Formalisation de preuves de sécurité concrète

Marion Daubignard

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Présentée par

Marion Daubignard

Thèse dirigée par le **Pr. Yassine Lakhnech**

préparée au sein du laboratoire **Verimag**
Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique

Formalisation de preuves de sécurité concrète

Thèse soutenue publiquement le **12 Janvier 2012**,
devant le jury composé de :

Dr. Gilles Barthe
IMDEA, Madrid, Président
Dr. Steve Kremer
LORIA, INRIA, Nancy, Rapporteur
Dr. David Pointcheval
LIENS, CNRS, Paris, Rapporteur
Dr. Bogdan Warinschi
University of Bristol , Rapporteur
Dr. Emmanuel Bresson
Emiraje Systems, Examinateur
Dr. Anupam Datta
Carnegie Mellon University, Examinateur
Dr. Bruce Kapron
University of Victoria , Examinateur
Pr. Yassine Lakhnech
Université de Grenoble, Directeur de thèse
Dr. Pascal Lafourcade
Université de Grenoble, Invité

Remerciements

Construire puis rédiger une thèse est un ouvrage qui se compare peut-être pour un scientifique à ce que l'adolescence représente dans la vie. Cette métamorphose, je ne l'ai pas accomplie seule. Pour construire une preuve, j'en ai détruit des dizaines. Concevoir un système de preuves ressemble un peu à construire un château de cartes en déplaçant souvent les cartes du dessous... Mais après trois ans, je suis devenue papillon, et je n'ai que ma chrysalide, ce manuscrit, à dédier à tous ceux qui m'ont accompagnée tout au long du chemin, même si finalement pour bon nombre d'entre eux tout ce qui est écrit là-dedans n'a pas vraiment de sens...

J'ai une pensée toute particulière pour mon chef, que j'appelle ainsi et que je continuerai sûrement à appeler ainsi longtemps. Il m'a transmis la passion de la recherche de l'argument juste, et cette volonté de tout remettre en question sans relâche tant que les bases ne sont pas certaines.

Je remercie sincèrement mes rapporteurs et les membres de mon jury de m'avoir fait l'honneur d'accepter de participer à ma soutenance, et d'être venu de (très) loin pour y assister.

Merci aussi à toute l'équipe de vérification de sécurité de Verimag, Cristian, Pierre, et surtout Pascal sans qui je n'aurais peut-être jamais connu la joie de soutenir ! Merci à Gilles de la confiance qu'il m'a accordée très tôt. J'espère que toutes nos aventures ne font que commencer...

Je veux dédier ce manuscrit à Benoît, qui, entre autres, a partagé toutes mes émotions tous les soirs et qui chaque matin m'a réveillée pour retourner à l'ouvrage ; qui a répondu patiemment, de jour comme au milieu de la nuit « Oui » à cette éternelle question « Tu crois que je vais réparer ma preuve ? » ; qui a appris les bases de la cryptographie prouvable lors de répétitions d'exposés le week-end... C'est grâce à toi, 'Teur, que je suis aujourd'hui devenue Doc'.

Maman et Papa, je suis un sacré mélange de toutes vos qualités (si, si, uniquement des qualités). Vous m'avez appris la ténacité et la rigueur qui me donnent cette force de ne pas m'arrêter tant qu'un résultat n'est pas sincèrement satisfaisant, cette volonté de tout donner pour quelque chose auquel on croit vraiment. Merci d'avoir été présents, d'avoir répondu au

téléphone à des heures impossibles, de m'avoir remise debout quand il fallait encore tout reprendre et qu'il n'y avait pas de temps (et un gros bisou de Cryptou)... J'essaie tous les jours de faire en sorte que vous soyez fiers de moi comme je suis fière de vous.

Merci à Vinz, mon co-bureau virtuel, d'avoir répondu patiemment à toutes les questions naïves d'une étudiante en Mathématiques qui débute en Informatique et qui enseigne quand même, d'avoir relu, corrigé, aidé à coder, d'avoir été connecté jour après jour sur gtalk, ainsi que d'être venu de loin pour me voir soutenir cette thèse !

Je dédie aussi tout ce travail à mes soeurs, dont je suis très fière, et qui partagent avec moi cet amour des choses bien faites qui nous rendra folles ! A ma Maï, qui m'a appris à lire, à compter sur un boulier chinois, que de chemin ai-je parcouru depuis ces balbutiements !

A tous mes amis de Verimag, qui ont rendu possible de supporter tous les aléas du quotidien, les preuves cassées, les étudiants qui exagèrent, les processus administratifs bourrés de deadlocks, les programmes qui ne compilent pas... Merci en particulier à tous ceux qui ont répondu présent : Marc - qui pas une seule fois n'a refusé d'interrompre son propre travail pour m'écouter et résoudre mes problèmes informatiques - Math' et Ju' - et notre trio infernal de thésards, aux bières partagées et aux débats sur les choses de la vie et de la science :-) Vous verrez, un jour, la thèse se termine vraiment... - Jacques et Sophie - qui ont partagé leur expérience avec moi et m'ont souvent faite rire et montré comment prendre les choses mieux - Raj et Selma - membres du bureau 41, qui ne se sont jamais plaints que je débatte avec ma machine à haute voix, et pour Selma qui m'a aidée à croire que « les choses finissent toujours par s'arranger », jour après jour après jour de thésard... Il y a encore d'autres Verimagiens, qui m'ont fait rire à la cafétéria (et qui ont joué à me couper l'appétit à midi), que je n'ai pas encore cité : Benoît, Martin, Jannick, Florent, mais aussi des plus anciens dont Simon, Guillaume et Hugo... Je maintiens que je préfère parler de vélo plutôt que de nombre d'autres sujets !

Je remercie également Christophe et Nicolas de m'avoir invitée à m'asseoir un jour pour prendre des décisions difficiles qui me rendent fière aujourd'hui.

Merci à Jérôme C. d'avoir pris soin de moi patiemment même quand je ne suivais pas ses conseils, à Jérémy d'avoir écouté mes râleries de thèse quand Benoît n'y arrivait bientôt plus et d'avoir écouté les problèmes de Benoît à écouter mes problèmes ! Merci à sa femme Chun Yan de ses petits plats et de ses encouragements dans la dernière ligne droite de la rédaction.

Merci enfin à toute la famille de Benoît, qui m'a supportée endormie aux fêtes de famille, et qui a pourtant toujours été fière de mon travail, au point de venir assister à des heures de soutenance en Anglais pour enfin me croiser quelques minutes à la fin.

« Si tu peux voir détruit l'ouvrage de ta vie,
Et sans dire un seul mot te mettre à reconstruire... »
... tu feras peut-être une preuve juste, ma fille !

Résumé

Les systèmes informatiques sont devenus omniprésents dans les objets de la vie quotidienne, et disposent de nombreux moyens de communiquer des informations entre eux. Aussi la capacité de garantir la confidentialité ou l'authenticité d'une information est-elle devenue un besoin crucial de nombreux utilisateurs de ces systèmes. La *cryptographie* offre des moyens de résoudre les problèmes inhérents à ce besoin, en définissant des techniques de communication sécurisée par des canaux non-fiables. Plus précisément, la cryptographie consiste en la conception et l'analyse de protocoles satisfaisant des aspects variés de sécurité de l'information comme l'intégrité, l'authenticité ou le caractère secret d'une donnée. Utiliser des outils cryptographiques ne résout pas tous les problèmes de sécurité de l'information, puisque la sécurité d'un système dépend aussi, par exemple, de ses conditions d'utilisation. Cependant, l'utilisateur doit pouvoir avoir confiance en les outils proposés par les cryptographes. Proposer des preuves formelles de l'évaluation exacte de la sécurité garantie par un protocole est l'objet de la *cryptographie prouvable*. C'est à la construction de telles preuves que s'attache le travail proposé dans cette thèse.

Bien que le besoin de confidentialité de communications, pour des raisons militaires par exemple, se soit fait sentir dès l'Antiquité, c'est Shannon [Sha48, Sha49], au milieu du XXe siècle, qui propose la première tentative de formuler une définition précise de la sécurité d'une information. Avant cela, beaucoup de procédés de chiffrement de messages avaient été inventés, mais jamais on n'avait *formalisé et prouvé* les garanties fournies par un système en terme de sécurité. Un système était considéré comme sûr tant que personne n'avait trouvé d'astuce pour extraire une quantité significative d'information sur les données transmises en l'utilisant. De nos jours, il existe deux grandes approches formelles pour traiter de la cryptographie prouvable : le *modèle symbolique* et le *modèle calculatoire*. Du point de vue symbolique [DY83], les messages sont formalisés par les termes d'une algèbre et un adversaire est représenté par un ensemble de règles d'inférence qu'il peut utiliser sur les termes qu'il intercepte sur les canaux de communication pour construire de nouvelles quantités. Dans le modèle calculatoire [GM84], les messages sont considérés comme des suites de bits, tandis que les adversaires sont des machines de Turing probabilistes qui interagissent avec les systèmes cryptographiques en appelant des oracles.

Dans cette thèse, nous remédions à l'absence de système de preuves qui soit dédié à la sécurité calculatoire des systèmes cryptographiques. De même que des techniques de vérification spécifiques ont été développées pour divers systèmes critiques, nous proposons un cadre propre aux systèmes cryptographiques dans lequel effectuer les preuves de satisfaction de critères de sécurité calculatoire. Par rapport aux systèmes concurrents, les systèmes étudiés en sécurité sont caractérisés par l'importante asymétrie des parties qui interagissent. En effet, les critères de sécurité sont en général définis au moyen de jeux probabilistes qui font intervenir deux parties : d'une part les éléments relevant de l'objet cryptographique, d'autre part l'adversaire de ce dernier. Aucune restriction n'est imposée aux adversaires, alors que l'on connait entièrement la spécification de la construction que l'on veut prouver. De plus, les théorèmes à établir ne sont en général pas formulés en terme d'existence ou de non-existence d'une trace satisfaisant une propriété, mais plutôt de l'une des manières suivantes. Soit on souhaite borner la probabilité que le système global (i.e. le système cryptographique et son adversaire) satisfasse une certaine condition, soit on veut prouver que du point de vue de l'adversaire, il est équivalent d'interagir avec l'une ou l'autre de deux versions du système cryptographique (par exemple, le système réel et un système complètement alétoire).

Contexte des travaux

Le modèle symbolique

Dans le modèle symbolique, les messages sont représentés par des termes algébriques et les adversaires comme l'ensemble des règles qui caractérisent leurs capacités de déduction. Ce modèle s'appuie sur l'hypothèse de la *cryptographie parfaite*, selon laquelle les schémas de chiffrement ne permettent à l'adversaire la déduction d'aucune information sur un message à la vue de son chiffré, à moins que l'adversaire ne connaisse la clé utilisée pour calculer ce chiffrement. L'analyse de protocoles cryptographiques, c.a.d. de programmes distribués permettant de communiquer en utilisant un canal contrôlé par l'adversaire, peut donc s'effectuer en utilisant cette modélisation.

Par comparaison au modèle calculatoire, l'approche symbolique a l'avantage de se placer à un plus haut niveau d'abstraction. Ceci a rendu possible l'analyse symbolique de protocoles en utilisant des techniques de vérification automatique dans les années 90. Il en a résulté une preuve éclatante de la criante nécessité de concevoir et vérifier les preuves des systèmes cryptographiques avec le plus grand soin. En effet, en 1995, Gavin Lowe a utilisé un model-checker pour vérifier le protocole de Needham-Shroeder, décrit en 1978 [NS78] et considéré comme sûr jusque là. Ce faisant, il a mis à jour l'existence d'attaques par interposition d'un tiers (appelées man-in-the-middle attack en Anglais) sur le protocole (cf. [Low95, Low96]).

Des efforts significatifs ont été dédiés à la création d'outils automatiques pour la vérification de critères de sécurité classiques, en présence d'adversaires passifs comme actifs. L'outil automatique ProVerif en est un exemple ; conçu par Blanchet et al. [BAF08], il est basé sur la résolution de clauses de Horn. Les deux principales plateformes maintenant disponibles sont CAPSL[1] et AVISPA[2], elles regroupent diverses techniques d'analyse et utilisent un certain

[1] Common Authentication Protocol Specification Language, cf. http ://www.csl.sri.com/users/millen/capsl/ pour l'outil et de plus amples références.

[2] Automated Validation of Internet Security Protocols and Applications, cf. http ://avispa-project.org/ pour l'outil et de plus amples références.

nombre d'autres outils d'analyse formelle.

Les garanties en matière de sécurité fournies par des preuves symboliques ne sont pas complètement claires vis-à-vis d'un contexte d'utilisation réelle du protocole. Cela constitue un important inconvénient de la modélisation symbolique des protocoles. Par comparaison au modèle calculatoire dans lequel les adversaires sont plus proches des capacités réelles d'un attaquant du système cryptographique, il apparaît préférable de développer des preuves calculatoires de ces systèmes. Dans l'article fondateur [AR00], Abadi et Rogaway suggèrent de mettre à profit le meilleur de chacune des approches en démontrant des résultats dits de correction calculatoire. De tels théorèmes sont typiquement des affirmations de la forme "s'il existe une preuve symbolique que le système cryptographique est sûr, alors il est également sûr dans le modèle calculatoire". De tels résultats, couvrant des situations de plus en plus diverses, ont été prouvés depuis. Le lecteur intéressé peut se référer à l'article de Cortier et al. [CKW11], dans lequel les auteurs inventorient les différentes variantes de ces techniques et résultats qui ont été développées.

Le modèle calculatoire

Suivant l'approche calculatoire, les messages sont formalisés par des suites de bits et les adversaires comme des machines de Turing probabilistes. Les preuves conçues dans ce modèle ne constituent pas des preuves à proprement parler de la sécurité des systèmes. En effet, ce sont des arguments de réduction basés sur la théorie de la complexité. Comme celle de Rabin [Rab79], qui justifie l'équivalence entre casser un cryptosystème de sa conception et la factorisation de nombres entiers, les preuves calculatoires attestent qu'il est au moins aussi difficile de briser la sécurité d'un système que de résoudre un problème difficile connu, ou un problème conjecturé difficile.

Pour effectuer une preuve par réduction, les problèmes sous-jacents aux critères de sécurité doivent être paramétrés, de manière à ce que l'on puisse parler de réduction efficace. En pratique, on fixe un *paramètre de sécurité*. Par exemple, dans le cas des schémas de chiffrement, il s'agit de la longueur des clés utilisées par le schéma. En plus de cela, l'adversaire est paramétré par deux quantités : une borne supérieure sur le temps dont il dispose pour déployer son attaque, ainsi qu'une borne supérieure sur le nombre d'appels qu'il effectue pour chaque oracle. Plus particulièrement, si k est une fonction qui associe un entier à chaque nom d'oracle et t est un entier désignant le temps, les adversaires paramétrés par (k, t) sont les machines de Turing dont le temps d'exécution est borné par t et qui effectuent au plus $k(o)$ appels à l'oracle o pour chaque oracle o disponible. Effectuer une réduction fournit deux informations : le temps nécessaire à l'exécution de l'adversaire et la probabilité de résoudre le problème difficile sous-jacent. Ces résultats peuvent être traités de deux manières. D'une part, suivant l'approche asymptotique, on peut considérer suffisant de s'assurer que tout adversaire ayant un temps d'exécution polynomial en le paramètre de sécurité possède une probabilité de succès négligeable en ce dernier[3]. D'autre part, on peut exhiber des preuves de sécurité concrète, c.a.d. en fournissant les bornes exactes nécessaires à la réduction. Les preuves de sécurité concrète (ou exacte) fournissent plus d'information à l'utilisateur du système cryptographique que leurs homologues asymptotiques : elles exhibent une borne sur les ressources nécessaires pour mener à bien une attaque contre le système. Cela permet en particulier d'effectuer un

[3]Une fonction est dite négligeable si et seulement si quel que soit le polynôme considéré, il existe toujours un stade à partir duquel la fonction devient plus petite que l'inverse du polynôme.

choix pertinent de paramètre de sécurité. Dans cette thèse, nous présentons un système dans lequel développer des preuves de sécurité concrète.

Nous avons mentionné plus haut que beaucoup de critères de sécurité sont construits autour de la notion d'équivalence entre deux comportements, ou entre deux jeux. Formellement, cela se traduit par l'indistingabilité de deux expériences probabilistes, ou de deux distributions, concept fondamental du modèle calculatoire. En termes asymptotiques, l'idée est de formaliser le fait que deux "mondes", c.a.d. deux distributions, sont indistingables si aucun adversaire polynômial n'est capable de trouver une stratégie de réponse significativement plus intelligente que tirer au hasard pour savoir duquel des deux mondes l'argument qui lui est fourni est issu. De manière similaire, on dit que deux expériences probabilistes sont indistingables si l'exécution de l'une ou de l'autre est indifférente à l'adversaire. L'indistingabilité de deux distributions est formalisée de la manière suivante.

Definition (Indistingabilité concrète de deux distributions). Soient $\mathcal{D}_0$ et $\mathcal{D}_1$ des distributions sur un ensemble de mémoires. Soit $\mathcal{O}$ un ensemble d'oracles que l'adversaire peut interroger. Etant donnée une fonction $\varepsilon : (k,t) \mapsto \varepsilon(k,t) \in [0,1]$, les distributions sont ε-indistingables si et seulement si pour tout adversaire paramétré par (k,t),

$$|\Pr[m \leftarrow \mathcal{D}_0; b \leftarrow \mathcal{A}^{\mathcal{O}}(m) : b = \mathsf{true}] - \Pr[m \leftarrow \mathcal{D}_1; b \leftarrow \mathcal{A}^{\mathcal{O}}(m) : b = \mathsf{true}]| \leq \varepsilon(k,t) \quad \square$$

Preuves effectuées dans le modèle calculatoire

Les preuves conçues directement dans le modèle calculatoire ne sont pas exemptes d'imperfections ou même d'erreurs. Dans son article [Sho02], Shoup présente une erreur dans la preuve de sécurité du schéma de chiffrement OAEP n'étant pas prise en compte par la preuve proposée par ses créateurs Bellare et Rogaway dans l'article [BR94]. Une preuve correcte de la sécurité de ce schéma dans le cas de l'utilisation de RSA comme primitive est plus tard présentée par Fujisaki et al. [FOPS04]. Peu après avoir trouvé cette erreur, Shoup rédige l'article [Sho04], dans lequel il propose d'utiliser les transformations de jeux probabilistes pour structurer et clarifier les preuves cryptographiques. La technique des jeux consiste à considérer l'expérience probabiliste définissant le critère de sécurité comme un jeu se déroulant entre l'adversaire et le cryptosystème. Il faut alors proposer une série de modifications successives de ce jeu pour le transformer en un jeu équivalent dans lequel l'adversaire doit résoudre un problème difficile, ou dans lequel il n'a aucune chance de gagner. Chaque pas de la preuve doit être justifié, et ce en fournissant une borne supérieure de la probabilité qu'un adversaire puisse déterminer s'il joue au jeu non-modifié ou au jeu comportant la modification effectuée à ce pas de preuve.

Cette technique de présentation des preuves est très utilisée par les cryptographes. Après l'article fondateur de Shoup, Halevi [Hal05] a plaidé en faveur de la conception de techniques de vérification formelle, et a exprimé la nécessité de l'implémentation d'un outil utilisable par la communauté des cryptographes. Dans [BR04], Bellare et Rogaway proposent d'effectuer des preuves en travaillant sur des expériences probabilistes exprimées en pseudo-code, que l'on peut modifier d'une manière qui rappelle la technique des jeux. Bien que ces propositions permettent d'organiser et de faciliter la lecture des preuves de sécurité, il n'en reste pas moins qu'elles ne fournissent pas de cadre formel dans lequel véritablement *concevoir* ces preuves.

Contributions

La logique CIL

Nous proposons une logique, CIL, permettant d'effectuer des preuves de sécurité concrète de cryptosystèmes, directement dans le modèle calculatoire. Notre cadre formel est versatile en ce qu'il n'est lié ni à un langage ni à des hypothèses particulières : on peut y réaliser des preuves dans le modèle à oracle aléatoire comme dans le modèle standard. Produit de l'analyse transverse de nombreuses preuves de sécurité, notre système de preuve remédie à l'absence de cadre formel dédié en proposant un petit nombre de règles traduisant les principes de raisonnement communs sous-jacents à de nombreuses preuves de sécurité. La formalisation de ces principes est basée sur des outils classiques de théorie des langages de programmation ou d'analyse de systèmes concurrents, comme les relations de bisimulation et les contextes. Par ailleurs, grâce à quelques règles supplémentaires, il est possible d'effectuer des hypothèses ou des morceaux de raisonnement nécessaires aux preuves mais vérifiés en dehors du cadre du système CIL (par exemple, des calculs simples de probabilités, du raisonnement en logique du premier ordre, etc.).

Dans la logique CIL, on raisonne avec des systèmes d'oracles. Il s'agit de systèmes à états qui formalisent l'interaction de l'adversaire avec un cryptosystème. La logique comporte deux jugements : un jugement d'indistingabilité, dénoté $\mathbb{O} \sim_\varepsilon \mathbb{O}'$, qui formalise l'indistingabilité à ε près de deux systèmes d'oracles, et un jugement $\mathbb{O} :_\varepsilon E$ qui traduit le fait que l'événement E possède une probabilité bornée de se réaliser lors de l'interaction du système $\mathbb{O}$ avec un adversaire. Le système de preuve et toute la formalisation qui lui est nécessaire sont présentés dans le chapitre III.

Par rapport aux possibilités existantes d'effectuer une preuve sur papier ou grâce à un outil automatique comme un assistant de preuve, CIL offre la possibilité de réaliser une preuve à un niveau d'abstraction intermédiaire. D'ailleurs, une formalisation des règles de CIL est développée au sein du laboratoire VERIMAG. Nous avons conçu CIL indépendamment de tout langage de programmation. Il est donc aisé de passer d'une représentation d'un système à une autre suivant ce qui paraît le plus approprié au pas de preuve à réaliser. De plus, la sémantique solide sur laquelle la logique s'appuie permet une compréhension et une vérification plus systématique des preuves de cryptographie concrète.

Preuve automatique de schémas de chiffrement asymétrique dans le modèle de l'oracle aléatoire

Nous proposons une logique de Hoare qui permet de prouver la sécurité concrète de schémas de chiffrement asymétrique spécifié dans un langage de programmation fixé de manière totalement automatique, en se plaçant dans le modèle de l'oracle aléatoire. Par comparaison à l'approche développée dans CIL, qui fournit des manières de raisonner de manière globale sur les systèmes à oracles, la logique de Hoare permet de raisonner de manière locale sur les propriétés de distributions des valeurs prises par les variables utilisées par l'implémentation du chiffrement. Il s'agit donc d'un point de vue orthogonal, qui permet de prouver des résultats qui peuvent ensuite être importés dans une preuve en CIL.

La logique de Hoare proposée s'appuie sur la composition de trois prédicats atomiques, qui fournissent des informations sur les distributions des variables du programme. L'un traite d'indistingabilité d'une valeur aléatoire, un second borne la capacité d'un adversaire à calculer

la valeur d'une variable, le troisième de la probabilité d'avoir déjà calculé le hachage d'une variable. Le caractère compositionnel de la logique permet de déduire de ces propriétés locales des distributions des conclusions correctes sur la distribution du résultat du programme considéré.

De manière assez classique, nous avons développé grâce à notre logique de Hoare un algorithme de recherche de preuve qui, bien qu'incomplet, permet de conclure de manière automatique pour des exemples intéressants tels le schéma proposé par Bellare et Rogaway dans [BR93] ou par Pointcheval dans [Poi00].

