying message m since it is masked by $F_k(a)$.

In the following, we provide a selection of non-exhaustive examples highlighting other applications of CPRFs.

Boneh and Waters [BW13] showed that CPRFs can be used to build broadcast encryption as well as identity-based and policy-based non-interactive key distribution mechanisms. Sahai and Waters [SW14] showed that puncturable PRFs can serve as the main ingredient in many applications of indistinguishability obfusecation ($i\mathcal{O}$) such as deniable encryption, chosen-ciphertext-secure public-key encryption, non-interactive zero-knowledge proofs, and more. Canetti and Chen [CC17] showed that *constraint-hiding* CPRFs imply powerful tools in cryptography. Constraint-hiding CPRFs generate constrained keys that reveal no information about the constraints.[1] In particular, the work of Canetti and Chen [CC17] shows that such CPRFs imply function-hiding secret-key functional encryption. The same work also demonstrates that constraint-hiding CPRFs capable of generating multiple constrained keys, also known as *multi-key* or *collusion-resistant* CPRFs, imply indistinguishability obfusecation ($i\mathcal{O}$). Tsabary [Tsa19] showed how constrained PRFs can be used to transform a selectively-secure attribute-based encryption to an adaptively-secure one.

Due to their various powerful applications, CPRFs have captured the attention of many cryptographers, leading to many efforts aimed at constructing CPRFs that support expressive constraints. These works try to improve state-of-the-art CPRFs in directions that, in essence, can be categorized as follows:

- ***Family of circuits***: Perhaps the first question that arises in constructing CPRFs is whether we can have constrained PRFs that support expressive families of constraints. Interesting families of constraints that we focus on in this thesis are NC^1 (the class of logarithmic-depth polynomial-sized circuits), $\mathsf{P/poly}$ (the class of polynomial-sized circuits), and inner-product predicates. For inner-product predicates, a constrained key $\mathsf{ck}_{\mathbf{z}}$ can evaluate on any input $\mathbf{x}$ *iff* $\langle \mathbf{x}, \mathbf{z} \rangle = 0$. It is worth to note that while constraint-hiding CPRFs for these classes of constraints are known from the learning with errors (LWE) assumption ([BTVW17, CC17, CVW18, PS18]), other families of standard assumption have so far failed to construct such CPRFs.

- ***Constraint-Hiding***: The notion of *constraint-hiding* CPRFs was first introduced by [BLW17] and indicates whether a constrained PRF generates constrained keys that hide the underlying constraints. While seemingly simple, adapting existing constructions to satisfy this property presents non-trivial challenges, even for basic constraints such as puncturing. For instance, the well-known tree-based PRF construction of [GGM86] is inherently a puncturable PRF. However, in this

[1] Constraint-hiding CPRFs are also known as private CPRFs.

construction, given a constrained-key, one can efficiently recover the punctured point. This notion of privacy regarding constrained keys contributes in various applications of CPRFs such as searchable encryption, watermarking PRFs, function-hiding functional encryption, $i\mathcal{O}$, *etc.*, and therefore many works are directed towards achieving this property. To this date, lattice-based constructions [BTVW17, CC17, CVW18, PS18, DKN+20] have shown to be more successful in this direction.

- ***Security (Adaptive vs. Selective)***: The most general security notions defined for CPRFs, e.g., that of [BW13], require that given a constrained key ck_C and oracle access to the evaluation of a CPRF, an adversary should not be able to distinguish the output of the CPRF on any unqueried input x, satisfying $C(x) = 1$, from random. This notion, called *adaptive* security, does not impose any restriction on the order of constrained key and evaluation queries. Achieving adaptive security proves to be quite challenging within the standard model and without using strong tools such as $i\mathcal{O}$, particularly for expressive constraints.[1] For instance, for inner-product predicates, the only known construction of adaptive CPRFs in the standard model is proposed by [DKN+20] from the learning with errors (LWE) assumption. On the contrary, *selective* security, restricts the adversary to submit the constrained key query before having access to the evaluation oracle. This relaxed notion of security still captures the security needs across different applications, and various methods, (e.g., complexity leveraging, using a random oracle or correlated-input secure hash functions, *etc.*) have been proposed to push it towards adaptive security.

- ***Collusion Resistance***: In the original security definition of CPRFs, e.g., in [BW13], an adversary is allowed to ask for *multiple* constrained keys for different constraints. However, it appears to be exceptionally difficult to construct CPRFs that allow for generation of multiple (even 2) constrained keys without compromising their security. To this day, the only known constructions that admit this property, called *collusion resistance*, leverage $i\mathcal{O}$ [BZ14, BLW17, HKKW19, DKN+20]. We recall that, due to the work of Canetti and Chen [CC17], constraint-hiding CPRFs that can generate two constrained keys imply $i\mathcal{O}$.

In Table 1.1, we summarize known CPRFs and their properties for constraints that can be expressed as $\mathsf{P/poly}$, NC^1, or inner-product predicates.

[1] We refer to standard model as the security model where no idealized assumption (e.g., access to a random oracle or common reference string, restricting adversary's access to group representations, etc.) is considered.

Setting	Constraint	Work	Adaptive	Hiding	Multi-Key	Comment
Lattice-Based	P/poly	[BV15]	✗	✗	✗	LWE⋆
		[BTVW17] [PS18]	✗	✓	✗	
	NC^1	[CC17] [CVW18]	✗	✓	✗	
	Inner-Product	[DKN+20]	✓	✓	✗	
Group-Based	NC^1	[AMN+18]	✗	✗	✗	DDHI‡
Other Tools	P/poly	[BZ14]	✗	✗	✓ (poly)	$i\mathcal{O}$
		[BLW17]	✗	✓	✓ (poly)	$i\mathcal{O}$
		[HKKW19]	✓	✗	✓ (poly)	$i\mathcal{O}$ + ROM§
		[DKN+20]	✓	✗	✓ $\mathcal{O}(1)$	$i\mathcal{O}$ + LWE⋆
	NC^1	[AMN+19]	✓	✗	✗	$i\mathcal{O}$ + SDA¶

⋆ LWE: Learning with Error Assumption
‡ DDHI: Decisional Diffie-Hellman Inversion Assumption
§ ROM: Random Oracle Model
¶ SDA: Subgroup Decision Assumption

Table 1.1 – State-of-the-art CPRFs for P/poly, NC^1 and inner-product predicates

Our Contributions

In this thesis, we discover connections between constrained PRFs and two other advanced cryptographic tools: (1) *homomorphic secret sharing* (HSS) protocols, and (2) *pseudorandom correlation functions* (PCFs). We propose a simple strategy for constructing CPRFs for rich families of constraints using HSS protocols, resulting in several instantiations of CPRFs from various assumptions. Furthermore, we show how CPRFs can be leveraged to construct PCFs for oblivious transfer correlations. We instantiate a public-key PCF, using our transformation, by applying a simple modification to the well-known PRF construction of Naor-Reingold [NR97].

Homomorphic secret sharing protocols and pseudorandom correlation functions were introduced within the context of *secure computation*. Secure computation is an active branch of cryptography that aims at providing efficient solutions for scenarios where n parties want to jointly evaluate a function f on their private inputs $x_1, \ldots, x_n$ without revealing any information about these inputs other than what might be implied from $f(x_1, \ldots, x_n)$. An extensive line of research in secure computation concerns reducing the cost of communication between parties, and in particular, achieving communication that is smaller than the size of the circuit. A generic approach for solving this problem is by using *Fully-Homomorphic Encryption* (FHE), proposed by [Gen09]. FHE is an encryption protocol that allows computation over encrypted data. In particular, to securely compute a function f, Alice and Bob, holding respective inputs x and y, can leverage FHE as follows: Alice first sends an FHE encryption of x to Bob. Next, Bob homomorphically

evaluates $f(\cdot, y)$ over the ciphertext received from Alice to get an encryption of $f(x, y)$. Bob sends back this ciphertext to Alice. Finally, Alice decrypts $f(x, y)$ and publishes the result. Homomorphic secret sharing and pseudorandom correlation functions can be considered as two alternative, more-tailored, approaches to this problem.

In the following, we provide a brief introduction of these tools, followed by our results regarding each of them.

1

Homomorphic Secret Sharing & Constrained PRFs

Homomorphic secret sharing protocols, introduced by [BGI16], allow distributed computations over *shares* of a secret. More precisely, a 2-party HSS protocol for a class of programs $\mathcal{P}$ works as follows: a user, holding a secret s, can "split" s and generate two shares $\mathsf{I}_0, \mathsf{I}_1$ that individually hide s.[1] Each share is then sent to a server. Given I_b and a program $P \in \mathcal{P}$, server b can run an evaluation algorithm over the given share and generate an output y_b, where $b \in \{0, 1\}$. Importantly, these two output shares should form additive shares of $P(s)$, i.e., $y_0 - y_1 = P(s)$. Homomorphic secret sharing gives rise to an alternative solution for secure computation with communication cost independent of the circuit size. Alice and Bob, with private inputs x and y, can use HSS as follows to compute $f(x, y)$: They first locally compute shares of their private inputs and send one part of these shares over a public channel to each other. Note that since shares of x and y are independent of f, the communication cost is unaffected by the circuit size. Next, They each locally compute shares of $f(x, y)$ by homomorphically evluating either $f(x, \cdot)$ (Bob) or $f(\cdot, y)$ (Alice) over the received shares. It is then enough for Alice and Bob to combine their shares and recover the value of $f(x, y)$. HSS protocols have found applications in secure computation with silent preprocessing [BCG+17, OSY21].[2]

Our Results. In this thesis, we show a new and surprising application of homomorphic secret sharing protocols towards constructing constrained PRFs. More precisely, we show how HSS schemes can be used to construct CPRFs for inner-product and NC^1 predicates. This transformation in particular leads to instantiations from (separate) different assumptions thanks to recent developments in HSS [RS21, OSY21, ADOS22]. Our results are obtained through the following steps:

1. **Introducing "Programmable" extensions of HSS**
   In a first step, we identify two natural extensions of homomorphic secret sharing that can be leveraged to build constrained PRFs. We term these extensions *homomorphic secret sharing with simulatable memory values* and *staged homomorphic secret sharing*. At a high level, both notions capture the ability to perform some limited form of *programming* of HSS shares, *i.e.*, to construct one of the two HSS shares of a secret x before knowing x.
   Importantly, we observe that most HSS constructions from the literature already satisfy the definition of these extensions. Thus, these extensions already exist from various assumptions.

[1] These shares are not required to be additive shares of the secret; instead, they represent some encoding of s.

[2] This model is explained in the paragraph discussing pseudorandom correlation functions

2. **Transformations from "Programmable" HSS to CPRFs**
   In this step, we discover two transformations that link our HSS extensions to constrained PRFs. Specifically, we show how an HSS scheme with simulatable memory values can be transformed into a CPRF that admits inner-product predicates. And, we show how a staged-HSS scheme can be used to construct a CPRF supporting the class of NC^1 predicates.

Plugging in existing constructions of our HSS extensions to these transformations leads to the following statement:

Theorem 1.1 (informal)**.** *Assuming any of the following assumptions:*

- *the Decisional Composite Residuosity (DCR) assumption,*
- *the hardness of the Joye-Libert encryption scheme,*
- *the Decisional Diffie-Hellman (DDH) and Decisional Cross Group Diffie-Hellman (DXDH) assumptions over class groups,*
- *the Hard Subgroup Membership assumption over class groups,*
- *the Learning with Errors (LWE) assumption with super-polynomial modulus-to-noise ratio,*

there exist (1-key, selectively secure) constraint-hiding CPRFs for inner product, and (1-key, selectively secure) CPRFs for NC^1*.*

Our results significantly expand the set of assumptions known to imply CPRFs for rich classes of constraints. In particular, our CPRF for NC^1 from DCR yields the first construction of a CPRF for a rich class of constraints from a well-established standard assumption beyond LWE-based constructions.

The technical overview of these results is provided in Section 3.1.

Pseudorandom Correlation Functions & Constrained PRFs

Pseudorandom correlation functions were introduced in the context of a popular paradigm in secure computation, called the *preprocessing model*. This approach builds on the impact of *correlated randomness* on the efficiency of protocols. More specifically, in the preprocessing model, a secure computation protocol is split into two phases: (1) a *preprocessing* phase that is run ahead of time and independently of the inputs and the circuit. In this phase, Alice and Bob engage in a process to generate long, *correlated* random strings. (2) An *online* phase, where Alice and Bob use the correlated randomness generated in the previous phase in order to run a fast protocol.

A correlated random pair consists of two values that independently look random and that are correlated to each other as specified by a joint distribution. For instance, a useful form of correlation, called *oblivious transfer* (OT) consists of pairs (y_0, y_1) of the form $y_0 = (r_0, r_1)$, where $r_0, r_1 \xleftarrow{\$} \{0,1\}^m$ are two random strings, and $y_1 = (b, r_b)$, where $b \xleftarrow{\$} \{0,1\}$ is a random bit. The seminal work of [GMW87], known as the GMW protocol, showed that if two parties have access to $\mathcal{O}(n)$ instances of random OT-correlated pairs, they can compute any circuit of size n while communicating as little as only four bits per AND gate.

1

Pseudorandom correlation generators (PCGs) [BCG+19] and pseudorandom correlation functions (PCFs) [BCG+20] were introduced as means of *silently* generating correlated pseudorandom strings in the preprocessing phase. More precisely, using a pseudorandom correlation generator, two parties receive two short correlated seeds, which they can expand locally, and *only once*, in order to generate long correlated strings that look like a random sample from the target correlation. Pseudorandom correlation functions can be viewed as generalizations of PCFs (similar to how PRFs generalize PRGs), where two parties, having two short correlated keys, called *evaluation keys*, can locally map multiple common inputs to pseudorandom correlated strings.

A PCF is said to be public-key if it allows non-interactive generation of the evaluation keys. This additional feature contributes to the usability of secure computation, especially over large-scale networks, where each pair of nodes might at some point want to run a fast secure computation; using a public-key PCF they don't require any prior setup for generating correlated pairs, and can therefore spontaneously engage in a secure computation.

Our Results. In this thesis we show how CPRFs can be used to construct PCFs for oblivious transfer correlations. In particular, we show how we can obtain efficient public-key PCFs for oblivious transfer correlations from a modified version of the Naor-Reingold PRF [NR97] that we call *Constrained Naor-Reingold*. Our results are obtained through the following steps:

1. **Naor-Reingold PRF ⇒ Pseudorandomly Constrained PRF**
   In the first step, we show that applying a slight modification to the Naor-Reingold PRF yields a constrained PRF for the class of inner-product membership (IPM) predicates. An IPM predicate C is associated with a vector $\mathbf{z}$ and a set S and satisfies $C(x) = 0$ iff $\langle \mathbf{x}, \mathbf{z} \rangle \in S$.
   We observe that the class of IPM predicates captures interesting predicates including puncturing, and, more importantly for our work, some *weak PRF* candidates from the literature. A weak PRF is an efficient function whose outputs look random on *random inputs* (contrary to PRFs that are required to look random on arbitrary inputs). This observation results in obtaining a construction of constrained PRFs that admit weak PRFs as constraints. We refer to these CPRFs as *pseudorandomly constrained PRFs*.

2. **Pseudorandomly Constrained PRFs ⇒ PCFs for OT Correlations**
   In this step, we show that PCFs for OT correlations can be constructed from pseudorandomly constrained PRFs through a simple transformation. We also prove that the resulting PCFs satisfy the notion of *precomputability* which captures the setting where one of the two parties can locally generate a PCF evaluation key and precompute its outputs even before knowing the identity of the other party.
   Combining this transformation with the previous step yields a PCF for OT correlations from the Naor-Reingold PRF.

3. **Public-Key PCF for OT from Constrained Naor-Reingold**
   Finally, we show that the construction of PCF for OT correlations, obtained in the previous step, can be modified to allow non-interactive generation of evaluation keys. This results in a very efficient public-key PCF construction from standard hardness assumptions.

The technical overview of these results is provided in Section 4.1.

Related Publications

The content of this thesis is based on the following two publications.

[CMPR23] Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Constrained pseudorandom functions from homomorphic secret sharing. *In EUROCRYPT 2023.*

[BCM+24] Dung Bui, Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Public-key silent ot and more from constrained Naor-Reingold. *To appear in EUROCRYPT 2024.*

Other Results

In addition to the contributions described earlier, two other results were obtained during the first year of my doctoral studies. These results concern the following topics. We briefly introduce the scope of each work and explain our contributions.

Non-Committing Encryption. Non-committing encryption (NCE), introduced by [CFGN96], is an encryption tool initially aimed at offering security against *adaptive* adversaries in multi-party computation (MPC) protocols. In the adaptive setting, an adversary can corrupt parties' internal states at any time during the execution of the protocol. This on-the-fly corruption allows the adversary to adapt its strategy, in particular, to the messages exchanged between parties. Simply put, an NCE scheme allows one to generate a "dummy" ciphertext that can be later opened to an arbitrary message. Therefore, using NCE in MPC protocols allows us to fool adaptive adversaries by revealing corrupted parties' internal states as arbitrary messages while maintaining their consistency with public transcript of the protocol.

> We present new constructions of NCE with constant ciphertext rate (separately) from the learning with errors (LWE), the decisional Diffie-Hellman (DDH), or the subgroup decision (SD) assumptions.

Our work follows the approach proposed by [BBD+20], where the authors introduce a bridge primitive, called *packed encryption with partial equivocality* (PEPE), that implies NCE with only a constant factor loss in the ciphertext rate. We improve the state-of-the-art NCE by proposing three new constructions of rate-1 PEPE: (1) from LWE with *ponlynomial modulus*, (2) from DDH, and (3) from SD assumption. Our SD-based construction requires a trusted setup in order to generate the composite order group (common reference string model). However, we observe that the trusted setup requirement appears to arise in any PEPE construction based on the hardness of factoring as long as no individidual party should learn the factorization.

[LPR22b] Benoît Libert, Alain Passelègue, and Mahshid Riahinia. New and improved constructions for partially equivocable public key encryption. *In SCN 2022.*

PointProofs. PointProofs is an efficient vector commitment (VC) scheme introduced by [GRWZ20]. Vector commitments allow a user to commit to a vector of messages, all at once, by generating a short commitment. Later, the commitment can be succinctly opened on arbitrary positions, without revealing any information about other unopened coordinates. Vector commitments allow reducing both storage (by storing a single commitment instead of a vector of commitments) and bandwidth (thanks to the ability of succinctly opening individual positions) in practical applications. PointProofs were introduced as vector commitments that additionally allow non-interactive *proof aggregations*. In other words, in PointProofs, one can aggregate proofs (of the same committed vector or across different commitments) into *a single* proof. This property makes PointProofs specially useful for distributed applications, such as blockchain propagation.

> We provide a new security analysis of PointProofs, which relies on the *Generalized Forking Lemma* [BCJ08] and *Local Forking Lemma* [BDL19]. Using these tools, we prove that PointProofs is binding in the random oracle model, under the n-Diffie-Hellman exponent (n-DHE) assumption.

The binding property of a VC requires that no efficient adversary can generate a commitment that can be opened to two different values at the same position. The original proof of [GRWZ20] shows that PointProofs are binding in the algebraic group model (AGM) as well as random oracle model under the weak n-bilinar Diffie-Hellman exponent (n-wBDHE) assumption. Algebraic group model is an idealized security model, where an adversary that outputs a group element z is required to also output a vector of exponents $(a_1, \ldots, a_n)$ such that $z = \prod_{i=1}^{n} L_i^{a_i}$, for known group elements $L_1, \ldots, L_n$. In our work, we get rid of AGM, and prove the binding property of PointProofs under the n-DHE assumption (which is implied by the n-wBDHE assumption), only in the random oracle model.

[LPR22a] Benoît Libert, Alain Passelègue, and Mahshid Riahinia. PointProofs, revisited. *In ASIACRYPT 2022.*

Chapter 2

Preliminaries

Contents

2.1 Basic Cryptographic Tools

2.1.1 Notation

In this thesis we use the following notations that are for the most part consistent with established terminology in the cryptography literature.

- **Security Parameter:** We use λ to denote the security parameter.

- **Sets:** For a natural integer $n \in \mathbb{N}$, the set $\{1, \ldots, n\}$ is denoted by $[n]$.

- **Vectors:** We mostly use bold lowercase letters (e.g., $\mathbf{r}$) to denote vectors. For a vector $\mathbf{r} = (r_1, \ldots, r_n)$, the vector $(g^{r_1} \ldots, g^{r_n})$ is sometimes denoted by $g^{\mathbf{r}}$.

- **Algorithms:** We write $\mathsf{poly}(\lambda)$ to denote an arbitrary polynomial function in λ. We denote by $\mathsf{negl}(\lambda)$ a negligible function in λ, which is a function that is asymptotically

smaller than the inverse of any polynomial function. PPT stands for probabilistic polynomial-time.

- **Sampling:** For a finite set S, we write $x \xleftarrow{\$} S$ to denote that x is sampled uniformly at random from S. For an algorithm $\mathcal{A}$, we denote by $y \leftarrow \mathcal{A}(x)$ the output y after running $\mathcal{A}$ on input x.

- **Indistinguishablity:** We say that two distributions $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{E} = \{\mathcal{E}_\lambda\}_{\lambda\in\mathbb{N}}$ defined over a set $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ are

  - **computationally indistinguishable** if for any $\lambda \in \mathbb{N}$ and any PPT algorithm $\mathcal{A}$, it holds that

    $$\left|\Pr\left[\mathcal{A}(1^\lambda, x) = 1 \middle| x \leftarrow \mathcal{D}_\lambda\right] - \Pr\left[\mathcal{A}(1^\lambda, x) = 1 \middle| x \leftarrow \mathcal{E}_\lambda\right]\right| \leq \mathsf{negl}(\lambda),$$

    in which case we write $\mathcal{D} \approx_c \mathcal{E}$.

  - **statistically indistinguishable** if for any $\lambda \in \mathbb{N}$ and any *unbounded* adversary it holds that

    $$\left|\Pr\left[\mathcal{A}(1^\lambda, x) = 1 \middle| x \leftarrow \mathcal{D}_\lambda\right] - \Pr\left[\mathcal{A}(1^\lambda, x) = 1 \middle| x \leftarrow \mathcal{E}_\lambda\right]\right| \leq \mathsf{negl}(\lambda),$$

    in which case we write $\mathcal{D} \approx_s \mathcal{E}$.
    We can equivalently define the statistical indistinguishability by stating that the statistical distance between $\mathcal{D}_\lambda$ and $\mathcal{E}_\lambda$ is a negligible function of λ:

    $$\Delta(\mathcal{D}_\lambda, \mathcal{E}_\lambda) = \frac{1}{2}\sum_{a\in\mathcal{X}_\lambda} |\Pr_{\mathcal{D}_\lambda}(a) - \Pr_{\mathcal{E}_\lambda}(a)| \leq \mathsf{negl}(\lambda)$$

- **Terminology:** In this thesis, we use the term *efficient* interchangeably with *polynomial-time*. For example, when we refer to "an efficient algorithm", we mean "an algorithm that terminates within polynomial time", and when we say "the two distributions cannot be distinguished efficiently", we imply that "the two distributions cannot be distinguished within polynomial time with more than negligible probability".

2.1.2 Idealized Security Models

Random Oracle Model

The *random oracle model* (ROM) is an abstract idealized model for analyzing security of cryptographic protocols. In this model, a protocol is proved to be secure under the assumption that all parties that are involved in the security game of the protocol, have oracle access to a global truly random function, called the *random oracle*. When these protocols are implemented in practice, the random oracle is replaced by a suitable hash function.

2.1.3 Hardness Assumptions

In this thesis we base the security of protocols on different cryptographic assumptions. This method is commonly used in cryptography to prove that the security of a given protocol is maintained as long as a certain mathematical problem (referred to as *hardness assumption*) cannot be solved in polynomial time. In this section, we go over the assumptions used in this thesis and briefly discuss their validity.

Decisional Composite Residuosity assumption

This assumption is defined over the multiplicative group $\mathbb{Z}^*_{N^2}$, where $N = pq$ for prime number p and q. The group $\mathbb{Z}^*_{N^2}$ can be written as a product of two subgroups, i.e., $\mathbb{Z}^*_{N^2} \cong \mathbb{H} \times \mathbb{NR}_N$, where $\mathbb{H} = \{(1+N)^i : i \in [N]\}$ is of order N, and $\mathbb{NR}_N = \{x^N : x \in \mathbb{Z}^*_{N^2}\}$ is the subgroup of N-th residues that has order $\phi(N)$. The decisional composite residuosity assumption states that a random element of $\mathbb{Z}^*_{N^2}$ cannot be distinguished from a random element of $\mathbb{NR}_N$ in polynomial time.

More precisely, let SampleModulus be a polynomial-time algorithm that on input the security parameter λ, outputs (N, p, q), where $N = pq$ for λ-bit primes p and q. The decision composite residuosity problem is as follows:

Definition 2.1 (Decisional Composite Residuosity (DCR) assumption, [Pai99])**.** Let λ be a security parameter. We say that the Decision Composite Residuosity (DCR) problem is hard relative to SampleModulus if

$$(N, x) \approx_c (N, x^N),$$

where $(N, p, q) \xleftarrow{\$} \mathsf{SampleModulus}(1^\lambda)$, $x \xleftarrow{\$} \mathbb{Z}^*_{N^2}$, and x^N is computed modulo N^2.

The DCR assumption is the assumption that there exists an algorithm SampleModulus relative to which the DCR problem is hard.

Decisional Diffie-Hellman Assumption and Variants

The Diffie-Hellman problem and its variants are built upon the discrete-logarithm problem. The discrete-logarithm problem is said to be hard over a cyclic group $\mathbb{G} = \langle g \rangle$ of order p, if given a random element $h \xleftarrow{\$} \mathbb{G}$, there exist no algorithm running in polynomial time that can output $x \in \mathbb{Z}_p$ such that $h = g^x$. The decisional Diffie-Hellman assumption states that given two random element g^x and g^y, an element of the form g^{xy} cannot be efficiently distinguished from a random element g^z, where $z \xleftarrow{\$} \mathbb{Z}_p$.

More precisely, let GenPar be a polynomial-time algorithm that on input the security parameter λ, outputs $(\mathbb{G}, g, p)$, where $\mathbb{G}$ is a cyclic group of prime order p generated by g.

Definition 2.2 (Decisional Diffie-Hellman (DDH) Assumption, [DH76])**.** Let λ be a security parameter. We say that the decisional Diffie-Hellman (DDH) problem is hard relative to GenPar if

$$(\mathbb{G}, p, g, g^a, g^b, g^{ab}) \approx_c (\mathbb{G}, p, g, g^a, g^b, g^c),$$

where $(\mathbb{G}, g, p) \xleftarrow{\$} \mathsf{GenPar}(1^\lambda)$ and $(a, b, c) \xleftarrow{\$} \mathbb{Z}_p$.

The DDH assumption is the assumption that there exists an algorithm GenPar relative to which the DDH problem is hard.

Random Self-Reducibility of DDH. An important property of the DDH problem, proved by [NR97] is its *random self-reducibility*. In general, a problem is said to be random self-reducible if solving any instane of the problem can be efficiently reduced to solving a random instance of the problem. The decisional Diffie-Hellman problem is random

self-reducible since given an instance $(\mathbb{G}, p, g, A = g^a, B = g^b, C = g^c)$, one can sample random elements $x, y, z \xleftarrow{\$} \mathbb{Z}_p$ and generate a random instance

$$(\mathbb{G}, p, g, A' = A^z g^x, B' = Bg^y, C' = C^z A^{zy} B^x g^{xy}).$$

Note that we have

$$A' = g^{az+x}, \quad B' = g^{b+y}, \quad C' = g^{cz+azy+bx+xy}.$$

Firstly, since x and y are random elements of $\mathbb{Z}_p$, $az+x$ and $b+y$ are also random. Secondly, when $c = ab$, we have that $C' = g^{(az+x)\cdot(b+y)}$. Otherwise, $C' = g^{(az+x)\cdot(b+y)+z(c-ab)}$, where the exponent is uniformly random since z is a random element of $\mathbb{Z}_p$. Therefore, the transformed instance is uniformly random and can be used to solve the original given instance.

2

Power-DDH Assumption

The security of the constructions presented in Chapter 4 are based on an assumption we term *Sparse Power-DDH*. This assumption can be viewed as a generalization of the Power-DDH assumption used in prior works such as [CNs07, AHI11]. In the following we first recall the power-DDH assumption, and then we introduce the sparse power-DDH assumption.

Definition 2.3 (Power-DDH Assumption, [CNs07, AHI11])**.** Let λ be a security parameter. We say that the power-DDH problem is hard relative to GenPar if for any polynomially-bounded $\ell \in \mathbb{N}$, it holds that

$$\left(\mathbb{G}, p, g, g^r, g^{r^2}, \ldots, g^{r^{\ell-1}}, g^{r^\ell}\right) \approx_c \left(\mathbb{G}, p, g, g^r, g^{r^2}, \ldots, g^{r^{\ell-1}}, g^t\right),$$

where $(\mathbb{G}, g, p) \xleftarrow{\$} \mathsf{GenPar}(1^\lambda)$, and $r, t \xleftarrow{\$} \mathbb{Z}_p^*$.

The power-DDH assumption is the assumption that there exists an algorithm GenPar relative to which the power-DDH problem is hard.

Definition 2.4 (Sparse Power-DDH Assumption)**.** Let λ be a security parameter. We say that the sparse power-DDH problem is hard relative to GenPar if for any polynomially-bounded $\ell \in \mathbb{N}$ and $S \subset [\ell]$, it holds that

$$\left(\mathbb{G}, p, g, (g^{r^s})_{s\in S}, (g^{r^s})_{s\in[\ell]\setminus S}\right) \approx_c \left(\mathbb{G}, p, g, (g^{r^s})_{s\in S}, (g^{t_s})_{s\in[\ell]\setminus S}\right),$$

where $(\mathbb{G}, g, p) \xleftarrow{\$} \mathsf{GenPar}(1^\lambda)$, and $r \xleftarrow{\$} \mathbb{Z}_p^*$, and $t_s \xleftarrow{\$} \mathbb{Z}_p^*$ for all $s \in [\ell] \setminus S$.

The sparse power-DDH assumption is the assumption that there exists an algorithm GenPar relative to which the power-DDH problem is hard.

The sparse power-DDH assumption is a static falsifiable assumption. It generalizes in a natural way the power-DDH assumption and is easily proven to hold in the generic group model since all exponents are distinct univariate monomials (e.g., using [BBG05, Corollary A.3] and observing that it is a special case of the uber-assumption family).

2.1.4 Basic Protocols

Public-Key Encryption

Public-key encryption is an important tool in cryptography that allows two parties to privately communicate to each other without having agreed on any secrets in advance. The idea of public-key encryption was proposed by Diffie and Hellman [DH76].

Definition 2.5 (Public-Key Encryption (PKE))**.** Let $\lambda \in \mathbb{N}$ be a security parameter. A public-key encryption scheme with message space $\mathcal{M}$ consists of the following three polynomial-time algorithms:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$: A probabilistic algorithm that on input the security parameter λ outputs a pair of public-key and secret-key $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$: A probabilistic algorithm that on input a public key pk and a message m outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to m$: A deterministic algorithm that on input a secret key sk and a ciphertext ct outputs a message m.

We require a public-key encryption scheme to satisfy the following two properties:

- **Correctness.** For any security parameter $\lambda \in \mathbb{N}$, and any message $m \in \mathcal{M}$, we have:
$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m : \quad (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)\right] \geq 1 - \mathsf{negl}(\lambda).$$
- **Security against Chosen-Plaintext Attacks (CPA-Security).** For any PPT adversary $\mathcal{A}$, we have:
$$\left|\Pr\left[b' = b : \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}) \\ b \xleftarrow{\$} \{0,1\} \\ \mathsf{ct}_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \\ b' \leftarrow \mathcal{A}(\mathsf{ct}_b) \end{array}\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)\ .$$

In this thesis, we use the term *semantically-secure* interchangeably with *CPA-secure*. Semantic security is another notion of security for public-key encryption schemes which roughly requires that any information that can be efficiently computed from a ciphertext, can be efficiently computed *only given the length of the underlying message*. Semantic security is the most natural security definition for PKE schemes and can be viewed as a computational-complexity analogue of Shannon's perfect secrecy. Goldwasser and Micali [GM84] showed that CPA security is equivalent to semantic security.

Commitment Schemes

Commitment schemes are tools that, in essence, mathematically simulate a sealed envelope; using a commitment scheme, one can generate a commitment to a message that reveals no information about the message (similarly to how a sealed envelope hides its content), and later on, this commitment can be opened *only* to the committed message (similarly to how the content of a sealed envelope cannot be changed).

Definition 2.6 (Commitment Scheme)**.** Let λ be a security parameter. A commitment scheme with message space $\mathcal{M}$ consists of the following four polynomial-time algorithms:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: A probabilistic algorithm that on input the security parameter λ outputs public parameters pp.
- $\mathsf{Com}(\mathsf{pp}, m) \to (\mathsf{com}, \mathsf{aux})$: A probabilistic algorithm that on input the public parameters pp and a message m outputs a commitment com together with auxiliary information aux.
- $\mathsf{Open}(\mathsf{pp}, \mathsf{com}, \mathsf{aux}) \to \pi$: A probabilistic algorithm that on input public parameters pp, a commitment com, and auxiliary information aux, outputs a proof π.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{com}, \pi, m) \to b \in \{0,1\}$: A deterministic algorithm that on input public parameters pp, a commitment com, a proof π, and a message m outputs a bit $b \in \{0,1\}$.

We require a commitment scheme to satisfy the following properties:

- **Correctness of Openings.** For all $\lambda \in \mathbb{N}$, and any message $m \in \mathcal{M}$, we have:

$$\Pr\left[\mathsf{Verify}(\mathsf{pp}, \mathsf{com}, \pi, m) = 1 : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{pp}, m) \\ \pi \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{com}, \mathsf{aux}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

- **Hiding.** For any PPT adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, we have:

$$\left|\Pr\left[b' = b : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pp}) \\ b \xleftarrow{\$} \{0,1\} \\ (\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{pp}, m_b) \\ b' \leftarrow \mathcal{A}(\mathsf{com}_b) \end{array}\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)\ .$$

- **Binding.** For any PPT adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, the probability that $\mathcal{A}$, on input $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, outputs a tuple $(\mathsf{com}, m, m', \pi, \pi')$, such that

$$\mathsf{Verify}(\mathsf{pp}, \mathsf{com}, \pi, m) = \mathsf{Verify}(\mathsf{pp}, \mathsf{com}, \pi', m') = 1 \quad \wedge \quad m \neq m',$$

  is negligible in λ.

2.1.5 Constructions

In this section we recall the description of the cryptographic schemes from the literature that are used in this thesis.

KDM-Secure Paillier Cryptosystem

Key-dependent message (KDM) security, is a notion of security for public-key encryption schemes that captures scenarios where an adversary has access to encryptions of messages that are functions of the secret key. In the following we recall the KDM-secure Paillier cryptosystem presented by Brakerski and Goldwasser in [BG10] which is introduced as a KDM-secure version of the Paillier encryption scheme [Pai99]. The security of the scheme follows from the DCR assumption. The scheme is parameterized by $\ell \in \mathbb{N}$ (polynomial in λ), and consists of three algorithms $(\mathsf{BG.KeyGen}, \mathsf{BG.Enc}, \mathsf{BG.Dec})$ defined as in Construction 2.1. We skip the proof of the KDM security of Construction 2.1 and refer to [BG10].

Construction 2.1: KDM-Secure Paillier Cryptosystem, [BG10]

- $\mathsf{BG.KeyGen}(1^\lambda)$:
  1. Sample $(N, p, q) \leftarrow \mathsf{SampleModulus}(1^\lambda)$.
  2. Sample $\mathbf{g} = (g_0, \ldots, g_{\ell-1}) \xleftarrow{\$} \mathbb{NR}_N^\ell$.
  3. Sample $\mathbf{d} = (d^{(0)}, \ldots, d^{(\ell-1)}) \xleftarrow{\$} \{0,1\}^\ell$.
  4. Compute $\hat{g} = \prod_{i=0}^{\ell-1} g_i^{d^{(i)}} \pmod{N^2}$.
  5. Output $\mathsf{pk} = (N, \mathbf{g}, \hat{g})$ and $\mathsf{sk} = \mathbf{d}$.
- $\mathsf{BG.Enc}(\mathsf{pk}, x)$:
  1. Sample $r \xleftarrow{\$} \mathbb{Z}_N$.
  2. Compute and output $\mathsf{ct} = (g_0^r, \ldots, g_{\ell-1}^r, \hat{g}^r \cdot (1+N)^x)$.
- $\mathsf{BG.Dec}(\mathsf{sk}, \mathsf{ct})$
  1. Parse $\mathsf{ct} = (c_0, \ldots, c_{\ell-1}, \hat{c})$.
  2. Compute $\bar{c} = (\prod_{i=0}^{\ell-1} c_i^{-d^{(i)}}) \cdot \hat{c} \pmod{N^2}$.
  3. Compute and output $x = (\bar{c} - 1)/N$.

Paillier-ElGamal Cryptosystem

The Paillier-ElGamal encryption scheme [CS02, DGS03, BCP03] is defined by three algorithms $(\mathsf{PaillierEG.KeyGen}, \mathsf{PaillierEG.Enc}, \mathsf{PaillierEG.Dec})$, and in essence, describes the ElGamal encryption scheme over the multiplicative group $\mathbb{Z}_{N^2}^\star$, where $N = pq$. This scheme is outlined in Construction 2.2. Assuming the DCR assumption (Definition 2.1), the Paillier-ElGamal cryptosystem is semantically secure. Observe that Paillier-ElGamal is a special case of the KDM-secure Paillier cryptosystem of [BG10], where $\ell = 1$.

Construction 2.2: Paillier-ElGamal Cryptosystem, [CS02, DGS03, BCP03]

- $\mathsf{PaillierEG.KeyGen}(1^\lambda)$:
  1. Sample $g' \xleftarrow{\$} \mathbb{Z}_{N^2}$, and let $g = (g')^{2N} \pmod{N^2}$.
  2. Sample $d \xleftarrow{\$} \mathbb{Z}_{N^2}$.
  3. Output $\mathsf{pk} = (g, g^d \bmod N^2)$, $\mathsf{sk} = d$.
- $\mathsf{PaillierEG.Enc}(\mathsf{pk}, x)$:
  1. Sample $r \xleftarrow{\$} \mathbb{Z}_N$, and output $\mathsf{ct} = (g^r, \mathsf{pk}^r \cdot (1+N)^x)$.
- $\mathsf{PaillierEG.Dec}(\mathsf{sk}, \mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1))$:
  1. Set $\mathsf{ct}' \leftarrow \mathsf{ct}_1 \cdot (\mathsf{ct}_0)^{-d} \pmod{N^2}$.
  2. Output $x = \frac{\mathsf{ct}'-1}{N}$.

Pedersen Commitment Scheme

Let p and q be prime numbers such that $q|p-1$. The Pedersen commitment scheme [Ped92] is defined over the subgroup of quadratic residues of p, $\mathsf{QR}_p = \{x^2 \mid x \in \mathbb{Z}_p\}$. This scheme consists of four algorithms $(\mathsf{Pedersen.Setup}, \mathsf{Pedersen.Com}, \mathsf{Pedersen.Open}, \mathsf{Pedersen.Verify})$ described in Construction 2.3. Pedersen commitments are perfectly hiding, and computationally binding assuming the hardness of the discrete logarithm problem over $\mathbb{Z}_p$.

Construction 2.3: Pedersen Commitment Scheme, [Ped92]

- $\mathsf{Pedersen.Setup}(1^\lambda)$:
  1. Sample a generator g of QR_p.
  2. Sample $a \xleftarrow{\$} \mathbb{Z}_q$, and set $h := g^a \pmod{p}$.
  3. Output $\mathsf{pp} = (g, h, p, q)$
- $\mathsf{Pedersen.Com}(\mathsf{pp}, m \in \mathbb{Z}_q)$:
  1. Sample $r \xleftarrow{\$} \mathbb{Z}_q$.
  2. Output $\mathsf{com} = g^r \cdot h^m \pmod{p}$, and $\mathsf{aux} = (m, r)$.
- $\mathsf{Pedersen.Open}(\mathsf{pp}, \mathsf{com}, \mathsf{aux})$:
  1. Parse $\mathsf{aux} = (m, r)$ and output $\pi = (m, r)$.
- $\mathsf{Pedersen.Verify}(\mathsf{pp}, \mathsf{com}, \pi = (m, r))$:
  1. Output 1 if $\mathsf{com} = g^r \cdot h^m$.

2.2 Constrained Pseudorandom Functions

2.2.1 Pseudorandom Functions

Definition 2.7 ((Weak) Pseudorandom Function (wPRF, PRF), [GGM84a, NR95])**.** Let $\lambda \in \mathbb{N}$ be a security parameter. A (weak) pseudorandom function with domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$, key space $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda\in\mathbb{N}}$, and range $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$, consists of the following two polynomial-time algorithms:

- $\mathsf{KeyGen}(1^\lambda) \to \mathsf{msk}$: A probabilistic algorithm that on input the security parameter λ, outputs a master secret key $\mathsf{msk} \in \mathcal{K}$.
- $\mathsf{Eval}(\mathsf{msk}, x) \to y$: A deterministic algorithm that on input the master secret key msk, and an input value $x \in \mathcal{X}$, outputs a value $y \in \mathcal{Y}$.

We say that the pair $(\mathsf{KeyGen}, \mathsf{Eval})$ is a

- **pseudorandom function (PRF)** if for any PPT adversary $\mathcal{A}$, it holds that

$$\left|\Pr\left[\mathcal{A}^{\mathsf{Eval}(\mathsf{msk},\cdot)}(1^\lambda) = 1 \middle| \mathsf{msk} \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)\right] - \Pr\left[\mathcal{A}^{RF(\cdot)}(1^\lambda) = 1 \middle| RF \xleftarrow{\$} \mathcal{F}\right]\right| = \mathsf{negl}(\lambda),$$

  where $\mathcal{F}$ is the set of all functions with domain $\mathcal{X}$ and range $\mathcal{Y}$.

- **weak pseudorandom function (wPRF)** if for any PPT adversary $\mathcal{A}$ and any polynomially bounded number $Q \in \mathbb{N}$, it holds that

$$\left\{\left((x_i, \mathsf{Eval}(\mathsf{msk}, x_i))_{i\in[Q]}\right) : \begin{array}{c}\mathsf{msk} \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda) \\ \forall i \in [Q] : x_i \xleftarrow{\$} \mathcal{X}\end{array}\right\} \approx_c \left\{\left((x_i, y_i)_{i\in[Q]}\right) : \begin{array}{c}\forall i \in [Q] : \\ x_i \xleftarrow{\$} \mathcal{X},\ y_i \xleftarrow{\$} \mathcal{Y}\end{array}\right\}.$$

Definition 2.7 captures the PRF security notion that is referred to as the *Real-or-Random* security. In Definition 2.8, we recall the Find-and-Guess security of a pseudorandom function which is equivalent to the Real-or-Random security up to a multiplicative gap of $\mathcal{O}(Q)$ between the advantage functions, where Q is the number of evaluation queries.

Definition 2.8 (Find-then-Guess Security)**.** Let λ be a security parameter. A function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is called a secure pseudorandom function if it is efficiently computable and the advantage of any PPT adversary $\mathcal{A}$ in the following game is negligible:

- **Setup.** The challenger chooses a random key $k \xleftarrow{\$} \mathcal{K}$ and a random bit $b \xleftarrow{\$} \{0, 1\}$, and initializes a set $S = \varnothing$.
- **Pre-Challenge Evaluation Queries.** $\mathcal{A}$ adaptively sends arbitrary inputs $x \in \mathcal{X}$ to the challenger. The challenger computes and returns $F_k(x)$ to $\mathcal{A}$. It also updates $S \leftarrow S \cup \{x\}$.

- **Challenge Phase.** $\mathcal{A}$ sends an input $x^* \in \mathcal{X}$ as its challenge query to the challenger with the restriction that $x^* \notin S$. If $b = 0$, then the challenger computes $y^* \leftarrow F_k(x^*)$. If $b = 1$, then the challenger samples a random element $y^* \xleftarrow{\$} \mathcal{Y}$. It then returns y^* to $\mathcal{A}$.

- **Post-Challenge Evaluation Queries.** $\mathcal{A}$ continues sending arbitrary inputs $x \in \mathcal{X}$ to the challenger with the restriction that $x \neq x^*$, and receives $F_k(x)$.

- **Guess.** $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$.

$\mathcal{A}$ wins if $b' = b$. We define the advantage of $\mathcal{A}$ in winning the game as $|2 \cdot \Pr[\mathcal{A} \text{ wins}] - 1|$, where the probability is over the internal coins of $\mathcal{A}$ and the challenger.

Remark 2.1 (Pseudorandom Predicate)**.** *We use the term* pseudorandom predicate *to refer to a pseudorandom function whose output range is limited to binary values, i.e.,* $\mathcal{Y} = \{0, 1\}$.

2

2.2.2 Constrained Pseudorandom Functions

Definition 2.9 (Constrained Pseudorandom Functions)**.** Let λ be a security parameter. A Constrained Pseudorandom Function (CPRF) with domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$, key space $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda\in\mathbb{N}}$, and range $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$, that supports a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$, where each $C_\lambda \in \mathcal{C}_\lambda$ has domain $\mathcal{X}_\lambda$ and range $\{0, 1\}$, consists of the following four polynomial-time algorithms:[1]

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pp}, \mathsf{msk})$: The master key generation algorithm is a probabilistic algorithm that on input the security parameter λ, outputs a public parameter pp and a master secret key $\mathsf{msk} \in \mathcal{K}$.

- $\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x) \to y$: The evaluation algorithm is a deterministic algorithm that on input the public parameter pp, the master secret key msk, and an input $x \in \mathcal{X}$, outputs a value $y \in \mathcal{Y}$.

- $\mathsf{Constrain}(\mathsf{msk}, C) \to \mathsf{ck}_C$: The constrained key generation algorithm is a probabilistic algorithm that on input the master secret key msk, and a circuit $C \in \mathcal{C}$, outputs a constrained key ck_C.

- $\mathsf{CEval}(\mathsf{pp}, \mathsf{ck}_C, x) \to y$: The constrained evaluation algorithm is a deterministic algorithm that on input the public parameter pp, a constrained key ck_C, and an input $x \in \mathcal{X}$, outputs a value $y \in \mathcal{Y}$.

Correctness. For any security parameter λ, any constrain $C \in \mathcal{C}$, and any input $x \in \mathcal{X}$ such that $C(x) = 0$, we have:

$$\Pr\left[\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x) \neq \mathsf{CEval}(\mathsf{pp}, \mathsf{ck}_C, x) : \begin{array}{l}(\mathsf{msk}, \mathsf{pp}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{ck}_C \leftarrow \mathsf{Constrain}(\mathsf{msk}, C)\end{array}\right] \leq \mathsf{negl}(\lambda).$$

1-Key Selective Security. We say that a CPRF is 1-key selectively secure if the advantage of any PPT adversary $\mathcal{A}$ in the following game is negligible:

[1] In this thesis, we drop the subscript λ when it is clear from context.

- **Setup:** The challenger runs $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, initializes a set $S_{\mathsf{eval}} = \varnothing$, and chooses a random bit $b \xleftarrow{\$} \{0,1\}$. It then sends pp to $\mathcal{A}$.

- **Selective Choice of Constraint:** The adversary chooses a (single) circuit $C \in \mathcal{C}$ and sends it to the challenger.

- **Constrained Key Generation:** The challenger computes $\mathsf{ck}_C \leftarrow \mathsf{Constrain}(\mathsf{msk}, C)$ and returns the constrained key ck_C to $\mathcal{A}$.

- **Pre-Challenge Evaluation Queries:** $\mathcal{A}$ can adaptively send arbitrary input values $x \in \mathcal{X}$ to the challenger. The challenger computes $y \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x)$ and returns y to $\mathcal{A}$. It also updates $S_{\mathsf{eval}} \leftarrow S_{\mathsf{eval}} \cup \{x\}$.

- **Challenge Phase:** $\mathcal{A}$ sends an input $x^* \in \mathcal{X}$ as its challenge query to the challenger with the restriction that $x^* \notin S_{\mathsf{eval}}$ and $C(x^*) \neq 0$. If it holds that $b = 0$, then the challenger computes $y^* \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x^*)$. Otherwise, if $b = 1$, the challenger samples a random value $y^* \xleftarrow{\$} \mathcal{Y}$. Finally, the challenger returns y^* to $\mathcal{A}$.

- **Post-Challenge Evaluation Queries:** $\mathcal{A}$ continues the queries as before, with the restriction that it cannot query x^* as an evaluation query.

- **Guess:** $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

1-Key Selective Constraint-Hiding. We say that a CPRF is selectively 1-key constraint-hiding if the advantage of any PPT adversary $\mathcal{A}$ in the following game is negligible:

- **Setup:** The challenger runs $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, and chooses a random bit $b \xleftarrow{\$} \{0,1\}$. It then sends pp to $\mathcal{A}$.

- **Selective Choice of Constraint:** The adversary chooses a (single) pair of circuits $(C_0, C_1) \in \mathcal{C}$ and sends the pair to the challenger.

- **Constrained Key Generation:** The challenger computes $\mathsf{ck}_b \leftarrow \mathsf{Constrain}(\mathsf{msk}, C_b)$, and returns ck_b to $\mathcal{A}$.

- **Evaluation Queries:** $\mathcal{A}$ can query the evaluation algorithm on arbitrary inputs $x \in \mathcal{X}$, with the restriction that $C_0(x) = C_1(x)$. On such inputs, the challenger computes and returns $y \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x)$ to $\mathcal{A}$.

- **Guess:** $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

In both of the games described above, $\mathcal{A}$ wins if $b' = b$. We also define the advantage of $\mathcal{A}$ in winning a game as $|2 \cdot \Pr[\mathcal{A} \text{ wins}] - 1|$, where the probability is over the internal coins of $\mathcal{A}$ and the challenger.

No-Evaluation Security. 1-key selective no-evaluation security (resp. 1-key selective no-evaluation constraint-hiding) is defined similarly with the extra restriction that the adversary cannot issue any pre-challenge or post-challenge query (resp. any evaluation query).

2.3 Homomorphic Secret Sharing

We start by recalling the standard definition of homomorphic secret sharing, as well as of Restricted Multiplication Straight-line (RMS) programs which is the common model of computation in the context of HSS.

Definition 2.10 (Homomorphic Secret Sharing)**.** Denote by λ a security parameter. A Homomorphic Secret Sharing (HSS) scheme for a class of programs $\mathcal{P}$ which is defined over a ring $\mathcal{R}$ and has input space $\mathcal{I} \subseteq \mathcal{R}$ consists of three PPT algorithms $(\mathsf{Setup}, \mathsf{Input}, \mathsf{Eval})$ such that:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1))$: On input the security parameter λ, the setup algorithm outputs a public key pk and a pair of evaluation keys $(\mathsf{ek}_0, \mathsf{ek}_1)$.
- $\mathsf{Input}(\mathsf{pk}, x) \to (\mathsf{I}_0, \mathsf{I}_1)$: On input the public key pk and an input $x \in \mathcal{I}$, the input algorithm outputs a pair of input information $(\mathsf{I}_0, \mathsf{I}_1)$.
- $\mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}_\sigma = (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), P) \to y_\sigma$: On input a party index $\sigma \in \{0, 1\}$, an evaluation key ek_σ, a vector of ρ input values $(\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)})$, and a program $P \in \mathcal{P}$, the evaluation algorithm outputs the party σ's corresponding share of the output y_σ.

We require an HSS scheme to satisfy the following two properties:

- **Correctness.** For any security parameter $\lambda \in \mathbb{N}$, and any program $P \in \mathcal{P}$ with input space $\mathcal{I} \subseteq \mathcal{R}$, we have:

$$\Pr\left[y_0 - y_1 = P(x^{(1)}, \ldots, x^{(\rho)})\right] \geq 1 - \mathsf{negl}(\lambda) \ ,$$

  where the probability is taken over $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{I}_0^{(i)}, \mathsf{I}_1^{(i)}) \leftarrow \mathsf{Input}(\mathsf{pk}, x^{(i)})$ for $i \in [\rho]$, and $y_\sigma \leftarrow \mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), P)$, for $\sigma \in \{0, 1\}$.

- **Security.** For any PPT adversaries $\mathcal{A}, \mathcal{A}'$, and any bit $\sigma \in \{0, 1\}$ the following value should be negligible in λ:

$$\left| \Pr\left[b' = b : \begin{array}{l} (x_0, x_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda) \\ b \xleftarrow{\$} \{0, 1\} \\ (\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{Input}(x_b) \\ b' \leftarrow \mathcal{A}'(\mathsf{state}, \mathsf{pk}, \mathsf{ek}_\sigma, \mathsf{I}_\sigma) \end{array} \right] - \frac{1}{2} \right|$$

2.3.1 RMS Programs

We now recall the definition of Restricted Multiplication Straight-line (RMS) programs. RMS programs form a class of programs which encompasses branching programs of polynomial-size and therefore NC^1 circuits. In an RMS program, the multiplication is restricted to happen between an input value and an intermediate value of the computation (so-called "memory" value).

Definition 2.11 (RMS Programs)**.** An RMS program with magnitude bound B is defined as a sequence of the instructions as follows:

- $\mathsf{ConvertInput}(\mathsf{I}^x) \to \mathsf{M}^x$: Loads an input x into memory.
- $\mathsf{Add}(\mathsf{M}^x, \mathsf{M}^y) \to \mathsf{M}^{x+y}$: Adds two memory values.
- $\mathsf{Mul}(\mathsf{I}^x, \mathsf{M}^y) \to \mathsf{M}^{x \cdot y}$: Multiplies an input value and a memory value to produce a memory value of their product.
- $\mathsf{Output}(\mathsf{M}^x, n) \to x \bmod n$: Outputs a memory value *w.r.t.* a modulus $n < B$.

2.4 Pseudorandom Correlation Functions

2.4.1 Reverse-Sampleable Correlations

Definition 2.12 (Reverse-Sampleable Correlation)**.** Let $1 \leq \ell_0(\lambda), \ell_1(\lambda) \leq \mathsf{poly}(\lambda)$ be output-length functions. Let $\mathcal{Y}$ be a probabilistic algorithm that, on input 1^λ, returns a pair of outputs $(y_0, y_1) \in \{0,1\}^{\ell_0(\lambda)} \times \{0,1\}^{\ell_1(\lambda)}$, defining a correlation on the outputs.

We say that $\mathcal{Y}$ defines a *reverse-sampleable* correlation if there exists a probabilistic polynomial time algorithm $\mathsf{RSample}$ which takes as input 1^λ, $\sigma \in \{0,1\}$, and $y_\sigma \in \{0,1\}^{\ell_\sigma(\lambda)}$, and outputs $y_{1-\lambda}^{\ell_{1-\sigma}(\lambda)}$, such that for all $\sigma \in \{0,1\}$ the following distributions are statistically close:

$$\{(y_0, y_1) \colon (y_0, y_1) \xleftarrow{\$} \mathcal{Y}(1^\lambda)\}, \text{ and}$$
$$\{(y_0, y_1) \colon (y_0', y_1') \xleftarrow{\$} \mathcal{Y}(1^\lambda), y_\sigma \leftarrow y_\sigma', y_{1-\sigma} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y_\sigma)\} .$$

Definition 2.13 (OT Correlation)**.** A *(1-out-of-2, bit) OT correlation* can be defined as being sampled as a pair $((r_0, r_1), (b, r_b))$, where $r_0, r_1, b \xleftarrow{\$} \{0,1\}$.

Remark 2.2 (OT Correlation is Reverse-Sampleable)**.** A (1-out-of-2, bit) OT correlation is reverse-sampleable. *Indeed, observe that the reverse-sampling can be performed as follows.* $\mathsf{RSample}(1^\lambda, \sigma, y_\sigma)$: *If* $\sigma = 0$*, parse* y_σ *as* $y_\sigma = (r_0, r_1)$*, sample* $b \xleftarrow{\$} \{0,1\}$*, and output* (b, r_b)*; otherwise (*i.e. *if* $\sigma = 1$*) parse* y_σ *as* $y_\sigma = (b, r)$*, sample* $r' \xleftarrow{\$} \{0,1\}$*, and output* $((1-b) \cdot r + b \cdot r', b \cdot r + (1-b) \cdot r')$.

2.4.2 Pseudorandom Correlation Functions

We recall the definition of pseudorandom correlation functions for reverse-sampleable correlations. We consider two different security levels of PCFs: *weak* PCFs (wPCF) and *strong* PCFs (sPCF). Analogously to weak PRFs, weak PCFs guarantee security given access only to evaluations on uniformly random and independent inputs, while strong PCFs guarantee security for arbitrary chosen inputs. Note that contrary to PRFs, weak PCFs are the notion that is commonly considered in the literature.

2.4.2.1 Weak Pseudorandom Correlation Functions (wPCF).

We start by defining the notion of a *weak* pseudorandom correlation functions.

Definition 2.14 ((Weak) Pseudorandom Correlation Function (wPCF), [BCG+20, Definition 4.3])**.** Let $\mathcal{Y}$ be a reverse-sampleable correlation with output length functions $\ell_0(\lambda), \ell_1(\lambda)$ and let $\lambda \leq n(\lambda) \leq \mathsf{poly}(\lambda)$ be an input length function. Let $\mathsf{RSample}$ be a reverse sampling algorithm of $\mathcal{Y}$. Let $(\mathsf{wPCF.Gen}, \mathsf{wPCF.Eval})$ be a pair of algorithms with the following syntax:

- $\mathsf{wPCF.Gen}(1^\lambda)$ is a probabilistic polynomial time algorithm that on input 1^λ, outputs a pair of keys (k_0, k_1); we assume that λ can be inferred from the keys.

- $\mathsf{wPCF.Eval}(\sigma, k_\sigma, x)$ is a deterministic polynomial time algorithm that on input $\sigma \in \{0,1\}$, key k_σ and input value $x \in \{0,1\}^{n(\lambda)}$, outputs a value $y_\sigma \in \{0,1\}^{\ell_\sigma(\lambda)}$.

We say that $(\mathsf{wPCF.Gen}, \mathsf{wPCF.Eval})$ is a pseudorandom correlation function (PCF) for $\mathcal{Y}$, if the following conditions hold:

- **(Weakly) pseudorandom $\mathcal{Y}$-correlated outputs.** For every non-uniform adversary $\mathcal{A}$ of size $B(\lambda)$, it holds that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Exp}^{\mathsf{w\text{-}pr}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{w\text{-}pr}}_{\mathcal{A},N,1}(\lambda) = 1]| \leq \epsilon(\lambda)$$

  where $\mathsf{Exp}^{\mathsf{w\text{-}pr}}_{\mathcal{A},N,b}$ ($b \in \{0,1\}$) is defined as in Figure 2.1. In particular, the adversary is given access to $N(\lambda)$ samples.

Experiment (Weakly) Pseudorandom Correlated Outputs

$\underline{\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda)}$:

$(k_0, k_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$

For $i = 1, \ldots, N(\lambda)$:

$x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

$(y_0^{(i)}, y_1^{(i)}) \hookleftarrow \mathcal{Y}(1^\lambda)$

$b \leftarrow \mathcal{A}(1^\lambda, (x^{(i)}, y_0^{(i)}, y_1^{(i)})_{i \in [N(\lambda)]})$

Output b

$\underline{\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda)}$:

$(k_0, k_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$

For $i = 1, \ldots, N(\lambda)$:

$x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

$y_0^{(i)} \xleftarrow{\$} \mathsf{wPCF.Eval}(0, k_0, x^{(i)})$

$y_1^{(i)} \xleftarrow{\$} \mathsf{wPCF.Eval}(1, k_1, x^{(i)})$

$b \leftarrow \mathcal{A}(1^\lambda, (x^{(i)}, y_0^{(i)}, y_1^{(i)})_{i \in [N(\lambda)]})$

Output b

Figure 2.1 – (Weakly) Pseudorandom $\mathcal{Y}$-correlated outputs of a (w)PCF

- **Security.** For every $\sigma \in \{0,1\}$ and every non-uniform adversary $\mathcal{A}$ of size $B(\lambda)$, it holds that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Exp}^{\mathsf{w\text{-}sec}}_{\mathcal{A},N,\sigma,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{w\text{-}sec}}_{\mathcal{A},N,\sigma,1}(\lambda) = 1]| \leq \epsilon(\lambda)$$

  where $\mathsf{Exp}^{\mathsf{w\text{-}sec}}_{\mathcal{A},N,\sigma,b}$ ($b \in \{0,1\}$) is defined as in Figure 2.2. In particular, the adversary is given access to $N(\lambda)$ samples (or simply N if there is no ambiguity).

Experiment (Weak) PCF Security

$\mathsf{Exp}^{\text{w-sec}}_{\mathcal{A},N,\sigma,0}(\lambda)$:

$(k_0, k_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$

For $i = 1, \ldots, N(\lambda)$:

$x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

$y^{(i)}_\sigma \leftarrow \mathsf{wPCF.Eval}(\sigma, k_\sigma, x^{(i)})$

$y^{(i)}_{1-\sigma} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y^{(i)}_\sigma)$

$b \leftarrow \mathcal{A}(1^\lambda, \sigma, k_\sigma, (x^{(i)}, y^{(i)}_{1-\sigma})_{i\in[N(\lambda)]})$

Output b

$\mathsf{Exp}^{\text{w-sec}}_{\mathcal{A},N,\sigma,1}(\lambda)$:

$(k_0, k_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$

For $i = 1, \ldots, N(\lambda)$:

$x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

$y^{(i)}_{1-\sigma} \leftarrow \mathsf{wPCF.Eval}(1-\sigma, k_{1-\sigma}, x^{(i)})$

$b \leftarrow \mathcal{A}(1^\lambda, \sigma, k_\sigma, (x^{(i)}, y^{(i)}_{1-\sigma})_{i\in[N(\lambda)]})$

Output b

Figure 2.2 – Security of a wPCF

2.4.2.2 Strong Pseudorandom Correlation Functions.

A strong PCF is syntactically defined in the same way as a weak PCF, but it instead satisfies stronger notions of *pseudorandom $\mathcal{Y}$-correlated outputs* and *PCF security*. For simplicity, we only provide these modified properties.

We say that $(\mathsf{sPCF.Gen}, \mathsf{sPCF.Eval})$ is an (N, B, ϵ)-secure strong pseudorandom correlation function (sPCF) for $\mathcal{Y}$, if the following conditions hold:

- **Strongly pseudorandom $\mathcal{Y}$-correlated outputs.** For every non-uniform adversary $\mathcal{A}$ of size $B(\lambda)$ asking at most $N(\lambda)$ queries to the oracle $\mathcal{O}_b(\cdot)$ (as defined in Figure 2.3), it holds that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Exp}^{\text{s-pr}}_{\mathcal{A},0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{s-pr}}_{\mathcal{A},1}(\lambda) = 1]| \leq \epsilon(\lambda)$$

  where $\mathsf{Exp}^{\text{s-pr}}_{\mathcal{A},b}$ ($b \in \{0,1\}$) is defined as in Figure 2.3.

- **Strong Security.** For every $\sigma \in \{0,1\}$ and every non-uniform adversary $\mathcal{A}$ of size $B(\lambda)$ asking at most $N(\lambda)$ queries to the oracle $\mathcal{O}_b(\cdot)$ (as defined in Figure 2.4), it holds that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Exp}^{\text{s-sec}}_{\mathcal{A},0,\sigma}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{s-sec}}_{\mathcal{A},1,\sigma}(\lambda) = 1]| \leq \epsilon(\lambda)$$

  where $\mathsf{Exp}^{\text{s-sec}}_{\mathcal{A},\sigma}$ is defined as in Figure 2.4. We recall that $\mathsf{RSample}$ is the algorithm for reverse sampling $\mathcal{Y}$ as in Definition 2.12.

Experiment Strongly Pseudorandom Correlated Outputs

$\underline{\mathsf{Exp}^{\text{s-pr}}_{\mathcal{A},b}(\lambda)}$:

$(k_0, k_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$

$\mathcal{Q} \leftarrow \varnothing$

$b \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_b(\cdot)}(1^\lambda)$

Output b

$\underline{\mathcal{O}_0(x)}$:

If $(x, y_0, y_1) \in \mathcal{Q}$**:**

Output (y_0, y_1)

Else:

$(y_0, y_1) \xleftarrow{\$} \mathcal{Y}(1^\lambda)$

$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(x, y_0, y_1)\}$

Output (y_0, y_1)

$\underline{\mathcal{O}_1(x)}$:

For $\sigma \in \{0, 1\}$**:**

$y_\sigma \leftarrow \mathsf{sPCF.Eval}(1^\lambda, \sigma, k_\sigma, x)$

Output (y_0, y_1)

Figure 2.3 – Strongly Pseudorandom $\mathcal{Y}$-correlated outputs of a sPCF

Experiment Strong PCF Security

$\underline{\mathsf{Exp}^{\text{s-sec}}_{\mathcal{A},b,\sigma}(\lambda)}$:

$(k_0, k_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$

$\mathcal{Q} \leftarrow \varnothing$

$b \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_b(\cdot)}(1^\lambda, \sigma, k_\sigma)$

Output b

$\underline{\mathcal{O}_0(x)}$:

$y_{1-\sigma} \leftarrow \mathsf{sPCF.Eval}(1 - \sigma, k_{1-\sigma}, x)$

Output $y_{1-\sigma}$

$\underline{\mathcal{O}_1(x)}$:

$y_\sigma \leftarrow \mathsf{sPCF.Eval}(\sigma, k_\sigma, x)$

$y_{1-\sigma} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y_\sigma)$

Return $y_{1-\sigma}$

Figure 2.4 – Security of a strong PCF.

Next, we define the precomputability property of a PCF. At a high level, a PCF is *precomputable* if the first party's key can be generated first, and the second key can be derived from the first.

Definition 2.15 (Precomputable Pseudorandom Correlation Function, [CMPR23])**.** Let $\mathcal{Y}$ be a reverse-sampleable correlation with output lengths $\ell_0(\lambda), \ell_1(\lambda)$ and let $\lambda \leq n(\lambda) \leq \mathsf{poly}(\lambda)$ be its input length. We say that a pseudorandom correlation function $(\mathsf{PCF.Gen}, \mathsf{PCF.Eval})$ is *precomputable* if the description of $\mathsf{PCF.Gen}$ contains the descriptions of two algorithms $(\mathsf{PCF.Gen}_0, \mathsf{PCF.Gen}_1)$ such that

- $\mathsf{PCF.Gen}_0(1^\lambda)$: On input the security parameter λ, returns a key k_0 and auxiliary output aux.
- $\mathsf{PCF.Gen}_1(1^\lambda, \mathsf{aux})$: On input the security parameter λ and an auxiliary input aux, outputs a key k_1.

We also require the following property to hold:
Precomputability. For any security parameter $\lambda \in \mathbb{N}$, the two following distributions are computationally indistinguishable:

$$\left\{(k_0, k_1) \colon \ (k_0, k_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda) \right\} \overset{c}{\approx} \left\{(k_0, k_1) \colon \begin{array}{l} (k_0, \mathsf{aux}) \leftarrow \mathsf{PCF.Gen}_0(1^\lambda) \\ k_1 \leftarrow \mathsf{PCF.Gen}_1(1^\lambda, \mathsf{aux}) \end{array} \right\}.$$

Chapter 3

Constrained Pseudorandom Functions from Homomorphic Secret Sharing

Contents

The content of this section is based on the results presented in [CMPR23].

3.1 Chapter Overview

In this chapter, we draw connections between constrained pseudorandom functions and (2-party) homomorphic secret sharing (HSS) protocols.[1] More precisely, we show how we can construct CPRFs for the class of inner-product predicates as well as for the class of NC^1 predicates from HSS. This transformation in particular leads to multiple instantiations from different assumptions thanks to the recent developments in HSS [RS21, OSY21, ADOS22].

Briefly recalling our discussion in the introduction, the results of this chapter are obtained via two steps: (1) we first extend HSS properties, and introduce two new extensions of

[1] For the sake of simplicity, in this thesis, we denote a two-party HSS protocol as HSS protocol.

homomorphic secret sharing that can be leveraged to build constrained PRFs, that we call *homomorphic secret sharing with simulatable memory shares* and *staged homomorphic secret sharing*. We observe that these extensions already exist from multiple assumption, as most HSS constructions from the literature already satisfy their definitions. Next, (2) we transform our HSS extensions to CPRFs. In particular, we show how our HSS extensions can be used to construct constrained PRFs for inner-product and NC^1 predicates. Combining these transformations with already-existing constructions of our HSS extensions leads to constructions of CPRFs for inner-product and NC^1 predicates from various assumptions.

Revisiting Applications of HSS to Secure Computation. Homomorphic secret sharing schemes have found notable applications for low-communication secure computation [BGI16], and secure computation with *silent* preprocessing [BCG+17, OSY21]. We revisit these applications in light of the properties of (already-existing) HSS extensions. This is done by my coauthors in our paper [CMPR23]. In short, (1) we show that staged HSS can be used to build a silent preprocessing 2-party protocol where one of the parties can entirely run the heavy offline computation before she even knows the identity of the other party. We call this model secure computation with *precomputable* silent preprocessing. This model is especially well-suited to a client-server setting, where a weak client wants to start the bulk of the computation a long time in advance, whereas the powerful server can run the heavy computation after its interaction with the client. Moreover, (2) using staged HSS, we obtain the first non-FHE-based constructions of one-sided statistically secure protocols with sublinear communication. More precisely, we obtain secure computation for any $\log\log$-depth circuits on inputs x, y with optimal communication, where x remains statistically hidden, provided that $|x| < |y|/\mathsf{poly}(\lambda)$, via a black-box use of staged HSS. We also achieve secure computation of any layered arithmetic circuit C of size s over a sufficiently large ring $\mathbb{Z}_n$, with sublinear communication $O(s/\log\log s)$ and one-sided statistical security, assuming the Paillier encryption scheme is circular-secure. The latter construction is non-black box and exploits the specific structure of a concrete Paillier-based staged HSS scheme from [OSY21]. We refer to our paper [CMPR23] for a comprehensive description of these results.

In the following, we first state the core idea underlying our transformations from HSS protocols to CPRFs. Although this idea turns out to be partly incorrect, it serves as the basis of our subsequent findings. We then show how we can fix this idea by imposing more requirements (tailored to our transitions to CPRFs) on the underlying HSS schemes, thereby introducing new HSS extensions.

3.1.1 General Strategy

Let us first explain a (partly wrong but insightful) strategy for constructing CPRFs from HSS. Let F denote a pseudorandom function with keyspace $\mathcal{K}$ and domain $\mathcal{X}$, and let $\mathcal{C} : \mathcal{X} \mapsto \{0,1\}$ be a class of constraints. Consider an HSS scheme $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{Eval})$ for a class of programs $\mathcal{P}$ that contains all functions of the form $P_x : (k, C) \mapsto C(x) \cdot F_k(x)$, for all $x \in \mathcal{X}$. Now consider the following construction:

- $\mathsf{KeyGen}(1^\lambda, C)$: Sample a PRF key $K \xleftarrow{\$} \mathcal{K}$. Run $(\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{I}_0^k, \mathsf{I}_1^k) \leftarrow \mathsf{Input}(\mathsf{pk}, k)$, and $(\mathsf{I}_0^C, \mathsf{I}_1^C) \leftarrow \mathsf{Input}(\mathsf{pk}, C)$. Finally, set $\mathsf{pp} := \mathsf{pk}$, and $\mathsf{msk} := (\mathsf{ek}_0, \mathsf{ek}_1, \mathsf{I}_0^k, \mathsf{I}_1^k, \mathsf{I}_0^C, \mathsf{I}_1^C)$.
- $\mathsf{Constrain}(\mathsf{msk}, C)$: Parse $\mathsf{msk} = (\mathsf{ek}_0, \mathsf{ek}_1, \mathsf{I}_0^k, \mathsf{I}_1^k, \mathsf{I}_0^C, \mathsf{I}_1^C)$ and output the constrained key $\mathsf{ck}_C := (\mathsf{ek}_1, \mathsf{I}_1^k, \mathsf{I}_1^C)$.
- $\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x)$: Run $y_0 \leftarrow \mathsf{Eval}(0, \mathsf{ek}_0, \mathsf{I}_0^k, \mathsf{I}_0^C, P_x)$ and output y_0.
- $\mathsf{CEval}(\mathsf{pp}, \mathsf{ck}_C, x)$: Run $y_1 \leftarrow \mathsf{Eval}(1, \mathsf{ek}_1, \mathsf{I}_1^k, \mathsf{I}_1^C, P_x)$ and output y_1.

By the correctness of HSS, for any input x, we have $y_0 - y_1 = C(x) \cdot F_k(x)$. Therefore, if $C(x) = 0$, $y_0 = y_1$ meaning that the CEval algorithm outputs the same value as the evaluation algorithm using msk. Also, if $C(x) = 1$, $y_0 = y_1 + F_k(x)$, and therefore y_0 looks random even given y_1 and ck_C. This is because y_0 is masked with $F_k(x)$ which is a pseudorandom value as k remains hidden, given I_1^k, due to the security of HSS.
The problem with the above construction is that the master secret key depends on the constraint C while it should be independent of it.[1] A way around this issue would be to use an HSS scheme with *programmable input shares*, *i.e.*, a scheme where I_0^C can be generated before knowing C, and the second share I_1^C can be constructed afterwards from I_0^C and C, when the constraint is chosen. Towards achieving some level of programmability of input shares, in this work, we identify weak properties which suffice to instantiate the above template, and show that they are satisfied by most known HSS constructions.

3.1.2 CPRF from HSS with Simulatable Memory Shares

As a start, we propose a first simple solution to circumvent the lack of programmability. This first property already allows to handle simple forms of constraints such as inner-product, and follows from the common design of HSS constructions. We start by providing a high-level description of HSS schemes, which applies to essentially all known HSS constructions (beside FHE-based constructions).
Known constructions of HSS schemes rely on additively homomorphic public-key encryption schemes with some form of linear decryption. The public key of the HSS scheme is the public key pk of the underlying encryption scheme, and evaluation keys $\mathsf{ek}_0, \mathsf{ek}_1$ are additive shares of the underlying secret key s. An HSS scheme uses two types of data: (1) **Input shares** $(\mathsf{I}_0, \mathsf{I}_1)$ which are generated by running $\mathsf{Input}(\mathsf{pk}, x)$ on some input x and consist in an encryption of $(x, x \cdot s)$, and (2) **Memory shares** (M_0, M_1) which are typically additive shares of $(x, x \cdot s)$ over $\mathbb{Z}$. Two types of operations are handled: **Additions of memory shares** (simply add the shares as $(x, x \cdot s) + (y, y \cdot s) = (x + y, (x + y) \cdot s)$), and a *restricted form* of **Multiplication**. Specifically, multiplication can only be performed between an *input share* of some value x and a *memory share* of some value y, and returns a *memory share* of their product $x \cdot y$. Typically, multiplication uses the memory share $(y, y \cdot s)$ to "linearly multiply-and-decrypt" the encryption of $(x, x \cdot s)$, getting some encoding of $(xy, xy \cdot s)$. Then, the encoding is converted into a valid memory share using a specific procedure, which depends on the concrete scheme and is often a form of *distributed discrete logarithm*. We provide more details about multiplication later. Note that one can transform any input share into a memory share of the same value by

[1] If the key could depend on C, one could just generate two independent PRF keys k_0, k_1 and define the evaluation as $F_{k_{C(x)}}(x)$. Revealing k_0 then allows to compute the evaluation on any x such that $C(x) = 0$ and reveals nothing about the key k_1 used when $C(x) = 1$.

multiplying it with a memory share of 1. At the end of a computation, each party recovers a memory value consisting in an additive share of $(z, z \cdot s)$, and therefore a share of the result z by dropping the second part. One can evaluate any polynomial-size program following the above restrictions, which precisely corresponds to *restricted multiplication straight-line* (RMS) programs, and encompasses branching programs, NC^1, and more.

HSS with simulatable memory shares. Our starting point is the result of two observations. First, we observe that any HSS following the above structure does in fact allow for a limited form of programming regarding memory values. Indeed, while input shares include a homomorphic encryption of the input (which cannot be generated without knowing the input), *memory shares* are simply additive shares. Thus, we can always *simulate* a memory share of one party before knowing the value to share, by generating a first random share u. The other share is later set to $x - u$ when the actual value x to share is known.
Second, we remark that two parties sharing input shares of some values $(x_1, \ldots, x_n)$ as well as memory shares of a value z can compute memory shares of $z \cdot P(x_1, \ldots, x_n)$ for any RMS program P. The trick is to evaluate all the operations of P "with z in front", i.e. by maintaining as an invariant that any memory share for any value y that should be used in the computation is replaced by a memory share for the value $z \cdot y$. This invariant being preserved by the two RMS operations (addition and multiplication), it is sufficient to guarantee that every memory value satisfies it when created. This is simply done by transforming an input x into a memory value by multiplying it with the memory share of z in order to get a memory share for $z \cdot x$ rather than for x.

CPRF for Linear Constraints. Combining these two observations leads to constructions of constrained PRFs for linear constraints (and in particular for inner-product). Looking back to the construction aforementioned, we just would like to be able to generate I_0^C, the share of C used for evaluation with the master secret key, without knowing the constraint C in advance. We do it by replacing I_0^C by a simulated *memory share* M_0 of the (yet unknown) constraint C. The constrained key for C is then computed from M_0 and C to generate the appropriate memory share M_1 (i.e. setting M_1 such that $\mathsf{M}_0 + \mathsf{M}_1 = C$). While this prevents the need for knowing the constraint ahead of time, this comes with a price: we now get a memory share of C rather than an input share, which reduces the set of functions one can evaluate. Still, thanks to our second observation, having a memory share of C and an input share of k allows to compute shares of $C \cdot P(k)$ for any RMS program P. Moreover, given memory shares of multiple C_i's, one can then compute any linear combination of shares $C_i \cdot P(k)$, by summing the latter additive shares. Notably, this allows computing shares of $\langle C, x \rangle \cdot F_k(x)$ as long as the function $k \mapsto F_k(x)$ is an RMS program (assuming F is in NC^1 is sufficient for that purpose).
We just constructed constrained pseudorandom functions for inner-product from any assumption that suffices to construct an HSS scheme for RMS programs satisfying the above conditions. For example, using the recent HSS scheme of [OSY21] yields a CPRF for inner products over $\mathbb{Z}$ (or any integer ring) under the DCR assumption (which also implies PRFs in NC^1). The construction extends immediately to any constant-degree polynomial constraints (by memory-sharing all the coefficients of C). It achieves 1-key selective security, as well as *constraint privacy*. To the best of our knowledge, this is the first construction of (1-key, selective, private) CPRF for inner products that does not rely on LWE.
Security analysis proceeds through a sequence of hybrid games. Recall that the adversary is given a constrained key ck_C of its choice, and access to an evaluation or-

acle $\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, \cdot)$. We first modify the evaluation oracle to return $C(x) \cdot F_K(x) + \mathsf{CEval}(\mathsf{pp}, k_C, x)$ on query x. By correctness of the HSS, the adversary's view remains identical to its view in the previous game though the game no longer relies in msk (and in particular now only relies on the evaluation key ek_1 from ck_C). This let us replace the input share I_1 of k in ck_C by an input share of a dummy value, thanks to HSS security. Then, the adversary does no longer have any information about k except in the evaluations, and we can use PRF security to replace evaluations of $F_K(\cdot)$ by truly random values, therefore proving pseudorandomness. Constraint privacy is proven in a similar fashion.