Indifférentiabilité d'un oracle aléatoire de fonctions de hachage itératives

Un troisième volet de nos contributions traite des fonctions de hachage. Le concours SHA-3 mis en place par le NIST pour l'établissement d'un nouveau standard de fonction de hachage cryptographique a donné lieu au développement de nombreux candidats et des preuves respectives de leur sécurité. L'indifférentiabilité est un concept proposé par Maurer et al. dans [MRH04] pour formaliser la différence de comportement existant entre un système donné et le même système dans lequel un composant a été remplacé par un autre (par exemple une idéalisation du composant). L'idée a été adaptée plus particulièrement aux fonctions de hachage, qu'on souhaite pouvoir remplacer par un oracle tirant aléatoirement ses réponses sur le même espace d'arrivée, dans le travail de Coron et al. [CDMP05]. Pour prouver qu'une fonction de hachage est indifférentiable d'un oracle aléatoire, il faut exhiber un simulateur qui soit capable de compenser les écarts de comportement entre la fonction réelle et son idéalisation aléatoire vis-à-vis d'un adversaire, et ce en donnant à l'adversaire un accès à tous les composants internes de la fonction de hachage.

Comme pour les autres critères de sécurité, beaucoup de preuves ont été produites pour un grand nombre de constructions, sans qu'un formalisme spécifique ne soit développé pour les concevoir et les vérifier. Notre contribution consiste en la preuve d'un théorème générique permettant de prouver l'indifférentiabilité d'un oracle aléatoire pour un sous-ensemble de fonctions de hachage itératives. Ce théorème fournit à la fois un simulateur générique et une manière de calculer une borne supérieure à l'indifférentiabilité de la construction traitée par rapport à un oracle aléatoire. La preuve de ce théorème est conçue et exposée en CIL, constituant un bon exemple de l'utilisation de notre système de preuve pour la formalisation de raisonnements de sécurité concrète. Nous illustrons notre théorème par son application sur l'un des finalistes du concours SHA-3, Keccak, et sur la construction Chop-Merkle-Damgård, pour laquelle nous obtenons des bornes satisfaisantes. L'application du théorème à Keccak a même permis de mettre en exergue une erreur dans sa preuve originale et de justifier l'apparition d'un terme supplémentaire dans le calcul de la borne d'indifférentiabilité, déjà pressenti mais pas justifié par les concepteurs de son concurrent Shabal dans [BCCM+08].

Contents

Chapter I

Introduction

With the increasing ubiquity of computing and communications, the need to guarantee security properties such as privacy, secrecy or authenticity has become critical. Cryptography addresses these issues by proposing techniques for secure communication over untrusted channels. In particular, cryptography consists in the conception and analysis of protocols achieving various aspects of information security such as data confidentiality, integrity or authentication. Cryptography is not a magical way to solve each and every security issue: a construction can never be more secure than its weakest component. However, users need to know that they can rely on the solutions proposed by cryptographers. This is what *provable cryptography* is about: the conception of proofs accounting for the exact amount of security supplied by cryptographic protocols.

Whereas the need for secret communications, e.g. for obvious military reasons, dates back to Antiquity, the first attempt to scientifically define security only took place during the second part of the twentieth century with the works of Shannon [Sha48, Sha49]. Before that, the essential processes for ciphering messages had been invented, but a cryptosystem had never been *proven secure*. It was simply *considered secure* until some cryptanalyst had found a clever way of breaking it. Nowadays, there exist two major formal approaches to provable cryptography: the symbolic and the computational model. In the symbolic approach [DY83], messages are defined as terms of an algebra and an adversary is represented as a set of deduction rules that can be used to deduce knowledge from the terms intercepted on the network. In the computational model [GM84], messages are bitstrings and adversaries are probabilistic Turing machines interacting with cryptosystems via oracle access.

In this thesis, we address the lack of proof systems dedicated to proving the computational security of cryptographic modes of operations directly in the computational model. In the same way specific verification techniques have been developed for several types of critical systems, we design a proper framework for cryptographic systems and their computational security criteria. Compared to concurrent systems, the systems studied in security are characterized by a profound asymmetry between interacting parties. Indeed, security criteria are generally defined as some kind of probabilistic games involving two kinds of parties: those behaving as specified by the cryptographic construction and the adversarial parties. No restrictions are

imposed on the adversarial parts of the system, whereas we know the complete specification of the design to be proven secure. Furthermore, the statements that we want to prove are usually not formalized as the existence or non-existence of a trace satisfying a property, but rather in one of the two following forms. We either want to bound the probability that the global system, i.e. the cryptosystem and the adversary, realizes some condition, or we want to prove equivalence for an adversary to

In the symbolic model, messages are algebraic terms, and adversaries are represented by a set of rules capturing their abilities. The symbolic view relies on the *perfect cryptography* assumption. The idea is to assume the cryptographic primitives to be perfect. As an example, encryption schemes are modeled so that the only way to extract a plaintext from a ciphertext is to know which key to use to decipher the ciphertext. The symbolic model proves useful for the analysis of cryptographic protocols, i.e. distributed programs for communicating over channels controlled by an attacker.

The symbolic approach has a great advantage on the computational one in that it has a higher level of abstraction. This has allowed the symbolic analysis of protocols using automated verification techniques in the nineties, confirming that security proofs have to be carefully designed and checked to avoid overlooking attacks. Indeed, in 1995, Gavin Lowe has run a model-checker to verify the Needham-Shroeder protocol, described in 1978 [NS78] and believed to be secure. He uncovered man-in-the-middle attacks on the construction (see [Low95, Low96] for details).

Consequently, significant effort has been devoted to the creation of automated tools to check usual security requirements, in the presence of passive as well as active adversaries. The automatic tool ProVerif, has been designed by Blanchet et. al. [BAF08] and is based on Horn clauses resolution. In a nutshell, two platforms are nowadays available, CAPSL (Common Authentication Protocol Specification Language[1]) and AVISPA (Automated Validation of Internet Security Protocols and Applications[2]), grouping various analysis techniques and built on top of a various number of other formal tools.

I.1 The Computational Model

As we said earlier, in the computational model, messages are represented by bitstrings and adversaries are probabilistic Turing machines with oracle access. Such machines are standardly denoted $\mathcal{A}^{\mathcal{O}}$, where $\mathcal{O}$ is the set of oracles which the adversary can query. The proofs performed in this model are not stand-alone arguments. Indeed, they are *complexity-theoretic based reduction proofs*. In the line of Rabin [Rab79], who proved the equivalence between breaking the security of a cryptosystem that he designed and performing integer factorization, computational proofs provide evidence that breaking the security of systems is as least as difficult as solving a known difficult problem, or as solving a problem strongly believed to be difficult.

To perform a reduction proof, we need to decide on a way to parameterize our problems, in order to decide on the efficiency of the reduction. This is done by fixing a *security parameter*. For example, in the case of encryption schemes, it is the length of the generated keys. Then, the adversarial Turing machine is parameterized by two quantities: an upper-bound on the

[1] See http://www.csl.sri.com/users/millen/capsl/ for the tool and references.
[2] See http://avispa-project.org/ for the tool and references.

time it takes to carry out its attack, and an upper-bound on the number of queries it can issue to each of its oracles. Given a function k mapping every oracle name to an integer and a time t, (k,t)-adversaries are Turing machines whose running time is upper-bounded by t and number of queries to an oracle $\mathcal{O}$ by function $k(\mathcal{O})$. Performing the reduction provides two results: a time necessary to run the adversary, and a probability of success to solve the underlying hard problem. These can be interpreted from two points of view. One can adopt an *asymptotic approach*; then, security follows from two conditions: the running time of the adversary must be bounded by a polynomial in the security parameter, and its probability of success must be negligible in the security parameter[3]. The other possibility is to provide *concrete security proofs*. They are more valuable than their asymptotic counterparts, in the sense that they provide a definite bound on the resources it takes to mount an attack against the cryptosystem. This in turn allows to choose the security parameter appropriately. In this thesis, we develop a framework to carry out concrete security proofs.

We mentioned that a great deal of security criteria involve the use of a notion of equivalence between two behaviors, or games. Formally, this is captured by the concept of *indistinguishability* of two probabilistic experiments, or two distributions, which is a cornerstone of computational security criteria. Intuitively, in asymptotic terms, as is illustrated in figure I.1, two "worlds", i.e. two distributions on a probability space, are said to be indistinguishable if no polynomial time adversary can tell in which world an argument was picked. Equivalently, two possible behaviors during a probabilistic experiment are indistinguishable iff an adversary cannot tell which behavior was chosen to compute the challenge it was provided. We provide a formal definition of the concrete indistinguishability of two distributions on memories, mapping variables to values: it captures that sampling a memory in one distribution or the other looks the same to an adversary.

Definition (Concrete Indistinguishability of Distributions). Let $\mathcal{D}_0$ and $\mathcal{D}_1$ be distributions on a set of memories. Let $\mathcal{O}$ be a set of oracles which the adversary can access. Given a function $\epsilon : (k,t) \mapsto \epsilon(k,t) \in [0,1]$, the distributions are (k,t,ε)-indistinguishable w.r.t. $\mathcal{O}$ iff for any (k,t)-adversary,

$$|\Pr[m \leftarrow \mathcal{D}_0; b \leftarrow \mathcal{A}^{\mathcal{O}}(m) : b = \mathsf{true}] - \Pr[m \leftarrow \mathcal{D}_1; b \leftarrow \mathcal{A}^{\mathcal{O}}(m) : b = \mathsf{true}]| \leq \varepsilon(k,t) \quad \square$$

I.2 Computational Proofs

I.2.1 — Using the Symbolic Model

One of the main drawbacks of the symbolic way of modeling protocols is that the security guarantees which proofs bring w.r.t. practical use of the protocols are not clear. In comparison, it seems highly preferable to develop a proof in the computational model, where adversaries look more realistic. In [AR00], Abadi and Rogaway suggested to make the best of both approaches by proving *computational soundness results*. Such theorems typically argue that if proven in the symbolic model, a security criterion also holds in the computational model. More and more relevant computational soundness results have been developed ever since. The interested reader can see the survey of Cortier, Kremer and Warinschi in [CKW11] for an inventory of the different flavors of results and practices.

[3] A function is said to be negligible if it is ultimately bounded by the inverse of every polynomial.

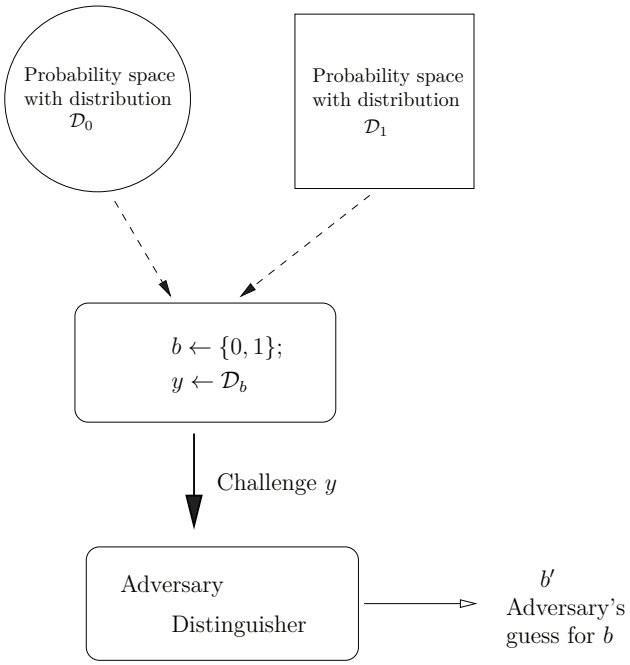


Figure I.1 – Indistinguishability of Distributions $\mathcal{D}_0$ and $\mathcal{D}_1$

I.2.2 — The Direct Approach

Proofs carried out directly in the computational approach are not immune to imprecisions or errors. In [Sho02], Shoup uncovered a mistake in the security proof of security of the widely used encryption scheme OAEP (for Optimal Asymmetric Encryption with Padding), originally proposed by Bellare and Rogaway in [BR94]. A corrected proof is later presented in [FOPS04] by Fujisaki et al. when using RSA as a one-way function in the implementation of the scheme. In [Sho04], Shoup presents the use of games as a way to clarify and structure them. The game-playing technique consists in considering the probabilistic experiment defining a security criterion as a game played between an adversary and a cryptosystem. Then, the idea is to provide a security-preserving sequence of modifications, which step after step turn the game into an equivalent one solving of a difficult problem. At each step, one has to justify security preservation by arguments such as indistinguishability between playing one game or the other for the adversary. As a result, one can conclude that the probability that an adversary wins the original game is bounded by the sum of the probability that it solves the difficult problem and the probabilities that it distinguishes between two consecutive game of the sequence. Figure I.2 illustrates this technique.

Game-playing has been extensively used by cryptographers. After the seminal article of Shoup, Halevi has argued in [Hal05] that the design of formal verification techniques and their implementation in a tool that could be used by the cryptographic community was crucial to support the trust in cryptographic proofs. In [BR04], Bellare and Rogaway propose code-based proofs resembling game-playing techniques, as a first step towards enabling automatic verification of the proofs. However, though the game-based technique yields well-organized proofs, it does not provide a formalism in which to carry them out.

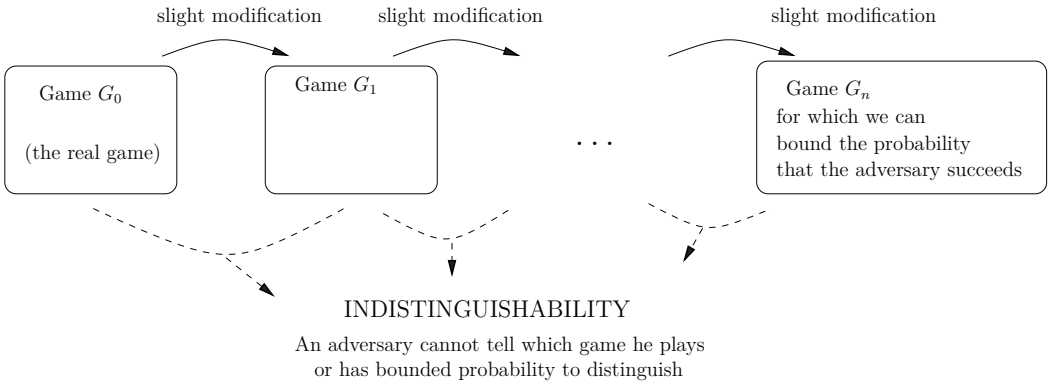


Figure I.2 – The Game-playing Technique

I.3 Contributions

I.3.1 — Computational Indistinguishability Logic

In this thesis, we present a general logic, the Computational Indistinguishability Logic (CIL), for proving concrete security of cryptographic schemes. It enables reasoning about schemes directly in the computational setting. Moreover, the framework is versatile, in that it is not committed to particular hypotheses on cryptographic primitives: proofs in the standard model as well as idealized models such as the random oracle model can be carried out in the logic. Product of the cross-analysis of many security proofs, our tool addresses the lack of proof systems by capturing common reasoning principles underlying security proofs. To this end, CIL features a small set of deduction rules allowing to apprehend these reasoning patterns. Their formalization relies on classic programming language and concurrency tools such as bisimulation relations and contexts. Furthermore, a few additional rules allow to interface with external reasoning (e.g. probability computations, first-order logic reasoning, etc.).

CIL allows to reason on *oracle systems*. They are stateful systems modeling adversarial interactions with cryptosystems. The logic is built around two judgments: firstly, statements of the form $\mathbb{O} \sim_\varepsilon \mathbb{O}'$ express the ε-indistinguishability of the pair of oracles systems $\mathbb{O}$ and $\mathbb{O}'$. Secondly, statement $\mathbb{O} :_\varepsilon E$ means that the probability that event E happens when an adversary interacts with $\mathbb{O}$ is bounded by ε. The framework and rules are presented in chapter III.

The main contribution of CIL is to support the design of proofs at a level of abstraction which allows to bridge the gap between pencil-and-paper fundamental proofs and existing practical verification tools. Namely, the proof system has been formalized in the proof-assistant Coq [4]. We have designed CIL independently of any programming language; thus, at any step of a proof, one can switch to the representation of oracle systems which is the most adapted to carry on with the step. Moreover, the solid semantic foundations on which the logic is built allows for a more systematic investigation of cryptographic proofs.

[4] References concerning the formal proof management system Coq, including an up-to-date reference manual can be found on coq.inria.fr

I.3.2 — A Hoare Logic to Prove Security of Asymmetric Schemes

In order to automate the generation of proofs for a subset of cryptographic primitives, we have designed a Hoare logic to prove concrete security statements, in particular Real-Or-Random-ciphertext security (ROR-ciphertext security) of asymmetric encryption schemes in the Random Oracle Model [BR93]. It is presented in chapter V.

The logic allows to reason on oracles described in a small fixed programming language. It is built out of three atomic predicates on distributions of the variables: the first captures ε-indistinguishability of a variable from random, the second bounds the probability that an adversary can recover a value for a variable, and the third bounds the probability that the value of a variable has been hashed previously. The semantics of the predicates is given with respect to resource-bounded computational adversaries, yielding a computational logic. Contrary to CIL rules, this framework is not based on global transformations of a program but on local properties and their conservation. It provides an orthogonal tool to obtain statements, which we can then import in CIL thanks to the dedicated set of interface rules.

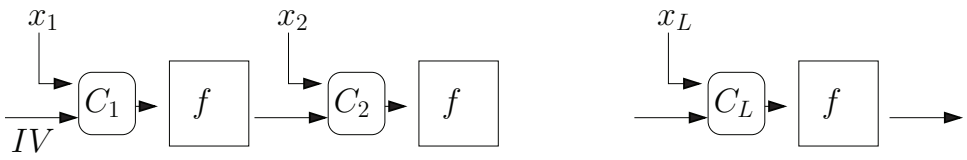
Hoare logics are quite standardly turned into automated verification procedures; ours can be too. Our logic thus yields an incomplete but fully automatic proof search algorithm. We illustrate the use of the logic on two asymmetric schemes: a first one proposed by Bellare and Rogaway in [BR93], and a second one proposed by Pointcheval in [Poi00].

I.3.3 — About Hash Functions and Their Security

There are two ways in which we look at hash functions in the present work. On one hand, when studying cryptographic primitives as encryption or signature specifications, hash functions are considered as atomic primitives. On the other hand, we look into the problem of how to design the hash constructions. In this case, these latter are no longer atomic primitives.

Designing a Hash Function. Hash functions are usually functions compressing their inputs, mapping them to a fixed output length. They are supposed to take inputs of arbitrary length (or "almost arbitrary", e.g. less than 2^{64}). Cryptographic hash functions are supposed to guarantee security properties such as non-malleability. In other words, two slightly different inputs must result in completely uncorrelated outputs; the input-output association should look like random to an adversary. Moreover, hash constructions should ideally be very unlikely to produce collisions, that is to say, that two different inputs map to the same output. This collision-resistance requirement seems quite far-fetched mathematically: there must be *huge sets* of inputs mapping to the same output given the respective size of input and output spaces. Consequently, collision-resistance rather requires that collisions are difficult to exhibit. Several formalizations can be proposed: there are various flavors of collision-resistance in the literature. A good overview and classification of various possibilities can be found in [RS04].

To address the problem of dealing with inputs of almost any length, hash designers classically provide two-tier constructions: they define a fixed input-output length compression function, and a way to iterate it to deal with any length: a domain extension technique. A *domain extender* is a series (C_j) of input transformations which create successive inputs to an underlying primitive out of previous outputs and the global domain extender input. This design paradigm has been proposed by Merkle in [Mer89] and Damgård in [Dam90]. Their construction, now called the Merkle-Damgård transform (or MD for short), is a domain extender where $C_j(h_{j-1}, x_j) = h_{j-1}||x_j$: up to padding to have a suitable length, the input



$MD^f(x) =$
$x := x_1||\ldots||x_L;$
$h_0 := IV;$
for $j = 1$ to L do
$h_j := f(h_{j-1}||x_j)$
endfor
return h_L

Here, $C_j(h_{j-1}, x_j) = h_{j-1}||x_j$.

Figure I.3 – A Domain Extender and the Merkle-Damgård Design

x is chopped into blocks of fixed length $x_1, \ldots, x_L$, before starting applying a compression function f to an initial value IV and x_1, and then iteratively apply f to its previous output and x_j. It is illustrated in figure I.3. Later on, tree-based hash designs have been proposed. We do not go into further details about these since we do not study them in this work.

Indifferentiability From a Random Oracle. In their seminal article [MRH04], Maurer, Renner and Holenstein introduce indifferentiability as a concept generalizing indistinguishability: an adversary of indifferentiability is provided with access to additional information about the internal state of systems. In this paper, they also prove indifferentiability to be a necessary and sufficient condition to impose on a couple of systems in order to soundly replace one by the other as cryptosystem components, as long as the composition of adversary and simulator yields an adversary (see [RSS11]). This question is fundamental when we try to figure out what guarantees may come of proofs in models such as the ideal cipher model (ICM) or the ROM. In particular, the question was raised when Canetti, Goldreich and Halevi have proven in [CGH04] that no hash function can implement a random oracle in case the pseudo-randomness parameter (e.g. the function key) is public. Then, soundness of modeling hash functions as random oracles became all the more debatable. Consequently, an argument quantifying the loss of security induced by such a replacement was much needed and can be found in indifferentiability. Rather than presenting the general definition, we only present indifferentiability from a random oracle as the notion tailored to hash functions by Coron et. al. in [CDMP05]. In this paper, they show that plain Merkle-Damgård does not satisfy the notion because of length-extension possibilities, and propose and prove a series of fixes such as chopping off some bits or using a prefix-free padding function on the input.

The idea behind indifferentiability from a random oracle is to measure the impact of the additional information provided to the adversary via a simulation-based argument. We consider an algorithm $\mathcal{H}$ depending on an ideal inner-primitive f. The adversary is provided with oracle access to $\mathcal{H}$ and f: direct access to f models the aforementioned additional public information. The idealization of $\mathcal{H}$ is a random oracle: it provides a randomly sampled output for every new input. To measure the gap between $\mathcal{H}$ and its idealization, we investigate

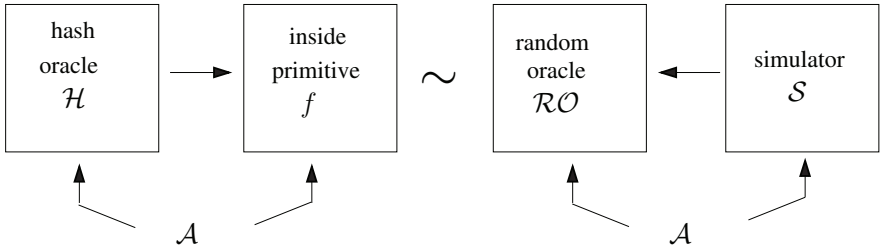


Figure I.4 – Indifferentiability From A Random Oracle

the existence of a simulator $\mathcal{S}$ such that for all adversaries, the pair $(\mathcal{RO}, \mathcal{S})$ consisting of the random oracle and the simulator is indistinguishable from the original pair $(\mathcal{H}, f)$. To compensate potential incoherence of behavior, in the idealized setting the simulator $\mathcal{S}$ is provided with (direct) access to $\mathcal{RO}$. This is depicted in figure I.4. The difficulty to elaborate a good simulator lies in the fact that the simulator *does not have access* to the random oracle memory. As a result, an adversary trying to distinguish both systems can issue a query to the random oracle, with the simulator being none the wiser. It is thus delicate for a simulator to mimic the real-world consistency between inner primitive queries.

Definition (Indifferentiability From a Random Oracle). A Turing machine $\mathcal{H}$ with oracle access to an ideal primitive f is (k, t, ε)-indifferentiable from an ideal primitive $\mathcal{RO}$ if there exists a simulator $\mathcal{S}$ such that for any (k, t)-adversary $\mathcal{A}$,

$$|\Pr[\mathcal{A}^{\mathcal{H},f} = \mathsf{true}] - \Pr[\mathcal{A}^{\mathcal{RO},\mathcal{S}} = \mathsf{true}]| \leq \varepsilon(k, t)$$

where $\mathcal{H}$ has oracle access to f and the simulator $\mathcal{S}$ has oracle access to $\mathcal{RO}$. □

If this criterion holds, then system $\mathcal{H}^f$ can replace $\mathcal{RO}$ in any cryptosystem, yielding a cryptosystem at least as secure as the former one in the following sense. To any environment interacting with a cryptosystem $\mathcal{S}ys$ and an adversary $\mathcal{A}$, there exists an adversary $\mathcal{A}'$ such that the interaction looks almost the same when cryptosystem and adversary $\mathcal{A}$ have access to $\mathcal{H}^f$ as when $\mathcal{S}ys$ and $\mathcal{A}'$ are provided with access to $\mathcal{RO}$. This can be formally put as follows.

Definition (At Least As Secure As). A cryptosystem $\mathcal{S}ys$ is ε-at least as secure in the f model with $\mathcal{H}$ as in the $\mathcal{RO}$ model if for all environment $\mathcal{E}$ and adversary (k, t)-$\mathcal{A}$, there exists a (k', t')-adversary $\mathcal{A}'$ such that

$$|\Pr[\mathcal{E}(\mathcal{S}ys^{\mathcal{H}}, \mathcal{A}^f) = \mathsf{true}] - \Pr[\mathcal{E}(\mathcal{S}ys^{\mathcal{RO}}, \mathcal{A}'^{\mathcal{RO}}) = \mathsf{true}| \leq \varepsilon(k, t).$$

□

The formal proof that indifferentiability of $\mathcal{H}$ from $\mathcal{RO}$ is sufficient to replace the former by the latter and obtain a system at least as secure is provided in [MRH04]. The intuition is illustrated in figure I.5. The idea is to consider the triple composed of environment, cryptosystem and adversary as a distinguisher between $(\mathcal{H}^f, f)$ and $(\mathcal{RO}, \mathcal{S}^{\mathcal{RO}})$. It allows to replace one by the other and provides a bound on the default of simulation that it can introduce. Then, adversary $\mathcal{A}$ and simulator $\mathcal{S}$ can be unified to form a new adversary $\mathcal{A}'$ interacting with the cryptosystem and $\mathcal{RO}$.

Acknowledging the weaknesses of existing standards SHA-1 and SHA-2, the National Institute of Standards and Technology has started a competition to decide on a new hash standard SHA-3. Though composition matters need to be dealt with carefully when using the

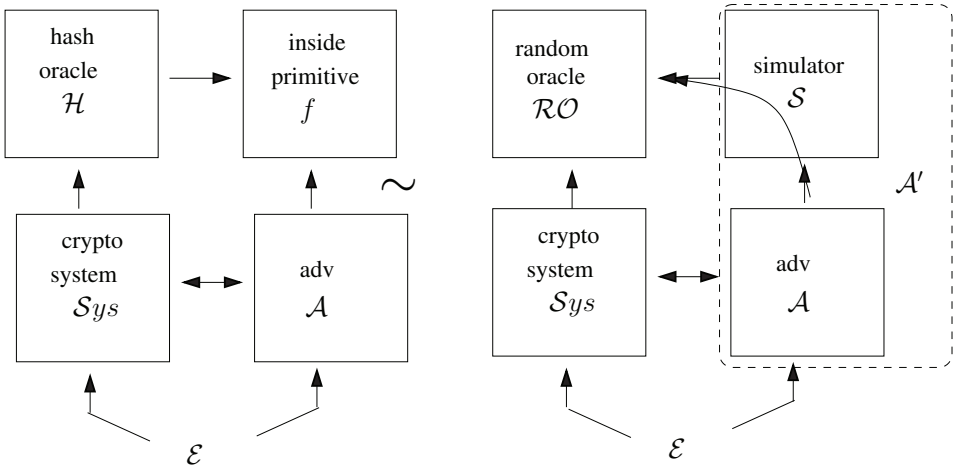


Figure I.5 – Motivation For Indifferentiability

indifferentiability security notion as defined originally (see [RSS11]), this criterion remains a standard security goal to be achieved by candidates. Roughly, it allows to track down structural design weaknesses allowing generic attacks. Therefore, following Coron et al. in [CDMP05], a lot of effort has been devoted to developing concrete security proofs of indifferentiability.

A Reduction Theorem for Indifferentiability. Submissions to the SHA-3 competition proposed by the NIST have been invited to include "any security argument that is applicable, such as a security reduction proof". Consequently, numerous candidates such as JH in [BMN10], Grøstl in [AMP10], Shabal in [BCCM+08], Keccak in [BDPA08] have been submitted along with a formal proof of indifferentiability. Surprisingly enough, as underlined in [BCCM+08], no real specific provable security framework was ever developed to carry out such proofs.

This is the issue we address in our last contribution, presented in chapter VI. Focusing on iterative hash constructions, we propose a generic strategy to design these proofs in our framework, in the form of a generic simulator implementation and a theorem proven in CIL providing a generic bound to the indifferentiability from a random oracle of a construction, using the generic simulator. Some extensions of our core framework are required to carry out the formalization of the two-tier structure composing hash designs, and to deal with graph structures. Eventually, we show on two examples that the bounds provided by our theorem are relevant: for the chop solution, we achieve the same result as Maurer and Tessaro in [MT07] in case of prefix-free padding and a better bound than that of Chang and Nandi in [CN08] in the general case. Moreover, the application of our result on the sponge construction (underlying the Keccak design) highlights the lack of an additional term in the bound provided by Bertoni et al. in [BDPA08], as was anticipated but not justified by Bresson et al. in [BCCM+08].

I.4 Related Work

I.4.1 — Logics for Cryptographic Proofs in the Computational Model

Impagliazzo and Kapron were the first to develop a logic to reason about indistinguishability in [IK06]. It is based on a more general logic whose soundness relies on non-standard arithmetic,

it does not support oracles or adaptive adversaries. Later, Zhang has proposed a logic built on top of Hofmann's SLR, named computational SLR [Zha08]. His logic, as that of Impagliazzo and Kapron, allows for a proof that next-bit unpredictability implies pseudo-randomness and to show a pseudo-random generator correctness. Nevertheless this logic presents the same drawbacks as the first one: there is no formalism to reason about oracles or adaptive adversaries. Another genre of framework is proposed by Corin and den Hartog in [CdH06]. They present an adapted version of a general probabilistic Hoare logic to model game-based proofs, and put it to work on to prove semantic security of ElGamal.

In [DDMR07], Datta, Derek, Mitchell and Roy present a survey of the Protocol Composition Logic (PCL), which is a logic in Hoare style to prove security properties of network protocols, using symmetric and asymmetric schemes. The logic is built on a process calculus based on atomic actions such as generating new nonces, sending and receiving a message, etc. The logic exhibits axioms and proof rules assuming the presence of an active Dolev-Yao style adversary. The rules allow both for sequential and a restricted parallel composition. It has been applied to prove security achievements of widely used protocols such as SSL/TLS, IEEE 802.11i and Kerberos V5. While PCL is more of a symbolic model tool, it has been provided with a computational semantics: it is the Computational Protocol Composition Logic presented in [BDD$^+$06]. Compared to PCL, the logic comprises a couple of new predicates, but the reasoning is not performed in the computational model directly: computational soundness follows from a theorem which assumes an IND-CCA encryption scheme. In [DDMW06], an extension of the logic to deal with security of key exchange protocols is presented. Weakening the requirements on the key by taking into account the way in which it is supposed to be used afterwards allows to capture more protocols. The article features the example of key-exchange protocol ISO-9798-3 followed by a session using the exchanged key. A new soundness result is provided, taking into account new primitives which the case study requires (e.g. decisional Diffie-Hellman assumption, unforgeability of a signature). Moreover, in [RDDM07], Roy et. al address the problem of proving secrecy properties. The fact that these latter are not trace-based properties (i.e. indistinguishability over a set of possible runs is not defined by summing the probability of indistinguishability on each run) seem to disable inductive reasoning. This motivates the introduction of secretive protocols, defined by a trace-based property dedicated to the treatment of secrecy, which allowing inductive and compositional reasoning. Finally, a further refinement of the logic is proposed in [RDM07], tailored to Diffie-Hellman based protocols. It is illustrated by proofs of the Diffie-Hellman variant of Kerberos and IKEv2, the revised standard key management protocol for IPSEC.

I.4.2 — Machine-Checked Proofs and Automation

Among the first to achieve machine-checked proofs of computational security without the perfect cryptography assumption, Barthe, Cederquist and Tarento have used the Coq-proof assistant to prove hardness of the discrete logarithm in the generic model [BCT04] and security of signed ElGamal encryption against interactive attacks [BT04]. Nowak [Now07] also shows a preliminary implementation of the game-based approach in Coq. Backes, Berg, Unruh [BBU08] propose a formalization of a language for games in the Isabelle proof assistant, and provide a proof of the fundamental lemma of game-playing; however, subsequent work [BMU10] rather presents symbolic analysis with computational soundness results.

Nowadays, there exist two principal automated verification frameworks for computa-

tional security concrete proofs: CryptoVerif and Certicrypt + EasyCrypt. We start with CryptoVerif [BP06, BJST08, Bla08], a tool developed by Bruno Blanchet. It is based on observational equivalence, which induces rewriting rules applicable in contexts satisfying specific properties. The rules allow to transform games into equivalent or almost equivalent ones. CryptoVerif can be used in two modes: it can either generate proofs of security by itself, or offer the possibility of human interaction to direct the proof towards its conclusion. In both cases, rewriting rules are applied until the adversary plays a game in which its probability of success is null. The global success probability is then the sum of the leaps taken from one game to the next. Cases studies include an automated proof of Kerberos 5 and Diffie-Hellman based constructions. Further progress have been reported in [BP10, Bla11].

Developed by Barthe, Grégoire and Zanella, CertiCrypt [BGZB09] is a framework enabling machine-checked design and verification of code-based concrete security proofs. It is built on top of the proof assistant Coq, and features reasoning elements of various domains, such as probabilities, algebra, complexity theory. The core layer of Certicrypt consists in a formalization of language pWhile, an imperative programming language with random assignments, structured data types and procedure calls. Moreover, a relational Hoare logic and a theory of observational equivalence are used to prove correctness of many classic transformations used in code-based proofs, e.g. lazy sampling. Case studies include existential unforgeability of the FDH signature scheme and semantic security of OAEP. Finally, Barthe et. al have recently presented EasyCrypt [BGHB11], an automated tool generating partial verifiable evidence of cryptographic proofs out of proof sketches. The sketches are checked using SMT solvers and automated theorem provers, and then compiled into verifiable proofs in the Certicrypt framework. Our line of research takes place right in the middle of the level of operation of Certicrypt and Easycrypt: we analyze extensively cryptographic proofs and provide a formalization of arguments which can in turn be implemented in a tool such as Certicrypt. For example, our framework has significantly helped in the conception of the proof of IND-CCA security presented in [BGLB11].

I.4.3 — Hash Functions

The problem is the same for hash functions as it is for public-key cryptography: while hash designs increase in number and complexity, their security proofs become more and more involved and difficult to check. To address this issue, a first possibility is to aim to broaden our view of what is required to ensure indifferentiability (e.g. [FGL10]). In this respect, Dodis et al. [DRS09] propose for example a new security notion, preimage-awareness, and show it to be a sufficient condition for indifferentiability.

In a slightly different approach, our ambition is to provide a framework and a generic simulator, along with a macro-theorem to compute an indifferentiability bound. Our work is not the first to aim to address the lack of unified provable security infrastructure in which to carry out indifferentiability proofs. It follows in the steps of Chang et al. who provide in [CLNY06] generic proofs for respectively most popular prefix-free padding designs, Bhattacharyya et al. who propose in [BMN09] a generic simulator and a list of events to optimally bound indifferentiability of a certain number of domain extenders, Shabal designers who set formal definitions of graph-based proofs and present their proof of Shabal as a roadmap to carry out others in [BCCM$^+$08]. A similar procedure is followed in Bhattacharyya et. al in the proof of JH [BMN10].

Our formalization of the two-tier structure of hash constructions captures all iterative designs that we are aware of, e.g. the SHA-3 finalists JH [Wu11], Grøstl [GKM+11], Keccak [BDPA11], Skein [FLS+10] and BLAKE [AHMP10], Shabal [BCCM+08], the EMD transform [BR06], HMAC and NMAC modes [BCK96]. Our definition generalizes that of generic domain extenders proposed by Bhattacharyya et al. in [BMN09] since it allows post-processing and multiple inner-primitives. Finally, the bounds that we obtain for the examples Sponge and ChopMD that we exhibit are the best currently known for these constructions.

Chapter II

Notations and Preliminaries

II.1 Notations

Memories. Memories map variables to values. To describe a particular memory, we can describe the mapping by enumerating variables between brackets [and] using symbol $\mapsto$ between each variable and its value. For example, $[x \mapsto 0, y \mapsto 1]$ denotes the memory mapping two variables x and y, to values of respectively 0 and 1. The value associated by m to variable x is denoted by $m.x$. Given a memory m, and a variable x in its domain, $m.[x \mapsto 1]$ denotes the memory m' mapping every variable in the domain of m to the same value as m but x, which is mapped to 1.

Functions. Given a function f ranging over real numbers and a real number a, we write $f < a$ as a shorthand for $f(x) < a$ for every x in the domain of definition of f. Given two functions f and g, $f \circ g$ denotes the usual composition of functions $(\forall x, f \circ g(x) = f(g(x)))$, assuming inclusion of the range of g in the domain of f. Function $\mathbf{1}_a$ is the function defined as for all element b, $\mathbf{1}_a(b) = 1$ iff $a = b$, $\mathbf{1}_a(b) = 0$ otherwise. Finally, a function $f : A \rightharpoonup B$ is a partial function from A to B.

Bitstrings. The set of bitstrings of length n is denoted by $\{0,1\}^n$, while $\{0,1\}^*$ stands for the set of bitstrings of finite length. For $bs \in \{0,1\}^k$ and $\ell \leq \ell' \in [1..k]$, $bs[\ell, \ell']$ denotes the bitstring corresponding to the bits of bs at positions $\ell, \ldots, \ell'$ and $bs[\ell]$ denotes $bs[1, \ell]$, i.e. the prefix of bs of length ℓ. The length of bs is denoted by $|bs|$. Moreover, given a bitstring bs of length $l \geq n$ $\mathsf{First}_n(bs)$ denotes the prefix of length n of bs, while $\mathsf{Last}_n(bs)$ denotes the suffix of length n of bs. Furthermore, given two bistrings bs_1 and bs_2, $bs_1||bs_2$ denotes their concatenation while $bs_1 \oplus bs_2$ denotes their bitwise exclusive or. A string of length 0 is denoted λ.

Probabilities. Given a set finite Set, $\mathcal{D}(Set)$ is the set of distributions on Set. $\mathcal{U}(Set)$ denotes the uniform distribution on the elements of Set. When Set is the set of bitstrings of a given length m, $\mathcal{U}(m)$ is preferred to $\mathcal{U}(\{0,1\}^m)$. The Dirac distribution putting all weight

on element $a \in Set$ is denoted by δ_a, or $\delta(a)$ when the description of element a is lengthy. Given a distribution X, $x \leftarrow X$ denotes the operation of sampling a value according to X and assigning it to x.

Lists. Given a set A, we denote by A^* the set of finite lists with elements in A. The empty list is denoted by $[\,]$.

— Given a list L of k-tuples (e.g. elements of the form $(a_1, \ldots, a_k)$) we denote by $\mathsf{dom}_i(L)$ the list obtained by projecting each tuple on its i-th component. As we mostly use $\mathsf{dom}_1(L)$, we shortly denote it by $\mathsf{dom}(L)$.

— Given a list L of k-tuples, concatenation of k-tuple t is denoted $L :: t$ no matter whether t already appears in L. When we want to build association lists, we rather use $L.t$, which systematically suppresses all occurrences of t in L and concatenates it at the end of the list. Elimination of all occurrences of t in list L is denoted as $L - t$. Moreover, if we write k-tuple t as (a, t') with a $(k-1)$-tuple t', $L \bullet (a, t')$ denotes the list mapping a to t', i.e. the list defined as $L :: (a, t')$ if $a \notin \mathsf{dom}(L)$, and $(L - (a, \_)) :: (a, t')$ if $a \in \mathsf{dom}(L)$.

— $L[i]$ denotes the i-th element of list L of length at least i.

Miscellaneous. The unit type is denoted by $\mathbf{1}$. Given sets A and B, $A + B$ is the disjoint union of A and B. We use _ to denote elements which we do not need to name. The set of integers between two given integers a and b ($a < b$) is denoted $[a..b]$, while $[a, b]$ denotes the real interval defined by a and b. Given a real number x, $\lceil x \rceil$ denotes the ceiling of x.

II.2 Computational Security Definitions

II.2.1 — One-Way Function

A one-way function is a function with the property that one can compute the image of any element in reasonable time, whereas it is difficult, in a complexity-theoretic sense, to find a pre-image to a randomly sampled element from its range.

Definition (One-way Function). Let $f : \{0,1\}^m \to \{0,1\}^n$ be a function. The function f is said to be ε-one-way iff there exists a function $\varepsilon(t)$ such that for all adversary $\mathcal{A}$ running in time at most t,

$|\Pr[x \leftarrow \mathcal{U}(m); y := f(x); s \leftarrow \mathcal{A}(f, y) : f(s) = y]| \leq \varepsilon(t)$

In the sequel, we generally write $OW(t)$ for $\varepsilon(t)$. □

II.2.2 — Asymmetric Encryption Schemes

An asymmetric encryption scheme consists of three algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$: the first generates the public and secret keys, the second one is the encryption algorithm and the third is the decryption algorithm. For the scheme to remain coherent (and the addressee to be able to recover its message), encryption and decryption are required to verify *functional correctness*: for any message m, $\mathcal{D} \circ \mathcal{E}(m) = m$.

We provide here a concrete formalization of indistinguishability under chosen plaintext attack (IND-CPA) (equivalent to the other classic notion of semantic security [GM84]), of an asymmetric encryption scheme.

Definition (ε-IND-CPA Security). Given a function $\epsilon : \mathbb{N} \times \mathbb{N} \to [0,1]$, an asymmetric encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is ε-secure iff for all (k,t)-adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ provided with access to the encryption oracle $\mathcal{E}$,

$$\begin{aligned}|\Pr[(pk,sk) \leftarrow \mathcal{K}; (m_0, m_1, \sigma) \leftarrow \mathcal{A}_1^{\mathcal{E}}(pk); y \leftarrow \mathcal{E}(pk, m_0); b \leftarrow \mathcal{A}_2^{\mathcal{E}}(y, \sigma) : b = 0] \\ - \Pr[(pk,sk) \leftarrow \mathcal{K}; (m, \sigma) \leftarrow \mathcal{A}_1^{\mathcal{E}}(pk); y \leftarrow \mathcal{E}(pk, m_1); \\ b \leftarrow \mathcal{A}_2^{\mathcal{E}}(y, \sigma) : b = 0]| \leq \varepsilon(k,t)\end{aligned}$$

□

A stronger criterion, indistinguishability under ciphertext attack (IND-CCA), consists in imposing the same bound but when providing the adversary with oracle access to the decryption algorithm too. In addition to these two, there exist a lot of variants of security criteria for asymmetric encryption schemes. For more details and classification of criteria, see for example [BDPR98, BBM00, PP04, BHK09]. In our work, we use a formalization of a criterion called Real-Or-Random Ciphertext (ROR-C) security, appearing in [BDJR97]. It resembles a lot to semantic security, except that the adversary is supposed to distinguish whether it is provided with an actual ciphertext corresponding to the plaintext it submitted, or rather with a randomly sampled bitstring of the same length as the actual ciphertext. Consequently, it is a stronger criterion than IND-CPA (and implies it). A frequent alternative to ROR-C security is Real-Or-Random Plaintext security, in which game the adversary gets either a ciphertext matching its message or the encryption of a random plaintext.

Definition (ε-Real-Or-Random Ciphertext Security). Given a function $\epsilon : \mathbb{N} \times \mathbb{N} \to [0,1]$, and $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an asymmetric encryption scheme. It is ε-Real-or-Random Ciphertext secure iff there exists a function ε ranging in $[0,1]$ such that for all (k,t) adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,

$$\begin{aligned}|\Pr[(pk,sk) \leftarrow \mathcal{K}; (m, \sigma) \leftarrow \mathcal{A}_1^{\mathcal{E}}; y \leftarrow \mathcal{E}(pk, m); b \leftarrow \mathcal{A}_2^{\mathcal{E}}(y, \sigma) : b = \mathsf{true}] \\ - \Pr[(pk,sk) \leftarrow \mathcal{K}; (m, \sigma) \leftarrow \mathcal{A}_1^{\mathcal{E}}; y \leftarrow \mathcal{E}(pk, m); \\ y \leftarrow \mathcal{U}(|y|); b \leftarrow \mathcal{A}_2^{\mathcal{E}}(y, \sigma) : b = \mathsf{true}]| \leq \varepsilon(k,t)\end{aligned}$$

□

II.2.3 — Signature Schemes

Besides privacy, we have briefly mentioned another security goal: authentication, i.e. the willingness to ensure that a message was issued by who we think it was. Though there exists designs to achieve this constraint in the symmetric setting, we only here present the asymmetric constructions for authentication, namely digital signature schemes. There is no reason other than our choice of examples to account for this restriction.

The idea behind the signature of a message is to create something characterizing uniquely the signer and the message, in the same way one can sign a paper. Moreover, the signer wants anybody to be able to verify that it is indeed them who signed a message, in the same way that we can read the name signed at the bottom of a letter. A notorious problem of pencil-and-paper signatures is forgery: it is quite easy to train and imitate correctly the signature of somebody else. Therefore, we would like the signer to add a secret in the creation process of message signatures, so that forgeries become difficult. On the contrary to asymmetric encryption, the idea is there to use a secret key to sign and then send a message, while a public key is used to verify the validity of signatures.

Formally, a digital signature scheme consists of three algorithms $(\mathcal{K}, \mathcal{S}, \mathcal{V})$. The first algorithm is used to generate keys, usually called a signature key (meant to remain secret) and a verification key (meant to be broadcasted). The signature algorithm takes as inputs a signature key and a message and produces a signed message. The verification algorithm

takes as input a signed message, a verification key, and the message, and verifies whether signed message and message are a match using the verification key. As in the encryption case, schemes are generally probabilistic, and we impose *functional correctness*: for all message m, $\mathcal{V}(\mathcal{S}(m), pk, m) = \mathsf{true}$.

Quite intuitively, security criteria imposed on signatures deal with hardness of forgery. We present existential forgery against chosen message attack (EF-CMA) (see [GMR88]). The idea is to assess the ability of the adversary to forge the valid signature of a message of its choice; a valid forgery is characterized by the fact that it is not the product of a query of the signature oracle, and the fact that it passes verification. The formalization of this criterion is as follows.