3.1.3 Handling more Constraints via Staged HSS

While the above already offers enough flexibility to evaluate linear functions (and extensions thereof, such as low-degree polynomials), we still cannot handle general computations like NC^1 circuits. To overcome this limitation, we show by a deeper analysis of known HSS schemes that most of them also achieve some specific, limited form of programmability, which turns out to be sufficient to construct CPRFs for all RMS programs (hence in particular for NC^1).

Concretely, for a vector $\mathbf{u} = (u_1, \ldots, u_\ell)$, our core observation is that it is possible to share $\mathbf{u}$ between parties P_0 and P_1 with two alternate sharing algorithms $(\overline{\mathsf{Input}}_0, \overline{\mathsf{Input}}_1)$ such that: (1) P_0's share of $\mathbf{u}$, obtained from $\overline{\mathsf{Input}}_0$, is *independent* of $\mathbf{u}$ (and can be generated without $\mathbf{u}$), (2) P_0 and P_1 can use specific $\overline{\mathsf{Eval}}_0, \overline{\mathsf{Eval}}_1$ evaluation algorithms to produce memory shares of $P(\mathbf{u})$ for any RMS program P, *provided that* P_1 *knows* $\mathbf{u}$ *in the clear*. We call staged-HSS an HSS scheme satisfying the latter properties, as it intuitively allows to split share generation and evaluation in 2 stages: a first *input-independent* stage, corresponding to P_0's view, and a second *input-dependent* stage corresponding to P_1's view.

At first sight, staged-HSS might not seem particularly useful: if P_1 knows $\mathbf{u}$ in the clear, then P_1 can already compute $P(\mathbf{u})$ for any RMS program P. The key observation is that P_0 and P_1 get *memory shares* of $P(\mathbf{u})$, and not just $P(\mathbf{u})$. This memory share can then be combined with the prior observations to let P_0, P_1 compute additive shares of $P(\mathbf{u}) \cdot Q(\mathbf{v})$, for any other RMS program P, Q, given *input shares* of $\mathbf{v}$. Setting $\mathbf{u}$ to be the description of the constraint C, P to be a universal circuit (with input x hardwired) which on input C returns $C(x)$, $\mathbf{v}$ to be a PRF key k, and Q to be the RMS program (with x hardwired) which on input k returns $F_k(x)$, parties P_0 and P_1 can then compute shares of $C(x) \cdot F_k(x)$, with shares of P_0 being independent of C. We can then instantiate our simple aforementioned strategy for constructing CPRFs while circumventing the need for C during KeyGen. As a result, we obtain (1-key selective) CPRFs for RMS programs (and therefore for NC^1) from any staged-HSS, i.e. from a wide variety of assumptions (including DCR [OSY21, RS21], class groups assumptions, or variants of QR [ADOS22, CLT22], and more.). The security analysis is similar to our construction for inner-product, though this new construction is no longer constraint-hiding, since the CEval algorithm now relies on knowing C (i.e. $\mathbf{u}$ above) in clear.

It remains to explain why known HSS schemes are also staged-HSS schemes. To illustrate this, we use the simple ElGamal-based HSS scheme from [BGI16].[1] We assume basic knowledge of ElGamal encryption in what follows. This scheme follows the general

[1] This scheme does not yield CPRFs as it does not achieve statistical correctness, but staged-HSS is easily illustrated with it.

structure detailed above by instantiating the additively homomorphic encryption scheme with ElGamal encryption. That is, an *input share* for x is an ElGamal encryption of the pair $(x, x \cdot s)$[1], i.e. a tuple $(c_0, c'_0, c_1, c'_1) = (g^{r_0}, h^{r_0} \cdot g^x, g^{r_1}, h^{r_1} \cdot g^{x \cdot s})$ with $s \in \mathbb{Z}_p$ being the secret key, $h = g^s$ being the public key, and $r_0, r_1 \xleftarrow{\$} \mathbb{Z}_p$ encryption randomness.[2] Multiplication between an input share (c_0, c'_0, c_1, c'_1) of x and a memory share $(\alpha_\sigma, \beta_\sigma)$ of y (which is just an additive share of $(y, y \cdot s)$ over $\mathbb{Z}_p$ owned by party P_σ) is done as follows. First, party P_σ computes $g_\sigma \leftarrow (c'_0)^{\alpha_\sigma} / c_0^{\beta_\sigma}$. Observe that $g_0 \cdot g_1 = (c'_0)^{\alpha_0 + \alpha_1} / c_0^{\beta_0 + \beta_1} = (g^{sr} \cdot g^x)^y / (g^r)^{sy} = g^{xy}$. Hence, parties get *multiplicative shares* g_0, g_1 of g^{xy}. Doing the same with c_1, c'_1 allows to get multiplicative shares of $g^{xy \cdot s}$. Then, an operation termed *distributed discrete logarithm* allows to transform these multiplicative shares of $(g^{xy}, g^{xy \cdot s})$ into additive shares of $(xy, xy \cdot s)$, i.e. memory shares for the value xy, as desired. Despite being at the core of HSS constructions, the details of the distributed discrete logarithm procedure do not matter here. The only important observation is that the $c_i = g^{r_i}$ components of input shares are independent of the input x; only the c'_i components actually depend on x. Furthermore, in the multiplication above, the only place where c'_i is involved is in the computation of $g_\sigma \leftarrow (c'_i)^{\alpha_\sigma} / c_i^{\beta_\sigma}$. Now, assume that one of the parties, say, P_1, already knows y in the clear: in this case, one can simply define $\alpha_1 \leftarrow y$ and $\alpha_0 \leftarrow 0$, which form valid additive shares of y. But now, P_0 does no longer need to know c'_i components either, since we now have $g_0 = 1/(c_i)^{\beta_0}$.

3

3.2 Homomorphic Secret Sharing and Extensions

The core notion underlying our constructions is homomorphic secret sharing (HSS), introduced by Boyle et al. in [BGI16]. In this section, we propose several extensions of HSS, and in particular define some special properties that play an important role in our constructions towards constrained PRFs. We further remark that these extensions are easily instantiated using the DCR-based HSS construction from [OSY21]. We first recall this instantiation.

Instantiation: HSS from DCR

Here, we recall the HSS construction of [OSY21] based on KDM-secure Paillier encryption (Section 2.1.5). The input space of the scheme is $\mathbb{Z}_N$ for a Blum integer $N = pq$.
First, we recall the following lemma due to [OSY21], where the authors introduce a distributed discrete logarithm algorithm for a subset of $\mathbb{Z}^*_{N^2}$, where $N = pq$ for λ-bit primes p and q.

Lemma 3.1. *There exists an algorithm* $\mathsf{DDLog}_N(g)$ *for which the following holds: Let* $g_0, g_1 \in \mathbb{Z}^*_{N^2}$, *such that* $g_0 = g_1(1+N)^x \pmod{N^2}$. *If* $z_0 = \mathsf{DDLog}_N(g_0)$ *and* $z_1 = \mathsf{DDLog}_N(g_1)$, *then* $z_0 - z_1 = x \pmod{N}$.

More precisely, $\mathsf{DDLog}_N(g)$ works as follows:

- $\mathsf{DDLog}_N(g)$
  - Write $g = h + h'N$, where $h, h' < N$, using the division algorithm.
  - Output $z = h'h^{-1} \bmod N$.

[1] An input share in fact consists of encryptions of x and $x \cdot s_i$'s for each bit s_i of s.
[2] The secret key s is encrypted bit-by-bit in the actual construction.

Instantiation 3.1: HSS from Paillier, [OSY21]

Requirements and notation:

- Let $(\mathsf{BG.KeyGen}, \mathsf{BG.Enc}, \mathsf{BG.Dec})$ be the KDM-secure Paillier encryption, as in Description 2.1.5, defined over $\mathbb{Z}^*_{N^2}$ for a Blum integer $N = pq$.
- Let $2^{-\kappa}$ be the correctness error of the scheme.
- Let $\mathcal{P}$ be the set of programs supported by the scheme, and $B_{\mathsf{msg}} = N/2^{\kappa}$ be the magnitude bound of programs in $\mathcal{P}$.

Algorithms:

▶ $\mathsf{Setup}(1^\lambda)$:

- Run $(\mathsf{BG.pk}, \mathsf{BG.sk}) \leftarrow \mathsf{BG.KeyGen}(1^\lambda)$, and parse $\mathsf{BG.pk} = (N, \mathbf{g}, \hat{g})$, and $\mathsf{BG.sk} = \mathbf{d} = (d^{(0)}, \ldots, d^{(\ell-1)})$.
- Sample $\langle 1 \rangle_0 \xleftarrow{\$} [2^\kappa]$, and set $\langle 1 \rangle_1 := \langle 1 \rangle_0 - 1 \bmod N$.
- For $i \in [\ell]$, sample $\langle d^{(i)} \rangle_0 \xleftarrow{\$} [2^\kappa]$, and set $\langle d^{(i)} \rangle_1 := \langle d^{(i)} \rangle_0 - d^{(i)} \bmod N$.
- For $i \in [\ell]$, compute $D^{(i)} \leftarrow \mathsf{BG.Enc}(\mathsf{BG.pk}, d^{(i)})$.
- Sample a PRF key k_{prf} for a PRF F that outputs values in $\mathbb{Z}_N$.
- Output $\mathsf{pk} = \left(\mathsf{BG.pk}, (D^{(i)})_{i\in[\ell]}\right)$, and $\mathsf{ek}_\sigma = \left(k_{\mathsf{prf}}, \langle 1 \rangle_\sigma, (\langle d^{(i)} \rangle_\sigma)_{i\in[\ell]}\right)$ for $\sigma \in \{0, 1\}$.

▶ $\mathsf{Input}(\mathsf{pk}, x)$:

- Parse $\mathsf{pk} = \left(\mathsf{BG.pk}, (D^{(i)} = (\mathbf{c}^{(i)}, \hat{c}^{(i)}))_{i\in[\ell]}\right)$, and $\mathsf{BG.pk} = (\mathbf{g}, \hat{g})$.
- Compute $X \leftarrow \mathsf{BG.Enc}(\mathsf{BG.pk}, x)$.
- For $i \in [\ell]$, compute $X^{(i)} \leftarrow (\mathbf{g}^{r'_i} \cdot (\mathbf{c}^{(i)})^x, \hat{g}^{r'_i} \cdot (\hat{c}^{(i)})^x)$, where $r'_i \xleftarrow{\$} \mathbb{Z}_N$.
- Set $\mathsf{I} = (X, X^{(0)}, \ldots, X^{(\ell-1)})$, and output $(\mathsf{I}_0 = \mathsf{I}, \mathsf{I}_1 = \mathsf{I})$.

▶ $\mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, (\mathsf{I}^{(0)}, \ldots, \mathsf{I}^{(n)}), P)$:

- $\mathsf{ConvertInput}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}_x = (X, X^{(0)}, \ldots, X^{(\ell-1)}))$
  - Set $\mathsf{M}^1_\sigma = (\langle 1 \rangle_\sigma, \langle d^{(0)} \rangle_\sigma, \ldots, \langle d^{(\ell-1)} \rangle_\sigma)$ for $\sigma \in \{0, 1\}$.
  - Compute $\mathsf{M}^x_\sigma \leftarrow \mathsf{Mult}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}^x, \mathsf{M}^1_\sigma)$.
- $\mathsf{Add}(\sigma, \mathsf{ek}_\sigma, \mathsf{M}^x_\sigma, \mathsf{M}^y_\sigma)$
  - Parse $\mathsf{M}^x_\sigma = (\langle x \rangle_\sigma, (\langle x d^{(i)} \rangle_\sigma)_{i\in[\ell]})$, and $\mathsf{M}^y_\sigma = (\langle y \rangle_\sigma, (\langle y d^{(i)} \rangle_\sigma)_{i\in[\ell]})$.
  - Compute $\langle z \rangle_\sigma = \langle x \rangle_\sigma + \langle y \rangle_\sigma$, and $\langle z d^{(i)} \rangle_\sigma = \langle x d^{(i)} \rangle_\sigma + \langle y d^{(i)} \rangle_\sigma$ for $i \in [\ell]$.
  - Output $\mathsf{M}^z_\sigma = (\langle z \rangle_\sigma, \langle z d^{(0)} \rangle_\sigma, \ldots, \langle z d^{(\ell-1)} \rangle_\sigma)$.

- $\mathsf{Mult}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}^x, \mathsf{M}^y_\sigma)$
  - Parse $\mathsf{I}^x = (X, (X^{(i)})_{i\in[\ell]})$ and $\mathsf{M}^y_\sigma = (\langle y\rangle_\sigma, (\langle yd^{(i)}\rangle_\sigma)_{i\in[\ell]})$.
  - Parse $X = (c_0, \ldots, c_{\ell-1}, \hat{c})$, and $X^{(i)} = (c_0^{(i)}, \ldots, c_{\ell-1}^{(i)}, \hat{c}^{(i)})$ for $i \in [\ell]$.
  - Compute $\langle z\rangle_\sigma = \mathsf{DDLog}_N\left(\mathsf{ct}'_\sigma\right) \pmod N + F_{k_{\mathsf{prf}}}(\mathsf{id})$, where
$$\mathsf{ct}'_\sigma = (\hat{c})^{\langle y\rangle_\sigma} \cdot \left(\prod_{i=0}^{\ell-1} c_i^{-\langle yd^{(i)}\rangle_\sigma}\right) \pmod{N^2}.$$
  - For $j \in [\ell]$, compute $\langle zd^{(j)}\rangle_\sigma = \mathsf{DDLog}_N\left(\mathsf{ct}'_{\sigma,j}\right) \pmod N + F_{k_{\mathsf{prf}}}(\mathsf{id})$, where
$$\mathsf{ct}'_{\sigma,j} = (\hat{c}^{(j)})^{\langle y\rangle_\sigma} \cdot \left(\prod_{i=0}^{\ell-1} (c_i^{(j)})^{-\langle yd^{(i)}\rangle_\sigma}\right) \pmod{N^2}.$$
  - Output $\mathsf{M}^z_\sigma = (\langle z\rangle_\sigma, \langle zd^{(0)}\rangle_\sigma, \ldots, \langle zd^{(\ell-1)}\rangle_\sigma)$.

▶ $\mathsf{Output}(\sigma, \mathsf{ek}_\sigma, \mathsf{M}^z_\sigma, n_{\mathsf{out}})$:

- Parse $\mathsf{M}^z_\sigma = (\langle z\rangle_\sigma, (\langle zd^{(i}\rangle_\sigma)_{i\in[\ell]})$, and output $\langle z\rangle_\sigma \pmod{n_{\mathsf{out}}}$.

3

3.2.1 HSS following the RMS Template

Similarly to [BCG+17], we first propose a more specific definition for HSS with additional algorithms that are relevant in the context of RMS programs.

Definition 3.1 (HSS Following the RMS Template)**.** A homomorphic secret sharing scheme $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{MemGen}, \mathsf{Eval})$ following the RMS template is an HSS scheme as defined in Definition 2.10 with an additional algorithm MemGen that generates memory values as follows:

- $\mathsf{MemGen}(\sigma, \mathsf{ek}_\sigma, x) \to \mathsf{M}_\sigma$: On input a party index $\sigma \in \{0, 1\}$, an evaluation key ek_σ, and an input $x \in \mathcal{I}$, the memory generator algorithm outputs a memory value M_σ.

Moreover, the Eval algorithm proceeds with sub-routines following the RMS operations ConvertInput, Add, Mul, Output as follows:

- $\mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, (\mathsf{I}^{(1)}_\sigma, \ldots, \mathsf{I}^{(\rho)}_\sigma), P) \to y_\sigma$: On input a party index $\sigma \in \{0, 1\}$, an evaluation key ek_σ, a vector of ρ input values $(\mathsf{I}^{(1)}_\sigma, \ldots, \mathsf{I}^{(\rho)}_\sigma)$, and an RMS program P, this algorithm follows the instructions of P and processes them as follows:
  - $\mathsf{ConvertInput}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}^x_\sigma) \to \mathsf{M}^x_\sigma$: This algorithm simply uses the MemGen and Mult algorithms as follows:
    - Run $\mathsf{MemGen}(\sigma, \mathsf{ek}_\sigma, 1) \to \mathsf{M}^1_\sigma$.
    - Run $\mathsf{Mult}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}^x_\sigma, \mathsf{M}^1_\sigma) \to \mathsf{M}^x_\sigma$.
  - $\mathsf{Add}(\sigma, \mathsf{ek}_\sigma, \mathsf{M}^x, \mathsf{M}^y) \to \mathsf{M}^{x+y}$: This algorithm directly adds the given memory values of x and y. Namely, $\mathsf{M}^{x+y}_\sigma = \mathsf{M}^x_\sigma + \mathsf{M}^y_\sigma$.

- $\mathsf{Mul}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}^x, \mathsf{M}^y) \to \mathsf{M}^{x \cdot y}$: It multiplies an input value I_x and a memory value M_y and outputs a memory value of $x \cdot y$. The template does not impose any non-black box requirement on this algorithm.
- $\mathsf{Output}(\sigma, \mathsf{M}^x, n) \to x \bmod n$: It uses M^x to output $x_\sigma \bmod n$.

Correctness and security properties are defined as in Definition 2.10, and we further require the following property:
Additively Homomorphic Memory. The memory values generated in HSS should be additively homomorphic. Meaning that for any two $x, y \in \mathcal{I}$ and any party index $\sigma \in \{0, 1\}$, it should hold that

$$\mathsf{M}_\sigma^x + \mathsf{M}_\sigma^y = \mathsf{M}_\sigma^{x+y} \ ,$$

where $\mathsf{M}_\sigma^z \leftarrow \mathsf{MemGen}(\sigma, \mathsf{ek}_\sigma, z)$, for $z \in \{x, y\}$, and $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$. Throughout this work, we may refer to memory values satisfying this property as "valid" memory values.

We now show that Construction 3.1 satisfies Definition 3.1.

Theorem 3.2. *The HSS scheme of [OSY21] (Construction 3.1) follows the RMS template.*

Proof. In Instantiation 3.2, we show how the MemGen algorithm of the template work in this construction. One can see that the other algorithms of the HSS construction exactly follow the template.
It is easy to see that the outputs of this algorithm are additively homomorphic. This follows from the fact that for any $x \neq 1 \in \mathcal{I}$, this algorithm uses the Input and Eval.ConvertInput algorithms to generate the memory values. Thus, if the HSS scheme works correctly, the generated memory values are intrinsically homomorphic. More specifically, for an input $z \in \mathcal{I}$, the memory value M_σ^z is of the form $\mathsf{M}_\sigma^z = (\langle z \rangle_\sigma, \langle z d^{(0)} \rangle_\sigma, \ldots, \langle z d^{(\ell-1)} \rangle_\sigma)$. Furthermore, when $x = 1$, this algorithm outputs a valid share for the vector $(1, d^{(0)}, \ldots, d^{(\ell-1)})$. □

Instantiation: HSS following the RMS template from DCR

Instantiation 3.2: HSS following the RMS template from Paillier

Requirements and notation: Same as in Instantiation 3.1.

Algorithms:

▶ Setup, Input, and Eval algorithms are the same as in Instantiation 3.1.

▶ $\mathsf{MemGen}(\sigma, \mathsf{ek}_\sigma, x)$:

- If $x = 1$, do:
  - Parse $\mathsf{ek}_\sigma = (k_{\mathsf{prf}}, \langle 1 \rangle_\sigma, \langle d^{(0)} \rangle_\sigma, \ldots, \langle d^{(\ell-1)} \rangle_\sigma)$.
  - Output $\mathsf{M}_\sigma^1 = (\langle 1 \rangle_\sigma, \langle d^{(0)} \rangle_\sigma, \ldots, \langle d^{(\ell-1)} \rangle_\sigma)$.
- Else, do:
  - Run $(\mathsf{I}_0^x, \mathsf{I}_1^x) \leftarrow \mathsf{Input}(\mathsf{pk}, x)$.
  - Run $\mathsf{M}_\sigma^x \leftarrow \mathsf{ConvertInput}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}_\sigma^x)$, and output M_σ^x.

3.2.2 Extended Evaluation and Simulatable Memory Values

Any HSS following the RMS template as defined above satisfies the following lemma, which states that one can evaluate share of $z \cdot P(x^{(1)}, \ldots, x^{(\rho)})$ using only a memory value of z (instead of an input value) together with the input values of the rest of variables $(x^{(1)}, \ldots, x^{(\rho)})$. This lemma plays a central role in our CPRF constructions.

Lemma 3.3. *Let* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{MemGen}, \mathsf{Eval})$ *be an HSS scheme following the RMS template. There exists an extended evaluation algorithm* ExtEval*:*

- $\mathsf{ExtEval}(\sigma, \mathsf{ek}_\sigma, \mathsf{M}_\sigma, (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), P) \to y_\sigma$*: On input a party index* $\sigma \in \{0,1\}$*, an evaluation key* ek_σ*, a single memory value* M_σ*, a vector of* ρ *input values* $(\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)})$*, and an RMS program* P*, return a value* y_σ *such that the following holds.*

For any security parameter $\lambda \in \mathbb{N}$ *and any RMS program* P*, we have:*

$$\Pr\left[y_0 - y_1 = z \cdot P(x^{(1)}, \ldots, x^{(\rho)})\right] \geq 1 - \mathsf{negl}(\lambda) \ , \tag{3.1}$$

where the probability is taken of the choice of $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{I}_0^{(i)}, \mathsf{I}_1^{(i)}) \leftarrow \mathsf{Input}(\mathsf{pk}, x^{(i)})$, $\mathsf{M}_\sigma \leftarrow \mathsf{MemGen}(\sigma, \mathsf{ek}_\sigma, z)$*, and* $y_\sigma \leftarrow \mathsf{ExtEval}(\sigma, \mathsf{ek}_\sigma, \mathsf{M}_\sigma, (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), P)$*, for* $\sigma \in \{0,1\}, i \in [\rho]$.

Proof. We design the extended evaluation algorithm using the original evaluation algorithm Eval. The idea is to recursively include the memory value in the computation. First, we define the modified input converting algorithm $\mathsf{ConvertInput}'(\sigma, \mathsf{ek}_\sigma, \mathsf{I}^x, \mathsf{M}^z)$ which converts input values into a memory values as follows

- $\mathsf{ConvertInput}'(\sigma, \mathsf{ek}_\sigma, \mathsf{I}^x, \mathsf{M}^z) \to \mathsf{M}^{x \cdot z}$:
  This algorithm runs $\mathsf{Mult}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}^x, \mathsf{M}^z) \to \mathsf{M}^{x \cdot z}$ and returns $\mathsf{M}^{x \cdot z}$.

In other words, for any input $x \neq z$ for which an input value I^x is provided, we can compute a memory value $\mathsf{M}^{x \cdot z}$ that is a memory value of $z \cdot x$. In this way, we make sure that each memory value represents a memory value of a multiple of z. Regarding this new shape of memory values, the two other algorithms Add' which adds memory values and Mult' which multiplies input and memory values can be simply defined as the original algorithms of Eval. Namely,

- $\mathsf{Add}'(\mathsf{M}^{x \cdot z}, \mathsf{M}^{y \cdot z}) \to \mathsf{M}^{z \cdot (x+y)}$:
  This algorithm runs $\mathsf{Add}(\mathsf{M}^{x \cdot z}, \mathsf{M}^{y \cdot z}) \to \mathsf{M}^{(x+y) \cdot z}$.

- $\mathsf{Mult}'(\mathsf{I}^x, \mathsf{M}^{y \cdot z}) \to \mathsf{M}^{x \cdot y \cdot z}$:
  This algorithm runs $\mathsf{Mult}(\mathsf{I}^x, \mathsf{M}^{y \cdot z}) \to \mathsf{M}^{x \cdot y \cdot z}$.

Finally, the output algorithm $\mathsf{Output}'(\sigma, \mathsf{ek}_\sigma, \mathsf{M}^{x \cdot z}, n)$ also works as the original algorithm of Eval:

- $\mathsf{Output}'(\sigma, \mathsf{ek}_\sigma, \mathsf{M}^{x \cdot z}, n) \to (x \cdot z) \mod n$:
  This algorithm runs $\mathsf{Output}(\sigma, \mathsf{ek}_\sigma, \mathsf{M}^{x \cdot z}, n) \to (x \cdot z) \mod n$

Conditioned on Eval satisfying the correctness property of HSS (Definition 2.10), algorithm ExtEval also works correctly and on input $(\sigma, \mathsf{ek}_\sigma, \mathsf{M}_\sigma, (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), P)$ outputs the value $z \cdot P(x^{(1)}, \ldots, x^{(\rho)})$. □

We now introduce an additional property termed *simulatable memory values*. Here, we require that for an input $x \in \mathcal{I}$, the memory value of one of the two parties can be generated ahead of time and without the knowledge of x using a simulation algorithm, while the other memory value can be generated given the pre-computed first memory value and the exact value of x. This simulation should not affect the correctness of ExtEval.

Definition 3.2 (HSS with Simulatable Memory Values)**.** Let $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{MemGen}, \mathsf{Eval})$ be an HSS following the RMS template as per Definition 3.1, with input space $\mathcal{I}$ over the ring $\mathcal{R}$. We say that HSS is simulatable with respect to its memory values if there exist algorithms Sim_0 and Sim_1 such that

- $\mathsf{Sim}_0(1^\lambda) \to \mathsf{M}_0$: on input the security parameter λ outputs a memory value M_0.
- $\mathsf{Sim}_1(\mathsf{M}_0, z, (\mathsf{ek}_0, \mathsf{ek}_1)) \to \mathsf{M}_1$: on input a memory value M_0, an element $z \in \mathcal{I}$, and two encoding keys $(\mathsf{ek}_0, \mathsf{ek}_1)$ outputs a memory value M_1.

We also require the two following properties:
Simulation Correctness. For any $\lambda \in \mathbb{N}$ and any $z \in \mathcal{I}$, the correctness condition given in Equation 3.1 holds when the memory value is simulated, i.e. when $\mathsf{M}_0 \leftarrow \mathsf{Sim}_0(1^\lambda)$ and $\mathsf{M}_1 \leftarrow \mathsf{Sim}_1(\mathsf{M}_0, z, (\mathsf{ek}_0, \mathsf{ek}_1))$.
Simulation Security. It should be computationally hard to distinguish the two memory values obtained via the simulation algorithms. That is, for any $\lambda \in \mathbb{N}$ and any $z \in \mathcal{I}$, we have $(z, \mathsf{M}_0) \approx_c (z, \mathsf{M}_1)$ for any $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{M}_0 \leftarrow \mathsf{Sim}_0(1^\lambda)$, and $\mathsf{M}_1 \leftarrow \mathsf{Sim}_1(\mathsf{M}, z, (\mathsf{ek}_0, \mathsf{ek}_1))$.

We now show that Construction 3.1 satisfies Definition 3.2.

Theorem 3.4. *The HSS scheme of [OSY21] (Construction 3.1) generates simulatable memory values.*

Proof. Regarding Definition 3.2, we need to show that there exist two algorithms Sim_0 and Sim_1 that simulate the output of MemGen. We define these algorithms in Instantiation 3.3 and prove their correctness and security in the following.
Simulation Correctness. For any $z \in \mathbb{Z}_N$, it holds that

$$\mathsf{M}_0 - \mathsf{M}_1 = (z, zd^{(0)}, \ldots, zd^{(\ell-1)}),$$

where $\mathsf{M}_0 \leftarrow \mathsf{Sim}_0(1^\lambda)$, and $\mathsf{M}_1 \leftarrow \mathsf{Sim}_1(\mathsf{M}, z, (\mathsf{ek}_0, \mathsf{ek}_1))$. Therefore, the simulated memory values of z are correctly formed as subtractive shares of vector $(z, zd^{(0)}, \ldots, zd^{(\ell-1)})$. Thus, they are valid shares. This guarantees the correctness of multiplication between this values and real input values, and finally the correctness of Equation 3.1 in Lemma 3.3 when M_σ is simulated.

Simulation Security. We need to prove that for any $x \in \mathcal{I}$, it holds that

$$(z, \mathsf{M}_0) \approx_s (z, \mathsf{M}_1),$$

where $\mathsf{M}_1 \leftarrow \mathsf{Sim}_1(\mathsf{M}_0, z, (\mathsf{ek}_0, \mathsf{ek}_1))$, and $\mathsf{M}_0 \leftarrow \mathsf{Sim}_0(1^\lambda)$.
Note that $\mathsf{M}_1 = \mathsf{M}_0 - (z, zd^{(0)}, \ldots, zd^{(\ell-1)})$, where each element of M_0 is chosen uniformly from $\mathbb{Z}_{2^\kappa N}$. Also, in a fixed vector $(z, zd^{(0)}, \ldots, zd^{(\ell-1)})$, z and each $zd^{(i)}$ for $i \in [\ell]$ are elements of $\mathbb{Z}_N$. Therefore, the distribution of each element of M_1 is within the statistical distance $2^{-\kappa}$ of the uniform distribution over $\mathbb{Z}_{2^\kappa N}$ which is the distribution of M_0. □

Instantiation: HSS with Simulatable Memory Values from DCR

Instantiation 3.3: HSS with Simulatable Memory Values from Paillier

Requirements and notation:

- Same as in Instantiation 3.1.

Algorithms:

▶ Setup, Input, MemGen and Eval algorithms are the same as in Instantiation 3.2.

▶ $\mathsf{Sim}_0(1^\lambda)$:

- Sample a random vector $(t, t_0, \ldots, t_{\ell-1}) \xleftarrow{\$} \mathbb{Z}_{2^\kappa \cdot N}^{\ell+1}$.
- Output $\mathsf{M}_0 = (t, t_0, \ldots, t_{\ell-1})$.

▶ $\mathsf{Sim}_1(\mathsf{M}, z, (\mathsf{ek}_0, \mathsf{ek}_1))$:

- Parse $\mathsf{ek}_\sigma = (\langle 1 \rangle_\sigma, \langle d^{(0)} \rangle_\sigma, \ldots, \langle d^{(\ell-1)} \rangle_\sigma)$ for both $\sigma \in \{0, 1\}$.
- For $i \in [\ell]$ reconstruct $d^{(i)} = \langle d^{(i)} \rangle_0 - \langle d^{(i)} \rangle_1 \bmod N$.
- Compute and output $\mathsf{M}_1 = \mathsf{M}_0 - (z, zd^{(0)}, \ldots, zd^{(\ell-1)})$.

3.2.3 Staged Homomorphic Secret Sharing

Finally, we define a new notion termed staged-HSS which is merely extending the idea of HSS with simulatable memory values to the case where we require the possibility of input values to be simulatable as well.

Definition 3.3 (staged-HSS)**.** Let $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{MemGen}, \mathsf{Input}, \mathsf{Eval})$ be an HSS scheme following the RMS template, with input space $\mathcal{I}$ over the ring $\mathcal{R}$. We say it is a staged-HSS if there exist additional algorithms $(\overline{\mathsf{Input}}_0, \overline{\mathsf{Input}}_1)$, and $(\overline{\mathsf{Eval}}_0, \overline{\mathsf{Eval}}_1)$ such that:

- $\overline{\mathsf{Input}}_0(\mathsf{pk}) \to (\bar{\mathsf{I}}_0, \mathsf{aux})$: On input a public key pk, return a value $\bar{\mathsf{I}}_0$ and an auxiliary output aux.
- $\overline{\mathsf{Input}}_1(\mathsf{pk}, x, \mathsf{aux}, (\mathsf{ek}_0, \mathsf{ek}_1)) \to \bar{\mathsf{I}}_1$: On input a public key pk, an input $x \in \mathcal{I}$, an auxiliary input aux, and two encoding keys $(\mathsf{ek}_0, \mathsf{ek}_1)$, return a value $\bar{\mathsf{I}}_1$.
- $\overline{\mathsf{Eval}}_0(\mathsf{ek}_0, (\bar{\mathsf{I}}_0^{(1)}, \ldots, \bar{\mathsf{I}}_0^{(\rho)}), P) \to \mathsf{M}_0$: On input an evaluation key ek_0, a vector of ρ input values $(\bar{\mathsf{I}}_0^{(1)}, \ldots, \bar{\mathsf{I}}_0^{(\rho)})$, and a program P, return a memory value M_0.
- $\overline{\mathsf{Eval}}_1(\mathsf{ek}_1, (\bar{\mathsf{I}}_1^{(1)}, \ldots, \bar{\mathsf{I}}_1^{(\rho)}), (x^{(1)}, \ldots, x^{(\rho)}), P) \to \mathsf{M}_1$: On input an evaluation key ek_1, a vector of ρ input values $(x^{(1)}, \ldots, x^{(\rho)})$ as well as $(\bar{\mathsf{I}}_1^{(1)}, \ldots, \bar{\mathsf{I}}_1^{(\rho)})$, and a program P, return a memory value M_1.

We further require the two following properties:

Correctness. We require that the outputs of $\overline{\mathsf{Eval}}_0$ and $\overline{\mathsf{Eval}}_1$ to be usable within the extended evaluation algorithm ExtEval (Lemma 3.3). Formally, for any $\lambda \in \mathbb{N}$ and any two RMS programs $P, Q \in \mathcal{P}$, it should hold that

$$\Pr[y_0 - y_1 = P(z^{(1)}, \ldots, z^{(\ell)}) \cdot Q(x^{(1)}, \ldots, x^{(\rho)})] \geq 1 - \mathsf{negl}(\lambda) \ ,$$

where $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{I}_0^{x^{(i)}}, \mathsf{I}_1^{x^{(i)}}) \leftarrow \mathsf{Input}(\mathsf{pk}, x^{(i)})$, for all $i \in [\rho]$, $(\bar{\mathsf{I}}_0^{z^{(i)}}, \mathsf{aux}^{(i)}) \leftarrow \overline{\mathsf{Input}}_0(\mathsf{pk})$, $\bar{\mathsf{I}}_1^{z^{(i)}} \leftarrow \overline{\mathsf{Input}}_1(\mathsf{pk}, z^{(i)}, \mathsf{aux}^{(i)}, (\mathsf{ek}_0, \mathsf{ek}_1))$, for all $i \in [\ell]$, $\mathsf{M}_0 \leftarrow \overline{\mathsf{Eval}}_0(\mathsf{ek}_0, (\bar{\mathsf{I}}_0^{z^{(1)}}, \ldots, \bar{\mathsf{I}}_0^{z^{(\ell)}}), P)$, $\mathsf{M}_1 \leftarrow \overline{\mathsf{Eval}}_1(\mathsf{ek}_1, (\bar{\mathsf{I}}_1^{z^{(1)}}, \ldots, \bar{\mathsf{I}}_1^{z^{(\ell)}}), (z^{(1)}, \ldots, z^{(\ell)}), P)$, and $y_\sigma \leftarrow \mathsf{ExtEval}(\sigma, \mathsf{ek}_\sigma, (\mathsf{M}_\sigma, \mathsf{I}_\sigma^{x^{(1)}}, \ldots, \mathsf{I}_\sigma^{x^{(\rho)}}), Q)$, for $\sigma \in \{0, 1\}$.

Security. The output of $\overline{\mathsf{Input}}_1$ and Input should be computationally indistinguishable. Formally, for any $\lambda \in \mathbb{N}$, and any $x \in \mathcal{I}$, the two following distributions should be computationally indistinguishable:

$$\left\{ \bar{\mathsf{I}}_1 : \begin{array}{l} (\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\bar{\mathsf{I}}_0, \mathsf{aux}) \leftarrow \overline{\mathsf{Input}}_0(\mathsf{pk}) \\ \bar{\mathsf{I}}_1 \leftarrow \overline{\mathsf{Input}}_1(\mathsf{pk}, x, \mathsf{aux}, (\mathsf{ek}_0, \mathsf{ek}_1)) \end{array} \right\} \overset{c}{\approx} \left\{ \mathsf{I}_1 : \begin{array}{l} (\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{Input}(\mathsf{pk}, x) \end{array} \right\} .$$

Theorem 3.5. *Assuming the hardness of DCR, the HSS scheme of [OSY21] (Construction 3.1) is a staged HSS.*

Proof. In Instantiation 3.4, we define the four algorithms $(\overline{\mathsf{Input}}_0, \overline{\mathsf{Input}}_1)$ and $(\overline{\mathsf{Eval}}_0, \overline{\mathsf{Eval}}_1)$ according to Definition 3.3. We now prove that the algorithms work correctly and satisfy the required properties.

Correctness. We show that a memory value M_σ^y outputted by $\overline{\mathsf{Eval}}_\sigma$ is in fact party σ's subtractive share of the vector $(y, yd^{(0)}, \ldots, yd^{(\ell-1)})$, thus it is a valid memory value. This guarantees the correctness of ExtEval algorithm when given as input a staged memory value and a vector of original input values.

Since the new evaluation algorithms $\overline{\mathsf{Eval}}_0$ and $\overline{\mathsf{Eval}}_1$ work the same as the original evaluation algorithm Eval except for the multiplication instruction, we briefly prove the correctness of multiplication in the following. Let $x, y \in \mathcal{I}$ be any two arbitrary input values. We show that

$$\Pr\left[z_0 - z_1 = xy\right] \geq 1 - \mathsf{negl}(\lambda),$$

and

$$\Pr\left[(zd^{(i)})_0 - (zd^{(i)})_1 = xyd^{(i)}\right] \geq 1 - \mathsf{negl}(\lambda),$$

for all $i \in [\ell]$, where
$\mathsf{M}_0^z = \left(z_0, (zd^{(0)})_0, \ldots, (zd^{(\ell-1)})_0\right) \leftarrow \overline{\mathsf{Mult}}_0(\mathsf{ek}_0, \bar{\mathsf{I}}_0^x, \mathsf{M}_0^y)$,
$\mathsf{M}_1^z = \left(z_1, (zd^{(0)})_1, \ldots, (zd^{(\ell-1)})_1\right) \leftarrow \overline{\mathsf{Mult}}_1(\mathsf{ek}_1, \bar{\mathsf{I}}_1^x, \mathsf{M}_1^y, y)$,
$(\bar{\mathsf{I}}_0^b, \mathsf{aux}^b) \leftarrow \overline{\mathsf{Input}}_0(\mathsf{pk})$ for $b \in \{x, y\}$,
$\bar{\mathsf{I}}_1^b \leftarrow \overline{\mathsf{Input}}_1(\mathsf{pk}, b, \mathsf{aux}^b)$, for $b \in \{x, y\}$,
$\mathsf{M}_0^y \leftarrow \overline{\mathsf{ConvertInput}}_0(\mathsf{ek}_0, \bar{\mathsf{I}}_0^y)$,
$\mathsf{M}_1^y \leftarrow \overline{\mathsf{ConvertInput}}_1(\mathsf{ek}_1, \bar{\mathsf{I}}_1^y, y)$, and
$(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$.

Regarding how $\overline{\mathsf{Mult}}_0$ and $\overline{\mathsf{Mult}}_1$ works, it holds that $z_b = \mathsf{DDLog}_N(\mathsf{ct}'_b)$, for $b \in \{0,1\}$. Thus, by Lemma 3.1, it's enough to prove that $\mathsf{ct}'_0 \cdot \mathsf{ct}'_1 = (1+N)^{xy}$. We have

$$\begin{aligned}
\mathsf{ct}'_0 \cdot \mathsf{ct}'_1 &= \prod_{i=0}^{\ell-1} (c_i)^{-\langle yd^{(i)} \rangle_0} \cdot (\hat{c})^y \cdot \prod_{i=0}^{\ell-1} (c_i)^{-\langle yd^{(i)} \rangle_1} \\
&= (\hat{c})^y \cdot \prod_{i=1}^{\ell} (c_i)^{-yd^{(i)}} \\
&= (1+N)^{xy} \cdot \prod_{i=1}^{\ell} (c_i)^{yd^{(i)}} \cdot \prod_{i=1}^{\ell} (c_i)^{-yd^{(i)}} \\
&= (1+N)^{xy} (\mathrm{mod}\ N^2).
\end{aligned}$$

The equation $\mathsf{ct}_0^{'(j)} \cdot \mathsf{ct}_1^{'(j)} = (1+N)^{xyd^{(j)}}$ for all $j \in [\ell]$ is proved similarly.

Security. Outputs of the $\overline{\mathsf{Input}}_1$ algorithm are in fact in the same form as the Input algorithm. More precisely, they are both Paillier encryptions of the vector $(x, xd^{(0)}, \ldots, xd^{(\ell-1)})$, where d is the secret key of the encryption scheme. Therefore, they are computationally indistinguishable. □

Instantiation: Staged HSS from DCR

3

Instantiation 3.4: HSS with Simulatable Memory Values from Paillier

Requirements and notation: Same as in Instantiation 3.1.

Algorithms:

▶ $\overline{\mathsf{Input}}_0(\mathsf{pp}) \to (\bar{\mathsf{I}}_0, \mathsf{aux})$

- Parse $\mathsf{pp} = (\mathsf{BG.pk}, D^{(0)}, \ldots, D^{(\ell-1)})$, and $\mathsf{BG.pk} = (N, \mathbf{g}, \hat{g})$.
- Sample $r \xleftarrow{\$} \mathbb{Z}_N$ and compute $\mathsf{ct}_{\mathsf{ind}} = \mathbf{g}^r$.
- For $i \in [\ell]$, sample $r_i \xleftarrow{\$} \mathbb{Z}_N$, and compute $\mathsf{ct}_{\mathsf{ind}}^{(i)} = \mathbf{g}^{r_i}$.
- Set $\bar{\mathsf{I}}_0 = (\mathsf{ct}_{\mathsf{ind}}, \mathsf{ct}_{\mathsf{ind}}^{(0)}, \ldots, \mathsf{ct}_{\mathsf{ind}}^{(\ell-1)})$.
- Set $\mathsf{aux} = (\mathbf{g}^r, \hat{g}^r, \{\mathbf{g}^{r_i}\}_{i\in[\ell]}, \{\hat{g}^{r_i}\}_{i\in[\ell]})$.
- Output $(\bar{\mathsf{I}}_0, \mathsf{aux})$.

▶ $\overline{\mathsf{Input}}_1(\mathsf{pp}, x, \mathsf{aux}, (\mathsf{ek}_0, \mathsf{ek}_1)) \to \bar{\mathsf{I}}_1$

- Parse $\mathsf{pp} = (\mathsf{BG.pk}, (D^{(i)})_{i\in[\ell]})$, $\mathsf{BG.pk} = (N, \mathbf{g}, \hat{g})$, and $\mathsf{aux} = (\mathbf{g}^r, \hat{g}^r, \{\mathbf{g}^{r_i}\}_{i\in[\ell]}, \{\hat{g}^{r_i}\}_{i\in[\ell]})$, and $\mathsf{ek}_\sigma = (k_{\mathsf{prf}}, \langle d^{(0)} \rangle_\sigma, \ldots, \langle d^{(\ell-1)} \rangle_\sigma)$ for $\sigma \in \{0,1\}$.
- Compute $\mathsf{ct} = (\mathbf{g}^r, \hat{g}^r \cdot (1+N)^x)$.
- For $i \in [\ell]$ do
  - Reconstruct $d^{(i)} = \langle d^{(i)} \rangle_0 - \langle d^{(i)} \rangle_1 \bmod N$.
  - Compute $\mathsf{ct}^{(i)} = (\mathbf{g}^{r_i}, \hat{g}^{r_i} \cdot (1+N)^{xd^{(i)}})$.
- Output $\bar{\mathsf{I}}_1 = (\mathsf{ct}, \mathsf{ct}^{(0)}, \ldots, \mathsf{ct}^{(\ell-1)})$.

▶ $\overline{\mathsf{Eval}}_0(\mathsf{ek}_0, (\bar{\mathsf{I}}_0^{(1)}, \ldots, \bar{\mathsf{I}}_0^{(\rho)}), P) \to \mathsf{M}_0$

- $\overline{\mathsf{ConvertInput}}_0(\mathsf{ek}_0, \bar{\mathsf{I}}_x)$ // same as in Eval
  - Parse $\mathsf{ek}_0 = (\langle 1\rangle_0, \langle d^{(0)}\rangle_0, \ldots, \langle d^{(\ell-1)}\rangle_0)$.
  - Set $\mathsf{M}_0^1 = (\langle 1\rangle_0, \langle d^{(0)}\rangle_0, \ldots, \langle d^{(\ell-1)}\rangle_0)$.
  - Compute $\mathsf{M}_0^x \leftarrow \overline{\mathsf{Mult}}_0(\mathsf{ek}_0, \bar{\mathsf{I}}_0, \mathsf{M}_0^1)$.
- $\overline{\mathsf{Add}}_0(\mathsf{ek}_0, \mathsf{M}_0^x, \mathsf{M}_0^y)$ // same as in Eval
  - Parse $\mathsf{M}_0^x = (\langle x\rangle_0, \langle xd^{(0)}\rangle_0, \ldots, \langle xd^{(\ell-1)}\rangle_0)$, and $\mathsf{M}_0^y = (\langle y\rangle_0, \langle yd^{(0)}\rangle_0, \ldots, \langle yd^{(\ell-1)}\rangle_0)$.
  - Compute $\langle z\rangle_0 = \langle x\rangle_0 + \langle y\rangle_0$, and $\langle zd^{(i)}\rangle_0 = \langle xd^{(i)}\rangle_0 + \langle yd^{(i)}\rangle_0$ for $i \in [\ell]$.
  - Output $\mathsf{M}_0^z = (\langle z\rangle_0, \langle zd^{(0)}\rangle_0, \ldots, \langle zd^{(\ell-1)}\rangle_0)$.
- $\overline{\mathsf{Mult}}_0(\mathsf{ek}_0, \bar{\mathsf{I}}_0^x, \mathsf{M}_0^y)$ // different from Eval
  - Parse $\bar{\mathsf{I}}_0^x = (\mathsf{ct}_{\mathsf{ind}}, \mathsf{ct}_{\mathsf{ind}}^{(0)}, \ldots, \mathsf{ct}_{\mathsf{ind}}^{(\ell-1)})$, and $\mathsf{M}_0^y = (\langle y\rangle_0, \langle yd^{(0)}\rangle_0, \ldots, \langle yd^{(\ell-1)}\rangle_0)$.
  - Parse $\mathsf{ct}_{\mathsf{ind}} = (c_0, \ldots, c_{\ell-1})$, and $\mathsf{ct}_{\mathsf{ind}}^{(i)} = (c_0^{(i)}, \ldots, c_{\ell-1}^{(i)})$ for $i \in [\ell]$.
  - Compute $\langle z\rangle_0 = \mathsf{DDLog}_N(\mathsf{ct}')(\mathrm{mod}\ N) + F_{k_{\mathsf{prf}}}(\mathsf{id})$, where
$$\mathsf{ct}' = \prod_{i=0}^{\ell-1} (c_i)^{-\langle yd^{(i)}\rangle_0} (\mathrm{mod}\ N^2).$$
  - For $j \in [\ell]$, compute $\langle zd^{(j)}\rangle_0 = \mathsf{DDLog}_N(\mathsf{ct}'_j)(\mathrm{mod}\ N) + F_{k_{\mathsf{prf}}}(\mathsf{id})$, where
$$\mathsf{ct}'_j = \prod_{i=0}^{\ell-1} (c_i^{(j)})^{-\langle yd^{(i)}\rangle_0} (\mathrm{mod}\ N^2).$$
  - Output $\mathsf{M}_0^z = (\langle z\rangle_0, \langle zd^{(0)}\rangle_0, \ldots, \langle zd^{(\ell-1)}\rangle_0)$.

▶ $\overline{\mathsf{Eval}}_1(\mathsf{ek}_1, (\bar{\mathsf{I}}_1^{(1)}, \ldots, \bar{\mathsf{I}}_1^{(\rho)}), (x^{(1)}, \ldots, x^{(\rho)}), P) \to \mathsf{M}_1$

- $\overline{\mathsf{ConvertInput}}_1(\mathsf{ek}_1, \bar{\mathsf{I}}_x, x)$ // same as in Eval
  - Parse $\mathsf{ek}_1 = (\langle 1\rangle_1, \langle d^{(0)}\rangle_1, \ldots, \langle d^{(\ell-1)}\rangle_1)$.
  - Set $\mathsf{M}_1^1 = (\langle 1\rangle_1, \langle d^{(0)}\rangle_1, \ldots, \langle d^{(\ell-1)}\rangle_1)$.
  - Compute $\mathsf{M}_1^x \leftarrow \overline{\mathsf{Mult}}_1(\mathsf{ek}_1, \bar{\mathsf{I}}_1, \mathsf{M}_1^1, x)$.
- $\overline{\mathsf{Add}}_1(\mathsf{ek}_1, \mathsf{M}_1^x, \mathsf{M}_1^y)$ // same as in Eval
  - Parse $\mathsf{M}_1^x = (\langle x\rangle_1, \langle xd^{(0)}\rangle_1, \ldots, \langle xd^{(\ell-1)}\rangle_1)$, and $\mathsf{M}_1^y = (\langle y\rangle_1, \langle yd^{(0)}\rangle_1, \ldots, \langle yd^{(\ell-1)}\rangle_1)$.
  - Compute $\langle z\rangle_1 = \langle x\rangle_1 + \langle y\rangle_1$, and $\langle zd^{(i)}\rangle_1 = \langle xd^{(i)}\rangle_1 + \langle yd^{(i)}\rangle_1$ for $i \in [\ell]$.
  - Output $\mathsf{M}_1^z = (\langle z\rangle_1, \langle zd^{(0)}\rangle_1, \ldots, \langle zd^{(\ell-1)}\rangle_1)$.

- $\overline{\mathsf{Mult}}_1(\mathsf{ek}_0, \bar{\mathsf{I}}_0^x, \mathsf{M}_0^y, y)$ // different from Eval
  - Parse $\bar{\mathsf{I}}_1^x = (\mathsf{ct}, \mathsf{ct}^{(0)}, \ldots, \mathsf{ct}^{(\ell-1)})$, and $\mathsf{M}_1^y = (\langle y \rangle_1, \langle y d^{(0)} \rangle_1, \ldots, \langle y d^{(\ell-1)} \rangle_1)$.
  - Parse $\mathsf{ct} = (c_0, \ldots, c_{\ell-1}, \hat{c})$, and $\mathsf{ct}^{(i)} = (c_0^{(i)}, \ldots, c_{\ell-1}^{(i)}, \hat{c}^{(i)})$ for $i \in [\ell]$.
  - Compute $\langle z \rangle_1 = \mathsf{DDLog}_N(\mathsf{ct}')(\mathrm{mod}\ N) + F_{k_{\mathsf{prf}}}(\mathsf{id})$, where
$$\mathsf{ct}' = (\hat{c})^y \cdot \prod_{i=0}^{\ell-1} (c_i)^{-\langle y d^{(i)} \rangle_1} (\mathrm{mod}\ N^2).$$
  - For $j \in [\ell]$, compute $\langle z d^{(j)} \rangle_1 = \mathsf{DDLog}_N(\mathsf{ct}'_j)(\mathrm{mod}\ N) + F_{k_{\mathsf{prf}}}(\mathsf{id})$, where
$$\mathsf{ct}'_j = (\hat{c}^{(j)})^y \cdot \prod_{i=0}^{\ell-1} (c_i^{(j)})^{-\langle y d^{(i)} \rangle_1} (\mathrm{mod}\ N^2).$$
  - Output $\mathsf{M}_1^z = (\langle z \rangle_1, \langle z d^{(0)} \rangle_1, \ldots, \langle z d^{(\ell-1)} \rangle_1)$.

3

3.3 Constrained Pseudorandom Functions

We now present our two transformations from homomorphic secret sharing to constrained pseudorandom functions.

3.3.1 CPRF for Inner-Product from HSS

Our first construction is a 1-key selectively secure constrained pseudorandom function for inner-product. The space input is $\mathcal{R}^n$ for some ring $\mathcal{R}$ and $n > 0$, and a constraint is defined by a vector $\mathbf{z} \in \mathcal{R}^n$. A constrained key for a vector $\mathbf{z}$ allows to compute the PRF evaluation on input $\mathbf{x} \in \mathcal{R}^n$ if and only if $\langle \mathbf{z}, \mathbf{x} \rangle = 0$. Specifically, the class of constraints is $\{C_{\mathbf{z}} \mid \mathbf{z} \in \mathcal{R}^n\}$ where the circuit $C_{\mathbf{z}} : \mathcal{R}^n \to \{0, 1\}$ is defined as $C_{\mathbf{z}}(\mathbf{x}) = 0$ if $\langle \mathbf{z}, \mathbf{x} \rangle = 0$, else 1.

The intuition behind our construction is that the master secret key and the constrained key (for a vector $\mathbf{z}$) are used to compute, via HSS, a share of $\langle \mathbf{x}, \mathbf{z} \rangle \cdot F_k(\mathbf{x})$, where k is a PRF key encoded via the HSS scheme. Then, if $\langle \mathbf{x}, \mathbf{z} \rangle = 0$, the two evaluations produce substractive shares of 0, i.e. equal shares, while if $\langle \mathbf{x}, \mathbf{z} \rangle \neq 0$, the shares differ by (a non-zero multiple of) $F_k(\mathbf{x})$. By the security of HSS, the PRF key k remains hidden to the constrained key owner, hence the actual PRF evaluation (the value of the share computed from the master secret key) is pseudorandom even given the value of the second share (which can be computed from the constrained key).

Before diving into our construction, we generalize Lemma 3.3, stating that not only one can produce shares of any evaluation of the form $z \cdot P(\mathbf{x})$ given a memory value for z and encoding of $\mathbf{x}$, but of any linear combination $\sum_i \alpha^{(i)} z^{(i)} \cdot P(\mathbf{x})$ with known coefficients given memory values for multiple $z^{(i)}$'s, i.e. for $\langle \mathbf{z}, \boldsymbol{\alpha} \rangle$ for a known vector $\boldsymbol{\alpha} = (\alpha^{(1)}, \ldots, \alpha^{(\ell)})$.

Corollary 3.1. *Let* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{MemGen}, \mathsf{Eval})$ *be an HSS scheme following the RMS template. There exists an extended evaluation algorithm* $\mathsf{LinExtEval}$*:*

- $\mathsf{LinExtEval}(\sigma, \mathsf{ek}_\sigma, (\mathsf{M}_\sigma^{(1)}, \ldots, \mathsf{M}_\sigma^{(\ell)}), (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), (\alpha^{(1)}, \ldots, \alpha^{(\ell)}), P) \to y_\sigma$*:* *On input a party index* $\sigma \in \{0,1\}$*, an evaluation key* ek_σ*, a vector of* ℓ *memory values* $\mathsf{M}_\sigma^{(1)}, \ldots, \mathsf{M}_\sigma^{(\ell)}$*, a vector of* ρ *input values* $(\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)})$*, a vector of* ℓ *ring elements* $\alpha^{(1)}, \ldots, \alpha^{(\ell)}$*, and an RMS program* P*, this algorithm outputs a value* y_σ *such that the following holds.*

For any security parameter $\lambda \in \mathbb{N}$*, any* $\alpha^{(i)} \in \mathcal{R}$ *for* $i \in [\ell]$*, and any RMS program* P*, we have:*

$$\Pr\left[y_0 - y_1 = \left(\sum_{i=1}^{\ell} \alpha^{(i)} \cdot z^{(i)}\right) \cdot P(x^{(1)}, \ldots, x^{(\rho)})\right] \geq 1 - \mathsf{negl}(\lambda) \ ,$$

where the probability is taken over the choice of $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$*,* $(\mathsf{I}_0^{(i)}, \mathsf{I}_1^{(i)}) \leftarrow \mathsf{Input}(\mathsf{pk}, x^{(i)})$*,* $\mathsf{M}_\sigma^{(j)} \leftarrow \mathsf{MemGen}(\sigma, \mathsf{ek}_\sigma, z^{(j)})$*, and over the shares* $y_\sigma \leftarrow \mathsf{LinExtEval}(\sigma, \mathsf{ek}_\sigma, (\mathsf{M}_\sigma^{(1)}, \ldots, \mathsf{M}_\sigma^{(\ell)}), (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), (\alpha^{(1)}, \ldots, \alpha^{(\ell)}), P)$*, with* $\sigma \in \{0,1\}, j \in [\ell], i \in [\rho]$*.*

The proof of the above statement follows from Lemma 3.3 by linearly combining the substractive shares obtained by applying ExtEval with each memory value.
We now have all the ingredients for our first construction.

Construction 3.1: CPRF for IP from HSS

Requirements and notation:

- Let $F : \mathcal{K} \times \mathcal{R}^n \to \mathcal{Y}$ be a PRF with evaluation in NC^1. For $\mathbf{x} \in \mathcal{R}^n$, we denote by $F_\bullet(\mathbf{x}) : \mathcal{K} \to \mathcal{Y}$ the function that maps $k \in \mathcal{K}$ to $F_k(\mathbf{x})$.
- Let $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{MemGen}, \mathsf{Eval})$ be a homomorphic secret sharing following the RMS template with simulatable memory values.

Algorithms:

▶ $\mathsf{KeyGen}(1^\lambda)$:

1. Run $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$.
2. Sample $k \xleftarrow{\$} \mathcal{K}$ for F
3. Run $(\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{Input}(\mathsf{pk}, k)$.
4. For $i \in \{1, \ldots, n\}$, set $\mathsf{M}_0^i \leftarrow \mathsf{Sim}_0(1^\lambda)$.
5. Set $\mathsf{msk} \leftarrow ((\mathsf{ek}_0, \mathsf{I}_0, (\mathsf{M}_0^i)_{i\in[n]}), (\mathsf{ek}_1, \mathsf{I}_1))$, and output $\mathsf{pp} = \mathsf{pk}$ and msk.

▶ $\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, \mathbf{x})$:

1. Parse $\mathsf{msk} = ((\mathsf{ek}_0, \mathsf{I}_0, (\mathsf{M}_0^i)_{i\in[n]}), (\mathsf{ek}_1, \mathsf{I}_1))$.
2. Compute $y_0 \leftarrow \mathsf{LinExtEval}(0, \mathsf{ek}_0, (\mathsf{M}_0^i)_{i\in[n]}, \mathsf{I}_0, \mathbf{x}, F_\bullet(\mathbf{x}))$.
3. Output y_0.

▶ $\mathsf{Constrain}(\mathsf{msk}, \mathbf{z})$:

1. Parse msk as $((\mathsf{ek}_0, \mathsf{l}_0, (\mathsf{M}_0^i)_{i\in[n]}), (\mathsf{ek}_1, \mathsf{l}_1))$
2. Parse $\mathbf{z} = (z_1, \ldots, z_n)$.
3. For $i \in \{1, \ldots, n\}$, set $\mathsf{M}_1^i \leftarrow \mathsf{Sim}_1(\mathsf{M}_0^i, z_i, (\mathsf{ek}_0, \mathsf{ek}_1))$.
4. Return $\mathsf{ck}_{\mathbf{z}} = (\mathsf{ek}_1, \mathsf{l}_1, (\mathsf{M}_1^i)_{i\in[n]})$.

▶ $\mathsf{CEval}(\mathsf{pp}, \mathsf{ck}_{\mathbf{z}}, \mathbf{x})$:

1. Parse $\mathsf{ck}_{\mathbf{z}} = (\mathsf{ek}_1, \mathsf{l}_1, (\mathsf{M}_1^i)_{i\in[n]})$.
2. Compute $y_1 \leftarrow \mathsf{LinExtEval}(1, \mathsf{ek}_1, (\mathsf{M}_1^i)_{i\in[n]}, \mathsf{l}_1, \mathbf{x}, F_\bullet(\mathbf{x}))$.
3. Output y_1.

3

Theorem 3.6. *Assuming F is a secure PRF with evaluation in NC^1 and HSS is a secure HSS scheme following the RMS template with simulatable memory values, then Construction 3.1 is a selective 1-key, constraint-hiding, secure CPRF for inner-product.*

Proof. We now prove correctness and security of Construction 3.1, starting with correctness.

Correctness. The idea of the scheme is to choose a random key k for the PRF F, and use HSS to compute shares of $\langle \mathbf{x}, \mathbf{z}\rangle \cdot F_k(\mathbf{x})$, respectively owned by the master key owner and the constrained key owner. It is easy to verify that if HSS is correct (with simulated memory values) and if $\langle \mathbf{x}, \mathbf{z}\rangle$, the two shares form substractive shares of $\langle \mathbf{x}, \mathbf{z}\rangle \cdot F_k(\mathbf{x}) = 0$ and therefore both evaluations match.

Pseudorandomness. Let us now prove selective security of our construction. The proof relies on a sequence of hybrid games. Let $\mathcal{A}$ denote a 1-key selective adversary against the pseudorandomness of the above construction. Selective security plays an important role as one needs to rely on the knowledge of the constraint $\mathbf{z}$ to answer evaluation oracle queries appropriately in the early stage of the proof.

Hybrid $\mathcal{H}_0$: This is the standard CPRF security game where the challenge is answered with the real PRF evaluation.

Hybrid $\mathcal{H}_1$: In this first hybrid game, we only change the definition of the evaluation oracle. Since we are in the selective setting, the challenger knows the constraint $\mathbf{z}$ before answering any evaluation query. Therefore, it can compute the constrained key $\mathsf{ck}_{\mathbf{z}}$ from the start. When asked for an evaluation query on input x, the challenger replies to it by computing $y_1 \leftarrow \mathsf{CEval}(\mathsf{pp}, \mathsf{ck}_{\mathbf{z}}, \mathbf{x})$ and returning $y_1 + \langle \mathbf{x}, \mathbf{z}\rangle \cdot F_k(\mathbf{x})$. The adversary's view remains identical to its view in $\mathcal{H}_0$ by correctness of HSS, therefore these two hybrid games are perfectly indistinguishable.

Hybrid $\mathcal{H}_2$: In this second hybrid game, we simply switch the memory value sampled via Sim_1 in the constrained secret key to memory value sampled from Sim_0. It is immediate that $\mathcal{H}_1$ and $\mathcal{H}_2$ are indistinguishable thanks to simulation security of the HSS.

Hybrid $\mathcal{H}_3$: We now remove the information about k in the constrained evaluation key

as follows: instead of defining $(\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{Input}(\mathsf{pk}, k)$, we set $(\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{Input}(\mathsf{pk}, 0)$.[1] A PRF key $k \xleftarrow{\$} \mathcal{K}$ is still sampled by the challenger, and evaluation (and challenge) queries are still answered on input $\mathbf{x}$ by having the challenger computing $y_1 \leftarrow \mathsf{CEval}(\mathsf{pp}, \mathsf{ck}_{\mathbf{z}}, \mathbf{x})$ and returning $y_1 + \langle \mathbf{x}, \mathbf{z} \rangle \cdot F_k(\mathbf{x})$.
By HSS security (for $\sigma = 1$), we claim that $\mathcal{H}_2$ and $\mathcal{H}_3$ are computationally indistinguishable. Suppose that an adversary $\mathcal{A}$ could distinguish between these two hybrids, we construct an adversary $\mathcal{B}$ against HSS security as follows: $\mathcal{B}$ first samples a key $k \xleftarrow{\$} \mathcal{K}$ and submits message $(k, 0)$ to its HSS challenger. It gets back $(\mathsf{pk}, \mathsf{ek}_1, \mathsf{I}_1)$ where I_1 is the second half of $\mathsf{Input}(k)$ or $\mathsf{Input}(0)$, depending on whether k or 0 was encoded by the challenger. Then, $\mathcal{B}$ computes the constrained key $\mathsf{ck}_{\mathbf{z}}$ as in the previous game, by sampling memory values using $\mathsf{Sim}_0(1^\lambda)$. It answers $\mathcal{A}$'s evaluation (and challenge) queries by computing $y_1 \leftarrow \mathsf{CEval}(\mathsf{pp}, \mathsf{ck}_{\mathbf{z}}, \mathbf{x})$ and returning $y_1 + \langle \mathbf{x}, \mathbf{z} \rangle \cdot F_k(\mathbf{x})$. When $\mathcal{A}$ halts with some output b', so does $\mathcal{B}$. It is clear that $\mathcal{B}$ simulates either $\mathcal{H}2$ or $\mathcal{H}3$, depending on whether it was given an encoding of k or of 0, which results in our claim.
Hybrid $\mathcal{H}_4$: In this hybrid, the challenger replies to the challenge query by returning a uniformly random value from $\mathbb{Z}_n$. Since the adversary's view does no longer contain any information about the PRF key k, the value of $F_k(\mathbf{x})$ is computationally indistinguishable from a random element of $\mathcal{Y}$ due to the security of the PRF. We also required $\mathcal{Y}$ to be such that F is pseudorandom on $\mathbb{Z}_n$. Therefore, hybrids $\mathcal{H}_3$ and $\mathcal{H}_4$ are computationally indistinguishable thanks to the security of the underlying PRF. Note that here that we only rely on Find-then-Guess security for the underlying PRF (See Definition 2.8).
The rest of the proof proceeds by reversing the sequence of hybrid games while leaving the challenge query answered by a uniformly random value.
Constraint-Hiding. We finally prove that our construction is also constraint-hiding. The proof essentially follows the same line as the proof of pseudorandomness except that one deviates at $\mathcal{H}_4$. Notice that in $\mathcal{H}_3$ already, the only place where $\mathbf{z}$ plays a role in the adversary's view is in the evaluations, since the constrained key is sampled using Sim_0 after $\mathcal{H}_2$.
Now, the hybrid game $\mathcal{H}_4$ for the constraint-hiding proof does the following: rather than using Find-then-Guess security and changing only the evaluation of the challenge (which no longer exists in the constraint-hiding security game), we use standard PRF security to replace answers to evaluation queries of the form $y_1 + \langle \mathbf{x}, \mathbf{z} \rangle \cdot F_k(\mathbf{x})$ by values $y_1 + \langle \mathbf{x}, \mathbf{z} \rangle \cdot f(\mathbf{x})$ where f is a truly random function (sampled lazily). This changes evaluation at points $\mathbf{x}$ such that $\langle \mathbf{x}, \mathbf{z} \rangle \neq 0$ to uniformly random (and independent) values, in particular these values are independent of the constraint $\mathbf{z}$. One can then switch the constraint $\mathbf{z}$ to $\mathbf{z}'$ easily, since the pair of constraints is required to satisfy $\langle \mathbf{x}, \mathbf{z} \rangle \neq 0$ if and only if $\langle \mathbf{x}, \mathbf{z}' \rangle \neq 0$ for all evaluation queries x.
This concludes the proof of Theorem 3.6. □

Remark 3.1. *In the above construction, we require the PRF range $\mathcal{Y}$ to be such that F is pseudorandom on $\mathbb{Z}_n$, for a fixed $n < B$, where B is the magnitude bound of the RMS programs that the HSS scheme used in the construction supports. We need to then reduce the outputs of the HSS evaluation algorithm modulo n by inputting n as the modulus to algorithm* Output *(See Definition 3.1). This is used in the security proof to ensure that masking with a pseudorandom value over $\mathcal{Y}$ causes the output to be pseudorandom.*

Combining Theorem 3.6 (CPRF for inner-product from HSS with simulatable memory values) with Theorem 3.4 (HSS with simulatable memory values from DCR), yields the

[1] By 0 we mean any fixed key, e.g. 0^λ if $\mathcal{K} = \{0,1\}^\lambda$.

following corollary.

Corollary 3.2 (Private CPRF for Inner-Product from DCR)**.** *There exist 1-key selectively-secure, constraint-hiding constrained pseudorandom functions for inner-product assuming the hardness of* DCR*.*

3.3.2 CPRF for NC^1 from HSS

We now describe CPRF for the class of NC^1 constraints. We consider the representation of an NC^1 circuit C with input size $n = \mathsf{poly}(\lambda)$ and depth $d = \mathcal{O}(\log n)$ to be a bit string $(C_1, \ldots, C_z) \in \{0,1\}^z$, where $z = \mathsf{poly}(n)$ is the description size. Also, we denote the universal circuit by $U(\cdot,\cdot)$ that on input a circuit $C \in \{0,1\}^z$ and $x = (x_1, \ldots, x_n) \in \{0,1\}^n$, outputs $U(C,x) = C(x)$. Due to the work of Cooks and Hoover [CH85], we know that there exists a universal circuit that correctly computes any NC^1 circuit and is itself an NC^1 circuit.

The strategy for our construction is similar as for inner-product. We aim to obtain substractive shares $U(C,x) \cdot F_k(x)$ via the (standard and constrained) evaluation algorithms, where F is a pseudorandom function with evaluation in NC^1, C denotes the constraint, and U denotes the above universal circuit.

A crucial point is that the master secret key should allow to compute such a share for any input x independently of the constraint C. Hence, we have to find a way to replace the encoding of C that is given to the evaluator by oblivious values that guarantee the correctness. In the inner-product case, where we want shares of $\langle \mathbf{x}, \mathbf{z} \rangle \cdot F_k(\mathbf{x})$, we used simulated memory values as the independent share of the undetermined constraint $\mathbf{z}$, and programmed the constrained key to guarantee correctness according to the constraint vector $\mathbf{z}$. However, this technique cannot be applied to the case of NC^1 constraints as we are dealing with non-linear evaluations.

The idea is again to use staged-HSS. We first compute a memory for $U(C,x)$ using $\overline{\mathsf{Eval}}_0$ and $\overline{\mathsf{Eval}}_1$. Then, this memory value is used in the $\mathsf{ExtEval}$ algorithm from Lemma 3.3 to compute a share of $U(C,x) \cdot F_k(x)$ additionally using an encoding of k.

The important point here, is that inputs of $\overline{\mathsf{Eval}}_0$ can be sampled obliviously using $(\bar{\mathsf{I}}_0, \mathsf{aux}) \leftarrow \overline{\mathsf{Input}}_0(\mathsf{pk})$, and therefore can be sampled during Setup without the knowledge of the constraint C. Yet, when computing the constrained key for C, the master key owner can use the full knowledge of C as well as auxiliary information generated during Setup to appropriately compute memory values for the i-th bit C_i of the description of C, using $\bar{\mathsf{I}}_1 \leftarrow \overline{\mathsf{Input}}_1(\mathsf{pk}, C_i, \mathsf{aux}, (\mathsf{ek}_0, \mathsf{ek}_1))$. The correctness of staged-HSS then guarantees the correctness of evaluations, while its security plays a role in the security proof to remove the need for both evaluation keys when computing $\bar{\mathsf{I}}_1$, therefore allowing to rely on HSS security to remove the information about the underlying PRF key k.

We now detail our construction. For any $x \in \{0,1\}^n$, we denote by $U(\cdot, x)$ the circuit that maps $C \in \{0,1\}^z$ to $U(C,x) = C(x) \in \{0,1\}$.

Construction 3.2: CPRF for NC^1 from HSS

Requirements and notation:

- Let $F : \mathcal{K} \times \{0,1\}^n \to \mathcal{Y}$ be a pseudorandom function with evaluation in NC^1, where $\mathcal{Y}$ is a finite cyclic group. For $\mathbf{x} \in \mathcal{R}^n$, we denote by $F_\bullet(\mathbf{x}) : \mathcal{K} \to \mathcal{Y}$ the function that maps $k \in \mathcal{K}$ to $F_k(\mathbf{x})$.
- Let $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{MemGen}, \mathsf{Input}, \mathsf{Eval})$ be a staged homomorphic secret sharing scheme and denote by $(\overline{\mathsf{Input}}_0, \overline{\mathsf{Input}}_1)$, and $(\overline{\mathsf{Eval}}_0, \overline{\mathsf{Eval}}_1)$ the additional algorithms defined in Definition 3.3.
- Let ExtEval be the modified evaluation algorithm as in Lemma 3.3.

Algorithms:

▶ $\mathsf{KeyGen}(1^\lambda)$:

- Run $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$.
- Choose a random key $k \xleftarrow{\$} \mathcal{K}$ for F and compute $(\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{Input}(\mathsf{pk}, k)$.
- For $i \in \{1, \ldots, z\}$, compute $(\bar{\mathsf{I}}_0^{(i)}, \mathsf{aux}^{(i)}) \leftarrow \overline{\mathsf{Input}}_0(\mathsf{pk})$.
- Output $\mathsf{pp} = \mathsf{pk}$, and $\mathsf{msk} = ((\mathsf{ek}_0, \mathsf{ek}_1, \mathsf{I}_0, \mathsf{I}_1), (\bar{\mathsf{I}}_0^{(1)}, \mathsf{aux}^{(1)}, \ldots, \bar{\mathsf{I}}_0^{(z)}, \mathsf{aux}^{(z)}))$.

▶ $\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x)$:

- Parse $\mathsf{pp} = \mathsf{pk}$, and $\mathsf{msk} = ((\mathsf{ek}_0, \mathsf{ek}_1, \mathsf{I}_0, \mathsf{I}_1), (\bar{\mathsf{I}}_0^{(1)}, \mathsf{aux}^{(1)}, \ldots, \bar{\mathsf{I}}_0^{(z)}, \mathsf{aux}^{(z)}))$.
- Run $\mathsf{M}_0 \leftarrow \overline{\mathsf{Eval}}_0(\mathsf{ek}_0, (\bar{\mathsf{I}}_0^{(1)}, \ldots, \bar{\mathsf{I}}_0^{(z)}), U(\cdot, x))$. Here, $\bar{\mathsf{I}}_0^{(i)}$ represents the input value of C_i for $i \in \{1, \ldots, z\}$.
- Run $y_0 \leftarrow \mathsf{ExtEval}(0, \mathsf{ek}_0, \mathsf{M}_0, \mathsf{I}_0, F_\bullet(x))$. Here, M_0 denotes the memory value of $U(C, x)$, and I_0 denotes the input value of k.
- Output y_0.

▶ $\mathsf{Constrain}(\mathsf{msk}, C)$:

- Parse $\mathsf{msk} = ((\mathsf{ek}_0, \mathsf{ek}_1, \mathsf{I}_0, \mathsf{I}_1), (\bar{\mathsf{I}}_0^{(1)}, \mathsf{aux}^{(1)}, \ldots, \bar{\mathsf{I}}_0^{(z)}, \mathsf{aux}^{(z)}))$, and $C = (C_1, \ldots, C_z) \in \{0,1\}^z$.
- For $i \in \{1, \ldots, z\}$, run $\bar{\mathsf{I}}_1^{(i)} \leftarrow \overline{\mathsf{Input}}_1(\mathsf{pk}, C_i, \mathsf{aux}^{(i)}, (\mathsf{ek}_0, \mathsf{ek}_1))$.
- Output $\mathsf{ck}_C = (\mathsf{ek}_1, \mathsf{I}_1, (\bar{\mathsf{I}}_1^{(1)}, \ldots, \bar{\mathsf{I}}_1^{(z)}), C)$.

▶ $\mathsf{CEval}(\mathsf{pp}, \mathsf{ck}_C, x)$:

- Parse $\mathsf{ck}_C = (\mathsf{ek}_1, \mathsf{I}_1, (\bar{\mathsf{I}}_1^{(1)}, \ldots, \bar{\mathsf{I}}_1^{(z)}), C)$.
- Run $\mathsf{M}_1 \leftarrow \overline{\mathsf{Eval}}_1(\mathsf{ek}_1, (\bar{\mathsf{I}}_1^{(1)}, \ldots, \bar{\mathsf{I}}_1^{(z)}), (C^{(1)}, \ldots, C^{(z)}), U(\cdot, x))$.
- Run $y_1 \leftarrow \mathsf{ExtEval}(1, \mathsf{ek}_1, \mathsf{M}_1, \mathsf{I}_1, F_\bullet(x))$.
- Output y_1.

Theorem 3.7 (CPRF for NC^1 from Staged HSS). *Assuming F is a secure pseudorandom function with evaluation in NC^1 and HSS is a secure staged-HSS scheme, Construction 3.2 is a selective 1-key secure constrained pseudorandom function for NC^1.*

Proof. We now prove correctness and pseudorandomness of the construction.
Correctness. The proof of correctness is roughly the same as for Construction 3.1 and directly follows from correctness properties of the underlying staged HSS scheme. The output of $\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x)$ and $\mathsf{CEval}(\mathsf{pp}, \mathsf{ck}_C, x)$ on an input x form substractive shares of $U(C, x) \cdot F_k(x) = C(x) \cdot F_k(x)$. When $C(x) = 0$, correctness of the staged HSS scheme guarantees that the outputs of evaluation and constrained evaluation algorithms form substractive shares of 0, thus they are equal.
Pseudorandomness. Here again, the proof follows a similar strategy as in the case of inner-product constraints. The goal is to remove the dependency to k the underlying PRF key in the constrained key such that the term $C(x^*) \cdot F_k(x^*)$ makes the challenge pseudorandom (since x^* is required to satisfy $C(x^*) = 1$). We proceed via a sequence of hybrid games.
Hybrid $\mathcal{H}_0$: This is the standard CPRF security game where the challenge is answered with the real PRF evaluation.
Hybrid $\mathcal{H}_1$: In this first hybrid game, we only change the definition of the evaluation oracle. Since we are in the selective setting, the challenger knows the constraint C before answering any evaluation query. Therefore, it can compute the constrained key ck_C from the start. When asked for an evaluation query on input x, the challenger now replies to it by computing $y_1 \leftarrow \mathsf{CEval}(\mathsf{pp}, \mathsf{ck}_C, x)$ and returning $y_1 + C(x) \cdot F_k(x)$. The adversary's view remains identical to its view in $\mathcal{H}_0$ by the correctness of HSS, therefore these two hybrid games are perfectly indistinguishable.
Hybrid $\mathcal{H}_2$: In this second hybrid game, instead of sampling the constrained key elements $\bar{\mathsf{I}}_1^{(i)}$ as $\bar{\mathsf{I}}_1^{(i)} \leftarrow \overline{\mathsf{Input}}_1(\mathsf{pk}, C_i, \mathsf{aux}^{(i)}, (\mathsf{ek}_0, \mathsf{ek}_1))$ for $i \in [z]$, we replace each of these values by $\mathsf{Input}(\mathsf{pk}, C_i)$. Computational indistinguishability between these two hybrid games follows from the staged-security of HSS.
Hybrid $\mathcal{H}_3$: We now remove the information about k in the constrained evaluation key as follows: instead of defining $(\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{Input}(\mathsf{pk}, k)$, we set $(\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{Input}(\mathsf{pk}, 0)$.[1] The PRF key $k \xleftarrow{\$} \mathcal{K}$ is still sampled by the challenger, and evaluation (and challenge) queries are still answered on input by having the challenger computing $y_1 \leftarrow \mathsf{CEval}(\mathsf{pp}, \mathsf{ck}_C, x)$ and returning $y_1 + C(x) \cdot F_k(x)$.
By HSS security (for $\sigma = 1$) and correctness of the HSS evaluation, we claim that $\mathcal{H}_2$ and $\mathcal{H}_3$ are computationally indistinguishable. Suppose that an adversary $\mathcal{A}$ could distinguish between these two hybrids, we construct an adversary $\mathcal{B}$ against HSS security as follows: $\mathcal{B}$ first samples a key $k \xleftarrow{\$} \mathcal{K}$ and submits message $(k, 0)$ to its HSS challenger. It gets back $(\mathsf{pk}, \mathsf{ek}_1, \mathsf{I}_1)$ where I_1 is the right-hand part of $\mathsf{Input}(k)$ or $\mathsf{Input}(0)$. Then, $\mathcal{B}$ computes the constrained key ck_C as described in $\mathcal{H}_2$. It answers $\mathcal{A}$'s evaluation (and challenge) queries x by computing $y_1 \leftarrow \mathsf{CEval}(\mathsf{pp}, \mathsf{ck}_C, x)$ and returning $y_1 + C(x) \cdot F_k(x)$. When $\mathcal{A}$ halts with some output b', so does $\mathcal{B}$. It is clear that $\mathcal{B}$ simulates either $\mathcal{H}_2$ or $\mathcal{H}_3$, depending on whether it was given an encoding of k or of 0, which results in our claim.
Hybrid $\mathcal{H}_4$: In this hybrid, the challenger now replies to the challenge query x^* by returning a uniformly random value of $\mathcal{Y}$. Since the adversary's view does no longer contain any information about the PRF key k, then $F_k(x^*)$ can be replaced by a random value of $\mathcal{Y}$. Also, since x^* must satisfy $C(x^*) = 1$, then $y_1 + C(x) \cdot F_k(x) = y_1 + F_k(x)$,

[1] By 0 we mean any fixed key, e.g. 0^λ if $\mathcal{K} = \{0,1\}^\lambda$.

which is computationally indistinguishable from a random element of $\mathcal{Y}$. Therefore, hybrids $\mathcal{H}_3$ and $\mathcal{H}_4$ are computationally indistinguishable thanks to the security of the underlying PRF. Note that here that we only rely on Find-then-Guess security for the underlying PRF.
The rest of the proof proceeds by reversing the sequence of hybrid games while leaving the challenge query answered by a uniformly random value. □

Remark 3.2. *We note that the above construction is not constraint-hiding, since the constrained evaluation algorithm relies on the knowledge of the constraint.*

Combining Theorem 3.7 (CPRF for NC^1 from staged-HSS) with Theorem 3.5 (staged-HSS from DCR), yields the following corollary.