Definition (Existential Forgery Security of Signature Schemes). Let $(\mathcal{K}, \mathcal{S}, \mathcal{V})$ be a digital signature scheme. It is ε-secure for existential forgery against chosen message attack iff for all (k, t)-adversary $\mathcal{A}$,

$|Pr[(pk, sk) \leftarrow \mathcal{K}; (m, s) \leftarrow \mathcal{A}^{\mathcal{S}}(pk) : \mathcal{V}(s, pk, m) = \mathsf{true} \wedge m \notin Query(\mathcal{S})] \leq \varepsilon(k, t)$ □

In the same way as for encryption, we could choose to provide $\mathcal{A}$ with oracle access to a verification oracle, alternatively to giving it the verification key. It influences a little the time bound we get in practice, but is essentially the same.

II.2.4 — Formalizing Randomness of Atomic Primitives

At the heart of security proofs of many probabilistic cryptographic schemes or hash constructions lies properties of randomness of one or several of its constituents. Hence, we propose to provide some details about the computational formalization of randomness properties.

We call a function family a map $F : Keys \times D \to R$, where $Keys$ is a finite set of keys and D and R are the domain and finite range of the functions in the family. The functions of the family are $(F_K)_{K \in Keys}$, where $F_K(x) = F(K, x)$. We let $\mathsf{Fun}(D, R)$ denote the family of *all* functions from D to R. What we formalize is not actually randomness of a function of $\mathsf{Fun}(D, R)$. We rather equip the key space with the uniform distribution, and assess the ability of an adversary to distinguish between two worlds: one in which a key is drawn at random and it interacts with $g = F_K$ and one in which a function g is drawn at random in $\mathsf{Fun}(D, R)$. As a result, the randomness property refers to the way the function g is sampled, rather than to a feature of the function g itself. What is expressed can be thought of as a property of the coverage of the whole set of functions from D to R by a family F. This is formalized in the following way.

Definition (Pseudo-Randomness of Families of Functions). Let $F : Keys \times D \to R$ be a family of functions. The family is ε-pseudo-random iff for all (k, t)-adversary $\mathcal{A}$,

$|\Pr[K \leftarrow \mathcal{U}(Keys); b \leftarrow \mathcal{A}^{F_K} : b = \mathsf{true}] - \Pr[g \leftarrow \mathcal{U}(\mathsf{Fun}(D, R)); b \leftarrow \mathcal{A}^{g} : b = \mathsf{true}]| \leq \varepsilon(k, t)$

□

It is all the more difficult to really keep in mind that pseudo-randomness is a property of function families than we often use a dynamic way of implementing a random function g (where "random", we insist, refers to the fact that g is sampled at random in $\mathsf{Fun}(D, R)$). In the dynamic view, instead of drawing at the beginning of the probabilistic experiment a function g, we draw new images for elements of D by g whenever the adversary asks for them. This is done by maintaining an association list L_g of pairs $(x, g(x))$ which the adversary has queried on, and drawing uniformly in R a new element in case of a fresh query. This

implementation is currently used in practice.

In the thesis presented here, though our framework is totally *independent* of any hypothesis we make on atomic cryptographic primitives, we propose proofs of constructions in two models. On the one hand, we prove hash constructions under the hypothesis that the fixed-input length inner primitives on top of which they are built are random functions or permutations. As these inner primitives are often instantiated by block-ciphers, we could say that we prove hash constructions in the *Ideal-Cipher Model* (ICM) - which means that we identify block-ciphers with random permutations and provide oracle access to them and their inverses to any adversary attacking the construction. Similarly, when the hash functions are considered as atomic primitives, it is often the case that security proofs are performed under the hypothesis that they are random oracles: this is the *Random Oracle Model* (ROM) [BR93]. Here again, the adversary is granted oracle access to all hash functions. Classically, in the examples we present, we instantiate random oracles using the dynamic approach described above.

Chapter III

The Computational Indistinguishability Logic

III.1 Semantics and Judgments

Usual security criteria of the computational model are defined in terms of the probability of occurence of an event at the end of a probabilistic experiment. These experiments involve three kinds of elements: drawings (e.g. keys), adversaries, and the oracles these latter can query. The event captures the capacity of an adversary to achieve a goal (e.g. compute a specific value) depending on the results of drawings and oracle queries. The probabilistic experiments describing security criteria involve two parties. On one side, there is an adversary, which can do whatever it wants and on which we do not impose any restriction but maximum total running time and maximum number of oracle queries. On the other side, we have all the remaining computations and drawings, e.g. initialization of keys, specification of oracle behavior, etc. We chose to mirror this decomposition in our formalization of security experiments as the interaction of two entities: adversaries and oracle systems.

III.1.1 — Oracle Systems and Adversaries

In this section, we define oracle systems and adversaries. They are modeled as stateful systems meant to interact with each another. Let us start with oracle systems.

Definition (Oracle System). An *oracle system* $\mathbb{O}$ is given by:

— a finite set $\mathsf{M}_{\mathbb{O}}$ of oracle memories (or states) and an initial memory $\bar{m}_{\mathbb{O}} \in \mathsf{M}_{\mathbb{O}}$,

— a finite set $\mathsf{N}_{\mathbb{O}}$ of oracle names,

— for each $o \in \mathsf{N}_{\mathbb{O}}$, a query domain $\mathsf{In}(o)$, an answer domain $\mathsf{Out}(o)$ and an implementation:

$$\mathsf{Imp}(o) : \mathsf{In}(o) \times \mathsf{M}_{\mathbb{O}} \to \mathcal{D}(\mathsf{Out}(o) \times \mathsf{M}_{\mathbb{O}})$$

— distinguished oracles o_{I} for initialization and o_{F} for finalization, such that $\mathsf{In}(o_{\mathrm{I}}) = \mathsf{Out}(o_{\mathrm{F}}) = \mathbf{1}$. We let $\mathsf{Res} = \mathsf{In}(o_{\mathrm{F}})$.

□

Note that the initialization oracle takes no input, while the finalization oracle has no

output. This is captured by requiring that the corresponding types are $\mathbf{1}$. However, these distinguished oracles can modify the state of the system. Intuitively, the finalization oracle is the oracle which the adversary is supposed to call with its result whenever it feels that it has achieved its security goal. This motivates the distinguished notation Res for the input type of oracle o_{F}.

Example 1. We consider a classic game that can be played by two parties as follows. One participant draws a number N between 0 and 100 at random and keeps it secret. The other player has to guess this secret number. Before providing its final guess, the second player can give a number N' to the first one, who replies whether N' is superior or inferior to N.

The first player can be modeled as an oracle system $\mathbb{O}$:

— memories $\mathsf{M}_{\mathbb{O}}$ of this system map variable N to a value between 0 and 100. We choose $[N \mapsto 0]$, the memory where N is assigned 0, as the initial memory.

— in addition to o_{I}, o_{F}, $\mathsf{N}_{\mathbb{O}}$ contains $Comp$.

— oracle $Comp$ has for input domain $\mathsf{In}(Comp) = [0..100]$ and outputs values in $\mathsf{Out}(Comp) = \{\leq, \geq\}$. Its implementation is the following:

$$\begin{array}{lrcl} \mathsf{Imp}(Comp): & [0..100] \times \mathsf{M}_{\mathbb{O}} & \to & \mathcal{D}(\{\leq, \geq\} \times \mathsf{M}_{\mathbb{O}}) \\ & (M, [N \mapsto N_0]) & \mapsto & \mathsf{if}\ M \leq N_0\ \mathsf{then}\ (\leq, [N \mapsto N_0]) \\ & & & \mathsf{else}\ (\geq, [N \mapsto N_0]) \end{array}$$

The initialization oracle has $\{0\}$ for output domain, and is implemented by:

$$\begin{array}{lrcl} \mathsf{Imp}(o_{\mathrm{I}}): & \mathbf{1} \times \mathsf{M}_{\mathbb{O}} & \to & \mathcal{D}(\{0\} \times \mathsf{M}_{\mathbb{O}}) \\ & (\_, \_) & \mapsto & \mathsf{let}\ N_0 \leftarrow \mathcal{U}([0..100])\ \mathsf{in}\ (0, [N \mapsto N_0]) \end{array}$$

Finally, the finalization oracle has $[0..100]$ as input domain and has an "idle" implementation. Namely,

$$\begin{array}{lrcl} \mathsf{Imp}(o_{\mathrm{F}}): & [0..100] \times \mathsf{M}_{\mathbb{O}} & \to & \mathcal{D}(\mathbf{1} \times \mathsf{M}_{\mathbb{O}}) \\ & (M, [N \mapsto N_0]) & \mapsto & (\mathbf{1}, [N \mapsto N_0]) \end{array}$$

$\Diamond$

For this example, we have used functional description of the implementations, showing explicitly the side-effect on memories. To describe more involved examples of implementations, we rather use usual imperative commands augmented with probabilistic sampling.

Definition (Compatibility). Two oracle systems are *compatible* iff they have the same set of names, and each name has identical query and answer domains in each system. However, state space and implementations may vary. $\square$

In practice, we often build compatible systems out of systems we have already defined by modifying the implementation of one of the oracles. When doing so, we only explain the modification and do not specify the whole system again.

Example 2. In the previous example, if we change the implementation of the oracle $Comp$ so that whenever queried on $M = N_0$, the oracle answers $\geq$, we obtain a new oracle system compatible with the original one. $\Diamond$

For a given oracle system $\mathbb{O}$, we call *exchange* a triple of the form (o, q, a), consisting of an oracle name $o \in \mathsf{N}_{\mathbb{O}}$, a query $q \in \mathsf{In}(o)$ and an answer $a \in \mathsf{Out}(o)$. Their set is denoted by Xch. Initial and final exchanges are distinguished by appropriate indices. Finally, the set Que of queries (resp. Ans of answers) is defined as $\{(o, q) \mid (o, q, a) \in \mathsf{Xch}\}$ (resp. $\{a \mid (o, q, a) \in \mathsf{Xch}\}$). We often abusively omit the oracle name in queries when it is clear from the context.

Once we have defined an oracle system $\mathbb{O}$, we can define matching adversaries. Intuitively, adversaries ask queries to one of the oracle in the system and receive answers. They are thus

formalized by a function to compute new queries and a function to update their state once they get an answer back.

Definition (Adversary). An $\mathbb{O}$-*adversary* $\mathbb{A}$ is given by a finite set $\mathsf{M}_{\mathbb{A}}$ of adversary memories, an initial memory $\bar{m}_{\mathbb{A}} \in \mathsf{M}_{\mathbb{A}}$ and (possibly partial) function for querying and a function for updating:

$$\begin{array}{rcl} \mathsf{A} & : & \mathsf{M}_{\mathbb{A}} \rightharpoonup \mathcal{D}(\mathsf{Que} \times \mathsf{M}_{\mathbb{A}}) \\ \mathsf{A}_{\downarrow} & : & \mathsf{Xch} \times \mathsf{M}_{\mathbb{A}} \to \mathcal{D}(\mathsf{M}_{\mathbb{A}}) \end{array}$$

□

III.1.2 — Formalization of the Interaction

Informally, the interaction between an oracle system and an adversary proceeds in three successive phases. During the first phase, the initialization oracle sets the initial memory distributions of the oracle system and of the adversary. Then, in a second phase, $\mathbb{A}$ performs computations, updates its state and submits a query to $\mathbb{O}$. In turn, $\mathbb{O}$ performs computations, updates its state, and replies to $\mathbb{A}$, which updates its state. This goes on until finally, $\mathbb{A}$ outputs a result by calling the finalization oracle, which is the third and last phase.

This interaction can be seen as the iteration of the pattern consisting in the query of an oracle, the computation of an answer by the oracle, and the update of its state by the adversary. To formalize these intuitions, we provide our definition of a probabilistic transition system and execution sequences.

Definition (Transition System). A *transition system* $\mathcal{S}$ consists of:

— a (countable non-empty) set M of memories (states), with a distinguished initial memory $\bar{m}$,

— a set Σ of actions, with distinguished subsets Σ_{I} and Σ_{F} of initialization and finalization actions,

— a (partial probabilistic) transition function $\mathsf{st} : \mathsf{M} \rightharpoonup \mathcal{D}(\Sigma \times \mathsf{M})$.

□

A *partial execution sequence* of $\mathcal{S}$ is a sequence η of the form

$$m_0 \xrightarrow{x_1} m_1 \xrightarrow{x_2} \ldots \xrightarrow{x_k} m_k$$

such that $m_0 = \bar{m}$, $x_i \in \Sigma$, $m_{i-1}, m_i \in \mathsf{M}$, and $\Pr[\mathsf{st}(m_{i-1}) = (x_i, m_i)] > 0$ for $i = 1 \ldots k$. Their set is denoted $\mathsf{PExec}(\mathcal{S})$. Any subsequence of the form $m \xrightarrow{x} m'$ is called a step, and often denoted σ. Given a partial execution $\eta = m_0 \ldots \xrightarrow{x_k} m_k$, the concatenation of η and a step σ of the form $m_k \xrightarrow{x_{k+1}} m_{k+1}$ is a partial execution $m_0 \ldots \xrightarrow{x_k} m_k \xrightarrow{x_{k+1}} m_{k+1}$ denoted by $\eta \cdot \sigma$. Moreover, if $1 \leq i \leq k$, $\mathsf{Pref}(\eta, i)$ denotes the prefix of length i of η, i.e. $m_0 \ldots \xrightarrow{x_i} m_i$. The i-th step of η, denoted $\eta[i]$, is $m_{i-1} \xrightarrow{x_i} m_i$ and $Last(\eta)$ denotes the last step of partial execution η.

If $x_1 \in \Sigma_{\mathrm{I}}$ and either $x_k \in \Sigma_{\mathrm{F}}$ or $m_k \notin \mathsf{dom}(\mathsf{st})$[1], then η is an *execution sequence* (or execution) of length k. An execution sequence is thus a partial execution sequence starting with an initialization action and terminating with a finalization action or in a state where no transition is defined. The set of executions is denoted by $\mathsf{Exec}(\mathcal{S})$.

[1] We consider this case to capture partial executions that reach an end without last step involving a finalization action. This is useful in the proof of rule *Cut*.

A finite partial execution sequence η is mapped to a probability corresponding to the product of the probabilities of taking each of its steps successively. Consequently, a probabilistic transition system $\mathcal{S}$ yields a sub-distribution on finite executions denoted again $\mathcal{S}$:

$$\Pr[\mathcal{S} = \eta] = \Pi_{i=1}^{k} \Pr[\mathsf{st}(m_{i-1}) = (x_i, m_i)],$$

when $\eta = m_0 \ldots \xrightarrow{x_k} m_k$. We notice that $\mathcal{S}$ is not a distribution since it is possible that infinite execution sequences have positive weight. If all executions of a system have length at most k, then we say that the system is of height k. In such a case, $\mathcal{S}$ is a distribution.

The interaction between adversary and oracles can be formalized in terms of a transition system, where a step consists in one occurence of the following three step pattern: the adversary computes a query to submit, the queried oracle computes an answer to this query and then the adversary updates its state with this answer.

Definition (Composition $\mathbb{A} \mid \mathbb{O}$). Let $\mathbb{O}$ be an oracle system and $\mathbb{A}$ be an $\mathbb{O}$-adversary . The composition $\mathbb{A} \mid \mathbb{O}$ is a transition system such that:

— the memories are $\mathsf{M} = \mathsf{M}_{\mathbb{A}} \times \mathsf{M}_{\mathbb{O}}$ and the initial memory is $(\bar{m}_{\mathbb{A}}, \bar{m}_{\mathbb{O}})$,

— the set of actions is $\Sigma = \mathsf{Xch}$, and initialization (resp. finalization) actions are initial (resp. final) exchanges (i.e. $\Sigma_{\mathrm{I}} = \mathsf{Xch}_{\mathrm{I}}$ and $\Sigma_{\mathrm{F}} = \mathsf{Xch}_{\mathrm{F}}$),

— the step function is given by:

$$\begin{aligned} \mathsf{st}_{\mathbb{A}|\mathbb{O}}(m_{\mathbb{A}}, m_{\mathbb{O}}) \stackrel{def}{=}\ & \mathsf{let}\ ((o,q), m'_{\mathbb{A}}) \leftarrow \mathsf{A}(m_{\mathbb{A}})\ \mathsf{in} \\ & \mathsf{let}\ (a, m'_{\mathbb{O}}) \leftarrow \mathsf{Imp}(o)(q, m_{\mathbb{O}})\ \mathsf{in} \\ & \mathsf{let}\ m''_{\mathbb{A}} \leftarrow \mathsf{A}_{\downarrow}((o,q,a), m'_{\mathbb{A}})\ \mathsf{in} \\ & \mathsf{return}\ ((o,q,a), (m''_{\mathbb{A}}, m'_{\mathbb{O}})) \end{aligned}$$

□

We are interested in expressing concrete security results. Therefore, we consider adversaries whose resources are bounded by two parameters: the number of queries they perform to the oracles and their running time. Bounds on the number of oracle calls are specified by giving a function $k : \mathsf{N}_{\mathbb{O}} \to \mathbb{N}$, mapping each oracle name to a maximum number of queries.

Definition ((k,t)-bounded Adversary). Given function $k : \mathsf{N}_{\mathbb{O}} \to \mathbb{N}$ and integer $t \in \mathbb{N}$, an $\mathbb{O}$-adversary is (k,t)-bounded iff:

— it runs in time at most t, i.e. the total time taken by the executions of the adversarial functions A and $\mathsf{A}_{\downarrow}$ (*but not* the time to get answers from oracles) is bounded by t,

— for all $o \in \mathsf{N}_{\mathbb{O}}$ and all executions η of $\mathbb{A} \mid \mathbb{O}$, the number of queries to o in η is at most $k(o)$.

The set of (k,t)-bounded adversaries is denoted by $\mathsf{Adv}(k,t)$. □

We say that an adversary is bounded iff there exists such a pair (k,t).

III.1.3 — Computing the Probability of Events

When reasoning about the security of a system, we do not want to impose any restriction on the state of adversaries. Thus, we reason on events abstracting away these latter, only keeping track of exchanges and oracle memories. This motivates the introduction of traces.

Definition (Trace). Let $\mathbb{O}$ be an oracle system. An $\mathbb{O}$-*partial trace* is a sequence τ of

the form

$$m_0 \xrightarrow{x_1} m_1 \xrightarrow{x_2} \ldots \xrightarrow{x_k} m_k$$

where $m_i \in \mathsf{M}_{\mathbb{O}}$ and $x_i = (o_i, q_i, a_i) \in \mathsf{Xch}$ such that $\Pr[\mathsf{Imp}(o_i)(q_i, m_{i-1}) = (a_i, m_i)] > 0$ for $i = 1 \ldots k$.

The projection mapping sequences of steps to partial traces is denoted by $\mathcal{T}$. An $\mathbb{O}$-partial trace is an $\mathbb{O}$-*trace* iff it is the projection of an execution sequence. □

Definition (Event). An $\mathbb{O}$-event E is a predicate over $\mathbb{O}$-traces.

An extended $\mathbb{O}$-event E is a predicate over partial $\mathbb{O}$-traces. □

To assign a probability to an event defined by a given predicate, we consider the subset of traces verifying the predicate. Several executions can project to the same trace. The probability of a trace τ can then be obtained by summing the weights of all executions projecting to τ. Similarly, the probability of an event is defined as the sum of weights of all traces satisfying the predicate.

Definition (Probability of an (Extended) Event). The probability of an (extended) event is derived from the definition of $\mathbb{A} \mid \mathbb{O}$:

$$\Pr[\mathbb{A}|\mathbb{O} : E] \quad = \quad \sum_{\{\eta \in \mathsf{Exec}(\mathbb{A}|\mathbb{O}) | E(\mathcal{T}(\eta)) = \mathsf{true}\}} \Pr[\mathbb{A}|\mathbb{O} : \eta]$$

In particular, the probability that a system yields a trace τ is the sum of probabilities that the system yields execution η projecting to τ, which we write:

$\Pr[\mathbb{A}|\mathbb{O} : \tau] = \sum_{\{\eta \in \mathsf{Exec}(\mathbb{A}|\mathbb{O}) | \mathcal{T}(\eta) = \tau\}} \Pr[\mathbb{A}|\mathbb{O} : \eta]$. □

Proofs in CIL use several common operations on (extended) events and traces. First, one can define the conjunction, disjunction and all other classic logic operators on events.

Definition (Step Predicate). We call a step-predicate any predicate ϕ on $\mathsf{Xch} \times \mathsf{M}_{\mathbb{O}} \times \mathsf{M}_{\mathbb{O}}$. □

We use temporal logic operators combined with step-predicates. We define for a step-predicate ϕ the event "eventually ϕ", denoted by F_ϕ, corresponding to ϕ being satisfied at one step of a trace. Furthermore, the event "always ϕ", denoted by G_ϕ, is true iff ϕ is satisfied at every step of the trace.

On the other hand, we introduce two more technical operators, corresponding to the usual "until" along with a stronger version of the latter, as follows. If E denotes an event, then $E \,\mathsf{U}\, \phi$ holds if ϕ becomes true at some step of the trace and E holds until this step. Formally, if τ is a trace of the form:

$$m_0 \xrightarrow{x_1} m_1 \xrightarrow{x_2} \ldots \xrightarrow{x_k} m_k$$

then $(E \,\mathsf{U}\, \phi)(\tau) = \mathsf{true}$ iff there exists $i \in [1..k]$ such that $\phi(x_i, m_{i-1}, m_i)$ holds and $E(\tau[i-1]) = \mathsf{true}$, where $\tau[i-1]$ is the partial trace from m_0 to m_{i-1}. Finally, we say that $(E; \phi)(\tau) = \mathsf{true}$ iff $E(\tau[k-1])$ and $\phi(x_k, m_{k-1}, m_k)$. Intuitively, $E; \phi$ holds if ϕ is true at the last transition, and E is always true before that.

Example 3. Let us provide some examples of events by expressing different winning conditions for the guessing game defined previously. The adversary plays the role of the second participant.

— The adversary wins if it queries the finalization oracle on the value N_0 of N. The predicate ϕ_1 formalizing "querying the finalization oracle with N_0" can be written as $\phi_1 = \lambda((o, q, a), m_{\mathbb{O}}, m'_{\mathbb{O}}).(o = o_{\mathrm{F}}) \wedge (q = m_{\mathbb{O}}.N_0)$. The winning event is then F_{ϕ_1}.

— The adversary wins if it manages to issue a query equal to N_0. "Querying an oracle

with N_0" is formalized via the step-predicate $\phi_2 = \lambda((o,q,a), m_{\mathbb{O}}, m'_{\mathbb{O}}).(q = m_{\mathbb{O}}.N_0)$. Our winning event is F_{ϕ_2}.

— The adversary wins if it queries the finalization oracle on N_0 but has never issued a query worth N_0 before. The winning condition is satisfied for traces for which at every step before last, $\neg\phi_2$ holds, and for the last step, ϕ_1 does. This is expressed by $\mathsf{G}_{(\neg\phi_2)}; \phi_1$.[2]

$\Diamond$

We often need to express that the finalization oracle is queried on a value satisfying a given constraint formalized by a property P. We thus introduce the notation R to designate the value of the query performed to o_{F}, and write $P(\mathrm{R})$ as a short for event $\mathsf{F}_{\lambda((o,q,a),m_{\mathbb{O}},m'_{\mathbb{O}}).(o=o_{\mathrm{F}})\wedge P(q)}$. If P depends on the memory and the finalization oracle modifies memories, we cannot use this notation and need to precise in which memory variables are to be evaluated.

III.1.4 — Two Judgments

Concrete security proofs use two kinds of statements on probabilities: the fact that the probability of an event is bounded by a function of the resources of the adversary, and the indistinguishability of the distributions yielded by systems. We formalize below these two types of statements.