Corollary 3.3 (CPRF for NC^1 from DCR)**.** *Assuming the* DCR *assumption holds, there exist 1-key selectively-secure constrained pseudorandom functions for* NC^1 *constraints.*

Remark 3.3 (Other Instantiations)**.** *Although not explicitly detailed in this work, our transformations from HSS to CPRF works using either of the schemes from [BKS19] based on the Learning With Errors (LWE) assumption with super-polynomial modulus, from [ADOS22] based on the hardness of Joye-Libert encryption scheme, from [ADOS22] based on the Decisional Diffie-Hellman (DDH) and Decisional Cross-Group Diffie-Hellman (DXDH) assumptions over class groups, or from [CLT22] based on the Hard Subgroup Membership (HSM) assumption over class groups. All of the above HSS schemes follow the same outline as the DCR-based scheme of [OSY21] when generating input and memory values. More precisely, input values are ciphertexts computed using a PKE scheme, and in all of the mentioned schemes, the used encryption tool generates ciphertexts that contain a separate part as a commitment to the encryption randomness which is independent of the underlying plaintext. This feature makes it feasible to generalize these schemes into staged-HSS schemes and then use it to construct CPRF for* NC^1 *constraints. These schemes also allow simulation of memory values which enables using the scheme to construct CPRF for inner-product constraints. This holds since a valid memory value of these schemes is a subtractive share of a secret vector dependent on the secret key of the used PKE, thus one share can be sampled obliviously and the other one can be correctly computed given the secret vector.*
Also, using HSS with only polynomial correctness (e.g., the DDH-based scheme of [BGI16]) still yields CPRFs for polynomial-size *domain. This leads to constructions of* poly-size domain *private CPRFs for inner-products and CPRFs for* NC^1 *from DDH, and from LWE with polynomial modulus-to-noise ratio.*
However, we note that instantiating our transformation for domains larger than polynomail from HSS schemes with polynomial correctness, leads not only to incorrect CPRFs, but to insecure ones! This is because the security proofs of our transformations rely on the fact that the two outputs of the underlying HSS scheme form correct subtractive shares of the program $C(x) \cdot F_k(x)$ *with overwhelming probability (see Hybrid* $\mathcal{H}_1$ *of the proofs of Theorems 3.6 and 3.7).*

Chapter 4

Public-Key Pseudorandom Correlation Functions from Constrained Pseudorandom Functions

Contents

The content of this section is based on the results presented in [BCM+24].

4.1 Chapter Overview

In this chapter, we draw connections between constrained pseudorandom functions and pseudorandom correlation functions (PCFs). More precisely, we show how constrained

PRFs can be used together with low-complexity weak PRFs to obtain pseudorandom correlation functions. Moreover, we show how this transformation yields efficient *public-key* PCFs, when the underlying constrained PRF is instantiated by modifying the PRF construction of Naor and Reingold [NR97].
Briefly recalling our discussing in the introduction chapter, the results of this chapter are obtained via three steps: (1) we first show that the Naor-Reingold PRF can be transformed into a CPRF for the class of inner-product membership (IPM) predicates. We observe that the class of IPM predicates contains many predicates of interest, including some weak PRF candidates. Therefore, we obtain CPRFs that admit weak PRFs as constraints, which we refer to as *pseudorandomly constrained PRFs*. Next, (2) we show that pseudorandomly constrained PRFs can be transformed into PCFs for oblivious transfer (OT) correlations. Combining this result with the previous step, we obtain a PCF for OT from the Naor-Reingold PRF. Finally, (3) we show that the PCF instantiation obtained in the previous step, can be upgraded to a *public-key* PCF, where the PCF evaluation keys can be generated non-interactively.
In the following, we provide a brief technical overview of each step.

4.1.1 Naor-Reingold PRF ⇒ Pseudorandomly Constrained PRF

Let us first recall the Naor-Reingold PRF [NR97] with binary input domain $\mathcal{X} = \{0,1\}^n$. This construction works over a cyclic group $\mathbb{G}$ of prime order p.

- $\mathsf{NR.KeyGen}(1^\lambda)$: Sample $g \xleftarrow{\$} \mathbb{G}$ and $\boldsymbol{a} = (a_1, a_2, \cdots, a_n) \xleftarrow{\$} (\mathbb{Z}_p^*)^n$.
  Output $\mathsf{msk} := (g, \boldsymbol{a})$.

- $\mathsf{NR.Eval}(\mathsf{msk}, \boldsymbol{x})$: On input $\boldsymbol{x} = (x_1, \cdots, x_n) \in \{0,1\}^n$, output $y = g^{\prod_{i=1}^{n} a_i^{x_i}}$.

Evaluating the Naor-Reingold PRF requires a few multiplications, followed by a single exponentiation and it falls in the complexity class NC^1. The security of this PRF follows from the Decisional Diffie-Hellman assumption over the group $\mathbb{G}$.
We first show how we can generate constrained keys for inner-product predicates:
Naor-Reingold Constrained PRF for Inner-Product. Let us recall the class of inner-product constraints with space input a set $\mathcal{R}$. This class is defined as $\mathsf{IP} = \{C_{\mathbf{z}} \mid \mathbf{z} \in \mathcal{R}^n\}$, where the circuit $C_{\mathbf{z}} : \mathcal{R}^n \to \{0,1\}$ is defined as $C_{\mathbf{z}}(\mathbf{x}) = 0$ if $\langle \mathbf{z}, \mathbf{x} \rangle = 0$, and $C_{\mathbf{z}}(\mathbf{x}) = 1$, otherwise.
The following two algorithms allow generating a constrained key $\mathsf{ck}_{\mathbf{z}}$ for a vector $\mathbf{z}$ which enables computing the output of the Naor-Reingold PRF on inputs $\mathbf{x} \in \mathcal{R}^n$ if and only if $\langle \mathbf{z}, \mathbf{x} \rangle = 0$.

- $\mathsf{NR.Constrain}(\mathsf{msk}, \mathbf{z})$: Sample $r \xleftarrow{\$} \mathbb{Z}_p^*$, and define $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_n)$, where $\alpha_i = r^{-z_i} \cdot a_i$ for $i \in [n]$. Output $\mathsf{ck}_{\mathbf{z}} = (g, \boldsymbol{\alpha})$.

- $\mathsf{NR.CEval}(\mathsf{ck}_{\mathbf{z}}, \mathbf{x})$: On input $\mathbf{x} = (x_1, \cdots, x_n) \in \{0,1\}^n$, output $y = g^{\prod_{i=1}^{n} \alpha_i^{x_i}}$.

Here, each element α_i of a constrained key is obtained by randomizing each master secret key element a_i using a term r^{-z_i}. The outputs of the Eval and CEval algorithms coincide when the blinding terms cancel out which happens exactly when it holds that $\langle \mathbf{x}, \mathbf{z} \rangle = 0 \pmod{\mathsf{ord}(r)}$, where $\mathsf{ord}(r)$ is the order of r in $\mathbb{G}$. For a safe prime $p = 2q + 1$, the order

of r is q or $2q$ with overwhelming probability. Therefore, when $q \gg n$, $\langle \mathbf{x}, \mathbf{z} \rangle = 0 \bmod q$ iff $\langle \mathbf{x}, \mathbf{z} \rangle = 0$ over the integers.
Regarding the security, when the adversary doesn't have access to the evaluation oracle, the pseudorandomness of the output of the above function on a challenge input $\mathbf{x}$, where $\langle \mathbf{x}, \mathbf{z} \rangle \neq 0$, holds as long as $g^{r^{\langle \mathbf{x}, \mathbf{z} \rangle}}$ looks random for a uniformly random $r \in \mathbb{Z}_p^*$. This requirement is satisfied unconditionally since the constrained key reveals no information about r as each a_i is uniformly random in $\mathbb{Z}_p^*$ and hides r.
Extending this idea, we can generate constrained keys for inner-product membership predicates as follows:
Naor-Reingold Constrained PRF for Inner-Product Membership. We define the class of inner-product membership predicates as $\mathsf{IPM} = \{C_{\mathbf{z}}^S \mid \mathbf{z} \in \mathcal{R}^n, S \subseteq \mathcal{I}\}$, for some sets $\mathcal{R}$ and $\mathcal{I}$, and $n > 0$, where $C_{\mathbf{z}}^S : \mathcal{R}^n \times 2^{\mathcal{I}} \to \{0,1\}$ is defined as $C_{\mathbf{z}}^S(\mathbf{x}) = 0$ iff $\langle \mathbf{z}, \mathbf{x} \rangle \in S$.
For this class, we define the Constrain and CEval algorithms as follows:

- $\mathsf{NR.Constrain}(\mathsf{msk}, \mathbf{z}, S)$: Sample $r \xleftarrow{\$} \mathbb{Z}_p^*$. For each $s \in S$, define $g_s := g^{r^s}$, and let $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_n)$, where $\alpha_i = r^{-z_i} \cdot a_i$, for $i \in [n]$. Output $\mathsf{ck} = (\mathbf{z}, (g_s)_{s \in S}, \boldsymbol{\alpha})$.
- $\mathsf{NR.CEval}(\mathsf{ck}, \mathbf{x})$: On input $\mathbf{x} = (x_1, \cdots, x_n) \in \{0,1\}^n$, output $y = g_s^{\prod_{i=1}^{n} \alpha_i^{x_i}}$, where $s = \langle \mathbf{x}, \mathbf{z} \rangle$.

The correctness follows from inspection and is similar to the correctness of the construction for inner-product predicates.
Regarding the security, our prior claim that the constrained key contains no information about r no longer holds since it now contains extra elements g^{r^s} for all $s \in S$, and therefore the no-evaluation security is no longer unconditional. We show that this construction is (no-evaluation) secure under a variant of the Diffie-Hellman assumption which we call *sparse power-DDH* assumption. The sparse power-DDH assumption states that for a subset $S \subseteq [\ell]$, where $\ell \in \mathbb{N}$ is polynomially-bounded, given g^{r^s} for various $s \in S$, it is infeasible to distinguish g^{r^s} for $s \in [\ell] \setminus S$ from uniformly random group elements. This assumption is a static falsifiable assumption, and it can be viewed as a natural generalization of the power-DDH assumption (which states that given g^{r^i} for $i \in \{1, \ldots, n\}$, it is infeasible to distinguish $g^{r^{n+1}}$ from random).
From no-evaluation security to full security. As discussed in [AMN+18], any no-evaluation secure CPRF can be turned into an adaptively secure CPRF (with any number of evaluation queries) in the random oracle model by hashing the output. Hence, proving the no-evaluation security suffices for our purpose.

On IPM predicates. The class of IPM predicates captures several predicates of interest. As already explained, it contains inner-product equality. We also note that since the range of inputs and constraints is polynomially bounded, the IPM-constrained Naor-Reingold also admits inner-product *inequality* predicates. Moreover, this class captures puncturing, which, to our knowledge, yields the first candidate puncturable pseudorandom function in the complexity class NC^1 (assuming that the hash function used to re-randomize the output is in NC^1).[1] Since puncturable PRFs have independent

[1] It is not too hard to build a puncturable PRF in NC^1 by following the blueprint of the GGM PRF [GGM84b] but using a λ-ary tree instead of a binary tree and instantiating the PRG with an NC^0 PRG with polynomial stretch. However, such constructions are inherently limited to superpolynomial-size domains, while our construction can handle subexponential-size domains.

applications, for instance in the context of indistinguishability obfuscation, we expect that this result could have other applications. Furthermore, the class of IPM predicates capture several variants of puncturing, such as puncturing a Hamming ball.
Most importantly for our work, this class also captures several candidate weak pseudorandom functions from the literature such as [BPR12, BIP+18, AR16]. We refer to these weak PRFs as IPM-wPRFs. We explain the details of these IPM-wPRFs in Section 4.2.3.3.

To draw a conclusion, the existence of IPM-wPRFs implies that the constrained Naor-Reingold PRF for the class of IPM predicates admits certain weak PRFs as constraints. This property, seemingly inappreciable, forms the basis of our transformation from CPRFs to PCFs for oblivious transfer correlations. Recognizing its importance, we refer to a CPRF that admits a (weak) pseudorandom predicate as a *(weakly) pseudorandomly constrained PRF*. If a CPRF admits both a pseudorandom predicate and its complement, we call it a *full-domain* pseudorandomly constrained PRF. The concept of full-domain pseudorandomly constrained PRFs is used in the transformation explained in the following part.

4.1.2 Pseudorandomly Constrained PRF ⇒ PCF for OT

Building on last part's conclusion, here we show how to construct PCFs for OT correlations from full-domain pseudorandomly constrained PRFs.
Let $F = (F.\mathsf{KeyGen}, F.\mathsf{Eval})$ be a (weak) pseudorandom predicate with key space $\mathcal{K}$. For a key $k \in \mathcal{K}$, let $F_k : x \mapsto F.\mathsf{Eval}(k, x)$. Also, let $\mathsf{CPRF} = (\mathsf{CPRF.KeyGen}, \mathsf{CPRF.Eval}, \mathsf{CPRF.Constrain}, \mathsf{CPRF.CEval})$ denote a full-domain pseudorandomly constrained PRF, meaning that it supports the class $\mathcal{F} = \{F_k\}_{k\in\mathcal{K}} \cup \{1 - F_k\}_{k\in\mathcal{K}}$ as constraints, i.e., all predicates "$F.\mathsf{Eval}(K, x)$ evaluates to b" for $b \in \{0, 1\}$ and $k \in \mathcal{K}$. Then, we construct a pseudorandom correlation function for oblivious transfer correlations as follows:

- $\mathsf{PCF.Gen}(1^\lambda)$:
  - Sender's key: Run $\mathsf{CPRF.KeyGen}(1^\lambda)$ to sample two independent master secret keys $\mathsf{msk}_0, \mathsf{msk}_1$. Set $k_0 = (\mathsf{msk}_0, \mathsf{msk}_1)$.
  - Receiver's key: Sample $k \xleftarrow{\$} \mathcal{K}$, and two constrained keys, ck_0 that is msk_0 constrained at "$F_k(x) = 0$", and ck_1 that is msk_1 constrained at "$F_k(x) = 1$". Set $k_1 = (k, \mathsf{ck}_0, \mathsf{ck}_1)$.
- $\mathsf{PCF.Eval}(0, k_0, x)$: On an input x, the sender evaluates the CPRF on x using both msk_0 and msk_1 to obtain two pseudorandom outputs (y_0, y_1).
- $\mathsf{PCF.Eval}(1, k_1, x)$: On an input x, the receiver computes $b \leftarrow F_k(x)$, and sets $y_b \leftarrow \mathsf{CPRF.CEval}(\mathsf{ck}_b, x)$. It then outputs (b, y_b).

In the above construction, the output pairs of the evaluation algorithm on a common input x are OT correlated. This is because for any input x, the predicate $F_k(x) = b$ is satisfied for some $b \in \{0, 1\}$, hence evaluating the CPRF using the constrained key ck_b yields the correct output y_b by the correctness of the CPRF. Regarding the security, the bit b looks random to the sender, by the security of the (weak) PRF F, since the key k

is hidden from the sender.[1] On the other hand, the sender security holds since for each input x, the predicate $F_k(x) = 1 - b$ is not satisfied, thus the constrained key ck_{1-b} fails to compute y_{1-b} and this element remains pseudorandom to the receiver.
Furthermore, the above PCF is *precomputable*, as recently defined in [CMPR23], as a property that allows one of the parties to locally generate its own PCF key and compute its correlated randomness entirely, before even knowing the identity of the other party. In the above construction, the sender can compute all pairs (y_0, y_1) prior to knowing the receiver, and it is therefore precomputable.
To conclude, plugging in our IPM-constrained Naor-Reingold to instantiate the full-domain pseudorandomly constrained PRF in above transformation yields a PCF for OT correlations from the sparse power-DDH assumption.

Optimizations

When we plug in our IPM-constrained Naor-Reingold construction to instantiate the transformation presented above, several optimizations can be applied in order to reduce the size of both sender and receiver keys. These optimizations are explained in details in our paper [BCM+24], and here, we provide a brief overview of them.
Note that when using the Naor-Reingold CPRF in the transformation, the sender key k_0 and the receiver key k_1 are as follows:

$$k_0 = (\mathsf{msk}_0 = (g, \boldsymbol{a}), \mathsf{msk}_1 = (g', \boldsymbol{a}')),$$

$$k_1 = (k, \mathsf{ck}_0 = (k, (g_s)_{s\in S}, \boldsymbol{\alpha}), \mathsf{ck}_1 = (k, (g'_s)_{s\notin S}, \boldsymbol{\alpha}')),$$

where $\boldsymbol{a} = (a_1, \ldots, a_n)$, $g_s = g^{r^s}$ for $s \in S$, and $\boldsymbol{\alpha} = (a_1 \cdot r^{-k_1}, \ldots, a_n \cdot r^{-k_n})$. Similarly, we have $\boldsymbol{a}' = (a'_1, \ldots, a'_n)$, $g_s = g^{r'^s}$ for $s \notin S$, and $\boldsymbol{\alpha} = (a'_1 \cdot r'^{-k_1}, \ldots, a'_n \cdot r'^{-k_n})$.

Halving the key size. This optimization builds on the random self-reducibility property of DDH (explained in Section 2.1). Due to this property, the two master secret keys in the sender's key k_0 can have common elements $(a_1, \cdots, a_n)$ provided that they use different bases g and g'. For the same reason, we can also set $r = r'$ without any security loss. This reduces the sender key size by a factor two, and compresses the receiver key size as well. Concretely, we have:

$$k_0 = (g, g', \boldsymbol{a} = (a_1, \ldots, a_n)),$$

$$k_1 = (\boldsymbol{\alpha} = (r^{-k_1} \cdot a_1, \ldots, r^{-k_n} \cdot a_n), (g^{r^s})_{s\in S}, (g'^{r^s})_{s\notin S})$$

Reusing the g_s elements. We note that the set S attributed to the weak PRF used in the transformation is public and depends only on the definition of the (weak) PRF used by the receiver. In a multiparty setting where the sender wants to compute PCF keys with multiple receivers, we can exploit this observation to define the g_t's once for all, and publish them as public parameters (or sender's public key) to be used by all receivers. This requires adding two additional terms $(a_{0,j}, a'_{0,j})$ in the sender key for each receiver j, to re-randomize the bases g, g'. That is, the sender now computes its pseudorandom OT messages with each receiver j as

$$y_{0,j}^{(x)} \leftarrow g^{a_{0,j} \cdot \prod_{i=1}^n a_i^{x_i}} \qquad\qquad y_{1,j}^{(x)} \leftarrow g'^{a'_{0,j} \cdot \prod_{i=1}^n a_i^{x_i}},$$

[1] We note that the common PCF security notion in the literature, requires the security to hold for *random* inputs. Therefore, in our transformation from CPRF to PCF for OT, it suffices to consider a *weak* PRF to generate the bit b.

Compressing the keys. When instantiating the group with a suitable elliptic curve, the size of each element a_i is 2λ bits in order to achieve λ bits of security against generic discrete log attacks. To further reduce the key size, these elements can be generated by applying a pseudorandom generator on a λ-bit seed. This optimization is explained in details in Section 4.2.2 and results in having a sender key of size only λ bits as well as reducing the receiver key size by 25%.

Exploiting the structure in the set S. This optimization builds on a special structure of the set S that is satisfied by some of the weak PRFs of the literature (in particular that of [BIP+18]). We refer to this structure as S being *anti-periodic*. More precisely, we say that S is m-antiperiodic when we *almost* have:

$$s \notin S \iff (s - m) \in S \ ,$$

where the *almost* stems from the fact that the equivalence breaks down at the extremities. In this case, we can rewrite the constraint $\langle \mathbf{x}, \mathbf{z} \rangle \notin S$ as "$\langle \mathbf{x}, \mathbf{z} \rangle - m \in S$". We can therefore consider only one master secret key msk for the sender by simply adding another exponent a_{n+1} to act as a *shift*. The keys therefore become:

$$k_0 = (g, a_1, \cdots, a_n, a_{n+1})$$

$$k_1 = \mathsf{ck} = (g, r^{-k_1} \cdot a_1, \cdots, r^{-k_n} \cdot a_n, r^m \cdot a_{n+1}, (g_s)_{s \in S'}),$$

On an input $\mathbf{x}$, the sender computes the two OT outputs as $y_b \leftarrow \mathsf{CPRF.Eval}(\mathsf{msk}, \mathbf{x}|b)$ for $b \in \{0, 1\}$. More precisely, we have:

$$y_b \leftarrow g^{\prod_{i=1}^{n} a_i^{x_i} \cdot a_{n+1}^b} \text{ for } b \in \{0, 1\}.$$

Thanks to the term $r^m \cdot a_{n+1}$ in the receiver key, for every input $\mathbf{x}$, there is only a single $b \in \{0, 1\}$ such that $\langle \mathbf{x}|b, \mathbf{z}| - m \rangle \in S$, i.e. such that $\langle x, z \rangle - b \cdot m \in S$. Compared to the previous construction, this (almost) halves the number of group elements g_t in the receiver key, going from $m \cdot R$ to $(m/2) \cdot (R + 1)$.
The BIPSW wPRF and the XOR-MAJ wPRF satisfy this property: the set S is m-antiperiodic with $m = 6$ for BIPSW, and $m = 49$ for our parameter choice with XOR-MAJ. Hence, this optimization can also be applied to these two instantiations.

We note that none of the optimizations above change the underlying security assumption, and the resulting construction is secure under the same assumption as the base construction.

4.1.3 Public-Key PCF for OT from Constrained Naor-Reingold

In this part we show that the PCF construction that is obtained from the IPM-constrained Naor-Reingold PRF can be transformed into a public-key PCF, i.e., the evaluation keys can be generated locally and non-interactively by each party. More precisely in a public-key PCF, the central $\mathsf{PCF.Gen}$ algorithm is replaced by two algorithms that are locally run by each party $b \in \{0, 1\}$: $\mathsf{PCF.Gen}(1^\lambda, b)$ that outputs $(\mathsf{sk}_b, \mathsf{pk}_b)$, and $\mathsf{PCF.KeyDer}(b, \mathsf{sk}_b, \mathsf{pk}_{1-b})$ that uses the other party's public-key to derive an evaluation key k_b.
The evaluation keys in this construction are as follows:

$$k_0 = (\mathsf{msk}_0 = (g, \boldsymbol{a}), \mathsf{msk}_1 = (g', \boldsymbol{a}')),$$

$$k_1 = (k, \mathsf{ck}_0 = (k, (g_s)_{s\in S}, \boldsymbol{\alpha}), \mathsf{ck}_1 = (k, (g'_s)_{s\in S}, \boldsymbol{\alpha}')),$$

where $\boldsymbol{a} = (a_1, \ldots, a_n)$, $g_s = g^{r^s}$ for $s \in S$, and $\boldsymbol{\alpha} = (a_1 \cdot r^{-k_1}, \ldots, a_n \cdot r^{-k_n})$, and similarly for $\boldsymbol{a}'$, $(g'_s)_{s\in S}$, and $\boldsymbol{\alpha}$.

A First Try via Bellare-Micali Non-Interactive OT: Focusing on the more correlated elements of the two keys, we have that $\alpha_i = a_i \cdot r^{-k_i}$ for each $i \in [n]$. These values can be viewed as multiplicative shares over $\mathbb{Z}_p^*$ of r^{-k_i} (up to inverting a_i locally). Therefore, assuming DDH over a suitable subgroup $\mathbb{G}'$ of $\mathbb{Z}_p^*$, such multiplicative shares can be directly obtained via the Bellare-Micali protocol.
We explain this construction in Section 4.4.2. Yet, this solution has two major downsides: Firstly, we cannot set $\mathbb{Z}_p^* = \mathbb{G}'$, since DDH over $\mathbb{Z}_p^*$ can be broken by computing the Legendre symbol. However, assuming that $p = 2q + 1$ is a safe prime (q is prime), we can set $\mathbb{G}'$ to be the subgroup QR_p of quadratic residues modulo p, where DDH is widely conjectured to hold for a sufficiently large p. This does not harm the security of the CPRF but changes slightly the underlying sparse power-DDH variant. A second, more troublesome, downside is the size of the modulus p: due to subexponential-time algorithms for discrete logarithm over finite fields, p should be taken much larger than 256 bits, at the very least 1024 bits. But in turn, this implies that the group $\mathbb{G}$ over which we instantiate our PCF should have order $p \geq 2^{1024}$, which considerably harms efficiency in terms of both the key size and the computation time, and prevents us in particular to rely on efficient 256-bit elliptic curves. We circumvent this issue by setting p to a smaller value, e.g., a 256-bit prime, and relying on *Paillier encryption*.

A More Optimized Solution via Paillier Encryption: This time we work over a Paillier group. We first recall the setting. Recall that the Naor-Reingold construction works over a group $\mathbb{G}$ of prime order $p = 2q+1$, where q is also a prime. Let $N = PQ$ be a Blum integer. The key generation and derivation of our public-key PCF work over the group $\mathbb{Z}_{N^2}^* \approx \mathbb{H} \times \mathbb{NR}_N$, where $\mathbb{H} = \{(1+N)^i : i \in [N]\}$, and $\mathbb{NR}_N = \{x^N : x \in \mathbb{Z}_{N^2}^*\}$. In essence, our solution is achieved through three steps:

1. First the two parties compute multiplicative shares of $(1+N)^{r'k}$, where $r' \xleftarrow{\$} \mathbb{Z}_q$.

   More precisely, let g_q be a generator of QR_p, and G, H two random elements of $\mathbb{NR}_{2N}$. The sender samples $r' \xleftarrow{\$} \mathbb{Z}_q$ and sets $r := g_q^{r'} \pmod p$. It then computes its public key as a Paillier-ElGamal encryption of r', i.e., $\mathsf{pk}_0 = (c_0 = G^t, c_1 = H^t \cdot (1+N)^{r'})$. The receiver holding a bit string $k_1, \ldots, k_n$ computes its public key as Pedersen commitments of each k_i over $\mathbb{NR}_{2N}$, i.e., $\mathsf{pk}_1 = ((\mathsf{com}_i = G^{s_i} \cdot H^{k_i})_{i\in[n]})$, where $s_i \xleftarrow{\$} \mathbb{Z}_N$.
   Knowing each other's public keys, the sender and receiver then respectively compute $A_i = \mathsf{com}_i^t$, and $B_i = c_0^{s_i} \cdot c_1^{k_i}$, for $i \in [n]$. Note that for each index i it holds that $B_i = A_i \cdot (1+N)^{r'k}$.

2. In the second step, the two parties convert their obtained multiplicative shares to subtractive shares modulo N using a distributed discrete log (introduced in [OSY21]), and they immediately convert these shares modulo N to shares modulo q using the fact that subtractive shares modulo N are with very high probability shares over the integers when the shared value is sufficiently smaller than the modulus.

   More precisely, let DDLog_N be the algorithm that on separate inputs g_0 and g_1 such that $g_0 = g_1 \cdot (1+N)^x \pmod{N^2}$, outputs z_0 and z_1, respectively, such that

$z_0 - z_1 = x \pmod N$. It is therefore enough for the sender and the receiver to respectively run $(r' \cdot k)_0 \leftarrow \mathsf{DDLog}_N(A_i)$, and $(r' \cdot k)_1 \leftarrow \mathsf{DDLog}_N(B_i)$. Note that these shares are over $\mathbb{Z}_N$. By setting $q \leq N/2^\lambda$, they also form correct shares over the integers with probability at least $1 - 1/2^\lambda$.

3. Finally the two parties convert their additive shares modulo q to multiplicative shares over $\mathbb{Z}_p^*$ via exponentiation.

   More precisely, the sender and receiver simply compute $a_i \leftarrow g_q^{(r' \cdot k)_0} \pmod p$, and $\alpha_i \leftarrow g_q^{(r' \cdot k)_1} \pmod p$. Note that it now holds that $\alpha_i = a_i \cdot r^{-k_i}$, which was our objective.

Given that the set S attributed with the IPM-wPRF is public, the sender can compute and publish the tuples $(g_s)_{s \in S}$ and $(g'_s)_{s \in S}$ as a part of its public key. The receiver can then use these tuples to complete its evaluation key.

4.2 Constraining the Naor-Reingold PRF

In this section, we first describe how to obtain a constrained PRF in the ROM from the Naor-Reingold PRF for the class of inner-product membership constraints, defined below. Then, we detail several optimizations and provide some simple applications for the resulting CPRF.

4.2.1 Inner Product Membership CPRF from Naor-Reingold

We define the class of inner-product membership (IPM) constraints as $\mathsf{IPM} = \{C_{\mathbf{z}}^S \mid \mathbf{z} \in \mathcal{R}^n, S \subseteq \mathcal{I}\}$, for some sets $\mathcal{R}$ and $\mathcal{I}$, and $n > 0$, where $C_{\mathbf{z}}^S : \mathcal{R}^n \to \{0,1\}$ is defined as $C_{\mathbf{z}}^S(\mathbf{x}) = 0$ iff $\langle \mathbf{z}, \mathbf{x} \rangle \in S$ for an input $\mathbf{x} \in \mathcal{R}^n$. In the following, we first consider binary inputs, i.e., $\mathcal{R} = \{0,1\}$, and later in the section, we explain how we can operate over non-binary inputs, e.g., considering $\mathcal{R} = \{0, 1, \ldots, p-1\}$ for a prime p.
In Construction 4.1, we describe our construction for constraining the Naor-Reingold PRF for the class of inner-product membership constraints.

Construction 4.1: Naor-Reingold CPRF for IPM (Binary Inputs)

Requirements and notation:

- p is a safe prime, i.e., $p = 2q + 1$ for some prime q.
- The input and constraint space is $\{0,1\}^n$.
- The inner-product space is $\mathcal{I} = \{0, 1, \ldots, n\}$.

Algorithms:

▶ $\mathsf{CPRF.KeyGen}(1^\lambda)$:

1. Run $(\mathbb{G}, g, p) \xleftarrow{\$} \mathsf{GenPar}(1^\lambda)$.
2. Sample $\mathbf{a} = (a_0, \ldots, a_n) \xleftarrow{\$} \mathbb{Z}_p^{n+1}$.
3. Output $\mathsf{msk} = \mathbf{a}$ and $\mathsf{pp} = (\mathbb{G}, g, p)$.

▶ $\mathsf{CPRF.Eval}(\mathsf{pp}, \mathsf{msk}, \mathbf{x} \in \{0,1\}^n)$:

1. Parse $\mathsf{pp} = (\mathbb{G}, g, p)$ and $\mathsf{msk} = \mathbf{a}$.
2. Output $y = g^{a_0 \cdot \prod_{i=1}^{n} a_i^{x_i}}$.

▶ $\mathsf{CPRF.Constrain}(\mathsf{pp}, \mathsf{msk}, (\mathbf{z}, S))$:

1. Parse $\mathsf{pp} = (\mathbb{G}, g, p)$, and $\mathsf{msk} = \mathbf{a}$.
2. Sample $r \xleftarrow{\$} \mathbb{Z}_p^*$.
3. For $i \in [n]$, set $\alpha_i := a_i \cdot r^{-z_i}$.
4. Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$.
5. For $s \in S$, compute $g_s := g^{a_0 \cdot r^s}$.
6. Output $\mathsf{ck} = (\boldsymbol{\alpha}, (g_s)_{s \in S}, \mathbf{z})$.

▶ $\mathsf{CPRF.CEval}(\mathsf{pp}, \mathsf{ck}, \mathbf{x} \in \{0,1\}^n)$:

1. Parse $\mathsf{ck} = (\boldsymbol{\alpha}, (g_s)_{s \in S}, \mathbf{z})$.
2. Let $s_{\mathbf{x}} := \langle \mathbf{z}, \mathbf{x} \rangle$.
3. If $\langle \mathbf{z}, \mathbf{x} \rangle \in S$, output $y = (g_{s_{\mathbf{x}}})^{\prod_{i=1}^{n} \alpha_i^{x_i}}$.
4. Otherwise, return $\perp$.

We first show that, our construction is no-evaluation secure under the sparse power-DDH assumption (Definition 2.4).

Theorem 4.1 (No-Evaluation Security)**.** *Assuming the hardness of sparse power-DDH (Definition 2.4), Construction 4.1 is a single-key, no-evaluation secure CPRF for the class of* IPM *constraints.*

Proof. We now prove correctness and no-evaluation security of Construction 4.1.

Correctness. Correctness follows by inspection after replacing each α_i by $r^{z_i} a_i$ in the output of the CEval algorithm.

No Evaluation Pseudorandomness. Let $\mathcal{A}$ denote a 1-key no-evaluation adversary against the pseudorandomness of the above construction. Consider the following sequence of hybrid games:

Hybrid $\mathcal{H}_0$: This is the standard CPRF security game where the challenge query is answered by returning the output of the CPRF evaluation algorithm. Here the view of the adversary is as follows:

$$\mathsf{View}_0^{\mathcal{A}} = (\mathsf{pp}, (\mathbf{z}, S), \mathsf{ck}_{(\mathbf{z},S)}, \mathbf{x}^*, y^*),$$

where $\mathsf{pp} = (\mathbb{G}, g, p)$, $\mathsf{ck}_{(\mathbf{z},S)} = (\boldsymbol{\alpha}, (g_s)_{s \in S}, \mathbf{z})$, and $y^* = g^{a_0 \cdot \prod_{i=1}^{n} a_i^{x_i^*}}$.

Hybrid $\mathcal{H}_1$: In this game, the challenger modifies the way it computes the constrained key. It sets the constrained key to be $\mathsf{ck} = (\boldsymbol{\alpha}, (g_s)_{s \in S}, \mathbf{z})$, where $g_s = g^{a_0 \cdot r^s}$ for each $s \in S$, same as in $\mathcal{H}_0$, but differently, for $\boldsymbol{\alpha}$, it samples a uniform vector $\boldsymbol{\alpha} \xleftarrow{\$} \mathbb{Z}_p^n$.
Note that in $\mathcal{H}_0$ we have $\alpha_i = r^{-z_i} a_i$, where a_i is uniformly sampled from $\mathbb{Z}_p$ for each $i \in [n]$. Therefore, the distribution of $\boldsymbol{\alpha}$ is identical in both games.

Hybrid $\mathcal{H}_2$: In this game, the challenger replies to the challenge query by returning g^t for a random element $t \xleftarrow{\$} \mathbb{Z}_p^*$. We claim that assuming the sparse power-DDH assumption (Definition 2.4), $\mathcal{H}_2$ and $\mathcal{H}_1$ are computationally indistinguishable. Suppose $\mathcal{A}$ succeeds in distinguishing these two hybrid games. We construct an adversary $\mathcal{B}$ that breaks Assumption 2.4 with respect to the set S and $\ell = n$.

$\mathcal{B}$ is given the group $\mathbb{G} = \langle g \rangle$ of order p and sets $\mathsf{pp} := (\mathbb{G}, g, p)$ which it sends to $\mathcal{A}$. Then, after receiving a constraint query $(\mathbf{z}, S)$ from $\mathcal{A}$, $\mathcal{B}$ asks its challenger for a challenge distribution with respect to the set S and $\ell = n$. It receives a tuple of the form:

$$\left(g, p, (g^{a \cdot r^s})_{s \in S}, (g^{t_s})_{s \in [n] \setminus S}\right),$$

where $a, r \xleftarrow{\$} \mathbb{Z}_p^*$, and for each g^{t_s}, where $s \in [n] \setminus S$, it either holds that $t_s = a \cdot r^s$ or $t_s \xleftarrow{\$} \mathbb{Z}_p^*$.
$\mathcal{B}$ then selects a random vector $\boldsymbol{\alpha} \xleftarrow{\$} \mathbb{Z}_p^n$, sets the constrained key $\mathsf{ck} = (\boldsymbol{\alpha}, (g^{a \cdot r^s})_{s \in S}, \mathbf{z})$, and returns it to $\mathcal{A}$.
To answer the challenge query $\mathbf{x}^*$ made by $\mathcal{A}$ which satisfies $\langle \mathbf{x}^*, \mathbf{z} \rangle \notin S$, it selects the $g^{t_{s^*}}$ corresponding to $s^* = \langle \mathbf{x}^*, \mathbf{z} \rangle \pmod p$ and outputs $(g^{t_{s^*}})^{\prod_{i=1}^{n} \alpha_i^{x_i^*}}$.

If $t_{s^*} = a \cdot r^{\langle \mathbf{x}^*, \mathbf{z} \rangle}$, then $\mathcal{B}$ simulates the view of $\mathcal{A}$ as in Hybrid $\mathcal{H}_1$, and otherwise, if $t_{s^*} \xleftarrow{\$} \mathbb{Z}_p^*$, it simulates $\mathcal{H}_2$. Therefore, distinguishing these two hybrids implies breaking Assumption 2.4 which proves our claim.
The rest of the proof proceeds by reversing the sequence of hybrid games while leaving the challenge query answered by a uniformly random value. □

While the CPRF of Construction 4.1 is no-evaluation secure, a simple attack can be mounted as soon as 1 evaluation query is allowed, as we remark below. Fortunately, no-evaluation secure CPRFs can be turned into standard secure CPRFs by known techniques [AMN+18], e.g. in the ROM. We provide more details below.

Remark 4.1 (A single query attack)**.** *Let $(\mathbf{z}, S)$ be the constraint selected by the adversary. For any input $\mathbf{x}$, let us define $s_{\mathbf{x}} = \langle \mathbf{z}, \mathbf{x} \rangle$ and $u_{\mathbf{x}} = \prod_{i=1}^{n} \alpha_i^{x_i}$, where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ is part of the constrained key obtained by the adversary. Then, remark that the output of the CPRF on any input $\mathbf{x}$ can be written as*

$$\mathsf{Eval}(\mathbf{x}) = g^{a_0 \cdot r^{s_{\mathbf{x}}} \cdot u_{\mathbf{x}}}$$

where $g^{a_0 \cdot r^{s_{\mathbf{x}}}}$ is part of the constrained key if and only only $s_{\mathbf{x}} \in S$.
Since $u_{\mathbf{x}}$ is computable for any $\mathbf{x}$ from the constrained key, an adversary $\mathcal{A}$ with access to the evaluation oracle, can ask for the output of the CPRF on any input $\mathbf{x}$ such that $\mathbf{s}_{\mathbf{x}} \notin S$ and recover $g^{a_0 \cdot r^{s_{\mathbf{x}}}}$ by raising the evaluation to the power $1/u_{\mathbf{x}}$. Given $g^{a_0 \cdot r^{s_{\mathbf{x}}}}$, $\mathcal{A}$ can now evaluate the CPRF on any input $\mathbf{x}'$ such that $s_{\mathbf{x}} = s_{\mathbf{x}'} \pmod p$, and therefore break the security by finding any input $\mathbf{x}^* \neq \mathbf{x}$ such that $s_{\mathbf{x}} = s_{\mathbf{x}^*} \pmod p$.

Achieving Selective and Adaptive Security. As shown in [AMN+18], our no-evaluation secure CPRF for IPM constraints (Construction 4.1) can be modified to achieve adaptive security using a hash function modeled as a random oracle. In order to prevent the attack explained above, we can simply hash the output of our no-evaluation secure CPRF. Modeling the hash function as a random oracle, the output of the evaluation function is perfectly random as long as an adversary cannot efficiently find the hash input, i.e., as explain in our attack, cannot find two values $\mathbf{x} \neq \mathbf{x}' \in \mathbb{Z}_p^n$ for which it holds that $s_{\mathbf{x}} = s_{\mathbf{x}'} \notin S$ and $u_{\mathbf{x}} = u_{\mathbf{x}'}$. Since each a_i (therefore each α_i) is a random element of $\mathbb{Z}_p$, the probability that $u_{\mathbf{x}} = u_{\mathbf{x}'}$ for any $\mathbf{x} \neq \mathbf{x}' \in \mathbb{Z}_p^n$ is $1/p$. Therefore, the probability of finding a collision is negligible. The proof of adaptive security proceeds in the same way as in [AMN+18] (Section 4.3).

Looking more closely, we can replace the random oracle by a correlation-robust hash function and achieve selective security. Variants of correlation-robust hash have been used in many previous works, see [IKNP03, KKRT16, AMN+18] for a small sample. As in these works, we note that this is a simple standard-model assumption that is likely to hold for classical hash functions such as SHA3.

Beyond Binary Inputs. We described the construction for binary inputs and constraints for simplicity, and to match with the original construction of Naor and Reingold [NR97]. However, the construction extends to the setting where $\mathbf{x}, \mathbf{z} \in [\pm B]^n$, where B is a polynomial-size bound (the security of the original Naor-Reingold construction for general inputs of this form was shown in [ABP15] to follow from a variant of the Diffie-Hellman assumption). We then have $|\langle \mathbf{x}, \mathbf{z} \rangle| \leq n \cdot B^2$ and assuming $n \cdot B^2 \ll q$, the inner product is again computed over the integers.

4.2.2 Compressing the keys

We now show how to generate the elements of a master secret key in our Naor-Reingold CPRF by evaluating pseudorandom generators on short seeds. We can thus store a shorter master secret key and communicate a shorter constrained key (for some indices of the constraint vector).
Due to the works of Nechaev [Nec94] and Shoup [Sho97], generic algorithms that solve the DLog problem over $\mathbb{F}_p$, for a prime p, run in $\sqrt{p}$ steps. Therefore, in order to achieve λ bits of security against such algorithms, it should hold that $\log(p) = 2\lambda$. As a result, we can choose the seeds of our PRGs (that generate the master secret key) as short as $\log(p)/2$ bits without any security loss.

Compressing msk**.** Using a pseudorandom generator (PRG) $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{(n+1)\cdot 2\lambda}$, we can generate the vector $\mathsf{msk} = (a_0, a_1, \ldots, a_n)$ as the output of G on a random λ-bit seed. Doing so, the size of the stored master secret key is reduced from $(n+1) \cdot 2\lambda$ bits to λ bits.

Compressing ck**.** Recall that for a constraint vector $\mathbf{z}$ and a set S, a constrained key generated in the CPRF of Construction 4.1 is of the form $\mathsf{ck}_{(\mathbf{z},S)} = (\boldsymbol{\alpha}, (g_s)_{s\in S}, \mathbf{z})$, where $\alpha_i = r^{-z_i} a_i$ for all $i \in [n]$. In the case of binary inputs, for each index $i \in [n]$, it holds that $\alpha_i = a_i$, if $z_i = 0$, and $\alpha_i = r^{-1} \cdot a_i$, if $z_i = 1$. In other words, for all the indices $i \in [n]$, where $z_i = 0$, the master secret key element a_i is included in the constrained key and given to the adversary in plain.
Here, we propose an alternative way of generating the master key elements $(a_1, \ldots, a_n) \in \mathbb{Z}_p^n$ which results in including shorter elements in the constrained key and thus reducing the constrained key size. The idea is to use a pseudorandom generator (PRG) $\mathsf{G}' : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ and to generate each a_i as the image of G' on a randomly sampled short seed $\mathsf{seed}_i \xleftarrow{\$} \{0,1\}^\lambda$, for all $i \in [n]$. Doing so, when generating a constrained key for a constraint vector $\mathbf{z}$, for all the indices $i \in [n]$ where $z_i = 0$, we can include the λ-bit seed_i instead of the 2λ-bit master secret key element a_i, resulting in a 25% reduction of the length of the constrained key.

Combining the two above solutions for reducing the msk and ck sizes, we can first use a PRG $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{(2+n)\lambda}$ to generate a vector $(a_0, \mathsf{seed}_1, \ldots, \mathsf{seed}_n)$ from a random λ-bit seed, and afterwards, use another PRG $\mathsf{G}' : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ to generate $a_i \leftarrow \mathsf{G}'(\mathsf{seed}_i)$ for all $i \in [n]$.

Construction 4.2 presents the modified construction with reduced key size. The steps that are different from the original construction (Construction 4.1) are marked by $\trianglerighteq$.

Construction 4.2: Naor-Reingold CPRF with Compressed Keys

Requirements and notation:

- p is a safe prime, i.e., $p = 2q + 1$ for some prime q.
- The input and constraint space is $\{0,1\}^n$.
- The inner-product space $\mathcal{I} = \{0, 1, \ldots, n\}$.
- $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{(n+2)\lambda}$, and $\mathsf{G}' : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ are PRGs.

Algorithms:

▶ $\mathsf{CPRF.KeyGen}(1^\lambda)$:

- Run $(\mathbb{G}, g, p) \xleftarrow{\$} \mathsf{GenPar}(1^\lambda)$.

$\trianglerighteq$ Sample $\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$.

$\trianglerighteq$ Output $\mathsf{msk} = \mathsf{seed}$, and $\mathsf{pp} = (\mathbb{G}, g, p)$.

▶ $\mathsf{CPRF.Eval}(\mathsf{pp}, \mathsf{msk}, \mathbf{x} \in \{0,1\}^n)$:

$\trianglerighteq$ Parse $\mathsf{pp} = (\mathbb{G}, g, p)$, and $\mathsf{msk} = \mathsf{seed}$.

$\trianglerighteq$ $(a_0, \mathsf{seed}_1, \ldots, \mathsf{seed}_n) \leftarrow \mathsf{G}(\mathsf{seed})$.

$\trianglerighteq$ For all $i \in [n]$: compute $a_i \leftarrow \mathsf{G}'(\mathsf{seed}_i)$.

- Output $y = g^{a_0 \cdot \prod_{i=1}^{n} a_i^{x_i}}$.

▶ $\mathsf{CPRF.Constrain}(\mathsf{pp}, \mathsf{msk}, (\mathbf{z}, S))$:

$\trianglerighteq$ Parse $\mathsf{pp} = (\mathbb{G}, g, p)$, $\mathsf{msk} = \mathsf{seed}$.

$\trianglerighteq$ $(a_0, (\mathsf{seed}_i)_{i=1}^n) \leftarrow \mathsf{G}(\mathsf{seed})$.

- Sample $r \xleftarrow{\$} \mathbb{Z}_p^*$.

$\trianglerighteq$ For $i \in [n]$:

1. If $z_i = 0$, set $\widetilde{\alpha}_i := \mathsf{seed}_i$.
2. If $z_i = 1$, set $\widetilde{\alpha}_i := r^{-1} \cdot \mathsf{G}'(\mathsf{seed}_i)$.

$\trianglerighteq$ Let $\widetilde{\alpha} = (\widetilde{\alpha}_1, \ldots, \widetilde{\alpha}_n)$.

- For $s \in S$, set $g_s := g^{a_0 \cdot r^s}$.

$\trianglerighteq$ Output $\mathsf{ck} = (\widetilde{\alpha}, (g_s)_{s \in S}, \mathbf{z})$.

▶ $\mathsf{CPRF.CEval}(\mathsf{pp}, \mathsf{ck}, \mathbf{x} \in \{0,1\}^n)$:

$\trianglerighteq$ Parse $\mathsf{pp} = (\mathbb{G}, g, p)$, and $\mathsf{ck} = (\widetilde{\alpha}, (g_s)_{s \in S}, \mathbf{z})$.

$\trianglerighteq$ For $i \in [n]$:

1. If $z_i = 0$, set $\alpha_i := \mathsf{G}'(\widetilde{\alpha}_i)$.
2. If $z_i = 1$, set $\alpha_i := \widetilde{\alpha}_i$.

- Let $s_{\mathbf{x}} := \langle \mathbf{z}, \mathbf{x} \rangle$.
- If $\langle \mathbf{z}, \mathbf{x} \rangle \in S$, output $y = (g_{s_{\mathbf{x}}})^{\prod_{i=1}^{n} \alpha_i^{x_i}}$.
- Otherwise, output $\perp$.

Security Analysis. We observe that computing the elements of the master secret as outputs of a pseudorandom generator on short seeds does not affect the no-evaluation security of the resulting CPRF except for imposing a negligible loss in the security reduction. More precisely, the proof of the no-evaluation security of the CPRF with optimized constrained key size follows from a sequence of hybrid games similar to the proof of Theorem 4.1 with an adaptation in Hybrid $\mathcal{H}_1$. We break this hybrid into the two following parts:

Hybrid $\mathcal{H}_1^0$: In this game, the challenger modifies the way it computes some elements of the vector $\boldsymbol{\alpha}$ of the constrained key. To generate a constrained key, the challenger first parses $\mathsf{msk} = \mathsf{seed}$ and computes $(a_0, \mathsf{seed}_1, \ldots, \mathsf{seed}_n) \leftarrow \mathsf{G}(\mathsf{seed})$. It then sets $\widetilde{\alpha}_i := \mathsf{seed}_i$, for all $i \in [n]$ such that $z_i = 0$. And differently from $\mathcal{H}_0$, it samples random elements $\widetilde{\alpha}_i \xleftarrow{\$} \mathbb{Z}_p$, for all $i \in [n]$ such that $z_i = 1$. It then sets the constrained key to be $\mathsf{ck} = (\widetilde{\boldsymbol{\alpha}}, (g_s)_{s\in S}, \mathbf{z})$, where $g_s = g^{a_0 \cdot r^s}$, for all $s \in S$.

Note that Hybrid $\mathcal{H}_1^0$ remains computationally indistinguishable from Hybrid $\mathcal{H}_0$. This is because the only difference between the two hybrids is that for all indices $i \in [n]$ where $z_i = 1$, the element $\widetilde{\alpha}_i$ is sampled as a random element in $\mathcal{H}_1$, rather than being computed as $\widetilde{\alpha}_i = r^{-1} \cdot \mathsf{G}'(\mathsf{seed}_i)$ in $\mathcal{H}_0$. Therefore, if a PPT adversary can distinguish between these two hybrids, there must exist an index i such that a random element $\widetilde{\alpha}_i \xleftarrow{\$} \mathbb{Z}_p$ is distinguished from an element of the form $\widetilde{\alpha}_i = r^{-1} \cdot \mathsf{G}'(\mathsf{seed}_i)$. Such an adversary can be leveraged to distinguish between the outputs of the pseudorandom generator G' with random elements of $\mathbb{Z}_p$.

Hybrid $\mathcal{H}_1^1$: In this game, the challenger samples the master secret key without using the PRG G anymore. In other words, the challenger samples $a_0 \xleftarrow{\$} \mathbb{Z}_p$, and $\mathsf{seed}_i \xleftarrow{\$} \{0,1\}^\lambda$ for all $i \in [n]$, and sets $\mathsf{msk} = (a_0, \mathsf{seed}_1, \ldots, \mathsf{seed}_n)$. As a result, when generating a constrained key, it no longer uses G. Here again, Hybrid $\mathcal{H}_1^1$ remains indistinguishable to $\mathcal{H}_1^0$ due to the security of the PRG G.
The rest of the proof is done exactly as in the proof of Theorem 4.1 with an adaptation that the constrained key element $\boldsymbol{\alpha}$ is now set as in $\mathcal{H}_1^1$.

4.2.3 On IPM Predicates

As mentioned in Section 4.1, the class of IPM predicates includes a variety of interesting predicates. In this part we explain the details of the construction for the inner-product equality predicate, the resulting puncturable PRF, and weak PRFs expressed as IPM predicates.

4.2.3.1 Inner-Product Equality

Inner-product equality (IPE) predicates are special cases of IPM predicates where the inner-product predicates are attributed with singleton sets. More specifically, for a vector $\mathbf{z} \in \mathcal{R}^n$, and a singleton set $S = \{s\}$, the constraint circuit $C_{\mathbf{z}}^S = 0$ iff $\langle \mathbf{z}, \mathbf{x} \rangle = s$. In the following we use the notation $C_{\mathbf{z}}^s$ for singleton sets $S = \{s\}$.

Construction. The CPRF construction supporting the IPE constraints is the same as Construction 4.1, with 2 small differences in the Constrain and CEval algorithms:

- $\mathsf{Constrain}(\mathsf{pp}, \mathsf{msk}, (\mathbf{z}, s))$:

- Parse $\mathsf{pp} = (\mathbb{G}, g, p)$, $\mathsf{msk} = \mathbf{a} = (a_0, \cdots, a_n)$, and $\mathbf{z} = (z_1, \ldots, z_n)$.
- Sample $r \xleftarrow{\$} \mathbb{Z}_p^*$.
- For $i \in [n]$, set $\alpha_i := r^{-z_i} a_i$. Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$.
- Compute $g_s := g^{a_0 \cdot r^s}$.
- Set and output $\mathsf{ck} := (\boldsymbol{\alpha}, g_s)$. \\ $\mathbf{z}$ is not included in ck

- $\mathsf{CEval}(\mathsf{pp}, \mathsf{ck}, \mathbf{x} = (x_1, \ldots, x_n))$:
  - Parse $\mathsf{pp} = (\mathbb{G}, g, p)$ and $\mathsf{ck} = (\boldsymbol{\alpha}, g_s)$.
  - Compute and output $y = (g_s)^{\prod_{i=1}^{n} \alpha_i^{x_i}}$. \\ The output is computed in this way for all inputs.

Security. The 1-key no-evaluation security of the scheme is also implied by the sparse power-DDH assumption (Assumption 2.4). Note that the attack explained in Remark 4.1 also breaks the security of the above construction in case of access to the evaluation oracle. Also, the same techniques explained in Section 4.2.1 can be used to achieve selective and adaptive security.

Constraint Hiding. Unlike for the IPM class, when considering inner-product equality, our CPRF construction is constraint-hiding. In other words, for a selective choice of constraints $(C^{s_0}_{\mathbf{z}_0}, C^{s_1}_{\mathbf{z}_1})$ and a random bit $b \xleftarrow{\$} \{0,1\}$, an adversary $\mathcal{A}$ on input $\mathsf{ck}_b \leftarrow \mathsf{Constrain}(\mathsf{pp}, \mathsf{msk}, (\mathbf{z}_b, s_b))$, cannot guess the bit b better than with probability $1/2$. We briefly go over the proof of constraint-hiding property using a sequence of hybrid games:

Hybrid $\mathcal{H}_0$: This is the original constraint hiding security game where the bit b is set to be 0.

Hybrid $\mathcal{H}_1$: In this game the challenger sets the vector $\boldsymbol{\alpha}$ in the constrained key ck_0 to be a random vector from $(\mathbb{Z}_p^*)^n$. This modification keeps the distribution of the constrained key statistically unchanged while removing the information of the constraint vector $\mathbf{z}_0$ from the constrained key ck_0.

Hybrid $\mathcal{H}_2$: Here the challenger chooses a random element $t \xleftarrow{\$} \mathbb{Z}_p^*$ and a random vector $\boldsymbol{\alpha} \xleftarrow{\$} (\mathbb{Z}_p^*)^n$ and returns $\mathsf{ck}_0 = (g^t, \boldsymbol{\alpha})$. Note that $\boldsymbol{\alpha}$ perfectly hides the randomness r, and therefore $g^{r^{s_0}}$ looks random. Therefore, this hybrid is indistinguishable from Hybrid $\mathcal{H}_1$. In this hybrid the constrained key ck is completely independent of the constraint pair $(\mathbf{z}_0, s_0)$, and thus the probability of $\mathcal{A}$ guessing the bit b correctly is $1/2$.

The rest of the proof proceeds by reversing the sequence of hybrid games but this time setting the bit b to be 1.

Remark 4.2 (No evaluation oracle access in the constraint-hiding game)**.** *Similarly to Remark 4.1, the constraint-hiding property does not hold if the adversary $\mathcal{A}$ is given access to the evaluation oracle. Let $(C^{s_0}_{\mathbf{z}_0}, C^{s_1}_{\mathbf{z}_1})$ be the two constraints selected by the adversary, and let b be the bit chosen by the challenger. When given access to the evaluation oracle, $\mathcal{A}$ can perform an attack as follows to determine the bit b: It first samples $\mathbf{x} \neq \mathbf{x}'$ such that $\langle \mathbf{z}_0, \mathbf{x} \rangle = \langle \mathbf{z}_0, \mathbf{x}' \rangle \pmod p$ and $\langle \mathbf{z}_1, \mathbf{x} \rangle \neq \langle \mathbf{z}_1, \mathbf{x}' \rangle \pmod p$. Then it queries the evaluation oracle on $\mathbf{x}$ and receives $y = g^{a_0 \cdot r^{-s_{\mathbf{x}}} \cdot u_{\mathbf{x}}}$, where $s_{\mathbf{x}} = \langle \mathbf{z}_b, \mathbf{x} \rangle$. It then queries the*

evaluation oracle on $\mathbf{x}'$ *and receives* $y' = g^{a_0 \cdot r^{-s_{\mathbf{x}'}} \cdot u_{\mathbf{x}'}}$, *where* $s_{\mathbf{x}'} = \langle \mathbf{z}_b, \mathbf{x}' \rangle$. *Computing* $u_{\mathbf{x}} = \prod_{i=1}^{n} \alpha_i^{x_i}$ *and* $u_{\mathbf{x}'} = \prod_{i=1}^{n} \alpha_i^{x'_i}$ *locally using* ck_b, $\mathcal{A}$ *computes* $(y)^{u_{\mathbf{x}}^{-1}}$ *and* $(y')^{u_{\mathbf{x}'}^{-1}}$ *and checks if they are equal. If the two elements are equal, then* $b = 0$. *Otherwise,* $b = 1$.
We note that using a hash function modeled as a random oracle also boosts the constraint-hiding game to allow evaluation queries. This is similar to what we explained in Section 4.2.1.

4.2.3.2 Application: A Puncturable PRF in NC^1

A puncturable PRF is a pseudorandom function that allows the generation of a constrained key ck which enables one to evaluate the output of the PRF on all inputs but one. The well-known construction of [GGM84a] from one-way functions offers a puncturable PRF that is computable by a linear-depth circuit in the size of the input. Noticing that the IPM class contains puncturing constraints (for certain parameters), our CPRF construction (Construction 4.1) which is essentially constrained Naor-Reingold PRF and can be evaluated by a log-depth circuit, offers a puncturable PRF in NC^1.

Construction. For this construction, we consider the input space to be $\{-1, 1\}^n$. Suppose that we want to puncture the Naor-Reingold PRF on an input $\mathbf{x}^* \in \{-1, 1\}^n$. In what follows, we show that this can be done using the Naor-Reingold CPRF construction (Construction 4.1) that supports the class of IPM constraints. Note that for a vector $\mathbf{x}^* \in \{-1, 1\}^n$ and any vector $\mathbf{x} \in \{-1, 1\}^n$, the inner-product $\langle \mathbf{x}, \mathbf{x}^* \rangle = n$ iff $\mathbf{x} = \mathbf{x}^*$. Also, all possible values of the inner-product between vectors in $\{-1, 1\}^n$ lie in the set $\mathcal{I} = \{-n, -n+2, -n+4, \ldots, n-2, n\}$. Therefore, setting the constraint set $S = \mathcal{I} \setminus \{n\}$, a CPRF supporting inner-product membership constraints for the set S, can be viewed as a puncturable PRF for any vector $\mathbf{x}^* \in \{-1, 1\}^n$. Parameters of the IPM constraints for this application is presented in Figure 4.1.

Naor-Reingold Puncturable PRF

Setting the parameters of $\mathsf{IPM} = \{C_{\mathbf{z}}^S \mid \mathbf{z} \in \mathcal{R}^n, S \subset \mathcal{I}\}$**:**

- Input and constraint vectors space: $\mathcal{R} = \{-1, 1\}^n$.
- Inner-product space: $\mathcal{I} = \{-n, -n+2, -n+4, \ldots, n-2, n\}$.
- Inner-product constraint set: $S = \mathcal{I} \setminus \{n\}$.

Figure 4.1 – Parameters of IPM for puncturing constraints.

Security. The 1-key selective no-evaluation security of the scheme is implied by the sparse power-DDH assumption. Importantly, we note that in the case of puncturing constraints, the 1-key selective no-evaluation security becomes equivalent to the standard selective security notion where an adversary is allowed to query the evaluation oracle. This is because the challenge query can only be issued on the punctured point $\mathbf{x}^*$. Therefore, in the selective setting, the challenger can compute the constrained key $\mathsf{ck}_{\mathbf{x}^*}$ in the beginning of the security game, and consequently, can answer evaluation queries for inputs $\mathbf{x} \neq \mathbf{x}^*$ by computing and returning the output of $\mathsf{CEval}(\mathsf{pp}, \mathsf{ck}_{\mathbf{x}^*}, \mathbf{x})$ to the adversary. In the following, we briefly go over the security proof:

Hybrid $\mathcal{H}_0$: This is the 1-key selective CPRF security game where the evaluation and challenge queries are answered by returning the output of the CPRF evaluation algorithm on the queried inputs. The view of the adversary in this game is as follows:

$$\mathsf{View}_0^{\mathcal{A}} = (\mathsf{pp}, \{\mathbf{x}_i, y_i\}_{i\in[Q]}, \mathsf{ck}, \mathbf{x}^*, y^*),$$

where $\mathsf{pp} = (\mathbb{G}, g, p)$, $\mathsf{ck} = (\boldsymbol{\alpha}, (g_s)_{s\in S})$, and $y = g^{a_0 \cdot \prod_{i=1}^{n} a_i^x}$ for all $(x, y) \in \{(x_i, y_i)_{i\in[Q]}, (\mathbf{x}^*, y^*)\}$.

Hybrid $\mathcal{H}_1$: In this game, we change how the evaluation oracle queries are answered. As we consider the selective setting, the challenger knows the punctured point $\mathbf{x}^*$ from the beginning. It can therefore compute the constrained key ck from the start, and answer an evaluation query on any input $\mathbf{x} \neq \mathbf{x}^*$ by computing $y \leftarrow \mathsf{CEval}(\mathsf{pp}, \mathsf{ck}, \mathbf{x})$. The view of the adversary remains identical to its view in $\mathcal{H}_0$ by the correctness of the CPRF.

Hybrid $\mathcal{H}_2$: In this game, the challenger sets the vector $\boldsymbol{\alpha}$ in the constrained key ck to be a random vector from $\mathbb{Z}_p^*$. Hybrids $\mathcal{H}_1$ and $\mathcal{H}_2$ remain statistically indistinguishable.

Hybrid $\mathcal{H}_3$: In this hybrid, the challenger replies to the challenge query by returning g^t for a random element $t \xleftarrow{\$} \mathbb{Z}_p^*$. Assuming the sparse power-DDH assumption (Definition 2.4), hybrids $\mathcal{H}_2$ and $\mathcal{H}_3$ are computationally indistinguishable. Let $\mathcal{A}$ be an adversary that distinguishes $\mathcal{H}_2$ and $\mathcal{H}_3$. In what follows, we construct an adversary $\mathcal{B}$ that breaks Assumption 2.4 with respect to the set $S' = \{0, 2, 4, \ldots, 2n-2\}$ and $\ell = 2n$. Knowing the group $\mathbb{G} = \langle g \rangle$ of order p, $\mathcal{B}$ sets $\mathsf{pp} := (\mathbb{G}, g, p)$ and sends it to $\mathcal{A}$.
Then, after receiving a punctured point $\mathbf{x}^*$ from $\mathcal{A}$, adversary $\mathcal{B}$ asks the assumption oracle for a challenge distribution with respect to the set $S' = \{0, 2, 4, \ldots, 2n-2\}$ and $\ell = 2n$ and receives a tuple of the following form:

$$\left(g, p, (g_s)_{s\in S'}, (g_s)_{s\in[2n]\setminus S'}\right),$$

where for all $s \in S'$ it holds that $g_s = g^{a\cdot r^s}$ for some $a, r \xleftarrow{\$} \mathbb{Z}_p^*$, and for all $s \in [2n] \setminus S'$ it either holds that $g_s = g^{a\cdot r^s}$ or $g_s = g^{t_s}$ for some $t_s \xleftarrow{\$} \mathbb{Z}_p^*$.
$\mathcal{B}$ then selects a random vector $\boldsymbol{\alpha} \xleftarrow{\$} \mathbb{Z}_p^n$ and sets the constrained key $\mathsf{ck} = (\boldsymbol{\alpha}, (g^{a\cdot r^s})_{s\in S'}, \mathbf{z})$. Note that with overwhelming probability, a can be written as $a = r^{-n} \cdot \beta$, for a random element $\beta \in \mathbb{Z}_p^*$. Therefore we can rewrite $g^{a\cdot r^s} = g^{\beta \cdot r^{s-n}}$ for all $s \in S'$. As a result, the tuple $(g^{a\cdot r^s})_{s\in S'}$ for the set $S' = \{0, 2, 4, \ldots, 2n-2\}$ simulates $(g^{\beta\cdot r^s})_{s\in S}$, where $S = \{-n, -n+2, \ldots, n-4, n-2\}$.
Finally, to answer the challenge query on the punctured input $\mathbf{x}^*$ which satisfies $\langle \mathbf{x}^*, \mathbf{x}^* \rangle = n$, $\mathcal{B}$ selects the g_{2n} and outputs $y^* = (g_{2n})^{\prod_{i=1}^{n} \alpha_i^{x_i^*}}$. If $\mathcal{A}$ can distinguish between hybrids $\mathcal{H}_3$ and $\mathcal{H}_2$, the adversary $\mathcal{B}$ can successfully break the sparse power-DDH assumption with respect to set S' and $\ell = 2n$.

□

4.2.3.3 Weak PRFs as IPM Predicates

Here, we show that the class of IPM predicates captures several weak PRF candidates from the literature. This observation initiates a ground for Section 4.3.1 where we show how a CPRF that admits a weak PRF as constraint can be used to construct a pseudorandom correlation function (PCF) for oblivious transfer (OT) correlations.

We first propose a definition for Inner-Product Membership (IPM) Pseudorandom Functions (PRFs), denoted as IPM-(w)PRFs. In short, we characterize these functions as (weak) PRFs whose evaluation corresponds to an Inner-Product Membership predicate on functions of the master secret key and inputs, with respect to a publicly defined set.

Definition 4.1 (Inner-Product Membership (weak) PRF, IPM-(w)PRF)**.** Let λ be a security parameter and let $\ell(\cdot)\colon \mathbb{N} \to \mathbb{N}$ denote a length function. An Inner-Product Membership (weak) Pseudorandom Function (IPM-(w)PRF) is a (weak) pseudorandom function $F = (\mathsf{KeyGen}, \mathsf{Eval})$ with domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$, key space $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda\in\mathbb{N}}$, and range $\mathcal{Y} = \{0,1\}$ (per Definition 2.7) for which there exist

- A collection $S = \{S_\lambda\}_{\lambda\in\mathbb{N}}$ of polynomial-size subsets, and
- Two collections of functions $f = \{f_\lambda\colon \mathcal{X}_\lambda \to \mathbb{Z}_\beta^{\ell(\lambda)}\}_{\lambda\in\mathbb{N}}$ and $g = \{g_\lambda\colon \mathcal{K}_\lambda \to \mathbb{Z}_\beta^{\ell(\lambda)}\}_{\lambda\in\mathbb{N}}$, where β is a polynomial-size bound,

such that for any $\lambda \in \mathbb{N}$, the following holds:

$$\mathsf{Eval}(\mathsf{msk}, x) = \begin{cases} 0 & \text{if } \langle f(x), g(\mathsf{msk})\rangle \in S \\ 1 & \text{otherwise.} \end{cases},$$

where $\mathsf{msk} \leftarrow \mathsf{KeyGen}(1^\lambda)$. We denote an IPM weak pseudorandom function as IPM-wPRF and an IPM pseudorandom function as IPM-PRF.

Candidate IPM-wPRFs

Here, we provide a brief overview of existing weak pseudorandom functions that satisfy our definition of IPM-wPRFs (as detailed in Definition 4.1). A more detailed discussion about these weak PRFs is provided in our paper [BCM+24].

▶ From the Learning with Rounding (LWR) Assumption

The Learning with Error (LWE) assumption, introduced by Regev [Reg05] is a well studied hardness assumption roughly stating that "noisy" inner-products of a secret vector of integers with public random vectors cannot be efficiently distinguished from random values. This assumption has received many attentions from cryptographers, in particular because for certain settings of parameters, this assumption follows from the hardness of solving some standard problems over lattices in the worst-case by *quantum* computers. Therefore, the protocols that are constructed based on this assumption, are considered to be secure even in the presence of quantum computers, and are called *post-quantum secure*.
The Learning with Rounding (LWR) assumption, introduced in [BPR12], can be viewed as a deterministic version of LWE where the "noise" is introduced *deterministically* by rounding the result of the inner-product. More precisely, the LWR assumption states that for two moduli $q' \ll q$, a random matrix $\boldsymbol{A} \xleftarrow{\$} \mathbb{Z}_q^{m\times n}$, and a random vector $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$, the structured pair $(\boldsymbol{A}, \lfloor \boldsymbol{A}\cdot\mathbf{s}\rceil_{q'})$ is computationally indistinguishable from $(\boldsymbol{A}, \mathbf{u})$, where $\mathbf{u}$ is a random vector of $\mathbb{Z}_{q'}^m$, and $\lfloor x\rceil_{q'}$ denotes rounding $x\cdot\frac{q'}{q}$ to the nearest integer modulo q'. In [BPR12], the authors showed that LWR can be reduced to the LWE assumption for certain parameters. In particular, their reduction required the modulus q to be superpolynomial. Later in [AKPW13], a refined reduction from LWR to LWE

was proposed that captured more general settings of parameter, in particular, for a polynomial modulus q.

Now we show the two following LWR(-style) weak PRF candidates from the literature can be expressed as inner-product membership weak PRFs.

- wPRF of [BPR12]: This candidate is simply defined as $F(\mathsf{msk}, \boldsymbol{x}) = \lfloor \langle \mathsf{msk}, \boldsymbol{x} \rangle \rceil_{q'}$, where $\mathsf{msk}, \boldsymbol{x} \in \mathbb{Z}_q^n$. As mentioned above, this candidate is proved to be a weak PRF under the LWR assumption, for a polynomial modulus q. While the proof does not extend to the case of $q' = 2$, there are no known attacks and the LWR assumption is believed to reduce to the LWE assumption in this regime. In this case, if we set

  - $S = \{s \in \mathbb{Z}_{n \cdot q^2} \ : \ \lfloor (s \mod q) \rceil_2 = 0\}$, and
  - f_λ and g_λ as identity functions over $\mathbb{Z}_q^n$,

  then we can express this candidate as an IPM-wPRF. Here $|S| \approx n \cdot q^2/2$, and for standard choices of the modulus q can be quite large.

- wPRF of [BIP+18]: This candidate is defined as $F(\mathsf{msk}, \boldsymbol{x}) = \lfloor \langle \mathsf{msk}, \boldsymbol{x} \rangle \pmod 6 \rceil$, where $\mathsf{msk}, \boldsymbol{x} \in \{0,1\}^n$, and the rounding function is defined as $\lfloor s \rceil = 0$ if $(s \bmod 6) \in \{0,1,2\}$, and $\lfloor s \rceil = 1$ if $(s \bmod 6) \in \{3,4,5\}$. This weak PRF can also be expressed as an IPM-wPRF, where

  - $S = \{s \in \mathbb{Z}_{n+1} \ : \ \exists k \leq n/6, i \in \{0,1,2\} \text{ s.t. } s = 6k + i\}$, and
  - f_λ and g_λ as identity functions over $\mathbb{Z}_q^n$. Analysis of [CCKK21] and [JMN23] suggest a key length of $n = 770$ for this candidate. In this case, the size of the set S, which is $n/2$, is as low as $|S| = 385$.

► From Goldreich's One-Way Function.

A one-way function (OWF) is defined as a function that can be efficiently computed on inputs, but is hard to invert given its output of a random input. Goldreich's one-way function [Gol00] is defined as the evaluation of a fixed low-arity predicate on fixed random small subsets of the input bits (*i.e.*, $f(x) = (P(x[S_1]), \cdots, P(x[S_m])$, where P is a predicate and $S_1, \cdots, S_n$ are fixed random subsets, also $x[S_i]$ denotes the substring of the bits of x indexed by S_i). Later works suggested that for a suitable choice of the predicate P, this function can be conjectured to be a pseudorandom generator when $m > |x|$. In [AR16], Applebaum and Raykov showed how Goldreich's random local functions can also yield plausible candidate wPRFs for suitable choices of P, when $|S_i| = \Omega(\log n)$ for $i \in [m]$. We denote this candidate wPRF as Goldreich-Applebaum-Raykov(GAR) wPRF.

More precisely, GAR wPRF is defined as $F(\mathsf{msk}, \boldsymbol{x}) = P(\mathsf{msk}[S_x])$, where $\mathsf{msk} \in \{0,1\}^n$, and $\boldsymbol{x} \in \{0,1\}^{\log^2(n)}$, and $P : \{0,1\}^n \times \{0,1\}^{\log^2(n)} \to \{0,1\}$. Also the set $S_{\boldsymbol{x}} = \{j_1, \ldots, j_k\}$ is the set of $k = \Omega(\log(n))$ indices where each index is an element of $[n]$, defined by parsing $\boldsymbol{x}$ as a k-tuple of distinct strings $\hat{j}_1, \cdots, \hat{j}_k \in \{0,1\}^{\log(n)}$, and computing each j_i as the integer representation of $\hat{j}_i$. This weak PRF can be expressed as an IPM-wPRF, up to the preprocessing of the input $\boldsymbol{x}$ as follows:

- $S = \{s \in [2^k] \ : \ P(s) = 0\}$, and

- f_λ is defined as a function mapping $\boldsymbol{x} \in \{0,1\}^{\log^2(n)}$ to a vector of length n with 0 everywhere, except at positions j_i, where it has the entry 2^{i-1}, for $i = 1, \ldots, k$. Also, g_λ is the identity function over $\{0,1\}^n$.

Note that with this encoding, we have

$$\langle f(\boldsymbol{x}), g(\mathsf{msk}) \rangle = \sum_{i=1}^{k} \mathsf{msk}_{j_i} \cdot 2^{i-1},$$

which is exactly the integer whose binary representation encodes $\mathsf{msk}[S_{\boldsymbol{x}}]$. For a k-ary predicate P, we have $|S| = \mathcal{O}(2^k)$, therefore when setting $k = \mathcal{O}(\log(n))$, we have a set of size $\mathcal{O}(n)$.

4.3 PCF for OT from Pseudorandomly Constrained PRFs

In this section, we provide a generic transformation from constrained PRFs to PCFs for OT correlations. We then show how it can be instantiated using the Naor-Reingold constrained PRF proposed in Section 4.2, which yields a concretely efficient construction.

4.3.1 A Generic Transformation

This transformation builds upon the following idea: a constrained pseudorandom function that supports a weak/strong PRF and its complement as constraint circuits can be used to build a PCF for OT correlations. We refer to such constrained PRFs as *Full-Domain Pseudorandomly Constrained PRFs*. These CPRFs are a special case of pseudorandomly constrained PRFs that admit a weak/strong PRF (but not necessarily its complement) as constraints.

We first formally define a pseudorandomly constrained PRF, and show how it can be transformed to a PCF for OT correlations. In the next section, we show that a CPRF for inner-product membership predicates is a full-domain pseudorandomly constrained PRF. This observation yields to as efficient instantiation of our paradigm from the Naor-Reingold constrained PRF proposed in Section 4.2.

Definition 4.2 (Pseudorandomly Constrained PRF)**.** A weakly/strongly pseudorandomly constrained PRF (PR-CPRF) is a constrained pseudorandom function (per Definition 2.9) that supports a family of weak/strong pseudorandom predicates as constraints. If a constrained pseudorandom function admits a family of weak/strong pseudorandom predicates $\{F(k,\cdot) \ : \ k \in \mathcal{K}_\lambda\}$ and their complement $\{1 - F(k,\cdot) \ : \ k \in \mathcal{K}_\lambda\}$, we refer to it as a full-domain pseudorandomly constrained PRF.

We now describe a transformation from full-domain pseudorandomly constrained PRFs to PCFs for 1-out-of-2 random OT correlations.[1]

[1] PCFs for 1-out-of-2 random OT correlations is the commonly considered notion of "PCF for OT correlations".

Construction 4.3: PCF for OT Correlations

Requirements and notation:

- $F\colon \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \{0,1\}$ is a weak/strong PRF; we denote $\overline{F}\colon \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \{0,1\}$ the function defined by $\overline{F}(k,x) := 1 - F(k,x)$.
- $\mathsf{CPRF} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ is a constrained PRF supporting as constraints $\{F(k,\cdot)\colon k \in \mathcal{K}_\lambda\} \cup \{\overline{F}(k,\cdot)\colon k \in \mathcal{K}_\lambda\}$.

Algorithms:

▶ $\mathsf{PCF.Gen}_0(1^\lambda)$:

1. Run $(\mathsf{pp}_0, \mathsf{msk}_0) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$.
2. Run $(\mathsf{pp}_1, \mathsf{msk}_1) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$.
3. Set and output $k_0 \leftarrow (\mathsf{msk}_0, \mathsf{msk}_1)$.
   // $\mathsf{pp}_0, \mathsf{pp}_1 \in k_0$, implicitly.

▶ $\mathsf{PCF.Gen}_1(1^\lambda, k_0)$:

1. Sample $k \xleftarrow{\$} \mathcal{K}_\lambda$.
2. Parse $k_0 = (\mathsf{msk}_0, \mathsf{msk}_1)$.
3. $\mathsf{ck}_0 \leftarrow \mathsf{Constrain}(\mathsf{msk}_0, F(k,\cdot))$.
   // For inputs x such that $F(k,x) = 0$.
4. $\mathsf{ck}_1 \leftarrow \mathsf{Constrain}(\mathsf{msk}_1, \overline{F}(k,\cdot))$.
   // For inputs x such that $F(k,x) = 1$.
5. Set and output $k_1 \leftarrow (\mathsf{ck}_0, \mathsf{ck}_1, k)$.
   // $\mathsf{pp}_0, \mathsf{pp}_1 \in k_1$, implicitly.