We generically denote by $\varepsilon : ((\mathsf{N}_{\mathbb{O}} \to \mathbb{N}) \times \mathbb{N}) \to [0,1]$ the function mapping adversarial resources (k,t) to a value between 0 and 1.

Definition (Judgment Bounded-by). A statement $\mathbb{O} :_{\varepsilon} E$ is *valid*, written $\vdash \mathbb{O} :_{\varepsilon} E$, iff for every (k,t)-bounded adversary $\mathbb{A}$,

$$\Pr[\mathbb{A}|\mathbb{O} : E] \leq \varepsilon(k,t)$$

$\square$

Example 4. Let $\mathbb{G}$ be a group of prime order q and $Gen = \mathbb{G} \setminus 1_{\mathbb{G}}$ the set of its generators. We want to express the Computational Diffie Hellman (CDH) assumption in $\mathbb{G}$. We define an oracle system $\mathbb{CDH}$ as follows:

— the memories map variable g to values in Gen, and variables α and β to values $[0..(q-1)]$,

— the initialization oracle draws uniformly at random a value for g and α, β, and outputs $(g, g^{\alpha}, g^{\beta})$.

— the finalization oracle takes as input an element of $\mathbb{G}$ (in addition to a memory).

Here, bounding the number of calls of the adversary to oracles is irrelevant; we let $\mathbf{1}_k$ denote the function mapping o_{I} and o_{F} to 1.

Given a function ε, the ε-CDH assumption holds for group $\mathbb{G}$ iff for all $(\mathbf{1}_k, t)$-adversary we have $\varepsilon - CDH \ \vdash \mathbb{CDH} :_{\varepsilon(\mathbf{1}_k,t)} \mathrm{R} = g^{\alpha.\beta}$. $\Diamond$

Definition (Judgment Indistinguishable from). Let $\mathbb{O}$ and $\mathbb{O}'$ be compatible oracle systems which expect a boolean as result (i.e. $\mathsf{Out}_{\mathrm{F}} = \mathbf{1}$). Statement $\mathbb{O} \sim_{\epsilon} \mathbb{O}'$ is *valid*, written $\vdash \mathbb{O} \sim_{\epsilon} \mathbb{O}'$, iff for every (k,t)-adversary $\mathbb{A}$,

$$|\Pr[\mathbb{A}|\mathbb{O} : \mathrm{R} = \mathsf{true}] - \Pr[\mathbb{A}|\mathbb{O}' : \mathrm{R} = \mathsf{true}]| \leq \epsilon(k,t)$$

$\square$

[2] Here $\mathsf{G}_{(\neg\phi_2)}\,\mathsf{U}\,\phi_1$ is also satisfactory since traces cannot contain two calls to the finalization oracle.

Example 5. We keep the notations introduced previously and provide the formalization of the Decisional Diffie-Hellman (DDH) hypothesis in group $\mathbb{G}$. We define two oracles systems $\mathbb{R}eal\mathbb{DH}$ and $\mathbb{R}and\mathbb{DH}$. The first system has memories mapping variable g to some fixed value g_0, and variables α and β to integers in $[0..(q-1)]$. Its initialization oracle draws values for α and β uniformly at random which we denote α_0 and β_0. The output is the triple $(g_0^{\alpha_0}, g_0^{\beta_0}, g_0^{\alpha_0 \cdot \beta_0})$. The finalization oracle is characterized by $\mathsf{Res} = \mathsf{Bool}$.

$\mathbb{R}and\mathbb{DH}$ is a system compatible with $\mathbb{R}eal\mathbb{DH}$, but with memories with an additional variable r mapped to $[0..(q-1)]$. The initialization oracle draws values for α, β and r, denoted with index 0. It outputs the triple $(g_0^{\alpha_0}, g_0^{\beta_0}, g_0^{r_0})$.

The ε-DDH hypothesis holds for group $\mathbb{G}$ iff for all $(\mathbf{1}_k, t)$-adversary we have $\varepsilon - DDH \vdash \mathbb{R}eal\mathbb{DH} \sim_{\varepsilon(\mathbf{1}_k,t)} \mathbb{R}and\mathbb{DH}$. $\Diamond$

Proofs often rely on assumptions (e.g. DDH). Hence, CIL deals with sequents of the form $\Delta \vdash \phi$, where Δ is a set of statements (the assumptions) possibly formalized outside our framework, and ϕ is a statement (the conclusion) in our framework, that is, either a bounded-by or an indistinguishability statement.

We omit hypotheses in the sequel of our presentation, as well as the usual structural and logical rules for sequent calculi, e.g. if $\Delta_1 \subseteq \Delta_2$ and $\Delta_1 \vdash \phi$ then $\Delta_2 \vdash \phi$.

III.2 Basic Rules

We start with three rules translating in the system that the indistinguishability relation is an equivalence relation.

Lemma III.1 (Refl, Sym, Trans). *The following rules Refl, Sym and Trans are sound.*

$$\frac{}{\mathbb{O} \sim_0 \mathbb{O}}\ \textit{Refl} \qquad \frac{\mathbb{O} \sim_\epsilon \mathbb{O}'}{\mathbb{O}' \sim_\epsilon \mathbb{O}}\ \textit{Sym} \qquad \frac{\mathbb{O} \sim_\epsilon \mathbb{O}' \qquad \mathbb{O}' \sim_{\epsilon'} \mathbb{O}''}{\mathbb{O} \sim_{\epsilon+\epsilon'} \mathbb{O}''}\ \textit{Trans}$$

We omit the proof of this lemma, which follows directly from the definition of indistinguishability for reflexivity and symmetry, and from the triangle inequality for transitivity.

In addition, CIL features a rule which proves useful to close branches in proof trees: given a bound on the probability that a step-predicate ϕ holds for one query, the rule allows to bound the probability of F_ϕ w.r.t. the total number of calls. To do so, we introduce for every $(o, q, a) \in \mathsf{Xch}$ and every m_1 a global bound on the probability that $\phi((o, q, a), m_1, m_1')$ holds for m_1'. In particular, we have that given (o, q, a) and memories m_1 and m_1',

$$Pr[\mathsf{Imp}(o)(q, m_1) = (a, m_1') \wedge \phi((o, q, a), m_1, m_1')] \leq \sum_{\substack{m' \in \mathsf{M}_{\mathbb{O}} \\ \phi((o,q,a),m,m')}} Pr[\mathsf{Imp}(o)(q, m) = (a, m')]$$

As a consequence, for an exchange with o,

$$\epsilon_\phi(o) = \max_{\substack{q \in \mathsf{Que}, m \in \mathsf{M}_{\mathbb{O}} \\ a \in \mathsf{Ans}}} \sum_{\substack{m' \in \mathsf{M}_{\mathbb{O}} \\ \phi((o,q,a),m,m')}} Pr[\mathsf{Imp}(o)(q, m) = (a, m')]$$

bounds the probability to satisfy ϕ when querying oracle o.

Lemma III.2 (Fail). *Rule Fail defined as follows is sound.*

$$\frac{}{\mathbb{O} :_\varepsilon \mathsf{F}_\phi}\ \textit{Fail} \quad \textit{where } \varepsilon = \lambda(k, t).\ \textstyle\sum_{o \in \mathsf{N}_{\mathbb{O}}} k(o)\epsilon_\phi(o).$$

Proof. Let $\mathbb{A}$ be a (k,t)-adversary for oracle system $\mathbb{O}$. We denote by T the set of traces satisfying F_ϕ,

To show our inequality, we divide traces of set T in subgroups using an equivalence relation. Two traces are related iff ϕ is true *for the first time* at step i for a query to oracle o. Classes are denoted $\mathcal{C}(i,o)$. They realize a partition of T.

Let $\tau \in \mathcal{C}(i,o)$. We recall that $\mathsf{Pref}(\eta,i)$ is the prefix of length i of partial execution η, and $\eta[i]$ its i-th step. Then, we have:

$$\begin{aligned}
&\sum_{\tau\in\mathcal{C}(i,o)} \Pr[\mathbb{A} \mid \mathbb{O} : \tau] \\
= &\sum_{\substack{\tau\in\mathcal{C}(i,o)\\ \mathcal{T}(\eta)=\tau}} \Pr[\mathbb{A} \mid \mathbb{O} : \eta] \\
\leq &\sum_{\substack{\tau\in\mathcal{C}(i,o)\\ \mathcal{T}(\eta)=\tau}} \Pr[\mathbb{A} \mid \mathbb{O} : \mathsf{Pref}(\eta,i)] \\
= &\sum_{\substack{\tau\in\mathcal{C}(i,o)\\ \mathcal{T}(\eta)=\tau}} \Pr[\mathbb{A} \mid \mathbb{O} : \mathsf{Pref}(\eta,i-1)]\Pr[\mathbb{A} \mid \mathbb{O} : \eta[i]] \\
\leq &\sum_{\substack{\tau\in\mathcal{C}(i,o),\mathcal{T}(\eta)=\tau\\ \mathcal{T}(\eta[i])=((o,q,a),m_{\mathbb{O}},m'_{\mathbb{O}})}} \Pr[\mathbb{A} \mid \mathbb{O} : \mathsf{Pref}(\eta,i-1)].\Pr[\mathsf{Imp}(o)(q,m_{\mathbb{O}}) = (a,m'_{\mathbb{O}})] \\
\leq &\sum_{\substack{\tau\in\mathcal{C}(i,o),\mathcal{T}(\eta)=\tau\\ \mathcal{T}(\eta[i])=((o,q,a),m_{\mathbb{O}},m'_{\mathbb{O}})}} \Pr[\mathbb{A} \mid \mathbb{O} : \mathsf{Pref}(\eta,i-1)].\epsilon_\phi(o) \\
\leq &\ \epsilon_\phi(o)
\end{aligned}$$

This last inequality results from the fact that the sum is taken on all possible last steps consisting in a query to o, and we take a maximum on all possible values of q, a and $m_{\mathbb{O}}$, leaving only a sum on values of $m'_{\mathbb{O}}$. Then, we use the fact that equivalence class form a partition to conclude:

$$\begin{aligned}
\Pr[\mathbb{A} \mid \mathbb{O} : \mathsf{F}_\phi] &= \textstyle\sum_{\tau\in T} \Pr[\mathbb{A} \mid \mathbb{O} : \tau] \\
&= \textstyle\sum_{i,o} \sum_{\tau\in\mathcal{C}(i,o)} \Pr[\mathbb{A} \mid \mathbb{O} : \tau] \\
&\leq \textstyle\sum_{i,o} \epsilon(o) \\
&\leq \textstyle\sum_{o} k(o)\epsilon(o)
\end{aligned}$$

■

Example 6. We illustrate the use of rule Fail on the example based on the principle of the French lottery.

We consider an oracle system $\mathbb{L}$ with empty memories and trivial initialization and finalization oracles. It contains an oracle *Loto* taking as input five distinct integers between 1 and 52, drawing its own five distinct integers at random and answering $WIN!$ if they coincide and $Try\ Again$ otherwise. As in the lottery, we emphasize that the tuple is drawn *at every query of the oracle.*

We let Win be the step-predicate $\lambda((o,q,a),\_,\_).(o = Loto) \wedge (a = WIN!)$. The probability that Win is true is 0 when an adversary calls initialization or finalization oracle and $\alpha = \frac{1}{52.51.50.49.48}$ when it calls oracle *Loto*. Applying Fail, we get $\models \mathbb{L} :_\varepsilon \mathsf{F}_{Win}$, where $\varepsilon(k,t) = \alpha.k(Loto)$. ◊

Lemma III.3 (Union Rule). *The following rule is sound for any countable set of index I.*

$$\frac{\mathbb{O} :_{\varepsilon_i} E_i\ (i \in I) \qquad E \Rightarrow \bigvee_{i\in I} E_i}{\mathbb{O} :_{\sum_{i\in I}\varepsilon_i} E}\ UR$$

Proof. Let $\mathbb{A}$ be an (k,t)-adversary.

$$\begin{array}{rcl}\Pr[\mathbb{A} \mid \mathbb{O} : E] & \leq & \Pr[\mathbb{A} \mid \mathbb{O} : \bigvee_{i\in I} E_i] \\ & \leq & \sum_{i\in I} \Pr[\mathbb{A} \mid \mathbb{O} : E_i] \\ & \leq & \sum_{i\in I} \varepsilon_i(k,t)\end{array}$$

■

The following rule allows to import in the proof system the preservation of invariants. We stress that proving the preservation is not the point of our logic. It can be for example obtained as a result of a Hoare logic specifically designed for a language in which the oracle system is written.

Lemma III.4 (Post). *We let E be an event holding for every execution of $\mathbb{A} \mid \mathbb{O}$, for all adversaries $\mathbb{A}$, which we denote by $\mathbb{O}\{E\}$. The following rule is sound.*

$$\frac{\mathbb{O}\{E\}}{\mathbb{O} :_0 \neg E}\ \mathit{Post}$$

We now introduce the first rule capturing a reduction-based argument. We consider an extended event E and a step-predicate ϕ. The purpose of the rule is to obtain tight reductions. It is based on the idea that as soon as we know an adversary able to achieve $E \,\mathsf{U}\, \phi$, we should be able to build one capable to achieve $E;\phi$ with the same probability. Intuitively, our adversary should terminate its run as soon as ϕ becomes true. Nevertheless, cutting short an execution only yields a partial execution. Consequently, we have to modify the state space by adding a sink state in which A gets stuck and make $\mathsf{A}_\downarrow$ transition into it as soon as ϕ gets true. According to our definition, this behavior results in a proper execution, which yields a trace. All this reasoning relies on the capacity for the adversary to test the truth value of ϕ at each step. As step-predicates take oracle memories as arguments, this testability cannot be taken for granted. It can be formalized as follows.

Definition (Testability). Let ϕ be a step-predicate. An effective function $\mathsf{Tester}_\phi : \mathsf{Xch}^* \times \mathsf{Que} \to \mathsf{Bool}$ is called a *ϕ-tester*, if for every trace τ of the form $m_0 \xrightarrow{x_1} \dots \xrightarrow{x_{k-1}} m_{k-1} \xrightarrow{(o_k,q_k,a_k)} m_k$, we have $\phi((o_k,q_k,a_k),m_{k-1},m_k) = \mathsf{Tester}_\phi([x_1,\dots,x_{k-1}],(o_k,q_k))$.

A step-predicate ϕ is called *testable*, if a *ϕ-tester* exists. □

We now state the rule capturing the reduction argument we have just developed.

Lemma III.5 (Cut). *Let E be an extended event and ϕ be a step-predicate. The following rule is sound.*

$$\frac{\mathbb{O} :_{\epsilon(k,t)} E;\phi \qquad \phi \text{ admits a } \phi\text{-tester } \mathsf{Tester}_\phi}{\mathbb{O} :_{\epsilon(f(\phi,k,t))} E \,\mathsf{U}\, \phi}\ \mathit{Cut}$$

where $T(\phi)$ is a bound on the time necessary for an evaluation of Tester_ϕ and $f(\phi,k,t) = (k, t + T(\phi).\sum_{o\in\mathsf{N}_\mathbb{O}} k(o))$.

Proof. Given an $\mathbb{O}$-adversary $\mathbb{A}$ and the ϕ-tester Tester_ϕ, we define an adversary $\mathbb{A}^{\mathsf{Tester}_\phi}$ that stops its execution as soon as ϕ occurs.

— The set of memories of $\mathbb{A}^{\mathsf{Tester}_\phi}$ is $\mathsf{M}'_\mathbb{A} = (\mathsf{M}_\mathbb{A} \times \mathsf{Xch}^*) + \{\bot_a\}$.

— The initial memory of $\mathbb{A}^{\mathsf{Tester}_\phi}$ is $(\bar{m}_\mathbb{A}, [\,])$, where $\bar{m}_\mathbb{A}$ is the initial memory of $\mathbb{A}$.

— $\mathsf{A}^{\mathsf{Tester}_\phi}$: $\mathsf{A}^{\mathsf{Tester}_\phi}(\bot_a)$ is undefined and
$\mathsf{A}^{\mathsf{Tester}_\phi}(m, list) =$
  $\mathsf{let}\ ((o,q),m') \leftarrow A(m)\ \mathsf{in}$
  $\mathsf{if}\ \mathsf{Tester}_\phi(list,(o,q)) = \mathsf{false}\ \mathsf{then}\ \mathsf{return}\ ((o,q),(m',list))$
  $\mathsf{else}\ \mathsf{return}\ ((o,q),\bot_a)$

— $\mathsf{A}_\downarrow^{\mathsf{Tester}_\phi}$: $\mathsf{A}_\downarrow^{\mathsf{Tester}_\phi}(\bot_a,(o,q,a)) = \delta_{\bot_a}$ and
$\mathsf{A}_\downarrow^{\mathsf{Tester}_\phi}((m,list),(o,q,a)) =$
  $\mathsf{let}\ m' \leftarrow \mathsf{A}_\downarrow(m,(o,q))\ \mathsf{in}$
  $\mathsf{return}\ (m', list\cdot(o,q,a))$

For every partial execution $\eta \in \mathsf{PExec}(\mathbb{A} \mid \mathbb{O})$ consisting of steps whose projection satisfying $\neg\phi$, by induction on the length of η, we have $\Pr[\mathbb{A} \mid \mathbb{O} : \eta] = \Pr[\mathbb{A}^{\mathsf{Tester}_\phi} \mid \mathbb{O} : \eta]$. Then, if we denote (m, m_a) the last state of η, $\Pr[\mathbb{A} \mid \mathbb{O} : \eta \xrightarrow{(o,q,a)} (m'_a, m') \wedge \phi((o,q,a),m,m')] = \Pr[\mathbb{A}^{\mathsf{Tester}_\phi} \mid \mathbb{O} : \eta \xrightarrow{(o,q,a)} (\bot_a, m')]$.

It follows that $\Pr[\mathbb{A} \mid \mathbb{O} : E\,\mathsf{U}\,\phi] = \Pr[\mathbb{A}^{\mathsf{Tester}_\phi} \mid \mathbb{O} : E;\phi]$.

Finally, if $\mathbb{A}$ is a (k,t)-bounded adversary then $\mathbb{A}^{\mathsf{Tester}_\phi}$ is bounded by $f(\varphi,k,t) = (k, t + T(\phi).\sum_{o\in\mathsf{N}_\mathbb{O}} k(o))$. The conclusion follows.

■

III.3 Contexts

In this section, we present the notion of context, which captures a lot of classical reduction proofs of security using the embedding of one adversary into another. Contexts act like an interface between adversaries and oracles: they can be seen as a top layer of an oracle system through which an adversary has to ask its queries, or they can be composed with adversaries themselves, which they run as subroutines. Thus, a context $\mathbb{C}$ can yield an oracle system $\mathbb{C}[\mathbb{O}]$ when composed with system $\mathbb{O}$, and an $\mathbb{O}$-adversary $\mathbb{A} \parallel \mathbb{C}$ when composed with a $\mathbb{C}[\mathbb{O}]$-adversary $\mathbb{A}$. It yields two transition systems, corresponding to $(\mathbb{A} \parallel \mathbb{C}) \mid \mathbb{O}$ and $\mathbb{A} \mid \mathbb{C}[\mathbb{O}]$. We argue that the intuition according to which these systems are two sides of a same coin is mathematically grounded. In conclusion of this section, we provide two new rules named I-Sub and B-Sub, allowing the embedding of a system in a context for statements based on both types of judgments.

III.3.1 — Definition and Composition with an Oracle System

In a lot of aspects, the definition of contexts resembles that of oracle systems. However, a context c alone does *not* define a system. It contains procedures, each of which is defined by two functions: $\mathsf{C}_c^\rightarrow$ to transform queries to c into queries to an oracle of $\mathbb{O}$, and $\mathsf{C}_c^\leftarrow$ to transform the result of this call back into an output of procedure c. These transformations are probabilistic, which is modeled by functions $\mathsf{C}_c^\rightarrow$ and $\mathsf{C}_c^\leftarrow$ outputting distributions. Finally, we also want to capture the possibility that no query to $\mathbb{O}$ is needed to compute the answer to a procedure query. In such a case, $\mathsf{C}_c^\rightarrow$ outputs $\bot$.

Definition (Context). An $\mathbb{O}$-*context* $\mathbb{C}$ is given by:

— finite sets $\mathsf{M}_\mathbb{C}$ of context memories and an initial memory $\bar{m}_\mathbb{C}$,

— a finite set $\mathsf{N}_{\mathbb{C}}$ of procedure names,

— for every $c \in \mathsf{N}_{\mathbb{C}}$, a query domain $\mathsf{In}(c)$, an answer domain $\mathsf{Out}(c)$, and two functions:

$$\mathsf{C}_c^{\rightarrow} : \mathsf{In}(c) \times \mathsf{M}_{\mathbb{C}} \to \mathcal{D}((\mathsf{Que} \cup \{\bot\}) \times \mathsf{M}_{\mathbb{C}})$$
$$\mathsf{C}_c^{\leftarrow} : \mathsf{In}(c) \times \mathsf{Xch} \times \mathsf{M}_{\mathbb{C}} \to \mathcal{D}(\mathsf{Out}(c) \times \mathsf{M}_{\mathbb{C}})$$

— distinguished initial and finalization procedures c_{I} and c_{F} such that $\mathsf{In}(c_{\mathrm{I}}) = \mathsf{Out}(c_{\mathrm{F}}) = \mathbf{1}$, and for any input, $\mathsf{C}_{c_{\mathrm{I}}}^{\rightarrow}$ (resp. $\mathsf{C}_{c_{\mathrm{F}}}^{\leftarrow}$) outputs a query to o_{I} (resp. o_{F}).

□

As was done in the case of oracle systems, we introduce the following notations: $\mathsf{Que}_{\mathbb{C}}$ is the set of context queries, $\mathsf{Ans}_{\mathbb{C}}$ the set of context answers, and $\mathsf{Xch}_{\mathbb{C}}$ the set of context exchanges.

Let us explain the intuition behind the composition of a context $\mathbb{C}$ and an oracle system $\mathbb{O}$, denoted $\mathbb{C}[\mathbb{O}]$. Composition yields a new oracle system, in which each procedure of $\mathbb{C}$ defines an oracle, including initialization and finalization oracles. However, no oracle of $\mathbb{O}$ is "directly" accessible to an adversary of the composed system. We now describe informally one step of the interaction of system $\mathbb{C}[\mathbb{O}]$ with a $\mathbb{C}[\mathbb{O}]$-adversary $\mathbb{A}$. It is represented in figure III.1. First, the adversary computes a query $(c, q_{\mathbb{C}})$, which it sends to system $\mathbb{C}[\mathbb{O}]$. Then, oracle c generates a query to an oracle of $\mathbb{O}$ using $\mathsf{C}_c^{\rightarrow}$. The query is forwarded to $\mathbb{O}$ and answered, then the resulting exchange is used as an input for $\mathsf{C}_c^{\leftarrow}$ to compute the answer $a_{\mathbb{C}}$ to query $(c, q_{\mathbb{C}})$.

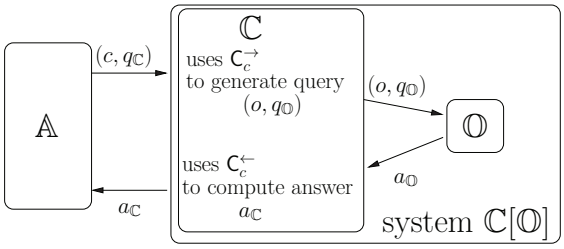


Figure III.1 – One Step of Interaction Between $\mathbb{C}[\mathbb{O}]$ and an Adversary $\mathbb{A}$

In our definition, we require that one query $(c, q_{\mathbb{C}})$ to a procedure in the context $\mathbb{C}$ yields one query to an oracle in $\mathbb{O}$. It is sometimes the case that no query is needed to compute the answer to $(c, q_{\mathbb{C}})$. To deal with this particular case, we add a dummy oracle, named $\bot$, to the oracle system $\mathbb{O}$. Its implementation is trivial: it returns the value it receives and does not change the state. This augmented system $\mathbb{O}$ is still denoted $\mathbb{O}$. Then, $\mathsf{C}_c^{\rightarrow}(q_{\mathbb{C}})$ can output $(\bot, \mathbf{1})$.