▶ $\mathsf{PCF.Eval}(1^\lambda, \sigma, k_\sigma, x)$:

1. If $\sigma = 0$:
   a) Parse $k_0 = (\mathsf{msk}_0, \mathsf{msk}_1)$.
   b) Set $r_0 \leftarrow \mathsf{Eval}(\mathsf{pp}_0, \mathsf{msk}_0, x)$.
   c) Set $r_1 \leftarrow \mathsf{Eval}(\mathsf{pp}_1, \mathsf{msk}_1, x)$.
   d) Set and output $y_0 \leftarrow (r_0, r_1)$.
2. If $\sigma = 1$:
   a) Parse $k_1 = (\mathsf{ck}_0, \mathsf{ck}_1, k)$.
   b) Compute $b := F(k,x)$.
   c) Compute $r \leftarrow \mathsf{CEval}(\mathsf{pp}_b, \mathsf{ck}_b, x)$.
   d) Set and output $y_1 \leftarrow (b, r)$.

Theorem 4.2. *Construction 4.3 is a secure weak/strong precomputable PCF for OT correlations.*

Proof. Firstly, Construction 4.3 explicitly provides two key generation algorithms $\mathsf{PCF.Gen}_0$ and $\mathsf{PCF.Gen}_1$, the precomputability property is evident and follows from definition. We proceed to prove the correctness of correlation between the outputs and the security of the scheme.

Weakly Pseudorandom OT-Correlated Outputs. Consider the following sequence of hybrid games:

Hybrid $\mathcal{H}_0$: This is the real-world experiment $\mathsf{Exp}^{\mathsf{w\text{-}pr}}_{\mathcal{A},N,1}(\lambda)$, where the view of an adversary consists of $(1^\lambda, (x^{(i)}, y_0^{(i)}, y_1^{(i)})_{i\in[N(\lambda)]})$, where each tuple $(x^{(i)}, y_0^{(i)}, y_1^{(i)})$ is generated by running the PCF evaluation algorithms of both parties on $x^{(i)}$ for $i \in [N(\lambda)]$.

Hybrid $\mathcal{H}_1$: In this game, we compute each $y_1^{(i)}$, for all $i \in [N]$, as follows:

1. Parse $k_1 = (\mathsf{ck}_0, \mathsf{ck}_1, k)$

2. Compute $b^{(i)} \leftarrow F(k, x^{(i)})$
3. Parse $y_0^{(i)} = (r_0^{(i)}, r_1^{(i)})$, and set $r^{(i)} \leftarrow r_{b^{(i)}}^{(i)}$.
4. Set and output $y_1^{(i)} \leftarrow (b^{(i)}, r^{(i)})$.

Hybrid $\mathcal{H}_1$ differs from Hybrid $\mathcal{H}_0$ only in the way we generate the second component of each $y_1^{(i)}$; either it is an output of CPRF.CEval (in Hybrid $\mathcal{H}_0$) or it is an output of CPRF.Eval (in Hybrid $\mathcal{H}_1$). These two hybrids are therefore indistinguishable by the correctness of the CPRF.

Hybrid $\mathcal{H}_2$: In this game, we replace $b^{(i)} \coloneqq F(k, x^{(i)})$ by random bits $b^{(i)} \xleftarrow{\$} \{0,1\}$ for all $i \in [N]$. This hybrid is indistinguishable from Hybrid $\mathcal{H}_1$ due to the weak/strong security of the PRF F.

Hybrid $\mathcal{H}_3$: In this game, for each input $x^{(i)}$, we compute $y_0^{(i)} = (r_0^{(i)}, r_1^{(i)}) \xleftarrow{\$} \mathcal{Y} \times \mathcal{Y}$, where $\mathcal{Y}$ is the output range of CPRF.
This hybrid is indistinguishable from Hybrid $\mathcal{H}_2$ due to the pseudorandomness of CPRF on all inputs when an adversary doesn't have access to neither the master secret key nor a constrained key.

Note that hybrid $\mathcal{H}_3$ has the same distribution as the ideal experiment $\mathsf{Exp}^{\text{w-pr}}_{\mathcal{A},N,0}(\lambda)$. This concludes the proof of pseudorandom OT-correlated outputs of the construction.

Weak PCF Security. Recall that for OT correlations (with message space $\mathcal{Y}$), RSample is defined as follows.
$\mathsf{RSample}(1^\lambda, \sigma, y_\sigma)$:

- If $\sigma = 0$:
  - Parse $y_0 = (r_0, r_1)$.
  - Sample $b \xleftarrow{\$} \{0,1\}$.
  - Set and output $y_1 = (b, r_b)$.
- If $\sigma = 1$:
  - Parse y_1 as $y_1 = (b, r_b)$.
  - Sample $r_{1-b} \xleftarrow{\$} \mathcal{Y}$.
  - Set and output $y_0 = (r_0, r_1)$

- For $\sigma = 0$: The only difference between $\mathsf{Exp}^{\text{w-sec}}_{\mathcal{A},N,\sigma=0,0}(\lambda)$ and $\mathsf{Exp}^{\text{w-sec}}_{\mathcal{A},N,\sigma=0,1}(\lambda)$ is whether the second component of each $(y_1^{(i)}$, for all $i \in [N]$, is sampled uniformly at random or is the an output of a (weak) PRF. Since the (weak) PRF key is hidden from the view of the adversary in both experiments, these two games are indistinguishable by security of the (weak) PRF F.
- For $\sigma = 1$: The only difference between the two experiments is for each $i \in [N]$, each bit $b^{(i)}$, and each output $y_0^{(i)} = (r_0^{(i)}, r_1^{(i)})$, whether $r_{1-b^{(i)}}^{(i)}$ is sampled uniformly at random from $\mathcal{Y}$, or is it computed as the output of the CPRF evaluation, i.e., $r_{1-b^{(i)}}^{(i)} \leftarrow \mathsf{CPRF.Eval}(\mathsf{pp}_{1-b^{(i)}}, \mathsf{msk}_{1-b^{(i)}}, x^{(i)})$.
  The indistinguishability between these two cases follows from the security of CPRF; For any index $i \in [N]$, input $x^{(i)}$, and bit $b^{(i)} \leftarrow F(k, x^{(i)})$, the constrained key $\mathsf{ck}_{1-b^{(i)}}$ fails in evaluating the output of the master secret key on $x^{(i)}$, and an adversary holding $\mathsf{ck}_{1-b^{(i)}}$ cannot distinguish the output from a random element of the range. Therefore $x^{(i)}$ is an unauthorized input for $\mathsf{ck}_{1-b^{(i)}}$, and the two experiments remain indistinguishable.

This concludes the proof of security of our PCF construction for OT correlations, against both parties. □

Remark 4.3. *We note that the fact that CPRFs for a class containing a PRF yield a PCF is not entirely new; for example, a similar observation was briefly mentioned in [BGMM20]. However, the few known constructions of sufficiently expressive CPRFs [BV15, AMN+18, CMPR23] are too expensive, and using them within the above transformation yields PCFs that are much less flexible than generic constructions based on homomorphic secret sharing or threshold FHE (that are not restricted to the OT correlation), and much less efficient than state-of-the-art PCFs [BCG+20, BCG+22]. Our key contribution is identifying that a simple tweak to the Naor-Reingold PRF [NR97] yields an efficient pseudorandomly constrained PRF as we explained in Section 4.2.*

4.3.2 Instantiations

In this section we show how the transformation from full-domain pseudorandomly constrained PRFs to PCFs for OT correlations can be efficiently instantiated. The core observation that makes the instantiation possible is that a constrained pseudorandom function for the class of inner-product membership predicates is a full-domain pseudorandomly constrained PRF.

Lemma 4.3 (IPM-CPRFs are full-domain PR-CPRFs)**.** *If* CPRF*, described by four algorithms* (KeyGen, Eval, Constrain, CEval)*, is a constrained pseudorandom function supporting the class of* IPM *predicates, then it is a full-domain pseudorandomly constrained PRF.*

Proof. The proof consists of the following steps:

1. There exist weak PRFs that can be expressed as IPM predicates, which we refer to as IPM-wPRFs.
   → Constructions of IPM-wPRFs were named and discussed in Section 4.2.3.3 which include the LWR-based weak PRFs of [BIP+18] and [BPR12], and Goldreich-Applebaum-Raykov wPRF.

2. The complement of an IPM-wPRF is itself an IPM-wPRF.
   → Let F be an IPM-wPRF with domain $\mathcal{X} = \{X_\lambda\}_{\lambda\in\mathbb{N}}$, key space $\mathcal{K} = \{K_\lambda\}_{\lambda\in\mathbb{N}}$, and range $\mathcal{Y} = \{0,1\}$. Let $S = \{S_\lambda\}_{\lambda\in\mathbb{N}}$ be the collection of subsets and $f = \{f_\lambda : \mathcal{X}_\lambda \to \mathbb{Z}_\beta^{\ell(\lambda)}\}$, and $g = \{g_\lambda : \mathcal{X}_\lambda \to \mathbb{Z}_\beta^{\ell(\lambda)}\}$ be the functions that, together with F, satisfy Definition 4.1.
   Let $\overline{F}$ denote the complement of the function F, i.e., for any $\lambda \in \mathbb{N}$ and all $x \in \mathcal{X}_\lambda$, it holds that $\overline{F}(k_\lambda, x) = 1 - F(k_\lambda, x)$. If we set $\overline{S}_\lambda = \{0,\dots,\beta^2 \cdot \ell(\lambda)\} \setminus S_\lambda$, then $\overline{F}$ can be expressed as an IPM-wPRF parametrized with $\overline{S}_\lambda$ and f, g.

□

We now have all the ingredients to state the following theorem:

Theorem 4.4 (PCF for OT from Constrained Naor-Reingold)**.** *Assuming the hardness of the sparse power-DDH assumption, and the existence of an* IPM-wPRF*, there exists a PCF for OT correlations in the random oracle model.*

4.4 Public-Key PCF for OT from Naor-Reingold

4.4.1 Public-Key PCF: Formal Definition

Here, we introduce and formalize the notion of public-key pseudorandom correlation function (PK-PCF). The main property of a public-key PCF is the non-interactive generation of evaluation keys. More precisely, in a PK-PCF, the generation of evaluation keys is done in two separate steps: a secret/public key generation $\mathsf{PCF.Gen}$, and an evaluation key derivation $\mathsf{PCF.KeyDer}$. Let $\sigma \in \{0,1\}$ denote the index of a party in a PK-PCF protocol. In the first step, each party locally runs $\mathsf{PCF.Gen}(\sigma)$ to generate a key pair $(\mathsf{sk}_\sigma, \mathsf{pk}_\sigma)$, and then broadcasts the public key pk_σ. In the second step, each party uses the secret key sk_σ and the public key of the other party $\mathsf{pk}_{1-\sigma}$ and runs the $\mathsf{PCF.KeyDer}$ algorithm in order to derive a PCF evaluation key k_σ. After this step, similarly to an interactive PCF, parties can use their keys k_σ and run the evaluation algorithm $\mathsf{PCF.Eval}(\sigma, k_\sigma, x)$ to compute a correlated output y_σ on an input x.

In the following, we state the formal definition and security requirements of a *weak* PK-PCF.

Definition 4.3 (Public-Key Pseudorandom Correlation Function (PK-PCF))**.** Let $\mathcal{Y}$ be a reverse-sampleable correlation with output length functions $\ell_0(\lambda), \ell_1(\lambda)$ and let $n(\lambda)$ be an input length function. A Public-Key Pseudorandom Correlation Function consists of the following four polynomial-time algorithms:

- $\mathsf{PCF.Setup}(1^\lambda)$: A probabilistic algorithm that on input the security parameter λ, outputs public parameters pp. For simplicity, we assume that all other algorithms have access to pp.
- $\mathsf{PCF.Gen}(\sigma)$: A probabilistic algorithm that on input $\sigma \in \{0,1\}$, outputs a pair of secret key and public key $(\mathsf{sk}_\sigma, \mathsf{pk}_\sigma)$.
- $\mathsf{PCF.KeyDer}(\sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma})$: A deterministic algorithm that on input $\sigma \in \{0,1\}$, a secret key sk_σ and a public key $\mathsf{pk}_{1-\sigma}$, outputs an evaluation key k_σ.
- $\mathsf{PCF.Eval}(\sigma, k_\sigma, x)$: A deterministic algorithm that on input $\sigma \in \{0,1\}$, a key k_σ and input value $x \in \{0,1\}^{n(\lambda)}$, outputs $y_\sigma \in \{0,1\}^{\ell_\sigma(\lambda)}$.

We say that $(\mathsf{PCF.Setup}, \mathsf{PCF.Gen}, \mathsf{PCF.KeyDer}, \mathsf{PCF.Eval})$ is a public-key pseudorandom correlation function (PK-PCF) for $\mathcal{Y}$, if the following conditions hold:

Pseudorandom $\mathcal{Y}$-correlated outputs. For any security parameter $\lambda \in \mathbb{N}$, and any non-uniform adversary $\mathcal{A}$ of size $B(\lambda)$:

$$\left|\Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,b}$ for each $b \in \{0,1\}$ is defined as in Figure 4.2.

Security. For $\sigma \in \{0,1\}$ and any non-uniform adversary $\mathcal{A}$ of size $B(\lambda)$:

$$|\Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda) = 1]| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,b}$ for each $b \in \{0,1\}$ is defined as in Figure 4.3. We recall that $\mathsf{RSample}$ is the algorithm for reverse sampling the correlation $\mathcal{Y}$.

Experiment Pseudorandom $\mathcal{Y}$-Correlated Outputs

$\underline{\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{PCF.Setup}(1^\lambda)$

For $\sigma = 0, 1$**:**

$(\mathsf{sk}_\sigma, \mathsf{pk}_\sigma) \leftarrow \mathsf{PCF.Gen}(\sigma)$

For $i = 1, \ldots, N(\lambda)$:

$x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

$(y_0^{(i)}, y_1^{(i)}) \hookleftarrow \mathcal{Y}(1^\lambda)$

$b \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}_0, \mathsf{pk}_1, (x^{(i)}, y_0^{(i)}, y_1^{(i)})_{i\in[N(\lambda)]})$

Output b

$\underline{\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{PCF.Setup}(1^\lambda)$

For $\sigma = 0, 1$**:**

$(\mathsf{sk}_\sigma, \mathsf{pk}_\sigma) \leftarrow \mathsf{PCF.Gen}(1^\lambda, \mathsf{pp}, \sigma)$

For $\sigma = 0, 1$**:**

$k_\sigma \leftarrow \mathsf{PCF.KeyDer}(\sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma})$

For $i = 1, \ldots, N(\lambda)$:

$x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

$y_0^{(i)} \leftarrow \mathsf{PCF.Eval}(0, k_0, x^{(i)})$

$y_1^{(i)} \leftarrow \mathsf{PCF.Eval}(1, k_1, x^{(i)})$

$b \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}_0, \mathsf{pk}_1, (x^{(i)}, y_0^{(i)}, y_1^{(i)})_{i\in[N(\lambda)]})$

Output b

Figure 4.2 – Pseudorandom $\mathcal{Y}$-correlated outputs of a PK-PCF

Experiment PCF Security

$\underline{\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{PCF.Setup}(1^\lambda)$

For $\hat{\sigma} = 0, 1$**:** $(\mathsf{sk}_{\hat{\sigma}}, \mathsf{pk}_{\hat{\sigma}}) \leftarrow \mathsf{PCF.Gen}(\hat{\sigma})$

$k_{1-\sigma} \leftarrow \mathsf{PCF.KeyDer}(1-\sigma, \mathsf{sk}_{1-\sigma}, \mathsf{pk}_\sigma)$

For $i = 1, \ldots, N(\lambda)$:

$x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

$y_{1-\sigma}^{(i)} \leftarrow \mathsf{PCF.Eval}(1-\sigma, k_{1-\sigma}, x^{(i)})$

Let $T = (x^{(i)}, y_{1-\sigma}^{(i)})_{i\in[N(\lambda)]}$

$b \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}_0, \mathsf{pk}_1, \sigma, \mathsf{sk}_\sigma, T)$

Output b

$\underline{\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{PCF.Setup}(1^\lambda)$

For $\hat{\sigma} = 0, 1$**:** $(\mathsf{sk}_{\hat{\sigma}}, \mathsf{pk}_{\hat{\sigma}}) \leftarrow \mathsf{PCF.Gen}(\hat{\sigma})$

$k_\sigma \leftarrow \mathsf{PCF.KeyDer}(\sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma})$

For $i = 1, \ldots, N(\lambda)$:

$x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

$y_\sigma^{(i)} \leftarrow \mathsf{PCF.Eval}(\sigma, k_\sigma, x^{(i)})$

$y_{1-\sigma}^{(i)} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y_\sigma^{(i)})$

Let $T = (x^{(i)}, y_{1-\sigma}^{(i)})_{i\in[N(\lambda)]}$

$b \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}_0, \mathsf{pk}_1, \sigma, \mathsf{sk}_\sigma, T)$

Output b

Figure 4.3 – Security of a PK-PCF.

4.4.2 A Public-Key PCF via Bellare-Micali Non-Interactive OT

In Section 4.3, we provided transformations from constrained PRFs supporting (w)PRF constraints and their complements to PCFs for OT correlations. More specifically, Construction 4.3 presents a simple instance of this transformation where in order to generate the PCF keys, two calls to the underlying CPRF are made. In this section, we discuss how to generate the resulting PCF keys of this transformation non-interactively when plugging in our Naor-Reingold CPRF.
Let $F : \mathcal{K} \times \mathcal{X} \to \{0,1\}$ be an IPM-wPRF associated with an IPM set S and predicate functions f, g (See Definition 4.1). Plugging in our Naor-Reingold CPRF construction (Construction 4.1) in the PCF transformation, the resulting PCF keys are:

$$k_0 = ((\mathsf{pp}_0, \mathsf{msk}_0), (\mathsf{pp}_1, \mathsf{msk}_1)), \quad k_1 = ((\mathsf{pp}_0, \mathsf{ck}_0), (\mathsf{pp}_1, \mathsf{ck}_1), g(\hat{\mathsf{k}})),$$

where $\mathsf{msk}_b \leftarrow \mathsf{CPRF.KeyGen}(1^\lambda)$, and $\mathsf{ck}_b \leftarrow \mathsf{CPRF.Constrain}(\mathsf{msk}_b, C_b)$, where $C_0(x) = F_{\hat{\mathsf{k}}}(x)$, and $C_1(x) = 1 - F_{\hat{\mathsf{k}}}(x)$ for a random key $\hat{\mathsf{k}} \xleftarrow{\$} \mathcal{K}$, and $b \in \{0,1\}$.

We now take a closer look at the correlation between a master secret key and a constrained key in Naor-Reingold CPRF. When working over a group $\mathbb{G} = \langle g \rangle$ of prime order p, a master secret key is of the form $\mathsf{msk} = (a_0, a_1, \ldots, a_n) \xleftarrow{\$} \mathbb{Z}_p^{n+1}$, and a constrained key for an IPM constraint (k, S) is $\mathsf{ck} = (\boldsymbol{\alpha}, (g_s)_{s\in S}, \mathsf{k})$, where $\alpha_i = a_i \cdot r^{-k_i}$ for all $i \in [n]$, and $g_s = g^{a_0 \cdot r^s}$ for all $s \in S$. Here, $\mathsf{k} = g(\hat{\mathsf{k}})$, where $\hat{\mathsf{k}}$ is the key of the wPRF F.

In order to generate the PCF keys non-interactively, we need to derive a valid pair of msk and ck non-interactively. Given that the set S attributed with the IPM-wPRF is public, party 0 can sample a random element $a_0 \xleftarrow{\$} \mathbb{Z}_p$ and compute and publish the tuple $(g_s)_{s\in S}$. Party 1 can then use this tuple as a part of its PCF key.
The more correlated elements of msk and ck are the vectors $(a_1, \ldots, a_n) \in \mathsf{msk}$ and $(\alpha_1, \ldots, \alpha_n) \in \mathsf{ck}$ that satisfy the equation $\alpha_i = a_i \cdot r^{-g(k)_i}$ for all $i \in [n]$. Note that the wPRF key k is only known to party 1 and the secret elements $(a_1, \ldots, a_n)$ and r are only known to party 0. In what follows, we discuss how to generate two pairs (a, r) and (α, b) such that $\alpha = a \cdot r^b$, where $\alpha, a, r \in \mathbb{Z}_p$ and $b \in \{0,1\}$. We can then extend the solution to generate all the elements of the two vectors $\mathbf{a}$ and $\boldsymbol{\alpha}$.

Bellare-Micali non-interactive OT [BM90] provides a simple but costly solution by using ElGamal encryption and Pedersen commitment to generate two such correlated pairs (a, r) and (α, b), for a bit $b \in \{0,1\}$ and elements $a, r, \alpha \in \mathsf{QR}_p$, where QR_p denotes the subgroup of quadratic residues modulo p. Let p and q be two prime numbers such that $p = 2q + 1$. The public elements g, h are randomly sampled from QR_p and $\mathsf{DLog}_g(h)$ is unknown to both parties. This protocol proceeds as follows.

$\mathsf{pp} = (p, g, h)$

Party 0		Party 1
$r \xleftarrow{\$} \mathsf{QR}_p$	$\mathsf{pk}_0 = (g^t, h^t \cdot r)$	$b \xleftarrow{\$} \{0,1\}$
$t \xleftarrow{\$} \mathbb{Z}_q$	$\mathsf{pk}_1 = g^s \cdot h^b$	$s \xleftarrow{\$} \mathbb{Z}_q$
Compute $a \leftarrow (g^s \cdot h^b)^t \pmod p$		Compute $\alpha \leftarrow (h^t \cdot r)^b (g^t)^s \pmod p$
Output (a, r)		Output (α, b)

Note that in this protocol, the element r should be a quadratic residue modulo p. Therefore, using this protocol for setting up a public key generation for our PCF protocol from the Naor-Reingold CPRF implies assuming the hardness of the sparse power-DDH problem for a quadratic residue $r \in \mathsf{QR}_p$. This variant is implied by the sparse power-DDH assumption. However, for the security of this non-interactive protocol we have to assume the hardness of DDH problem over QR_p which imposes choosing a large-enough prime p due to subexponential-time attacks on DDH over finite fields. This makes the resulting public-key PCF inefficient in terms of both the size of public-key and evaluation keys and computation time.
In the next section, we propose an alternative efficient way of setting up the public keys.

4.4.3 A Better Construction from Paillier-ElGamal

In this section, we present an efficient public-key PCF construction where in order to derive a pair of OT-correlated evaluation keys, we perform the non-interactive OT protocol of Bellare-Micali [BM90] over a Paillier group. This is in essence the Non-interactive VOLE protocol of [OSY21], followed by additional steps in order to derive the correlated keys over $\mathbb{Z}_p^*$.
Let $N = PQ$ be a Blum integer, meaning that P and Q are prime numbers of the form $P = 2P' + 1$ and $Q = 2Q' + 1$ for λ-bit prime numbers P' and Q'. The key generation and derivation of our public-key PCF work over the group $\mathbb{Z}_{N^2}^* \approx \mathbb{H} \times \mathbb{NR}_N$, where $\mathbb{H} = \{(1+N)^i : i \in [N]\}$ is of order N, and $\mathbb{NR}_N = \{x^N : x \in \mathbb{Z}_{N^2}^*\}$ is of order $\varphi(N)$. We first recall the following lemma due to [OSY21], where they introduce a distributed discrete logarithm algorithm for a subset of $\mathbb{Z}_{N^2}^*$.

Lemma 4.5 ([OSY21]). *There exists an algorithm* $\mathsf{DDLog}_N(g)$ *for which the following holds: Let* $g_0, g_1 \in \mathbb{Z}_{N^2}^*$*, such that* $g_0 = g_1(1+N)^x \pmod{N^2}$*. If* $z_0 = \mathsf{DDLog}_N(g_0)$ *and* $z_1 = \mathsf{DDLog}_N(g_1)$*, then* $z_0 - z_1 = x \pmod{N}$*. More precisely,* $\mathsf{DDLog}_N(g)$ *works as follows:*

- $\mathsf{DDLog}_N(g)$
  - *Write* $g = h + h'N$*, where* $h, h' < N$*, using the division algorithm.*
  - *Output* $z = h'h^{-1} \pmod{N}$.

Construction idea: Let g_q be a generator of QR_p, and G, H two random elements of $\mathbb{NR}_{2N}$. Party 0 samples $r' \xleftarrow{\$} \mathbb{Z}_q$ and sets $r := g_q^{r'} \pmod{p}$. It then computes its public key as a Paillier-ElGamal encryption of r', i.e., $\mathsf{pk}_0 = (G^t, H^t \cdot (1+N)^{r'})$. Party 1 holding a bit $k \in \{0,1\}$ computes its public key as a Pedersen commitment of k over $\mathbb{NR}_{2N}$, i.e., $\mathsf{pk}_1 = G^s \cdot H^k$. Following the same computations as in the Bellare-Micali non-interactive OT protocol, at the end of the protocol, the two parties derive two pairs (A, r) and (B, k), where $B = A \cdot (1+N)^{r' \cdot k}$. In other words, the parties derive multiplicative shares of $(1+N)^{r' \cdot k}$. We now use the DDLog_N algorithm of Lemma 4.5 to locally convert these multiplicative shares to subtractive shares. More precisely, Party 0 and Party 1 respectively compute $\hat{a} \leftarrow \mathsf{DDLog}_N(A)$ and $\hat{\alpha} \leftarrow \mathsf{DDLog}_N(B)$, where $\hat{a} - \hat{\alpha} = r' \cdot k \pmod{N}$. Note that we can furthermore view these elements as the shares of $r' \cdot k$ over the integers, if it holds that $r' \cdot k \ll N$. Therefore, setting $q < N/2^\lambda$, it is now possible for Party 0 and Party 1 to respectively compute $a = g_q^{\hat{a}}$ and $\alpha = g_q^{\hat{\alpha}}$ such that $\alpha = a \cdot r^{-k} \pmod{p}$.
Recall that the goal of the protocol is to derive a pair of Naor-Reingold master secret key

and constrained key for a constraint vector $\mathsf{k} \in \{0,1\}^n$ that is a wPRF key. We need to therefore derive n such correlated elements. To do so, it is enough for Party 1 to publish n Pedersen commitments to each bit of its wPRF key k. The details of the construction are presented in Construction 4.4.

Construction 4.4: PK-PCF for OT Correlations from sparse power-DDH and DCR

Requirements and notation:

- Let $F : \mathcal{K} \times \mathcal{X} \to \{0,1\}$ be an IPM-wPRF with respect to an IPM set S and predicate functions f, g (See Definition 4.1).
- Let p be a safe prime, i.e., $p = 2q + 1$ for a prime q, and $N = PQ$ for primes P and Q.
- Let DDLog_N be distributed discrete logarithm algorithm as in Lemma 4.5.
- Let H be a hash function modeled as a random oracle.

Algorithms:

▶ $\mathsf{PCF.Setup}(1^\lambda)$:

1. $(\mathbb{G}, g, p) \xleftarrow{\$} \mathsf{GenPar}(1^\lambda)$.
2. Sample a generator g_q of QR_p.
3. Sample $G' \xleftarrow{\$} \mathbb{Z}_{N^2}$, and set $G \leftarrow (G')^{2N} \pmod{N^2}$.
4. Sample $d \xleftarrow{\$} \mathbb{Z}_{N^2}$, and set $H \leftarrow G^d \pmod{N^2}$.
5. Output $\mathsf{pp} = (\mathbb{G}, p, g_q, (G, H), F)$.

▶ $\mathsf{PCF.Gen}_0(1^\lambda)$:

1. Sample $h, h' \xleftarrow{\$} \mathbb{G}$.
2. Sample $r' \xleftarrow{\$} \mathbb{Z}_q$, and set $r := g_q^{r'} \pmod{p}$.
3. For $s \in S$, compute and set $h_s := h^{r^s}$, and $h'_s := (h')^{r^s}$.
4. Sample $t \xleftarrow{\$} \mathbb{Z}_N$ and compute $(c_0, c_1) = (G^t, H^t \cdot (1+N)^{r'})$.
   // Paillier-ElGamal encryption of r'.
5. Set $\mathsf{sk}_0 = (h, h', r, t)$, and $\mathsf{pk}_0 = \left\{(c_0, c_1), (h_s)_{s\in S}, (h'_s)_{s\in S}\right\}$.
6. Output $(\mathsf{sk}_0, \mathsf{pk}_0)$.

▶ $\mathsf{PCF.Gen}_1(1^\lambda)$:

1. Sample $\hat{\mathsf{k}} \xleftarrow{\$} \mathcal{K}$.
2. Compute $\mathsf{k} \leftarrow g(\hat{\mathsf{k}})$.
3. For $i \in [n]$:
   a) parse $\mathsf{k} = (k_1, \ldots, k_n)$.
   b) sample $s_i \xleftarrow{\$} \mathbb{Z}_N$.
   c) set $\mathsf{com}_i = G^{s_i} \cdot H^{k_i} \pmod{N^2}$.
      // Pedersen commitments of k_i.
4. Set $\mathsf{sk}_1 = (\mathsf{k}, (s_i)_{i\in[n]})$.
5. Set $\mathsf{pk}_1 = (\mathsf{com}_1, \cdots, \mathsf{com}_n)$.
6. Output $(\mathsf{sk}_1, \mathsf{pk}_1)$.

▶ $\mathsf{PCF.KeyDer}(0, \mathsf{sk}_0, \mathsf{pk}_1)$:

1. Parse $\mathsf{sk}_0 = (h, h', r, t)$, and $\mathsf{pk}_1 = (\mathsf{com}_1, \cdots, \mathsf{com}_n)$.
2. For $i \in [n]$:
   a) compute $A_i = \mathsf{com}_i^t \pmod{N^2}$.
      $//A_i = G^{s_i \cdot t} \cdot H^{k_i \cdot t} \pmod{N^2}$.
   b) compute $\hat{a}_i \leftarrow \mathsf{DDLog}_N(A_i)$.
      $//\hat{a}_i = (r' \cdot k_i)_0$.
   c) set $a_i := g_q^{\hat{a}_i} \pmod{p}$.
      $//a_i = g_q^{(r' \cdot k_i)_0}$.
3. Set $\mathbf{a} = (a_1, \cdots, a_n)$.
4. Output $k_0 = (h, h', \mathbf{a})$.

▶ $\mathsf{PCF.KeyDer}(1, \mathsf{sk}_1, \mathsf{pk}_0)$:

1. Parse $\mathsf{sk}_1 = (\mathsf{k}, (s_i)_{i \in [n]})$, and $\mathsf{pk}_0 = \left\{(c_0, c_1), (h_s, h'_s)_{s \in S}\right\}$.
2. For $i \in [n]$:
   a) compute $B_i = c_0^{s_i} \cdot c_1^{k_i} \pmod{N^2}$.
      $//B_i = A_i \cdot (1+N)^{r' \cdot k_i}$.
   b) compute $\hat{\alpha}_i = \mathsf{DDLog}_N(B_i)$.
      $//\hat{\alpha}_i = (r' \cdot k_i)_1$.
   c) set $\alpha_i := g_q^{\hat{\alpha}_i} \pmod{p}$.
      $//\alpha_i = g_q^{(r' \cdot k_i)_1}$.
3. Set $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$.
4. Output $k_1 = (\boldsymbol{\alpha}, (h_s)_{s \in S}, (h'_s)_{s \in S}, \mathsf{k})$.

▶ $\mathsf{PCF.Eval}(1^\lambda, 0, k_0, \hat{\mathsf{x}})$:

1. Parse $k_0 = (h, h', \mathbf{a})$.
2. Compute $\mathbf{x} \leftarrow f(\hat{\mathsf{x}})$.
3. Compute $r_0 = h^{\prod_{i=1}^{n} a_i^{x_i}}$.
4. Compute $r_1 = h'^{\prod_{i=1}^{n} a_i^{x_i}}$.
5. Output $y_0 \leftarrow (\mathsf{H}(r_0), \mathsf{H}(r_1))$.

▶ $\mathsf{PCF.Eval}(1^\lambda, 1, k_1, \hat{\mathsf{x}})$:

1. Parse $k_1 = (\boldsymbol{\alpha}, (h_s, h'_s)_{s \in S}, \mathsf{k})$.
2. Compute $\mathbf{x} \leftarrow f(\hat{\mathsf{x}})$.
3. Let $s = \langle \mathbf{x}, \mathsf{k} \rangle$, and $b = \mathbb{1}_S(s)$.
4. If $b = 0$, compute $r_\bullet = (h_s)^{\prod_{i=1}^{n} \alpha_i^{x_i}}$.
5. If $b = 1$, compute $r_\bullet = (h'_s)^{\prod_{i=1}^{n} \alpha_i^{x_i}}$.
6. Output $y_1 \leftarrow (b, \mathsf{H}(r_\bullet))$.

Security Analysis

We first state the following lemma. The proof follows from inspection. We refer the reader to the construction idea explained in Section 4.4.3 for more details.

Lemma 4.6 (Correlated Evaluation Keys). *Let* $\mathsf{PCF.Gen}$ *and* $\mathsf{PCF.KeyDer}$ *be the algorithms described in Construction 4.4. And let* $(\mathsf{sk}_\sigma, \mathsf{pk}_\sigma) \leftarrow \mathsf{PCF.Gen}(\sigma)$, *and* $k_\sigma \leftarrow \mathsf{PCF.KeyDer}(\sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma})$ *for* $\sigma \in \{0, 1\}$. *It holds that* $k_0 = (h, h', \boldsymbol{a})$, $k_1 = (\boldsymbol{\alpha}, (h_s)_{s \in S}, (h'_s)_{s \in S}, \mathsf{k})$, *where for an element* $r \in \mathbb{Z}_p^*$:

- $h_s = h^{r^s}$ *and* $h'_s = (h')^{r^s}$ *for all* $s \in S$, *and*
- $\boldsymbol{a} = (a_1, \ldots, a_n)$ *and* $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ *such that* $\alpha_i = a_i \cdot r^{-k} \pmod{p}$ *for all* $i \in [n]$.

We also state the following remark regarding the randomness space of operations over $\mathbb{NR}_{2N}$ in our construction:

Remark 4.4. *The key generation and derivation phases of our PK-PCF protocol contain operations over the subgroup $\mathbb{NR}_{2N}$ that is of order $P'Q'$. However, since the two parties should perform their computations obliviously to the factorization of N, they sample their required randomness for computing commitments and encryptions from $\mathbb{Z}_N$ instead of $\mathbb{Z}_{P'Q'}$. This does not change the distribution of the resulting elements over $\mathbb{NR}_{2N}$, since*

$$\Delta(\{r \pmod{P'Q'} : r \xleftarrow{\$} \mathbb{Z}_N\}, \mathcal{U}(\mathbb{Z}_{P'Q'})) = \frac{N \pmod{P'Q'}}{N} = \frac{2P' + 2Q' + 1}{4P'Q' + 2P' + 2Q' + 1} \leq \frac{1}{2^\lambda},$$

where $\Delta(\mathcal{D}_1, \mathcal{D}_2)$ denotes the statistical distance between the distributions $\mathcal{D}_1$ and $\mathcal{D}_2$.

We now prove the security of our PK-PCF construction.

Theorem 4.7 (PCF Security)**.** *Assuming the hardness of sparse power-DDH (Definition 2.4) and DCR (Definition 2.1), Construction 4.4 is a secure public-key pseudorandom correlation function for OT correlations.*

Proof. We prove that our construction satisfies the properties of a PK-PCF.
Pseudorandom OT-Correlated Outputs. We consider the following sequence of hybrid games:
Hybrid $\mathcal{H}_0$: This is the game $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda)$, where the view of an adversary consists of $(1^\lambda, \mathsf{pk}_0, \mathsf{pk}_1, (x^{(i)}, y_0^{(i)}, y_1^{(i)})_{i\in[N(\lambda)]})$, where each pair $(y_0^{(i)}, y_1^{(i)})$ is generated by running the PCF evaluation algorithms of both parties on $x^{(i)}$ for $i \in [N(\lambda)]$.
Hybrid $\mathcal{H}_1$: The goal of this hybrid is showing that the public-keys do not reveal any information. We break this hybrid into following sub-hybrids:

- **Hybrid** $\mathcal{H}_1^1$: In this hybrid, after running $(\mathsf{sk}_\sigma, \mathsf{pk}_\sigma) \leftarrow \mathsf{PCF.Gen}(1^\lambda, \mathsf{pp}, \sigma)$, for $\sigma \in \{0, 1\}$, we do the following instead of running the key derivation algorithm:
  - Parse $\mathsf{sk}_0 = (h, h', r, t)$, $\mathsf{pk}_0 = \{(c_0, c_1), (h_s)_{s\in S}, (h'_s)_{s\in S}\}$, and $\mathsf{sk}_1 = (\mathsf{k}, (s_i)_{i\in[n]})$, $\mathsf{pk}_1 = (\mathsf{com}_1, \ldots, \mathsf{com}_n)$.
  - Sample $\boldsymbol{a} = (a_1, \ldots, a_n) \xleftarrow{\$} (\mathbb{Z}_p^*)^n$.
  - For $i \in [n]$, compute $\alpha_i = a_i \cdot r^{-\mathsf{k}_i}$, and set $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$.
  - Set $k_0 = (h, h', \boldsymbol{a})$, and $k_1 = (\boldsymbol{\alpha}, (h_s)_{s\in S}, (h'_s)_{s\in S}, \mathsf{k})$.

  Hybrid $\mathcal{H}_1^1$ is indistinguishable from Hybrid $\mathcal{H}_0$ due to the correctness of the key derivation algorithm (see Lemma 4.6).

- **Hybrid** $\mathcal{H}_1^2$: In this hybrid, we run $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda, \mathsf{pp}, 1)$. Alternatively, we replace $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{PCF.Gen}(1^\lambda, \mathsf{pp}, 0)$ by the following:
  - Sample $h, h' \xleftarrow{\$} \mathbb{G}$.
  - Sample $r' \xleftarrow{\$} \mathbb{Z}_q$, and set $r := g_q^{r'} \pmod{p}$.
  - For $s \in S$, compute and set $h_s := h^{r^s}$, and $h'_s := (h')^{r^s}$.
  - Sample $t \xleftarrow{\$} \mathbb{Z}_N$ and compute $(c_0, c_1) = (G^t, H^t \cdot (1+N)^0)$.
    // Instead of encrypting r'.
  - Set and output $\mathsf{sk}_0 = (h, h', r, t)$, and $\mathsf{pk}_0 = \left\{(c_0, c_1), (h_s)_{s\in S}, (h'_s)_{s\in S}\right\}$.

We do the remaining of the experiment as in Hybrid $\mathcal{H}_1^1$. Hybrid $\mathcal{H}_1^2$ remains computationally indistinguishable to $\mathcal{H}_1^1$ due to the CPA security of Paillier-ElGamal encryption scheme under the DCR assumption.

- **Hybrid** $\mathcal{H}_1^3$: In this hybrid, we do exactly as in $\mathcal{H}_1^2$ for all the steps, except for $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda, \mathsf{pp}, 1)$ which is replaced by the following:
  - Sample $\hat{\mathsf{k}} \xleftarrow{\$} \mathcal{K}$ and compute $\mathsf{k} \leftarrow g(\hat{\mathsf{k}})$.
  - For $i \in [n]$:
    - parse $\mathsf{k} = (k_1, \ldots, k_n)$.
    - sample $s_i \xleftarrow{\$} \mathbb{Z}_N$, and compute $\mathsf{com}_i = G^{s_i} \cdot H^0 \pmod{N^2}$.
      // Instead of committing to k_i.
  - Set and output $\mathsf{sk}_1 = (\mathsf{k}, (s_i)_{i\in[n]})$, and $\mathsf{pk}_1 = (\mathsf{com}_1, \cdots, \mathsf{com}_n)$.