Definition (Composition of Context with an Oracle System). The composition of an $\mathbb{O}$-context $\mathbb{C}$ with $\mathbb{O}$ defines an oracle system $\mathbb{C}[\mathbb{O}]$ such that:

— the set of memories is $\mathsf{M}_{\mathbb{C}} \times \mathsf{M}_{\mathbb{O}}$, and the initial memory is $(\bar{m}_{\mathbb{C}}, \bar{m}_{\mathbb{O}})$,

— the oracles are the procedures of $\mathbb{C}$, and their query and answer domains are given by $\mathbb{C}$. The initialization and finalization oracles are the initialization and finalization procedures of $\mathbb{C}$,

— the implementation $\mathsf{Imp}(c)$ of an oracle c is the function:

$$\begin{array}{l}\lambda(q_{\mathbb{C}},(m_{\mathbb{C}},m_{\mathbb{O}})).\\ \quad \mathsf{let}\ ((o,q_{\mathbb{O}}),m'_{\mathbb{C}}) \leftarrow \mathsf{C}_c^{\rightarrow}(q_{\mathbb{C}},m_{\mathbb{C}})\ \mathsf{in}\\ \quad \mathsf{let}\ (a_{\mathbb{O}},m'_{\mathbb{O}}) \leftarrow \mathsf{Imp}(o)(q_{\mathbb{O}},m_{\mathbb{O}})\ \mathsf{in}\\ \quad \mathsf{let}\ (a_{\mathbb{C}},m''_{\mathbb{C}}) \leftarrow \mathsf{C}_c^{\leftarrow}(q_{\mathbb{C}},(o,q_{\mathbb{O}},a_{\mathbb{O}}),m'_{\mathbb{C}})\ \mathsf{in}\\ \quad \mathsf{return}\ (a_{\mathbb{C}},(m''_{\mathbb{C}},m'_{\mathbb{O}}))\end{array}$$

□

The important thing to notice about this definition is that a context can only obtain outputs of oracles of the inner system. Namely, *no way of reading or writing in memories of the inner oracle system is provided to a context.* This limits the expressivity of the concept but plays a key-role in the composability of adversaries and contexts: it guarantees black-box access to oracles of the inner system for a composed adversary. Details about that follow shortly (see III.3.2).

Example 7. We recall briefly the encryption algorithm of ElGamal in a group $\mathbb{G}$ of prime order q, with a generator g. Key generation computes a pair of matching public and secret keys (pk, sk) as follows:

$$x \leftarrow \mathcal{U}(\mathbb{Z}_q); pk := g^x; sk := x.$$

To encrypt a message $g^m \in \mathbb{G}$, the following steps are performed:

$$y \leftarrow \mathcal{U}(\mathbb{Z}_q); a := g^y; b := pk^y; c := (a, b.g^m).$$

The ciphertext resulting is the pair c. We want to build an oracle system $\mathbb{E}l\mathbb{G}amal$ which we can use later to express semantic security of this scheme. It consists in three oracles: one for key generation, one for the computation of a challenge, and a finalization oracle. We consider the oracle system $\mathbb{R}eal\mathbb{D}\mathbb{H} = (o_{\mathrm{I}}, o_{\mathrm{F}})$ defined in III.1.4, and provide a context $\mathbb{C}$ such that its composition with $\mathbb{R}eal\mathbb{D}\mathbb{H}$ yields the desired system $\mathbb{E}l\mathbb{G}amal$. Context $\mathbb{C}$ is given by:

— memories containing a variable for pk, a and b.

— Intuitively, we want c_{I} to perform key generation. As a result, it must call o_{I} to get a value for variable pk. This satisfies the requirement that c_{I} calls o_{I} (expressed in the last point of the definition of context). On result (v_0, v_1, v_2) from o_{I}, we want c_{I} to store these values in pk, a and b and to output pk. This is done by means of function $\mathsf{C}_{c_{\mathrm{I}}}^{\leftarrow}$.

— Oracle $\mathcal{E}$ is a one-time cipher. On its first query $q = g^\mu$, it outputs the pair $(a, b.g^\mu)$. We set it to answer λ if queried more than once.

— The finalization oracle of the context forwards the answer it gets to o_{F}.

The system $\mathbb{E}l\mathbb{G}amal = \mathbb{C}[\mathbb{R}eal\mathbb{D}\mathbb{H}]$ models the oracles to which an adversary against semantic security has access.

An interesting thing to notice about this example is that, would we want to write the decryption oracle of the ElGamal scheme as a context of $\mathbb{R}eal\mathbb{D}\mathbb{H}$, we could not do it without computing the logarithm of the public key to obtain the secret key (namely, α given $a = g^\alpha$). The reason is that, as underlined previously, states of $\mathbb{C}$ and $\mathbb{R}eal\mathbb{D}\mathbb{H}$ are separated, and the value stored in α is not output when querying oracles of $\mathbb{R}eal\mathbb{D}\mathbb{H}$. ◊

III.3.2 — Composition of a Context and an Adversary

The intuition behind the composition of context $\mathbb{C}$ and a $\mathbb{C}[\mathbb{O}]$-adversary $\mathbb{A}$ is pretty similar to that of the composition with an oracle system. This time, the context is considered as a part of an adversary playing with system $\mathbb{O}$. One step of interaction between $\mathbb{A}$ composed with $\mathbb{C}$ on the one hand, and $\mathbb{O}$ on the other hand is illustrated in figure III.2. The idea is that the composed adversary first runs $\mathbb{A}$, which outputs a query $(c, q_{\mathbb{C}})$ to an oracle c of $\mathbb{C}[\mathbb{O}]$; then, $\mathsf{C}_c^{\rightarrow}$ is applied to generate a query to an oracle of system $\mathbb{O}$. This query is then submitted to $\mathbb{O}$, which answers it. Finally, to be usable by the update part of $\mathbb{A}$, this answer is put through $\mathsf{C}_c^{\leftarrow}$. We can see here that this composition really corresponds to the usual embedding of an adversary into another one which runs it as a subroutine and simulates the oracles the subroutine needs.

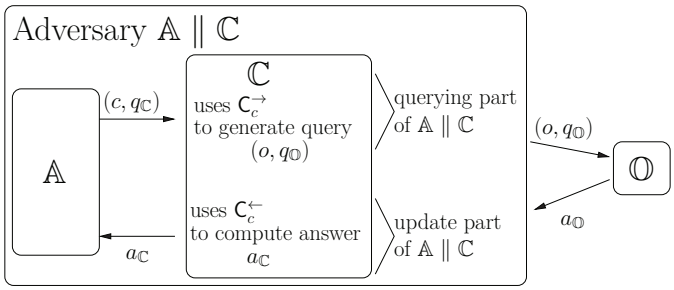


Figure III.2 – One Step of Interaction Between Composed Adversary $\mathbb{A} \parallel \mathbb{C}$ and $\mathbb{O}$

As for the composition with an oracle system, we are now left with adding states where they are needed. We notice that the new adversary needs to store the current query in its state because the procedure name is required to know which function $\mathsf{C}_c^{\leftarrow}$ to apply, and value $q_{\mathbb{C}}$ is needed as an input of this function.

Definition (Composition of Context and Adversary). The composition of an $\mathbb{O}$-context $\mathbb{C}$ to a $\mathbb{C}[\mathbb{O}]$-adversary $\mathbb{A}$ defines an $\mathbb{O}$-adversary $\mathbb{A} \parallel \mathbb{C}$ such that:

— the set of memories is $\mathsf{M}_{\mathbb{C}} \times \mathsf{M}_{\mathbb{A}} \times \mathsf{Que}_{\mathbb{C}}$, and the initial memory is $(\bar{m}_{\mathbb{C}}, \bar{m}_{\mathbb{A}}, \_)$,

— the querying function is:

$$
\begin{array}{l}
\lambda(m_{\mathbb{C}}, m_{\mathbb{A}}, \_). \\
\quad \mathsf{let}\ ((c, q_{\mathbb{C}}), m'_{\mathbb{A}}) \leftarrow \mathsf{A}(m_{\mathbb{A}})\ \mathsf{in} \\
\quad \mathsf{let}\ ((o, q_{\mathbb{O}}), m'_{\mathbb{C}}) \leftarrow \mathsf{C}_c^{\rightarrow}(q_{\mathbb{C}}, m_{\mathbb{C}})\ \mathsf{in} \\
\quad \mathsf{return}\ ((o, q_{\mathbb{O}}), (m'_{\mathbb{C}}, m'_{\mathbb{A}}, (c, q_{\mathbb{C}})))
\end{array}
$$

— the update function is:

$$
\begin{array}{l}
\lambda((o, q_{\mathbb{O}}, a_{\mathbb{O}}), (m_{\mathbb{C}}, m_{\mathbb{A}}, (c, q_{\mathbb{C}}))). \\
\quad \mathsf{let}\ (a_{\mathbb{C}}, m'_{\mathbb{C}}) \leftarrow \mathsf{C}_c^{\leftarrow}(q_{\mathbb{C}}, (o, q_{\mathbb{O}}, a_{\mathbb{O}}), m_{\mathbb{C}})\ \mathsf{in} \\
\quad \mathsf{return}\ (m'_{\mathbb{C}}, \mathsf{A}_{\downarrow}((c, q_{\mathbb{C}}, a_{\mathbb{C}}), m_{\mathbb{A}}), \_)
\end{array}
$$

□

Now that we have given a semantics to the composition of contexts and adversaries, we can see why it is essential that states of $\mathbb{C}$ and $\mathbb{O}$ are disjoint. If it was not the case, we could not define $\mathbb{A} \parallel \mathbb{C}$ as an adversary. Indeed, this latter could have information or side-effect

on oracle memories via other means than querying oracles, and this is not captured by our definition of adversary.

As the judgments in our logic express concrete security statements, we need to relate resource consumption of the composed adversary to that of the inner adversary and time consumption of the context computations. To do so, we define two auxiliary functions for contexts. On the one hand, we assume that there exists an upper bound $T(c)$ on the time needed to compute both $\mathsf{C}_c^{\rightarrow}$ and $\mathsf{C}_c^{\leftarrow}$, for any of their arguments. On the other hand, we let $\mathfrak{Calls} : \mathsf{N}_{\mathbb{C}} \times \mathsf{N}_{\mathbb{O}} \to \{0,1\}$ be a function such that $\mathfrak{Calls}(c,o) = 1$ iff c may call o, i.e., there are $m_c \in \mathsf{M}_{\mathbb{C}}$ and $q_c \in \mathsf{In}(c)$ such that $Pr[\mathsf{C}_c^{\rightarrow}(m_c, q_c) = ((o,q), m'_c)] > 0$ for some $q \in \mathsf{In}(o)$ and $m'_c \in \mathsf{M}_{\mathbb{C}}$.

Lemma III.6. *Let $\mathbb{C}$ be an $\mathbb{O}$-context and $\mathbb{A}$ be a $\mathbb{C}[\mathbb{O}]$-adversary. If $\mathbb{A}$ has resources bounded by (k,t), then $\mathbb{A} \parallel \mathbb{C}$ has resources bounded by*

$$f(\mathbb{C},k,t) = (\lambda o.\ \sum_{c \in \mathsf{N}_{\mathbb{C}}} \mathfrak{Calls}(c,o)k(c),\ t + \sum_{c \in \mathsf{N}_{\mathbb{C}}} k(c)T(c))$$

Proof. Let us consider one step of $(\mathbb{A} \parallel \mathbb{C}) \mid \mathbb{O}$ involving procedure c. The time of execution of this step corresponds to the time used up by $\mathbb{A}$ plus the time taken by the computations of $\mathsf{C}_c^{\rightarrow}$ and $\mathsf{C}_c^{\leftarrow}$, bounded by $T(c)$. Concerning queries, we notice that at most one call is placed to an oracle in $\mathbb{O}$, but we have no means of anticipating which one (we recall that this choice is probabilistic). To be safe, we increase by one the number of calls of *every oracle of* $\mathbb{O}$ which the computation of $\mathsf{C}_c^{\rightarrow}$ *can result in calling*. Function $\lambda o.\mathfrak{Calls}(c,o)$, defined to be worth 1 if c may call o, bounds the number of calls to o performed during the step.

This reasoning has been performed for one step involving a particular c. The conclusion follows by summing on all procedures. ■

III.3.3 — Links Between $\mathbb{A} \mid \mathbb{C}[\mathbb{O}]$ and $(\mathbb{A} \parallel \mathbb{C}) \mid \mathbb{O}$

The two ways of composing an $\mathbb{O}$-context bring forth two transition systems, namely $\mathbb{A} \mid \mathbb{C}[\mathbb{O}]$ and $(\mathbb{A} \parallel \mathbb{C}) \mid \mathbb{O}$. They informally seem to bear quite a lot of similarities. In fact, we can see the same computations take place in both systems, and only the fact that we decompose transition systems into adversary and oracle system introduces a difference between them. In this subsection, we formalize this intuition by showing that $\mathbb{A} \mid \mathbb{C}[\mathbb{O}]$ and $(\mathbb{A} \parallel \mathbb{C}) \mid \mathbb{O}$ are two projections of a same underlying transition system which we call $\mathbb{A} \times \mathbb{C} \times \mathbb{O}$. This latter merely corresponds to the description of computations carried out without imposing the split between an adversarial part and an oracle part. Let us start with the definition of $\mathbb{A} \times \mathbb{C} \times \mathbb{O}$.

Definition (Product System $\mathbb{A} \times \mathbb{C} \times \mathbb{O}$). For an oracle system $\mathbb{O}$, an $\mathbb{O}$-context $\mathbb{C}$ and a $\mathbb{C}[\mathbb{O}]$ adversary $\mathbb{A}$, we define their product probabilistic transition system $\mathbb{A} \times \mathbb{C} \times \mathbb{O}$ as follows:

— the set of memories of $\mathbb{A} \times \mathbb{C} \times \mathbb{O}$ is $\mathsf{M}_{\mathbb{A}} \times \mathsf{M}_{\mathbb{C}} \times \mathsf{M}_{\mathbb{O}}$ and the initial memory is $(\bar{m}_{\mathbb{A}}, \bar{m}_{\mathbb{C}}, \bar{m}_{\mathbb{O}})$;

— actions are pairs of exchanges in $\mathsf{Xch}_{\mathbb{C}} \times \mathsf{Xch}_{\mathbb{O}}$, with as initialization actions those of the form $((c_{\mathrm{I}}, -, -), (o_{\mathrm{I}}, -, -))$ and as finalization actions $((c_{\mathrm{F}}, -, -), (o_{\mathrm{F}}, -, -))$;

— the probabilistic transition function $\mathsf{st}_{\mathbb{A}\times\mathbb{C}\times\mathbb{O}}$ is:

$$
\begin{array}{l}
\lambda(m_{\mathbb{A}}, m_{\mathbb{C}}, m_{\mathbb{O}}).\\
\quad \mathsf{let}\ ((c, q_{\mathbb{C}}), m'_{\mathbb{A}}) \leftarrow \mathsf{A}(m_{\mathbb{A}})\ \mathsf{in}\\
\qquad \mathsf{let}\ ((o, q_{\mathbb{O}}), m'_{\mathbb{C}}) \leftarrow \mathsf{C}^{\rightarrow}_{c}(q_{\mathbb{C}}, m_{\mathbb{C}})\ \mathsf{in}\\
\qquad \mathsf{let}\ (a_{\mathbb{O}}, m'_{\mathbb{O}}) \leftarrow \mathsf{Imp}(o)(q_{\mathbb{O}}, m_{\mathbb{O}})\ \mathsf{in}\\
\qquad \mathsf{let}\ (a_{\mathbb{C}}, m''_{\mathbb{C}}) \leftarrow \mathsf{C}^{\leftarrow}_{c}(q_{\mathbb{C}}, (o, q_{\mathbb{O}}, a_{\mathbb{O}}), m'_{\mathbb{C}})\ \mathsf{in}\\
\qquad \mathsf{let}\ m''_{\mathbb{A}} \leftarrow \mathsf{A}_{\downarrow}((c, q_{\mathbb{C}}, a_{\mathbb{C}}), m'_{\mathbb{A}})\ \mathsf{in}\\
\qquad \mathsf{return}\ (((c, q_{\mathbb{C}}, a_{\mathbb{C}}), (o, q_{\mathbb{O}}, a_{\mathbb{O}})), (m''_{\mathbb{A}}, m''_{\mathbb{C}}, m'_{\mathbb{O}}))
\end{array}
$$

□

Similarly to what we did for oracle systems, we can define product traces as executions of the product system where adversary memories have been erased. Formally, it yields the following definition.

Definition (Product Execution, Product Trace). A *partial product execution* is a sequence η^{prod}

$$(m^0_{\mathbb{C}}, m^0_{\mathbb{O}}, m^0_{\mathbb{A}}) \xrightarrow{(x_1,y_1)} (m^1_{\mathbb{C}}, m^1_{\mathbb{O}}, m^1_{\mathbb{A}}) \xrightarrow{(x_2,y_2)} \dots \xrightarrow{(x_k,y_k)} (m^k_{\mathbb{C}}, m^k_{\mathbb{O}}, m^k_{\mathbb{A}})$$

where $(m^i_{\mathbb{C}}, m^i_{\mathbb{O}}, m^i_{\mathbb{A}}) \in \mathsf{M}_{\mathbb{C}} \times \mathsf{M}_{\mathbb{O}} \times \mathsf{M}_{\mathbb{A}}$ for all i in $[0, k]$ and $(m^0_{\mathbb{C}}, m^0_{\mathbb{O}}, m^0_{\mathbb{A}}) = (\bar{m}_{\mathbb{C}}, \bar{m}_{\mathbb{O}}, \bar{m}_{\mathbb{A}})$, and for i in $[1, k]$, $x_i = (o_i, q^i_{\mathbb{O}}, a^i_{\mathbb{O}}) \in \mathsf{Xch}_{\mathbb{O}}$ and $y_i = (c_i, q^i_{\mathbb{C}}, a^i_{\mathbb{C}}) \in \mathsf{Xch}_{\mathbb{C}}$ are such that there is a positive probability that the step occurs, i.e. there exists $m'_{\mathbb{C}}, m'_{\mathbb{A}}$ s.t.

$$
\begin{cases}
\Pr[\mathsf{A}(m^{i-1}_{\mathbb{A}}) = ((c_i, q^i_{\mathbb{C}}), m'_{\mathbb{A}})] > 0\\
\Pr[\mathsf{C}^{\rightarrow}_{c_i}(q^i_{\mathbb{C}}, m^{i-1}_{\mathbb{C}}) = ((o_i, q^i_{\mathbb{O}}), m'_{\mathbb{C}})] > 0\\
\Pr[\mathsf{Imp}(o_i)(q^i_{\mathbb{O}}, m^{i-1}_{\mathbb{O}}) = (a^i_{\mathbb{O}}, m^i_{\mathbb{O}})] > 0\\
\Pr[\mathsf{C}^{\leftarrow}_{c_i}(q^i_{\mathbb{C}}, (o_i, q^i_{\mathbb{O}}, a^i_{\mathbb{O}}), m'_{\mathbb{C}}) = (a^i_{\mathbb{C}}, m^i_{\mathbb{C}})] > 0\\
\Pr[\mathsf{A}_{\downarrow}((c_i, q^i_{\mathbb{C}}, a^i_{\mathbb{C}}), m'_{\mathbb{A}}) = m^i_{\mathbb{A}}] > 0
\end{cases}
$$

A *product execution* is a partial product execution ending with an exchange with c_{F} and o_{F} or in a sink state.

A *partial product trace* τ^{prod} is a the projection on $\mathsf{M}_{\mathbb{C}} \times \mathsf{M}_{\mathbb{O}}$ of a subsequence of a partial execution (i.e. it does not necessarily start with the initial states).

We call *product step* a partial product trace of length 1. Finally, a *product trace* is the projection of a product execution. □

We underline that if we project a product trace to keep only $\mathbb{O}$-elements of the traces, we obtain a trace of interaction of $\mathbb{A} \parallel \mathbb{C}$ with $\mathbb{O}$, while if we keep only $\mathbb{C}$-elements, we obtain the result of an interaction between $\mathbb{A}$ and $\mathbb{C}[\mathbb{O}]$. The only element we miss is the correspondence between probabilities granted to these traces. If we consider a product step and want to associate a probability to its projection as an $\mathbb{O}$-step, we naturally think of summing all probabilities of $\mathbb{C}$-elements projecting to this particular $\mathbb{O}$-step. This is indeed the way we choose to deal with probabilities in projected transition systems.

Definition (Projected Transition System). Consider a probabilistic transition system $\mathcal{S}$ with a set of actions $\Sigma = (a : \Sigma_a) \times (b : \Sigma_b)$, that is, Σ is the cartesian product of two set of actions Σ_a and Σ_b.

The a-projection of $\mathcal{S}$, denoted by $\pi_a(\mathcal{S})$, is a probabilistic transition system that has the same set of memories as $\mathcal{S}$, the same initial memory, Σ_a as alphabet and the transition

probability function $\mathsf{st}_a : \mathsf{M} \to \mathcal{D}(\Sigma_a \times \mathsf{M})$ defined as follows:

$$\lambda m.\ \mathsf{let}\ ((a,b),m') \leftarrow \mathsf{st}(m)\ \mathsf{in}\ \ \mathsf{return}\ (a,m')$$

Similarly, we can define the b-projection $\pi_b(\mathcal{S})$ of $\mathcal{S}$. □

This definition captures exactly the relation existing between $\mathbb{A} \times \mathbb{C} \times \mathbb{O}$ and $\mathbb{A} \mid \mathbb{C}[\mathbb{O}]$ on the one hand, and $\mathbb{A} \times \mathbb{C} \times \mathbb{O}$ and $(\mathbb{A} \parallel \mathbb{C}) \mid \mathbb{O}$ on the other hand. We write $\pi_{\mathbb{C}[\mathbb{O}]}(\mathbb{A} \times \mathbb{C} \times \mathbb{O})$ for $\pi_{\mathsf{Xch}_{\mathbb{C}}}(\mathbb{A} \times \mathbb{C} \times \mathbb{O})$ and $\pi_{\mathbb{O}}(\mathbb{A} \times \mathbb{C} \times \mathbb{O})$ for $\pi_{\mathsf{Xch}_{\mathbb{O}}}(\mathbb{A} \times \mathbb{C} \times \mathbb{O})$.

Lemma III.7. *For every $\mathbb{C}[\mathbb{O}]$-adversary $\mathbb{A}$, we have:*

1. $\mathbb{A} \mid \mathbb{C}[\mathbb{O}] = \pi_{\mathbb{C}[\mathbb{O}]}(\mathbb{A} \times \mathbb{C} \times \mathbb{O})$,
2. $(\mathbb{A} \parallel \mathbb{C}) \mid \mathbb{O} = \pi_{\mathbb{O}}(\mathbb{A} \times \mathbb{C} \times \mathbb{O})$.

Proof. Firstly, the initial memory of $\mathbb{A} \mid \mathbb{C}[\mathbb{O}]$, $\mathbb{A} \times \mathbb{C} \times \mathbb{O}$ and its projections and $(\mathbb{A} \parallel \mathbb{C}) \mid \mathbb{O}$ are identical modulo tuple isomorphism. Secondly, the following equalities obviously follow from code rewriting:

1. $\mathsf{st}_{\mathbb{A}|\mathbb{C}[\mathbb{O}]} = \mathsf{st}_{\pi_{\mathbb{C}[\mathbb{O}]}}$;
2. $\mathsf{st}_{(\mathbb{A}\|\mathbb{C})|\mathbb{O}} = \mathsf{st}_{\pi_{\mathbb{O}}}$.

We provide a few details for the first item, the second one being completely similar. The definition of the transition function of $\mathbb{A} \mid \mathbb{C}[\mathbb{O}]$ states:

$$\begin{array}{lll} \mathsf{st}_{\mathbb{A}|\mathbb{C}[\mathbb{O}]}(m_{\mathbb{A}},(m_{\mathbb{C}},m_{\mathbb{O}})) \stackrel{def}{=} & & \mathsf{let}\ ((c,q_{\mathbb{C}}),m'_{\mathbb{A}}) \leftarrow \mathsf{A}(m_{\mathbb{A}})\ \mathsf{in} \\ & (*) & \mathsf{let}\ (a_{\mathbb{C}},(m'_{\mathbb{C}},m'_{\mathbb{O}})) \leftarrow \mathsf{Imp}(c)(q_{\mathbb{C}},(m_{\mathbb{C}},m_{\mathbb{O}})\ \mathsf{in} \\ & & \mathsf{let}\ m''_{\mathbb{A}} \leftarrow \mathsf{A}_{\downarrow}((c,q_{\mathbb{C}},a_{\mathbb{C}}),m'_{\mathbb{A}})\ \mathsf{in} \\ & & \mathsf{return}\ ((c,q_{\mathbb{C}},a_{\mathbb{C}}),(m''_{\mathbb{A}},m'_{\mathbb{O}})) \end{array}$$

Besides, the definition of $\mathsf{st}_{\pi_{\mathbb{C}[\mathbb{O}]}}$ provides by developing $\mathsf{st}_{\mathbb{A}\times\mathbb{C}\times\mathbb{O}}$:

$$\begin{array}{l} \lambda(m_{\mathbb{A}},(m_{\mathbb{C}},m_{\mathbb{O}})). \\ \quad \mathsf{let}\ ((c,q_{\mathbb{C}}),m'_{\mathbb{A}}) \leftarrow \mathsf{A}(m_{\mathbb{A}})\ \mathsf{in} \\ \qquad \mathsf{let}\ ((o,q_{\mathbb{O}}),m'_{\mathbb{C}}) \leftarrow \mathsf{C}_c^{\rightarrow}(q_{\mathbb{C}},m_{\mathbb{C}})\ \mathsf{in} \\ \qquad \mathsf{let}\ (a_{\mathbb{O}},m'_{\mathbb{O}}) \leftarrow \mathsf{Imp}(o)(q_{\mathbb{O}},m_{\mathbb{O}})\ \mathsf{in} \\ \qquad \mathsf{let}\ (a_{\mathbb{C}},m''_{\mathbb{C}}) \leftarrow \mathsf{C}_c^{\leftarrow}(q_{\mathbb{C}},(o,q_{\mathbb{O}},a_{\mathbb{O}}),m'_{\mathbb{C}})\ \mathsf{in} \\ \qquad \mathsf{let}\ m''_{\mathbb{A}} \leftarrow \mathsf{A}_{\downarrow}((c,q_{\mathbb{C}},a_{\mathbb{C}}),m'_{\mathbb{A}})\ \mathsf{in} \\ \qquad \text{– here } \mathsf{st}_{\mathbb{A}\times\mathbb{C}\times\mathbb{O}} \text{ would return } (((c,q_{\mathbb{C}},a_{\mathbb{C}}),(o,q_{\mathbb{O}},a_{\mathbb{O}})),(m''_{\mathbb{A}},m''_{\mathbb{C}},m'_{\mathbb{O}})) \\ \qquad \mathsf{return}\ ((c,q_{\mathbb{C}},a_{\mathbb{C}}),(m''_{\mathbb{A}},m'_{\mathbb{O}})) \end{array}$$

To see the equality between both functions, one just has to detail computations performed at line $(*)$ in the first one. ■

III.3.4 — Rules Involving Contexts

In this subsection, we provide new rules allowing the use of contexts in the proof system. Two instances of rules can be stated, one based on the preservation of each judgment of the logic. We first present the rules and then proceed to their proofs. These latter are performed using the link between $(\mathbb{A} \parallel \mathbb{C}) \mid \mathbb{O}$ and $\mathbb{A} \mid \mathbb{C}[\mathbb{O}]$.

Let us start by looking at what happens for indistinguishability statements. We assume that we have two compatible systems $\mathbb{O}$ and $\mathbb{O}'$ and a function ε such that $\models \mathbb{O} \sim_{\varepsilon} \mathbb{O}'$. Given

some $\mathbb{O}$-context, we want to find a function ε' such that we can prove $\models \mathbb{C}[\mathbb{O}] \sim_{\varepsilon'} \mathbb{C}[\mathbb{O}']$. First, we have to put a little restriction to the kind of context $\mathbb{C}$ which can be used. This simply follows from the fact that we want to talk about indistinguishability of $\mathbb{C}[\mathbb{O}]$ and $\mathbb{C}[\mathbb{O}']$: we must impose that the finalization oracle c_{F} take a boolean as input. We also assume that it merely forwards its input to inner finalization oracles o_{F} and o'_{F}. Such contexts are called *indistinguishability contexts*. As for ε', we must take into account the change in resources needed by an $\mathbb{C}[\mathbb{O}]$-adversary $\mathbb{A}$ and an adversary $\mathbb{A} \parallel \mathbb{C}$ (see lemma III.6). Given this computation, it is actually easier to express ε as a function of ε'. We have the following rule.

Lemma III.8 (I-Sub). *If $\mathbb{C}$ is an indistinguishability $\mathbb{O}$-context, rule I-Sub defined as follows is sound, where $\varepsilon = \lambda(k,t).\varepsilon'(f(\mathbb{C},k,t))$.*

$$\frac{\mathbb{O} \sim_{\varepsilon} \mathbb{O}'}{\mathbb{C}[\mathbb{O}] \sim_{\varepsilon'} \mathbb{C}[\mathbb{O}']}\ \textit{I-Sub}$$

To present the second rule, we proceed backwards, in the sense that we suppose that we are provided with a system such that $\models \mathbb{C}[\mathbb{O}] :_{\varepsilon} E$ for an event E on $\mathbb{C}[\mathbb{O}]$-traces. We have to define the counterpart in $\mathbb{O}$ of event E, which we denote $E \circ \mathbb{C}$. Here we provide intuition about the definition on $E \circ \mathbb{C}$. The formal proof of the rule provides more details about the preservation of probabilities. We use the projections introduced in the previous subsection. The $\mathbb{C}[\mathbb{O}]$-event E induces an event E_{prod} on product traces, which holds for a product trace τ^{prod} iff $\pi_{\mathbb{C}[\mathbb{O}]}(\tau^{prod})$ verifies E. If we identify the events with the set of traces they characterize, we can reformulate the definition of E_{prod} as $E_{prod} = \pi^{-1}_{\mathbb{C}[\mathbb{O}]}(E)$. Now that we have a product event equivalent to E, we can deduce from it an $\mathbb{O}$-event which describes the same underlying set of product traces, namely $\pi_{\mathbb{O}}(E_{prod})$. We thus let $E \circ \mathbb{C} = \pi_{\mathbb{O}}(\pi^{-1}_{\mathbb{C}[\mathbb{O}]}(E))$, and state the following rule.

Lemma III.9 (B-Sub). *If $\mathbb{C}$ is an $\mathbb{O}$-context and E a $\mathbb{C}[\mathbb{O}]$-event, rule B-Sub defined as follows is sound, where $\varepsilon = \lambda(k,t).\varepsilon'(f(\mathbb{C},k,t))$, and $E \circ \mathbb{C} = \pi_{\mathbb{O}}(\pi^{-1}_{\mathbb{C}[\mathbb{O}]}(E))$.*

$$\frac{\mathbb{O} :_{\varepsilon} E \circ \mathbb{C}}{\mathbb{C}[\mathbb{O}] :_{\varepsilon'} E}\ \textit{B-Sub}$$

Before actually proving the rules, we introduce a useful intermediate lemma about projected transition systems. We believe that citing them in the general setting and then applying them makes the proof easier to read.

Lemma III.10. *Let $\mathcal{S}$ be a probabilistic transition system provided with a set of actions $\Sigma = (a : \Sigma_a) \times (b : \Sigma_b)$. Then,*

- *for all executions η of $\mathcal{S}$, $Pr[\pi_a(\mathcal{S}) = \eta] = Pr[\mathcal{S} : \pi_a^{-1}(\eta)]$*
- *for any set of executions $E \subseteq \mathsf{Exec}(\mathcal{S})$ such that $\pi_a^{-1}(\pi_a(E)) = E$, $Pr[\pi_a(\mathcal{S}) : \pi_a(E)] = Pr[\mathcal{S} : E]$.*

Similar statements hold for b.

Proof. First item results directly from the more general following claim.
Claim. For any partial execution $\eta \in \mathsf{PExec}(\pi_a(\mathcal{S}))$,
$Pr[\pi_a(\mathcal{S}) = \eta] = \sum_{\eta' \in \mathsf{PExec}(\mathcal{S}),\, \eta' \in \pi_a^{-1}(\eta)} Pr[\mathcal{S} = \eta']$.

This can be proven by induction on the length of partial executions. If η is of length 1, then η is a step (m, a, m'), and $\eta' \in \mathsf{PExec}(\mathcal{S}) \cap \pi_a^{-1}(\eta)$ iff there exists $b \in \Sigma_b$ such that η' is step $(m, (\mathrm{a}, b), m')$. Then,

$$\begin{array}{rcl} Pr[\pi_a(\mathcal{S}) = \eta] & = & Pr[\mathsf{st}_a(m) = (\mathrm{a}, m')] \\ & \stackrel{def}{=} & \sum_{b \in \Sigma_b} Pr[\mathsf{st}(m) = ((\mathrm{a}, b), m')] \\ & = & \sum_{\eta' \in \mathsf{PExec}(\mathcal{S}),\, \eta' \in \pi_a^{-1}(\eta)} Pr[\mathcal{S} = \eta'] \end{array}$$

The induction step follows from decomposing the partial execution of length n into one of length $(n-1)$ for which the induction hypothesis holds and one last step for which we can do the same reasoning as we just did. We omit the details of the computation.

Second item can be justified thanks to the first one and the hypothesis on E: $Pr[\pi_a(\mathcal{S}) : \pi_a(E)] = Pr[\mathcal{S} : \pi_a^{-1}(\pi_a(E))] = Pr[\mathcal{S} : E]$. ■

We now continue with the proof of our context rules, which are quite easy to derive now that we have all the material we need.

Proof. We start with I-Sub. The rule results of the following equality:

$$Pr[\mathbb{A} \mid \mathbb{C}[\mathbb{O}] : \mathrm{R} = \mathsf{true}] = Pr[(\mathbb{A} \parallel \mathbb{C}) \mid \mathbb{O} : \mathrm{R} = \mathsf{true}]$$

If tt denotes the set of executions of the product system $\mathbb{A} \times \mathbb{C} \times \mathbb{O}$ ending with action $(c_{\mathrm{F}}, \mathsf{true}, \_), (o_{\mathrm{F}}, \mathsf{true}, \_)$, we have:

$\pi_{\mathbb{O}}(\mathsf{tt}) = \mathcal{T}_{\mathbb{O}}^{-1}(\mathrm{R} = \mathsf{true})$ and $\pi_{\mathbb{C}[\mathbb{O}]}(\mathsf{tt}) = \mathcal{T}_{\mathbb{C}[\mathbb{O}]}^{-1}(\mathrm{R} = \mathsf{true})$.

We can deduce that:

$$\begin{array}{rl} Pr[\mathbb{A}|\mathbb{C}[\mathbb{O}] : \mathrm{R} = \mathsf{true}] = & Pr[\mathbb{A}|\mathbb{C}[\mathbb{O}] : \mathcal{T}_{\mathbb{C}[\mathbb{O}]}^{-1}(\mathrm{R} = \mathsf{true})] \\ = & Pr[\mathbb{A}|\mathbb{C}[\mathbb{O}] : \pi_{\mathbb{C}[\mathbb{O}]}(\mathsf{tt})] \\ = & Pr[\pi_{\mathbb{C}[\mathbb{O}]}(\mathbb{A} \times \mathbb{C} \times \mathbb{O}) : \pi_{\mathbb{C}[\mathbb{O}]}(\mathsf{tt})] \end{array}$$

and similarly that

$$\begin{array}{rl} Pr[\mathbb{A} \parallel \mathbb{C}|\mathbb{O} : \mathrm{R} = \mathsf{true}] = & Pr[\mathbb{A} \parallel \mathbb{C}|\mathbb{O} : \mathcal{T}_{\mathbb{O}}^{-1}(\mathrm{R} = \mathsf{true})] \\ = & Pr[\mathbb{A} \parallel \mathbb{C}|\mathbb{O} : \pi_{\mathbb{O}}(\mathsf{tt})] \\ = & Pr[\pi_{\mathbb{O}}(\mathbb{A} \times \mathbb{C} \times \mathbb{O}) : \pi_{\mathbb{O}}(\mathsf{tt})] \end{array}$$

After noticing that the fact that we impose the context finalization oracle to forward the same answer and do nothing else yields $\pi_{\mathbb{C}[\mathbb{O}]}^{-1}(\pi_{\mathbb{C}[\mathbb{O}]}(\mathsf{tt})) = \mathsf{tt} = \pi_{\mathbb{O}}^{-1}(\pi_{\mathbb{O}}(\mathsf{tt}))$, we apply the second item of our lemma III.10 and deduce

$$\begin{array}{rcl} Pr[\pi_{\mathbb{C}[\mathbb{O}]}(\mathbb{A} \times \mathbb{C} \times \mathbb{O}) : \pi_{\mathbb{C}[\mathbb{O}]}(\mathsf{tt})] & = & Pr[\mathbb{A} \times \mathbb{C} \times \mathbb{O} : \mathsf{tt}] \\ & = & Pr[\pi_{\mathbb{O}}(\mathbb{A} \times \mathbb{C} \times \mathbb{O}) : \pi_{\mathbb{O}}(\mathsf{tt})] \end{array}$$

This allows us to conclude to the desired equality.

As for rule B-Sub, we want to prove that:

$$Pr[\mathbb{A} \mid \mathbb{C}[\mathbb{O}] : \ E] \leq Pr[(\mathbb{A} \parallel \mathbb{C}) \mid \mathbb{O} : \ \pi_{\mathbb{O}}(\pi_{\mathbb{C}[\mathbb{O}]}^{-1}(E))]$$

which we can rewrite as

$$Pr[\pi_{\mathbb{C}[\mathbb{O}]}(\mathbb{A} \times \mathbb{C} \times \mathbb{O}) : \ E] \leq Pr[\pi_{\mathbb{O}}(\mathbb{A} \times \mathbb{C} \times \mathbb{O}) : \ \pi_{\mathbb{O}}(\pi_{\mathbb{C}[\mathbb{O}]}^{-1}(E))]$$

Besides, item 1 of lemma III.10 yields two equalities:

— $Pr[\pi_{\mathbb{C}[\mathbb{O}]}(\mathbb{A} \times \mathbb{C} \times \mathbb{O}) : \ E] = Pr[\mathbb{A} \times \mathbb{C} \times \mathbb{O} : \ \pi_{\mathbb{C}[\mathbb{O}]}^{-1}(E)]$,

— $Pr[\pi_{\mathbb{O}}(\mathbb{A} \times \mathbb{C} \times \mathbb{O}) : \ \pi_{\mathbb{O}}(\pi_{\mathbb{C}[\mathbb{O}]}^{-1}(E))] = Pr[\mathbb{A} \times \mathbb{C} \times \mathbb{O} : \ \pi_{\mathbb{O}}^{-1}(\pi_{\mathbb{O}}(\pi_{\mathbb{C}[\mathbb{O}]}^{-1}(E)))]$.

As for any set Set, $Set \subseteq \pi_{\mathbb{O}}^{-1}(\pi_{\mathbb{O}}(Set))$, we then have the inequality $Pr[\mathbb{A} \times \mathbb{C} \times \mathbb{O} :\ \pi_{\mathbb{C}[\mathbb{O}]}^{-1}(E)] \leq Pr[\mathbb{A} \times \mathbb{C} \times \mathbb{O} :\ \pi_{\mathbb{O}}^{-1}(\pi_{\mathbb{O}}(\pi_{\mathbb{C}[\mathbb{O}]}^{-1}(E)))]$, which yields the result we want. ■

III.3.5 — Bits and Pieces of ElGamal Security Proofs

In this subsection, we provide a detailed illustration of the use of the context rules introduced in this section. We examine the possibility to deduce the confidentiality of a plaintext ciphered with the ElGamal algorithm from the computational Diffie-Hellman assumption, and how to prove ROR-plaintext security. In addition to allowing us to show instances of applications of context rules, these two examples allow us to acknowledge the limitations of the set of rules we have established so far. In the sequel, we suppose that we have fixed a group $\mathbb{G}$ of prime order q.

Confidentiality. Our goal is to prove that provided with a tuple $(g, g^\alpha, g^\beta, g^{\alpha.\beta}.g^\mu)$, an adversary has a probability to retrieve the value of g^μ bounded by that of computing $g^{\alpha.\beta}$. This can naturally be translated as a CIL "bounded-by" statement via the introduction of system $\mathbb{CE}l\mathbb{G}amal$ consisting of only the two mandatory oracles and states mapping variables g, α, β and μ to values in $\mathbb{G}$ for g and in $[0..(q-1)]$ for the others. Initialization draws uniformly at random values g_0, α_0, β_0 and μ_0 for g, α, β and μ, before releasing values for $(g_0, g_0^{\alpha_0}, g_0^{\beta_0}, g_0^{\mu_0}.g_0^{\alpha_0.\beta_0})$ (and the adequately updated state). In the sequel we omit indices differentiating variables and values for the sake of readability. Finalization oracle has an input type $\mathbb{G}$ and does not modify the state. Then, confidentiality is captured by:

$$\mathbb{CE}l\mathbb{G}amal :_\epsilon \mathrm{R} = g^\mu$$

We would like to transform $\mathbb{CE}l\mathbb{G}amal$ into $\mathbb{CE}l\mathbb{G}amal'$, identical in everything except that the initialization oracle outputs $(g, g^\alpha, g^\beta, g^\mu)$. It is intuitively obvious that both systems are yielding the same distribution on states and initialization output modulo the fact that in $\mathbb{CE}l\mathbb{G}amal'$, $\mu - \alpha.\beta$ plays the role of what is μ in $\mathbb{CE}l\mathbb{G}amal$. Thus, we should be able to prove that an adversary has the same probability of computing g^μ given $(g, g^\alpha, g^\beta, g^\mu.g^{\alpha.\beta})$ as to compute $g^\mu.(g^{\alpha.\beta})^{-1}$ given $(g, g^\alpha, g^\beta, g^\mu)$. In CIL, it translates into:

$$\frac{\mathbb{CE}l\mathbb{G}amal' :_\epsilon \mathrm{R} = g^\mu.(g^{\alpha.\beta})^{-1}}{\mathbb{CE}l\mathbb{G}amal :_\epsilon \mathrm{R} = g^\mu}$$

However, we find there the first limit of the rules we have presented until now: we are at a loss when it comes to changing the system into another one yielding same distribution on states and translating soundly the event we want to bound according to changes performed. At this point, we admit this step[3] and continue with the proof of $\mathbb{CE}l\mathbb{G}amal' :_\epsilon \mathrm{R} = g^\mu.(g^{\alpha.\beta})^{-1}$.

We use the oracle system $\mathbb{CDH}$ introduced in example III.1.4, and decompose $\mathbb{CE}l\mathbb{G}amal$ into a context $\mathbb{C}$ of $\mathbb{CDH}$. $\mathbb{C}$ has a memory containing a variable for μ, and consists in two procedures, c_I and o_F, such that:

— c_I prompts o_I, which returns a triple. It then draws an integer $\mu_0 \in [0..(q-1)]$, outputs the triple together with the value of g^μ together with an updated memory $[\mu \mapsto \mu_0]$.

[3] A formal proof in CIL of the step can be found in III.4.3

— c_{F} gets an input in and calls o_{F} on $g^{\mu}.(in)^{-1}$.

We denote t' the time necessary to perform these computations. The application of B-Sub to this system yields the following proof:

$$\frac{\mathbb{CDH} :_{\epsilon(\mathbf{1}_k,t+t')} \mathrm{R} = g^{\alpha.\beta}}{\mathbb{CE}l\mathbb{G}amal' :_{\epsilon(\mathbf{1}_k,t)} \mathrm{R} = g^{\mu}.(g^{\alpha.\beta})^{-1}} \text{ B-Sub}$$

We must now justify that the composition of event $\mathrm{R} = g^{\mu}.(g^{\alpha.\beta})^{-1}$ with $\mathbb{C}$ equals event $\mathrm{R} = g^{\alpha.\beta}$. Figure III.3 displays a representation of the product traces of our system, where m denotes the state mapping variables g, α, β, μ to their values with index 0. It allows us to deduce that $in_{\mathbb{C}} = g^{\mu}.(g^{\alpha.\beta})^{-1}$ iff $in = g^{\mu}.(in_{\mathbb{C}})^{-1} = g^{\mu}.(g^{\mu}.(g^{\alpha.\beta})^{-1})^{-1} = g^{\alpha.\beta}$.

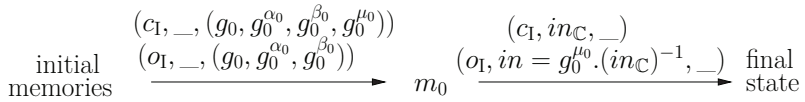


Figure III.3 – Product traces

ROR-plaintext Security. We now turn to proving ROR-plaintext security of the ElGamal encryption scheme. We recall that we have defined the oracle system $\mathbb{E}l\mathbb{G}amal$ in example 7. We formalize ROR-plaintext security with an adversary provided with access to a one-time cipher oracle and the public key. In the sequel we suppose that a generator g of $\mathbb{G}$ is fixed. We define the system outputting the random plaintext encryption instead of the real encryption of a message g^{μ}. Named $\mathbb{RE}l\mathbb{G}amal$, this system is identical to $\mathbb{E}l\mathbb{G}amal$ except that its state contains an additional variable r and that implementation of $\mathcal{E}$ consists in drawing a random integer $r \in [0..(q-1)]$, and outputting $(a, b.g^r)$ (together with an updated memory) instead of $(a, b.g^{\mu})$. We aim at showing $\mathbb{RE}l\mathbb{G}amal \sim_{\varepsilon} \mathbb{E}l\mathbb{G}amal$ for some function ε.