  Hybrid $\mathcal{H}_1^3$ is perfectly indistinguishable from $\mathcal{H}_1^2$ due to the perfectly-hiding property of Pedersen commitments.

Combining all the above arguments, Hybrid $\mathcal{H}_1$ is indistinguishable from $\mathcal{H}_0$. Note that in Hybrid $\mathcal{H}_1$ the key generation and derivation algorithms are no more used, and the view of the adversary contains simulated public keys pk_0 and pk_1 that are independent of any secrets.
Hybrid $\mathcal{H}_2$: This is the same as Hybrid $\mathcal{H}_1$ except that in this hybrid, instead of running $y_\sigma \leftarrow \mathsf{PCF.Eval}(\sigma, k_\sigma, x^{(i)})$, for $\sigma \in \{0,1\}$, we run $(y_0^{(i)}, y_1^{(i)}) \xleftarrow{\$} \mathsf{OT}(1^\lambda)$. We claim that Hybrids $\mathcal{H}_2$ and $\mathcal{H}_1$ are computationally indistinguishable. Since the two public keys are simulated in Hybrid $\mathcal{H}_1$, this hybrid is in essence the experiment $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda)$ of the *secret-key* PCF construction that is obtained by plugging in our Naor-Reingold IPM-CPRF into the transformation presented in Construction 4.3. Consequently, since the Naor-Reingold CPRF is a secure constrained PRF (see Theorem 4.1), the PCF construction generates pseudorandom OT-correlated outputs (see Theorem 4.2). Therefore Hybrids $\mathcal{H}_2$ and $\mathcal{H}_1$ are indistinguishable.
Hybrid $\mathcal{H}_3$: This is the same as Hybrid $\mathcal{H}_2$ except that here, we undo the changes from hybrids $\mathcal{H}_1^3$ and $\mathcal{H}_1^2$, and replace them with

$$(\mathsf{sk}_\sigma, \mathsf{pk}_\sigma) \leftarrow \mathsf{PCF.Gen}(1^\lambda, \mathsf{pp}, \sigma), \text{ for } \sigma \in \{0,1\} .$$

This hybrid remains indistinguishable to Hybrid $\mathcal{H}_2$ due to the same arguments explained in hybrids $\mathcal{H}_1^2$ and $\mathcal{H}_1^3$. Note that the two evaluation keys are not used in $\mathcal{H}_2$ anymore, and therefore, it does not change the view of the adversary that we do not generate them in this hybrid.

Note that hybrid $\mathcal{H}_3$ has the same distribution as $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda)$. This concludes the proof of pseudorandom OT-correlated output of the construction.

Security. We prove the security for each $\sigma \in \{0,1\}$.

- For $\sigma = 0$, consider the following sequence of hybrid games:

  Hybrid $\mathcal{H}_0$: This is the game $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma=0,0}(\lambda)$, where the view of party 0, considered as the adversary, consists of

$$(1^\lambda, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{sk}_0, (x^{(i)}, y_0^{(i)}, y_1^{(i)})_{i\in[N(\lambda)]}),$$

where $y_1^{(i)} \leftarrow \mathsf{PCF.Eval}(1, k_1, x^{(i)})$ for all $i \in [N(\lambda)]$.

Hybrid $\mathcal{H}_1$: This is the same as hybrid $\mathcal{H}_0$, but here, we remove the steps

$$(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{PCF.Gen}_1(1^\lambda), \text{ and } k_1 \leftarrow \mathsf{PCF.KeyDer}(1, \mathsf{sk}_1, \mathsf{pk}_0),$$

and generate pk_1 and k_1 as follows:

- Let $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{PCF.Gen}_0(1^\lambda)$.
- Parse $\mathsf{sk}_0 = (h, h', r, t)$, and $\mathsf{pk}_0 = \{(c_0, c_1), (h^{r^s})_{s\in S}, ((h')^{r^s})_{s\in S}\}$.

  // To generate pk_1:
- For $i \in [n]$:
  1. Sample $s_i \xleftarrow{\$} \mathbb{Z}_N$.
  2. Compute $\mathsf{com}_i = G^{s_i} \cdot H^0$.
- Output $\mathsf{pk}_1 = (\mathsf{com}_1, \ldots, \mathsf{com}_n)$.

  // To generate k_1:
- Run $k_0 \leftarrow \mathsf{PCF.KeyDer}(0, \mathsf{sk}_0, \mathsf{pk}_1)$.
- Parse $k_0 = (h, h', \boldsymbol{a})$.
- Sample $\hat{\mathsf{k}} \xleftarrow{\$} \mathcal{K}$, and compute $\mathsf{k} = g(\hat{\mathsf{k}})$.
- For $i \in [n]$, compute and set $\alpha_i = a_i \cdot r^{-k_i} \pmod p$.
- Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$.
- Set $k_1 = (\boldsymbol{\alpha}, (h_s)_{s\in S}, (h'_s)_{s\in S}, \mathsf{k})$.

Note that they key k_1 is of the same form as in $\mathcal{H}_0$ due to the correctness of the key derivation algorithm (see Lemma 4.6). Also, here, differently from Hybrid $\mathcal{H}_0$, the public key pk_1 contains commitments that are independent from the evaluation key k_1. This public key retains the same distribution as in $\mathcal{H}_0$, since the Pedersen commitment scheme is perfectly hiding. Therefore, Hybrid $\mathcal{H}_1$ remains indistinguishable from Hybrid $\mathcal{H}_0$.

Hybrid $\mathcal{H}_2$: This is the same as Hybrid $\mathcal{H}_1$ except that here, we compute the key $k_0 \leftarrow \mathsf{PCF.KeyDer}(0, \mathsf{sk}_0, \mathsf{pk}_1)$, and then for each $i \in [N(\lambda)]$, we replace the step $y_1^{(i)} \leftarrow \mathsf{PCF.Eval}(1, k_1, x^{(i)})$ in $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma=0,0}(\lambda)$ with

$$y_0^{(i)} \leftarrow \mathsf{PCF.Eval}(0, k_0, x^{(i)}), \text{ and } y_1^{(i)} \leftarrow \mathsf{RSample}(1^\lambda, 0, y_0^{(i)}).$$

Note that in Hybrid $\mathcal{H}_1$, the public key pk_1 was simulated and therefore, this hybrid is in essence the experiment $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma=0,0}(\lambda)$ of a *secret-key* PCF that is constructed from the Naor-Reingold CPRF. Regarding the security of the Naor-Reingold CPRF (Theorem 4.1) and the CPRF-to-PCF transformation (Theorem 4.2), hybrids $\mathcal{H}_2$ and $\mathcal{H}_1$ are indistinguishable.

Hybrid $\mathcal{H}_3$: This is the same as hybrid $\mathcal{H}_2$ except that here, we undo the changes that we made in hybrid $\mathcal{H}_1$ and bring back the algorithm $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda, \mathsf{pp}, 1)$. The evaluation key k_1 is not used in hybrid $\mathcal{H}_2$ anymore, and therefore, removing it does not change the view of party 0. This hybrid remains indistinguishable to hybrid $\mathcal{H}_2$ for the same arguments explained in hybrid $\mathcal{H}_1$.
Note that Hybrid $\mathcal{H}_3$ has the same distribution as $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma=0,1}(\lambda)$. This concludes the proof of security of our PK-PCF construction against party 0.

- For $\sigma = 1$, consider the following sequence of hybrid games:

  Hybrid $\mathcal{H}_0$: This is the game $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma=1,0}(\lambda)$.

  Hybrid $\mathcal{H}_1$: This is the same as hybrid $\mathcal{H}_0$, but here, we remove the steps

  $$(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{PCF.Gen}_0(1^\lambda), \quad \text{and} \quad k_0 \leftarrow \mathsf{PCF.KeyDer}(0, \mathsf{sk}_0, \mathsf{pk}_1),$$

  and generate pk_0 and k_0 as explained in the following.

  - Let $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{PCF.Gen}_1(1^\lambda)$.
  - Parse $\mathsf{sk}_1 = (\mathsf{k}, (s_i)_{i\in[n]})$, and $\mathsf{pk}_1 = (\mathsf{com}_1, \ldots, \mathsf{com}_n)$.

    // To generate pk_0:
  - Sample $h, h' \xleftarrow{\$} \mathbb{G}$.
  - Sample $r \xleftarrow{\$} \mathbb{Z}_p^*$.
  - For $s \in S$, compute and set $h_s := h^{r^s}$ and $h'_s := (h')^{r^s}$.
  - Compute $(c_0, c_1) = (G^t, H^t \cdot (1+N)^0)$, where $t \xleftarrow{\$} \mathbb{Z}_N$.
  - Set $\mathsf{pk}_0 = ((c_0, c_1), (h_s)_{s\in S}, (h'_s)_{s\in S})$.

    // To generate k_0:
  - Run $k_1 \leftarrow \mathsf{PCF.KeyDer}(1, \mathsf{sk}_1, \mathsf{pk}_0)$.
  - Parse $k_1 = (\boldsymbol{\alpha}, (h_s)_{s\in S}, (h'_s)_{s\in S}, \mathsf{k})$.
  - For $i \in [n]$, compute and set $a_i = \alpha_i \cdot r^{k_i} \pmod p$.
  - Let $\boldsymbol{a} = (a_1, \ldots, a_n)$, and set $\mathsf{k}_0 = (h, h', \boldsymbol{a})$.

  Note that they key k_0 is of the same form as in $\mathcal{H}_0$ due to the correctness of the key derivation algorithm (see Lemma 4.6). Also, here, differently from Hybrid $\mathcal{H}_0$, the public key pk_0 contains ciphertexts that are independent from the evaluation key k_0. This public key retains the same distribution as in $\mathcal{H}_0$, since the Paillier-ElGamal encryption scheme is CPA-secure. As a result, Hybrid $\mathcal{H}_1$ remains indistinguishable from Hybrid $\mathcal{H}_0$.

  Hybrid $\mathcal{H}_2$: This is the same as Hybrid $\mathcal{H}_1$ except that here, we compute the key $k_1 \leftarrow \mathsf{PCF.KeyDer}(0, \mathsf{sk}_1, \mathsf{pk}_0)$, and then replace each $y_0^{(i)} \leftarrow \mathsf{PCF.Eval}(0, k_0, x^{(i)})$ in $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma=1,0}(\lambda)$ with

  $$y_1^{(i)} \leftarrow \mathsf{PCF.Eval}(1, k_1, x^{(i)}), \text{ and } y_0^{(i)} \leftarrow \mathsf{RSample}(1^\lambda, 1, y_1^{(i)}).$$

  Similarly to the security proof for $\sigma = 0$, hybrid $\mathcal{H}_1$ is in essence the experiment $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma=1,0}(\lambda)$ of a *secret-key* PCF that is constructed from the Naor-Reingold CPRF. Regarding the security of the Naor-Reingold CPRF (Theorem 4.1) and the CPRF-to-PCF transformation (Theorem 4.2), hybrids $\mathcal{H}_2$ and $\mathcal{H}_1$ are indistinguishable.

  Hybrid $\mathcal{H}_3$: This is the same as hybrid $\mathcal{H}_2$ except that here, we undo the changes that we made in hybrid $\mathcal{H}_1$ and bring back the algorithms

  $$(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{PCF.Gen}_0(1^\lambda), \text{ and } (\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{PCF.Gen}_1(1^\lambda).$$

  The evaluation key k_0 is not used in hybrid $\mathcal{H}_2$ anymore, and therefore, removing it does not change the view of the adversary (party 1). This hybrid remains indistinguishable to hybrid $\mathcal{H}_2$ for the same arguments explained in hybrid $\mathcal{H}_1$.

  Note that hybrid $\mathcal{H}_3$ has the same distribution as $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma=1,1}(\lambda)$. This concludes the proof of security of our PK-PCF construction against party 1.

□

4.4.4 Reducing The Public Keys Size to $\mathcal{O}(n^{2/3})$

In the public-key PCF of Construction 4.4, the public keys are of the form

$$\mathsf{pk}_0 = \left\{(c_0, c_1), (h_s)_{s\in S}, (h'_s)_{s\in S}\right\}, \text{ and } \mathsf{pk}_1 = (\mathsf{com}_1, \cdots, \mathsf{com}_n),$$

where (c_0, c_1) is a Paillier-ElGamal ciphertext and $(\mathsf{com}_1, \cdots, \mathsf{com}_n)$ are n Pedersen commitments over $\mathbb{NR}_{2N}$. Regarding the key sizes, pk_0 contains 2 elements of $\mathbb{NR}_{2N}$, while pk_1 contains n such elements. In this section we aim to find a better balance for the size of pk_0 and pk_1. The key observation is that pk_1 that includes a list of Pedersen commitments can be easily made compact using generalized Pedersen commitments which allow committing to n different values by generating a single commitment. However, this must be done while maintaining the correct correlation of derived PCF keys for both parties. In the following, we explain how we achieve the correctness by including more Paillier-ElGamal ciphertexts in pk_0 while reducing the total size of the public keys. Let $0 < m \leq n$ be a block size. The idea is the following:

- Let $G, H_1, \ldots, H_m$ be random elements of $\mathbb{NR}_{2N}$.
- Party 1 divides the wPRF key k into consecutive subvectors $\mathsf{k}_1, \ldots, \mathsf{k}_\delta$, each of length m. It then commits to each subvector k_u by generating generalized Pedersen commitments $\mathsf{com}_u = G^{s_u} \cdot \prod_{j=1}^{m} H_j^{k_u^{(j)}}$, for all $u \in [\delta]$.
- Party 0 generates m^2 Paillier-ElGamal encryptions of r' with randomness reuse as follows:

$$\begin{aligned}
(c_1^0, c_1^1, \ldots, c_1^m) &= (G^{t_1}, H_1^{t_1} \cdot (1+N)^{r'}, H_2^{t_1}, \ldots, H_m^{t_1}) \\
(c_2^0, c_2^1, \ldots, c_2^m) &= (G^{t_2}, H_1^{t_2}, H_2^{t_2} \cdot (1+N)^{r'}, \ldots, H_m^{t_2}) \\
&\vdots \\
(c_m^0, c_m^1, \ldots, c_m^m) &= (G^{t_m}, H_1^{t_m}, H_2^{t_m}, \ldots, H_m^{t_m} \cdot (1+N)^{r'}).
\end{aligned}$$

- Party 0 publishes the m^2 ciphertexts $(c_i^0, c_i^j)_{i,j\in[m]}$ as a part of pk_0 and party 1 publishes the n/m generalized commitments $\mathsf{com}_1, \ldots, \mathsf{com}_\delta$ as a part of pk_1.

By inspection, one can see that for each $u \in [\delta]$ and each $v \in [m]$, it holds that

$$\mathsf{com}_u^{t_v} \cdot (1+N)^{r' \cdot k_u^{(v)}} = (c_v^0)^{s_u} \cdot \prod_{j=1}^{m} (c_v^j)^{k_u^{(j)}}.$$

Therefore, the two parties can first derive multiplicative shares of $(1+N)^{r' \cdot k_u^{(v)}}$, and as before, after applying the DDLog algorithm over their shares and mapping the result in $\mathbb{G}$, they can compute their PCF keys. Doing as explained above yields publishing $m^2 + 2m + n/m$ elements (including $G, H_1, \ldots, H_m$ elements of the public parameters), where n denotes the length of the wPRF key k. Minimizing with respect to m results in $\mathcal{O}(n^{2/3})$ elements. The resulting optimized public-key PCF is detailed in Construction 4.5.

Construction 4.5: PK-PCF for OT Correlations from sparse power-DDH and DCR (with compressed public keys)

Requirements and notation:

- Let $F : \mathcal{K} \times \mathcal{X} \to \{0,1\}$ be an IPM-wPRF with respect to an IPM set S and predicate functions f, g (See Definition 4.1).
- Let p be a safe prime, i.e., $p = 2q + 1$ for a prime q, and $N = PQ$ for primes P and Q.
- Let DDLog_N be the distributed discrete logarithm algorithm as in Lemma 4.5.
- Let H be a hash function modeled as a random oracle.

Algorithms:

▶ $\mathsf{PCF.Setup}(1^\lambda)$:

1. $(\mathbb{G}, g, p) \xleftarrow{\$} \mathsf{GenPar}(1^\lambda)$.
2. Sample a generator g_q of QR_p.
3. Sample $G' \xleftarrow{\$} \mathbb{Z}_{N^2}$, and set $G \leftarrow (G')^{2N} \pmod{N^2}$.
4. For $i \in [m]$, sample $d_i \xleftarrow{\$} \mathbb{Z}_{N^2}$, and set $H_i \leftarrow G^{d_i} \pmod{N^2}$.
5. Output $\mathsf{pp} = (\mathbb{G}, p, g_q, (G, H_1, \ldots, H_m), F)$.

▶ $\mathsf{PCF.Gen}_0(1^\lambda)$:

1. Sample $h, h' \xleftarrow{\$} \mathbb{G}$.
2. Sample $r' \xleftarrow{\$} \mathbb{Z}_q$, and $r := g_q^{r'} \pmod{p}$.
3. For $i \in [m]$ do:
   a) Sample $t_i \xleftarrow{\$} \mathbb{Z}_N$.
   b) Compute $\mathbf{c}_i = (c_i^0, c_i^1, \ldots, c_i^m)$, where $c_i^0 = G^{t_i}$, and $c_i^i = H_i^{t_i} \cdot (1+N)^{r'}$, and $c_i^j = H_j^{t_i}$ for all $j \neq i \in [m]$.

   // m^2 Paillier-ElGamal encryptions of r'.
4. For $s \in S$, set $h_s := h^{r^s}$, and $h'_s := (h')^{r^s}$.
5. Set $\mathsf{sk}_0 = (h, h', r, (t_i)_{i\in[m]})$.
6. Set $\mathsf{pk}_0 = \left\{(c_i^0, (c_i^j))_{i,j\in[m]}, (h_s)_{s\in S}, (h'_s)_{s\in S}\right\}$.
7. Output $(\mathsf{sk}_0, \mathsf{pk}_0)$.

▶ $\mathsf{PCF.Gen}_1(1^\lambda)$:

1. Sample $\hat{\mathsf{k}} \xleftarrow{\$} \mathcal{K}$.
2. Compute $\mathsf{k} \leftarrow g(\hat{\mathsf{k}})$.
3. Partition k into δ subvectors $\mathsf{k}_1, \mathsf{k}_2, \ldots, \mathsf{k}_\delta$ of length m.
4. For $u \in [\delta]$:
   a) Parse $\mathsf{k}_u = (k_u^{(1)}, \ldots, k_u^{(m)})$.
   b) Compute $\mathsf{com}_u = G^{s_u} \cdot \prod_{j=1}^{m} H_j^{k_u^{(j)}}$, where $s_u \xleftarrow{\$} \mathbb{Z}_N$.

   // Generalized Pedersen commitment of k_u.
5. Set $\mathsf{sk}_1 = ((\mathsf{k}_u)_{u\in[\delta]}, (s_u)_{u\in[\delta]})$.
6. Set $\mathsf{pk}_1 = (\mathsf{com}_1, \ldots, \mathsf{com}_\delta)$.
7. Output $(\mathsf{sk}_1, \mathsf{pk}_1)$.

▶ $\mathsf{PCF.KeyDer}(\sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma})$:

1. If $\sigma = 0$:
   a) Parse $\mathsf{sk}_0 = (h, h', r, (t_i)_{i\in[m]})$.
   b) Parse $\mathsf{pk}_1 = (\mathsf{com}_1, \cdots, \mathsf{com}_\delta)$.
   c) For $u \in [\delta]$, and $v \in [m]$:
      set $A_i = (\mathsf{com}_u)^{t_v}$, where
      $i = m(u-1) + v$.
      $//A_i = G^{t_v \cdot s_u} \cdot \prod_{j=1}^{m} H_j^{k_u^{(j)} \cdot t_v}$.
   d) For $i \in [n]$:
      i. compute $\hat{a}_i \leftarrow \mathsf{DDLog}_N(A_i)$.
      ii. set $a_i := g_q^{\hat{a}_i}$.
   e) Set $\mathbf{a} = (a_1, \cdots, a_n)$.
   f) Set and output $k_0 = (h, h', \mathbf{a})$.

2. If $\sigma = 1$:
   a) Parse $\mathsf{sk}_1 = ((\mathsf{k}_u)_{u\in[\delta]}, (s_u)_{u\in[\delta]})$, and
      $\mathsf{pk}_0 = \left\{(c_i^0, (c_i^j))_{i,j\in[m]}, (h_s)_{s\in S}, (h'_s)_{s\in S}\right\}$.
   b) For $u \in [\delta]$, and $v \in [m]$:
      set $B_i = (c_v^0)^{s_u} \cdot \prod_{j=1}^{m} (c_v^j)^{k_u^{(j)}}$, where
      $i = m(u-1) + v$.
      $//B_i = G^{t_v \cdot s_u} \cdot \prod_{j=1}^{m} H_j^{k_u^{(j)} \cdot t_v} \cdot (1+N)^{r' \cdot k_u^{(v)}}$.
   c) For $i \in [n]$:
      i. compute $\hat{\alpha}_i \leftarrow \mathsf{DDLog}_N(B_i)$.
      ii. set $\alpha_i = g_q^{\hat{\alpha}_i} \pmod p$.
   d) Set $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$.
   e) Output $k_1 = (\boldsymbol{\alpha}, (h_s)_{s\in S}, (h'_s)_{s\in S}, \mathsf{k})$.

▶ $\mathsf{PCF.Eval}(1^\lambda, \sigma, k_\sigma, \hat{\mathsf{x}} = (\hat{x}_1, \ldots, \hat{x}_n))$:

1. If $\sigma = 0$:
   a) Parse $k_0 = (h, h', \mathbf{a})$.
   b) Let $\mathbf{x} = (x_1, \ldots, x_n) \leftarrow f(\hat{\mathsf{x}})$.
   c) Compute $r_0 = h^{\prod_{i=1}^{n} a_i^{x_i}}$.
   d) Compute $r_1 = h'^{\prod_{i=1}^{n} a_i^{x_i}}$.
   e) Output $y_0 \leftarrow (\mathsf{H}(r_0), \mathsf{H}(r_1))$.

2. If $\sigma = 1$:
   a) Parse $k_1 = (\boldsymbol{\alpha}, (h_s)_{s\in S}, (h'_s)_{s\in S}, \mathsf{k})$.
   b) Let $\mathbf{x} = (x_1, \ldots, x_n) \leftarrow f(\hat{\mathsf{x}})$.
   c) Let $s = \langle \mathbf{x}, \mathsf{k} \rangle$, and $b = \mathbb{1}_S(s)$.
   d) If $b = 0$, compute $r_\bullet = (h_s)^{\prod_{i=1}^{n} \alpha_i^{x_i}}$.
   e) If $b = 1$, compute $r_\bullet = (h'_s)^{\prod_{i=1}^{n} \alpha_i^{x_i}}$.
   f) Set and output $y_1 \leftarrow (b, \mathsf{H}(r_\bullet))$.

The security analysis of this construction is similar to that of the public-key PCF presented in Construction 4.4 (See the proof of Theorem 4.7), where we now leverage the perfectly-hiding property of generalized Pedersen commitments and semantic security of Paillier-ElGamal ciphertexts with randomness reuse over $\mathbb{NR}_{2N}$.

Conclusion and Open Problems

In this thesis we demonstrate connections between constrained PRFs and two protocols that were introduced in the context of secure computation: homomorphic secret sharing and pseudorandom correlation functions.

In Chapter 3, we provided transformations from two HSS extensions to constrained PRFs. More specifically, we showed (1) how HSS with simulatable memory values implies constrained PRFs that support inner-product predicates, and (2) how staged-HSS can be used to construct CPRFs for NC^1 predicates. These transformations immediately imply new CPRF constructions from various assumptions, and initiate new directions for future constructions of CPRFs.
It seems to us that these transformations could yield CPRFs for broader classes of constraints. More precisely, consider a family of circuits for which there exists a polynomial-size representation where, given this representation of a circuit in this family and an input, one can evaluate the output by doing only addition and multiplication with constants (where the input is considered a constant). When using HSS to generate constrained keys for such constraints, it suffices to generate "fake" memory values (additive shares) for a given circuit and include them in a constrained key. Given memory values of the circuit representation, one can linearly compute (additive shares of) the output of the circuit on different inputs. As a result, HSS with memory values suffices for constructing CPRFs for such families of constraints.
Contrary to HSS with simulatable memory values, staged-HSS protocols yield CPRFs that admit families of circuits which do not have a polynomial-size *linear* representation in the sense that is explained above. Such families could potentially include P/poly circuits. Although this observation does not yield new instantiations of CPRFs for classes larger than NC^1, [1] it can be still viewed as a new path to constructing CPRFs for broader classes of constraints.

We find many interesting problems left to be solved regarding CPRFs and their connections with HSS protocols. We start by revisiting the long-standing problem of achieving collusion-resistant constrained PRFs for expressive families of constraints in the standard model.

Question 1. *Can we construct collusion-resistant constrained PRFs from HSS protocols? Alternatively stated, what are the specific properties that, if satisfied by an HSS protocol, imply collusion-resistant CPRFs?*

[1] since we do not have (staged-)HSS protocols for classes larger than NC^1.

A question that can be considered as a counterpart to Question 1, is *"What can be achieved, in the context of constrained PRFs, from multi-party HSS protocols?"*. Notably, a primary observation is that this does not inherently lead to multi-key CPRFs. Exploring this connection is left to future works.

Another interesting question, as raised in [BW13, BGI14], concerns the feasibility of *delegations* in constrained PRFs. Using a *delegatable CPRF*, a party holding a constrained key that allows evaluating the PRF on a subset S, can locally generate sub-constrained keys for subsets of S.[1] This property can be viewed as a relaxation of collusion resistance property. To this day, known constructions of delegatable CPRFs use strong tools such as indistinguishability obfuscation ($i\mathcal{O}$) [DDM17, AMN+19, DKN+20]. In this regard, we propose the following question.

Question 2. *Can we construct delegatable constrained PRFs from HSS protocols?*

In Chapter 4, we showed how constrained PRFs that admit (weak) PRFs as constraints can be used to construct pseudorandom correlation functions for OT correlations. We also showed how to instantiate this transformation by slightly changing the Naor-Reingold PRF into a constrained PRF for inner-product membership predicates. We note that our construction of constrained PRF for inner-product membership predicates is inherently delegatable in the sense that given a constrained key $\mathsf{ck}_{(\mathbf{z},S)}$ that evaluates the PRF on all inputs $\mathbf{x}$ such that $\langle \mathbf{x}, \mathbf{z} \rangle \in S$, one can locally generate and output a constrained key $\mathsf{ck}_{(\mathbf{z},S')}$ for $S' \subset S$. A primary question regarding this result is the following:

Question 3. *Can we build (public-key) PCFs for other correlations from CPRFs?*

In the same chapter, we showed how our PCF for OT construction, when instantiated from the Naor-Reingold IPM-CPRF, can allow for non-interactive generation of evaluation keys, resulting in an efficient public-key PCF construction for OT correlations.

The security proof of our (public-key) pseudorandom correlation function for OT correlations relies on the sparse power-DDH assumption. This assumption reduces to solving the discrete logarithm problem. The discrete logarithm problem was shown by Shor [Sho94] to be solvable in polynomial time using quantum computations. Therefore, the security proof of our PCF construction holds as long as an adversary does not have access to quantum computers. Therefore, an interesting question is the following:

Question 4. *Can we build post-quantum-secure PCFs for OT or other correlations from CPRFs?*

For the case of OT correlations, this question can be rephrased to whether we could have constrained PRFs for inner-product membership predicates from post-quantum assumptions.
We observe that the weak PCF security, the notion commonly considered in the literature, is defined with respect to *random inputs*. As a result, when constructing PCFs for OT correlations from CPRFs using our transformation, the security of the resulting PCF boils down to the security of the underlying CPRF *on random inputs*. Thus, for this transformation, a *weak* CPRF suffices.

[1] Delegatable constrained PRFs were also introduced under the name *hierarchical functional PRFs* in [BGI14].

Another interesting question left to future work is the following:

Question 5. *Can we build constrained PRFs from PCFs?*

Finally, we revisit the question of achieving adaptive security for CPRFs supporting expressive families of constraints in the standard model, which was not realized in this thesis. Recently, in [DKN+20], the authors showed possibility of achieving this level of security in the standard model for inner-product predicates using *admissible hash functions* [BB04] and borrowing lattice techniques from [LST18]. Also, the work of [Yan23] proposed a new direction for achieving adaptive security for puncturing predicates, using *explainable hash functions*. It remains an interesting direction to consider applying their techniques to more generalized settings.

Question 6. *Can we have adaptively-secure constrained PRFs for* NC^1 *or* P/poly *in the standard model without relying on* $i\mathcal{O}$*?*

List of Publications

[BCM+24] Dung Bui, Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and **Mahshid Riahinia**. Public-key silent ot and more from constrained Naor-Reingold. *To appear in EUROCRYPT 2024*.
Available at eprint.iacr.org/2024/178.

[CMPR23] Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and **Mahshid Riahinia**. Constrained pseudorandom functions from homomorphic secret sharing. *In EUROCRYPT 2023*.
Available at eprint.iacr.org/2023/387
doi:10.1007/978-3-031-30620-4_7.

[LPR22a] Benoît Libert, Alain Passelègue, and **Mahshid Riahinia**. PointProofs, revisited. *In ASIACRYPT 2022*.
Available at hal.science/hal-03903981/file/asiacrypt2022-final314.pdf
doi:10.1007/978-3-031-22972-5_8.

[LPR22b] Benoît Libert, Alain Passelègue, and **Mahshid Riahinia**. New and improved constructions for partially equivocable public key encryption. *In SCN 2022*.
Available at eprint.iacr.org/2022/1733
doi:10.1007/978-3-031-14791-3_9.

Bibliography

[ABP15] Michel Abdalla, Fabrice Benhamouda, and Alain Passelègue. An algebraic framework for pseudorandom functions and applications to related-key security. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 388–409. Springer, Heidelberg, August 2015. Citations: § 65

[ADOS22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 421–452. Springer, Heidelberg, August 2022. Citations: § 8, 31, 35, and 53

[AHI11] Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. In Bernard Chazelle, editor, *Innovations in Computer Science - ICS 2011, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 45–60. Tsinghua University Press, 2011. Citations: § 16

[AKPW13] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited - new reduction, properties and applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 57–74. Springer, Heidelberg, August 2013. Citations: § 71

[AMN+18] Nuttapong Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Constrained PRFs for NC^1 in traditional groups. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 543–574. Springer, Heidelberg, August 2018. Citations: § 7, 57, 64, 65, and 76

[AMN+19] Nuttapong Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Adaptively single-key secure constrained PRFs for NC^1. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 223–253. Springer, Heidelberg, April 2019. Citations: § 7 and 92

[AR16] Benny Applebaum and Pavel Raykov. Fast pseudorandom functions based on expander graphs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-*

B, Part I, volume 9985 of *LNCS*, pages 27–56. Springer, Heidelberg, October / November 2016. Citations: § 58 and 72

[BB04] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 443–459. Springer, Heidelberg, August 2004. Citations: § 93

[BBD+20] Zvika Brakerski, Pedro Branco, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Constant ciphertext-rate non-committing encryption from standard assumptions. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 58–87. Springer, Heidelberg, November 2020. Citations: § 11

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440–456. Springer, Heidelberg, May 2005. Citations: § 16

[BCG+17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2105–2122. ACM Press, October / November 2017. Citations: § 8, 32, and 38

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Heidelberg, August 2019. Citations: § 10

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080. IEEE Computer Society Press, November 2020. Citations: § 3, 10, 25, and 76

[BCG+22] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 603–633. Springer, Heidelberg, August 2022. Citations: § 76

[BCJ08] Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 449–458. ACM Press, October 2008. Citations: § 12

[BCM+24] Dung Bui, Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Public-key silent ot and more from constrained naor-reingold, to appear in EUROCRYPT 2024. Citations: § 11, 55, 59, 71, and 95

[BCP03] Emmanuel Bresson, Dario Catalano, and David Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its

applications. In Chi-Sung Laih, editor, *ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 37–54. Springer, Heidelberg, November / December 2003. Citations: § 19 and 20

[BDL19] Mihir Bellare, Wei Dai, and Lucy Li. The local forking lemma and its application to deterministic encryption. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 607–636. Springer, Heidelberg, December 2019. Citations: § 12

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992. Citations: § 3

[BG10] Zvika Brakerski and Shafi Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 1–20. Springer, Heidelberg, August 2010. Citations: § 19

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014. Citations: § 3 and 92

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Heidelberg, August 2016. Citations: § 8, 32, 35, 36, and 53

[BGMM20] James Bartusek, Sanjam Garg, Daniel Masny, and Pratyay Mukherjee. Reusable two-round MPC from DDH. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 320–348. Springer, Heidelberg, November 2020. Citations: § 76

[BIP+18] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J. Wu. Exploring crypto dark matter: New simple PRF candidates and their applications. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 699–729. Springer, Heidelberg, November 2018. Citations: § 58, 60, 72, and 76

[BKS19] Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 3–33. Springer, Heidelberg, May 2019. Citations: § 53

[BLW17] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 494–524. Springer, Heidelberg, March 2017. Citations: § 5, 6, and 7

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, 1984. Citations: § 2

[BM90] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 547–557. Springer, Heidelberg, August 1990. Citations: § 79 and 80

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 719–737. Springer, Heidelberg, April 2012. Citations: § 58, 71, 72, and 76

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 264–302. Springer, Heidelberg, November 2017. Citations: § 5, 6, and 7

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 1–30. Springer, Heidelberg, March 2015. Citations: § 7 and 76

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013. Citations: § 3, 5, 6, and 92

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 480–499. Springer, Heidelberg, August 2014. Citations: § 6 and 7

[CC17] Ran Canetti and Yilei Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 446–476. Springer, Heidelberg, April / May 2017. Citations: § 5, 6, and 7

[CCKK21] Jung Hee Cheon, Wonhee Cho, Jeong Han Kim, and Jiseung Kim. Adventures in crypto dark matter: Attacks and fixes for weak pseudorandom functions. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 739–760. Springer, Heidelberg, May 2021. Citations: § 72

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *28th ACM STOC*, pages 639–648. ACM Press, May 1996. Citations: § 11

[CH85] Stephen A. Cook and H. James Hoover. A depth-universal circuit. *SIAM J. Comput.*, 14(4):833–839, 1985. Citations: § 50

[CLT22] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. Threshold linearly homomorphic encryption on $z/2^k z$. *Cryptology ePrint Archive*, 2022. Citations: § 35 and 53

[CMPR23] Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Constrained pseudorandom functions from homomorphic secret sharing. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 194–224. Springer, Heidelberg, April 2023. Citations: § 11, 29, 31, 32, 59, 76, and 95

[CNs07] Jan Camenisch, Gregory Neven, and abhi shelat. Simulatable adaptive oblivious transfer. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 573–590. Springer, Heidelberg, May 2007. Citations: § 16

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, Heidelberg, April / May 2002. Citations: § 19 and 20

[CVW18] Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 577–607. Springer, Heidelberg, August 2018. Citations: § 5, 6, and 7

[DDM17] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Constrained pseudorandom functions for unconstrained inputs revisited: Achieving verifiability and key delegation. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 463–493. Springer, Heidelberg, March 2017. Citations: § 92

[DGS03] Ivan Damgård, Jens Groth, and Gorm Salomonsen. *The Theory and Implementation of an Electronic Voting System*, pages 77–99. Springer US, Boston, MA, 2003. Citations: § 19 and 20

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976. Citations: § 15 and 17

[DKN+20] Alex Davidson, Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Adaptively secure constrained pseudorandom functions in the standard model. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 559–589. Springer, Heidelberg, August 2020. Citations: § 6, 7, 92, and 93

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012. Citations: § 3

[FKOS15] Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to MPC with preprocessing using OT. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 711–735. Springer, Heidelberg, November / December 2015. Citations: § 3

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009. Citations: § 7

[GGM84a] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479. IEEE Computer Society Press, October 1984. Citations: § 21 and 69

[GGM84b] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 276–288. Springer, Heidelberg, August 1984. Citations: § 57

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. 33(4), 1986. Citations: § 2, 3, and 5

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *21st ACM STOC*, pages 25–32. ACM Press, May 1989. Citations: § 3

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. Citations: § 17

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987. Citations: § 9

[Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs. Cryptology ePrint Archive, Report 2000/063, 2000. https://eprint.iacr.org/2000/063. Citations: § 72

[GRWZ20] Sergey Gorbunov, Leonid Reyzin, Hoeteck Wee, and Zhenfei Zhang. Pointproofs: Aggregating proofs for multiple vector commitments. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2007–2023. ACM Press, November 2020. Citations: § 12

[HILL99] Johan HÅstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999. Citations: § 3

[HKKW19] Dennis Hofheinz, Akshay Kamath, Venkata Koppula, and Brent Waters. Adaptively secure constrained pseudorandom functions. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 357–376. Springer, Heidelberg, February 2019. Citations: § 6 and 7

[HSS20] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. *Journal of Cryptology*, 33(4):1732–1786, October 2020. Citations: § 3

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003. Citations: § 65

[JMN23] Thomas Johansson, Willi Meier, and Vu Nguyen. Differential cryptanalysis of mod-2/mod-3 constructions of binary weak prfs. In *2023 IEEE International Symposium on Information Theory (ISIT)*, pages 477–482. IEEE, 2023. Citations: § 72

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 818–829. ACM Press, October 2016. Citations: § 65

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013. Citations: § 3

[LPR22a] Benoît Libert, Alain Passelègue, and Mahshid Riahinia. PointProofs, revisited. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 220–246. Springer, Heidelberg, December 2022. Citations: § 12 and 95

[LPR22b] Benoît Libert, Alain Passelègue, and Mahshid Riahinia. New and improved constructions for partially equivocable public key encryption, SCN 2022. Citations: § 11 and 95

[LPSY15] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 319–338. Springer, Heidelberg, August 2015. Citations: § 3

[LST18] Benoît Libert, Damien Stehlé, and Radu Titiu. Adaptively secure distributed PRFs from LWE. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 391–421. Springer, Heidelberg, November 2018. Citations: § 93

[Nec94] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994. Citations: § 65

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Heidelberg, August 2012. Citations: § 3

[NR95] Moni Naor and Omer Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. In *36th FOCS*, pages 170–181. IEEE Computer Society Press, October 1995. Citations: § 21

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, October 1997. Citations: § 7, 10, 15, 56, 65, and 76

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708. Springer, Heidelberg, October 2021. Citations: § 8, 31, 32, 34, 35, 36, 37, 39, 41, 43, 53, 61, and 80

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999. Citations: § 15 and 19

[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. Springer, Heidelberg, August 1992. Citations: § 20

[PS18] Chris Peikert and Sina Shiehian. Privately constraining and programming PRFs, the LWE way. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 675–701. Springer, Heidelberg, March 2018. Citations: § 5, 6, and 7

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. Citations: § 71

[RS21] Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 687–717, Virtual Event, August 2021. Springer, Heidelberg. Citations: § 8, 31, and 35

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994. Citations: § 92

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997. Citations: § 65

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014. Citations: § 5

[Tsa19] Rotem Tsabary. Fully secure attribute-based encryption for t-CNF from LWE. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 62–85. Springer, Heidelberg, August 2019. Citations: § 5

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 39–56. ACM Press, October / November 2017. Citations: § 3

[Yan23] Rupeng Yang. Privately puncturing PRFs from lattices: Adaptive security and collusion resistant pseudorandomness. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 163–193. Springer, Heidelberg, April 2023. Citations: § 93