Since we defined $\mathbb{E}l\mathbb{G}amal$ as a context $\mathbb{C}$ of system $\mathbb{R}eal\mathbb{DH}$, it seems a good step forward to apply rule I-Sub to the DDH hypothesis $\mathbb{R}eal\mathbb{DH} \sim_{\varepsilon} \mathbb{R}and\mathbb{DH}$ (defined in example 5). Every execution of $\mathbb{C}[.]$ generates exactly one query to inner initialization and finalization oracles. If we denote by t' the time necessary to compute the result of an $\mathcal{E}$ query, rule I-Sub provides:

$$\frac{\mathbb{R}eal\mathbb{DH} \sim_{\varepsilon(\mathbf{1}_k,t)} \mathbb{R}and\mathbb{DH}}{\mathbb{C}[\mathbb{R}eal\mathbb{DH}] \sim_{\varepsilon(k,t-k(\mathcal{E}).t')} \mathbb{C}[\mathbb{R}and\mathbb{DH}]} \text{ I-Sub}$$

Unfortunately, while the left term is $\mathbb{E}l\mathbb{G}amal$ by definition, $\mathbb{C}[\mathbb{R}and\mathbb{DH}]$ is not exactly the same system as $\mathbb{RE}l\mathbb{G}amal$. Figure III.4 shows a table displaying synoptic descriptions of the pair of systems on both extremities. The column in the middle provides the description of a system which is yielding the same distribution on traces as each of its neighbors. As a result, the middle system is 0-indistinguishable of its left and right counterparts[4]. Once we have justified this latter claim, we can conclude to by applying the transitivity rule that $\mathbb{C}[\mathbb{R}and\mathbb{DH}] \sim_{\varepsilon(k,t-k(\mathcal{E}).t')} \mathbb{RE}l\mathbb{G}amal$.

Unfortunately, we are again in cases for which the current means at our disposal seem insufficient to prove the claim. Equality, or 0-indistinguishability, of systems on the left is

[4]We recall that the encryption oracle is a one-time cipher.

$\mathbb{C}[\mathbb{R}and\mathbb{DH}]$	$\mathbb{I}nterm$	$\mathbb{RE}l\mathbb{G}amal$
o_{I} :	o_{I} :	o_{I} :
draws α, β, r	draws α, β	draws α, β
return g^{α}	return g^{α}	return g^{α}
$\mathcal{E}(g^{\mu})$:	$\mathcal{E}(g^{\mu})$:	$\mathcal{E}(g^{\mu})$:
return $(g^{\beta}, g^{r}g^{\mu})$	draws r	draws r
	return $(g^{\beta}, g^{r}g^{\mu})$	return $(g^{\beta}, g^{\alpha.\beta}g^{r})$

Figure III.4 – Transition Between $\mathbb{C}[\mathbb{R}and\mathbb{DH}]$ and $\mathbb{RE}l\mathbb{G}amal$

obtained in proofs of the literature by using so-called lazy sampling lemmas, stating that we can draw values for variables exactly when we need them instead of any place before in the execution of an experiment. The reverse move is symmetrically named eager sampling. We generalize this kind of arguments in determinization rules. They are introduced at section III.5.

As for the other equality, it resembles the one which we have encountered in the confidentiality proof above. Indeed, we can easily justify that after every step, outputs are equally distributed in both systems by computing the probabilities. This leads us to a new set of rules to formalize this sort of arguments, namely bisimulation rules, introduced in the next section. Formal proofs of $\mathbb{C}[\mathbb{R}and\mathbb{DH}] \sim_0 \mathbb{RE}l\mathbb{G}amal$ and $\mathbb{I}nterm \sim_0 \mathbb{RE}l\mathbb{G}amal$ can be found respectively in III.5.3 and III.4.3.

III.4 Forward Bisimulation up to Relations

In the previous section, we show that we have no formal way to deal with certain types of transformations on oracles systems. If we try to perform an analogy with the game-based framework introduced in [Sho04], these are generally situations corresponding to "bridging steps" (e.g. those we have highlighted in our previous trial to prove security of ElGamal) and "failure events". About "bridging steps", Victor Shoup writes that they are transitions based on "purely conceptual changes", which are "typically a way of restating how certain quantities can be computed in a completely equivalent way". However, for our purpose of design of a framework to enable automatic verification or security-dedicated proof-assistance, this is not a formal enough notion to capture the proper nature of the steps. As for "transitions based on failure events", the notion encompasses a more precise set of transformations: games are identical "unless some failure event occurs", in which case we are provided with a way of bounding the difference between probabilities of winning in both games. Though our work is not directly abstracting away from game-based proofs, we do perform our proofs by successive transformations of the oracle systems. As a result, situations similar to those mentioned here occur. In this section, we introduce a tool to reason about transformations of oracle systems into systems that are equivalent, or equivalent up to the fact that some property is verified. Namely, we formalize these near-equivalence relations through our definition of the relation of bisimulation up to between systems. We then introduce rules of the logic putting this powerful concept to use.

III.4.1 — Definition of Forward Bisimulation Up to Relations

The definition provided below is inspired from the classical notion of bisimulation between probabilistic transition systems (e.g. see [BH97]). Yet bisimulation on its own is insufficient to capture some particular cases of imperfect bisimulation, which occur quite frequently, and for which we can still derive some interesting properties of probabilities. Instead of just adapting the definition of bisimulation to our setting, we therefore define bisimulation up to a step-property ϕ. Intuitively, ϕ is there to ensure that the step performed remains within the conditions of preservation of the bisimulation. While ϕ is true, the condition of "compatibility" guarantees that probabilistic bisimulation holds for the next step.

We now introduce formally these relations between systems. We start by presenting the definition of probabilistic forward bisimulation in our framework, and then add a step-property to the already quite involved definition. However, rules are directly written and proven for the complete version of bisimulation up to relations. The term 'forward' is used to distinguish two different bisimulation: forward and backwards bisimulation.

Let us consider two compatible oracle systems $\mathbb{O}$ and $\mathbb{O}'$. For every oracle name, we let $\mathsf{M}_{\mathbb{O}+\mathbb{O}'}$ be $\mathsf{M}_{\mathbb{O}} + \mathsf{M}_{\mathbb{O}'}$ and for every $o \in \mathsf{N}_{\mathbb{O}}$, we let $\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)$ be the disjoint sum of $\mathsf{Imp}_{\mathbb{O}}(o)$ and $\mathsf{Imp}_{\mathbb{O}'}(o)$, i.e.

$$\begin{array}{rlcl} \mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o): & \mathsf{In}(o) \times \mathsf{M}_{\mathbb{O}+\mathbb{O}'} & \to & \mathcal{D}(\mathsf{Out}(o) \times \mathsf{M}_{\mathbb{O}+\mathbb{O}'}) \\ & (q, m) & \mapsto & \begin{cases} \mathsf{Imp}_{\mathbb{O}}(o)(q,m) & \text{if } m \in \mathsf{M}_{\mathbb{O}} \\ \mathsf{Imp}_{\mathbb{O}'}(o)(q,m) & \text{otherwise} \end{cases} \end{array}$$

We write $m_1 \xrightarrow{(o,q,a)}_{>0} m_2$ iff $Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, m_1) = (a, m_2)] > 0$.

Definition (Forward Bisimulation for Oracle Systems). Let $R \subseteq \mathsf{M}_{\mathbb{O}+\mathbb{O}'} \times \mathsf{M}_{\mathbb{O}+\mathbb{O}'}$ be an equivalence relation. $\mathbb{O}$ and $\mathbb{O}'$ are bisimilar w.r.t. R, written $\mathbb{O} \equiv_R \mathbb{O}'$, iff $\bar{m}\ R\ \bar{m}'$, and for all $m_1 \xrightarrow{(o,q,a)}_{>0} m_2$ and m_3 such that $m_1\, R\, m_3$, we have

$$Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, m_1) \in (a, \mathcal{C}(m_2))] = Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, m_3) \in (a, \mathcal{C}(m_2))]$$

where $\mathcal{C}(m_2)$ is the equivalence class of m_2 under R and
$Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, m_1) \in (a, \mathcal{C}(m_2))] = \sum_{m\, R\, m_2} Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, m_1) \in (a, m)]$. □

The idea behind this definition is the following: states are grouped in classes according to an equivalence relation R. This relation is relevant if given two states in a same class, they offer the same possibility of evolution with the same probabilities. This is exactly what is captured by the equality between probabilities above. If the relation is relevant, then the systems are bisimilar w.r.t. the relation. Intuitively, we can foresee that, as we impose initial states to be in relation, then partial traces starting in the initial state and going through states related by R at each step are going to weigh the same in $\mathbb{O}$ and $\mathbb{O}'$.

After having provided this definition to familiarize the reader with the notion of bisimulation in our setting, we formally introduce the more complete concept of bisimulation up to as follows.

Definition (Forward Bisimulation Up to For Oracle Systems). Let $\phi \subseteq \mathsf{Xch} \times \mathsf{M}_{\mathbb{O}+\mathbb{O}'} \times \mathsf{M}_{\mathbb{O}+\mathbb{O}'}$ be a step-predicate and let $R \subseteq \mathsf{M}_{\mathbb{O}+\mathbb{O}'} \times \mathsf{M}_{\mathbb{O}+\mathbb{O}'}$ be an equivalence relation. $\mathbb{O}$ and $\mathbb{O}'$ are bisimilar up to ϕ, written $\mathbb{O} \equiv_{R,\phi} \mathbb{O}'$, iff $\bar{m}\ R\ \bar{m}'$, and for all $m_1 \xrightarrow{(o,q,a)}_{>0} m_2$ and $m_3 \xrightarrow{(o,q,a)}_{>0} m_4$ such that $m_1\, R\, m_3$, the following properties hold:

— *stability:* if $m_2 \, R \, m_4$ then

$$\phi((o,q,a),m_1,m_2) \quad \Leftrightarrow \quad \phi((o,q,a),m_3,m_4)$$

— *compatibility:* if $\phi((o,q,a),m_1,m_2)$, then

$$Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q,m_1) \in (a,\mathcal{C}(m_2))] = Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q,m_3) \in (a,\mathcal{C}(m_2))]$$

where $\mathcal{C}(m_2)$ is the equivalence class of m_2 under R, and
$Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q,m_1) \in (a,\mathcal{C}(m_2))] = \sum_{m \, R \, m_2} Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q,m_1) \in (a,m)]$.

□

Here we capture the notion of imperfect bisimulation. The addition of ϕ is quite intuitive for compatibility: we require equality between probabilities only if ϕ is true for the step we perform. On the contrary, when ϕ does not hold, we do not know anything about the evolution of probabilities. The need for stability is quite easy to imagine. Since we want to reason about classes of states in which we can end up after a step, we intuitively need ϕ to have the same truth-value on whole classes.

We now turn to traces to see what kind of conclusions we can draw from two systems being bisimilar up to some property. In the remainder of this subsection, we consider fixed $\mathbb{O}$, $\mathbb{O}'$, R and ϕ satisfying the above definition. The relation defined on states can be lifted to (partial) executions and traces quite easily.

Definition (Lifting The Relation). Let η and η' be two partial executions of $\mathbb{A} \mid \mathbb{O}$ or $\mathbb{A} \mid \mathbb{O}'$ of length k such that $\eta = (m_0, m_a^0) \xrightarrow{x_1} (m_1, m_a^1) \xrightarrow{x_2} \ldots \xrightarrow{x_k} (m_k, m_a^k)$ and $\eta' = (m'_0, m_a^0) \xrightarrow{x_1} (m'_1, m_a^1) \xrightarrow{x_2} \ldots \xrightarrow{x_k} (m'_k, m_a^k)$. They are said to be *in relation by* R, denoted $\eta \; R \; \eta'$, iff $m_i \; R \; m'_i$ for all $i \in [0..k]$.

Similarly, two traces $\tau = \bar{m}_{\mathbb{O}} \xrightarrow{x_1} m_1 \xrightarrow{x_2} \ldots \xrightarrow{x_k} m_k$ and $\tau' = \bar{m}_{\mathbb{O}} \xrightarrow{x_1} m'_1 \xrightarrow{x_2} \ldots \xrightarrow{x_k} m'_k$ are in relation ($\tau \; R \; \tau'$) iff $m_i \; R \; m'_i$ for all $i \in [0..k]$. □

Now that we can speak of equivalence classes for partial executions, we can assign them a probability.

Definition (Probability of an Equivalence Class w.r.t. Relation R of Partial Executions). Let η be a partial execution of $\mathbb{A} \mid \mathbb{O}$ or $\mathbb{A} \mid \mathbb{O}'$. The equivalence class of η is defined by $\mathcal{C}(\eta) = \{\eta' \mid \eta \; R \; \eta'\}$. The $\mathbb{O}$-class of η, denoted $\mathcal{C}_{\mathbb{O}}(\eta)$, is the intersection with $\mathbb{O}$-traces of $\mathcal{C}(\eta)$. Its probability is given by:

$$Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\eta)] = \sum_{\eta' \in \mathcal{C}_{\mathbb{O}}(\eta)} Pr[\mathbb{A} \mid \mathbb{O} : \eta']$$

A similar definition can be written for $\mathbb{O}'$. □

Consider a state $m_1 \in \mathsf{M}_{\mathbb{O}}$, in relation with $m_3 \in \mathsf{M}_{\mathbb{O}'}$. According to the definition of bisimulation up to, if we perform one step for which ϕ holds from m_1 and its successor or m_3 and its successor, then we can move to the same equivalence classes of states with the same probability. Say we have gone through such a step: $m_1 \xrightarrow{x} m_2$ and $m_3 \xrightarrow{x} m_4$, and $m_2 \; R \; m_4$. We can iterate the same reasoning on m_2 and m_4. Informally, we can anticipate that if we perform a series of steps for which ϕ holds, it yields equivalence classes on executions with same probabilities in $\mathbb{O}$ and $\mathbb{O}'$.

Lemma III.11. *Let η be a partial execution of $\mathbb{A} \mid \mathbb{O}$ of length k such that $\eta = (m_0, m_a^0) \xrightarrow{x_1}$*

$(m_1, m_a^1) \xrightarrow{x_2} \ldots \xrightarrow{x_k} (m_k, m_a^k)$.

$$Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\eta)] = \Pi_{i=1}^{k} Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}((m_{i-1}, m_a^{i-1}) \xrightarrow{x_i} (m_i, m_a^i))]$$

where

$$Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}((m_{i-1}, m_a^{i-1}) \xrightarrow{x_i} (m_i, m_a^i))] = \sum_{\tilde{m}_i \ R \ m_i} \Pr[(m_{i-1}, m_a^{i-1}) \xrightarrow{x_i} (\tilde{m}_i, m_a^i)]$$

Proof. We prove the result by induction on the length of partial executions. The case for $k = 1$ is obvious: then the partial execution is just a step.

Let us assume that the result holds for partial executions of length $k-1$. We notice that:

$$\mathcal{C}_{\mathbb{O}}(\eta) = \{\eta' = (\tilde{m}_0, m_a^0) \xrightarrow{x_1} (\tilde{m}_1, m_a^1) \xrightarrow{x_2} \ldots \xrightarrow{x_k} (\tilde{m}_k, m_a^k) | \tilde{m}_i \ R \ m_i, \tilde{m}_i \in \mathsf{M}_{\mathbb{O}}\}$$

Then,

$$\begin{aligned}
& Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\eta)] \\
\stackrel{def}{=} & \textstyle\sum_{\eta' \in \mathcal{C}_{\mathbb{O}}(\eta)} Pr[\mathbb{A} \mid \mathbb{O} : \eta'] \\
= & \sum_{\tilde{m}_i \ R \ m_i, \ m_i \in \mathsf{M}_{\mathbb{O}}, \ i=1..k} Pr[\mathbb{A} \mid \mathbb{O} : \mathsf{Pref}(\eta', k-1)] \Pr[\mathbb{A} \mid \mathbb{O} : \eta'[k]] \\
= & \sum_{\substack{\tilde{m}_i \ R \ m_i, \ m_i \in \mathsf{M}_{\mathbb{O}}, \ i=1..(k-1) \\ \eta' = (\tilde{m}_0, m_a^0) \xrightarrow{x_1} (\tilde{m}_1, m_a^1) \xrightarrow{x_2} \ldots \xrightarrow{x_k} (\tilde{m}_k, m_a^k)}} \Pr[\mathbb{A} \mid \mathbb{O} : \mathsf{Pref}(\eta', k-1)] \sum_{\substack{\tilde{m}_k \ R \ m_k \\ \tilde{m}_k \in \mathsf{M}_{\mathbb{O}}}} \Pr[\mathbb{A} \mid \mathbb{O} : \eta'[k]] \\
= & \sum_{\substack{\tilde{m}_i \ R \ m_i, \ m_i \in \mathsf{M}_{\mathbb{O}}, \ i=1..(k-1) \\ \eta' = (\tilde{m}_0, m_a^0) \xrightarrow{x_1} (\tilde{m}_1, m_a^1) \xrightarrow{x_2} \ldots \xrightarrow{x_k} (\tilde{m}_k, m_a^k)}} \Pr[\mathbb{A} \mid \mathbb{O} : \mathsf{Pref}(\eta', k-1)]. \Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}((m_{k-1}, m_a^{k-1}) \xrightarrow{x_k} (m_k, m_a^k))]
\end{aligned}$$

and we can conclude using the induction hypothesis. ■

We can now continue with a lemma formally translating the intuition we have provided above. It also shows that given related traces for which ϕ holds at every step, we have equal probabilities to make a next step *not* verifying ϕ when interacting with $\mathbb{A} \mid \mathbb{O}$ as when interacting with $\mathbb{A} \mid \mathbb{O}'$.

Lemma III.12. *Let η be a partial execution of $\mathbb{A} \mid \mathbb{O}$ of length k such that ϕ holds for each of its steps:*

$$\eta = (m_0, m_a^0) \xrightarrow{x_1} (m_1, m_a^1) \xrightarrow{x_2} \ldots \xrightarrow{x_k} (m_k, m_a^k) \text{ and } \forall i = 1..k, \phi(x_i, m_{i-1}, m_i)$$

— *Let $\sigma = (m_k, m_a^k) \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})$ be a step. Let $\eta' = \eta \cdot \sigma$. If $\phi(x_{k+1}, m_k, m_{k+1})$ then $Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\eta')] = Pr[\mathbb{A} \mid \mathbb{O}' : \mathcal{C}_{\mathbb{O}'}(\eta')]$*

— $Pr[\mathbb{A} \mid \mathbb{O} : \eta_0 \cdot \sigma_0 \wedge (\eta_0 \ R \ \eta) \wedge \neg\phi(\sigma_0)] = Pr[\mathbb{A} \mid \mathbb{O}' : \eta_0 \cdot \sigma_0 \wedge (\eta_0 \ R \ \eta) \wedge \neg\phi(\sigma_0)]$

Proof. The first item can be proven by mathematical induction on the length n of the resulting partial execution η'. For the case when $n = 1$, η' is a step σ which we can write $\sigma = (\bar{m}_{\mathbb{O}}, \bar{m}_{\mathbb{A}}) \xrightarrow{x_1} (m_1, m_a^1)$, where $x_1 = (o_{\mathrm{I}}, q_1, a_1)$. Then we have:

$$\Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\sigma)] = \sum_{\tilde{m}_1 \ R \ m_1, \tilde{m}_1 \in \mathsf{M}_{\mathbb{O}}} \Pr[\mathbb{A} \mid \mathbb{O} : (\bar{m}_{\mathbb{O}}, \bar{m}_{\mathbb{A}}) \xrightarrow{x_1} (\tilde{m}_1, m_a^1)]$$

Moreover, if we denote $\tilde{\sigma} = (\bar{m}_{\mathbb{O}}, \bar{m}_{\mathbb{A}}) \xrightarrow{x_1} (\tilde{m}_1, m_a^1)$, by definition of the probability of a partial step, $\Pr[\mathbb{A} \mid \mathbb{O} : \tilde{\sigma}]$ is:

$\sum_{m'_a} \Pr[\mathsf{A}(\bar{m}_{\mathbb{A}}) = ((o_{\mathrm{I}}, q_1), m'_a)].$
$\Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o_{\mathrm{I}})(q_1, \bar{m}_{\mathbb{O}}) \in (a_1, \tilde{m}_1)]. \Pr[\mathsf{A}_{\downarrow}((o_{\mathrm{I}}, q_1, a_1), m'_a) = m_a^1]$

We continue by swapping both sums in our computation of $\Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\sigma)]$. As the adversary's execution does not depend on $\tilde{m}_1$, we can consider that terms containing adversarial computations are multiples of

$$\sum_{\tilde{m_1}\ R\ m_1,\ \tilde{m}_1 \in \mathsf{M}_{\mathbb{O}}} \Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o_{\mathrm{I}})(q_1, \bar{m}_{\mathbb{O}}) \in (a_1, \tilde{m}_1)]$$

We know $\phi(x_1, \bar{m}_{\mathbb{O}}, m_1)$ holds, hence we can use compatibility to replace the above term by the following one:

$$\sum_{\tilde{m_1}\ R\ m_1,\ \tilde{m}_1 \in \mathsf{M}_{\mathbb{O}'}} \Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o_{\mathrm{I}})(q_1, \bar{m}'_{\mathbb{O}}) \in (a_1, \tilde{m}_1)]$$

After this, we can undo all of the manipulations we just did: first put adversarial probabilities inside the sum on states in relation "$\sum_{\tilde{m_1}\ R\ m_1,\ \tilde{m}_1 \in \mathsf{M}_{\mathbb{O}'}}$", then swap both sum symbols and replace our three terms with $\Pr[\mathbb{A} \mid \mathbb{O}' : \tilde{\sigma}]$, to finally conclude that

$$\sum_{\tilde{m_1}\ R\ m_1,\ \tilde{m}_1 \in \mathsf{M}_{\mathbb{O}'}} \Pr[\mathbb{A} \mid \mathbb{O}' : \tilde{\sigma}] = \Pr[\mathbb{A} \mid \mathbb{O}' : \mathcal{C}_{\mathbb{O}'}(\sigma)]$$

which is our result for $n = 1$.

For the induction step, if η is a partial execution of length $n > 1$ and $\sigma = (m_n, m_a^n) \xrightarrow{x_{n+1}} (m_{n+1}, m_a^{n+1})$, is a step, we use lemma III.11 to write

$$Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\eta \cdot \sigma)] = Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\eta)] Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\sigma)]. \ (*)$$

A similar equation can be written for $\mathbb{A} \mid \mathbb{O}'$. By performing the same reasoning as for the initial case, we can obtain easily $Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\sigma)] = Pr[\mathbb{A} \mid \mathbb{O}' : \mathcal{C}_{\mathbb{O}'}(\sigma)]$. The induction hypothesis provides $Pr[\mathbb{A} \mid \mathbb{O}' : \mathcal{C}_{\mathbb{O}'}(\eta)] = Pr[\mathbb{A} \mid \mathbb{O}' : \mathcal{C}_{\mathbb{O}'}(\eta)]$. The conclusion follows from (*).

As for the second item, it is justified by the computations below:

$$\begin{aligned}
& Pr[\mathbb{A} \mid \mathbb{O} : \eta_0 \cdot \sigma_0 \wedge \eta_0\ R\ \eta \wedge \neg\phi(\mathcal{T}(\sigma_0))] \\
= \ & 1 - Pr[\mathbb{A} \mid \mathbb{O} : \eta_0 \cdot \sigma_0 \wedge \eta_0\ R\ \eta \wedge \phi(\mathcal{T}(\sigma_0))] \\
= \ & 1 - \textstyle\sum_{\sigma_0 \mid \phi(\mathcal{T}(\sigma_0))} Pr[\mathbb{A} \mid \mathbb{O} : \eta_0 \cdot \sigma_0 \wedge \eta_0\ R\ \eta] \quad (1)
\end{aligned}$$

where (1) holds by definition of the probability of a step-predicate (or an event) as the sum of probabilities of elements satisfying it. We divide the set of steps verifying ϕ into distinct classes of equivalence of $\sigma_1, \ldots, \sigma_m$. Then we can regroup terms and go on with our computation as follows:

$$\begin{aligned}
& Pr[\mathbb{A} \mid \mathbb{O} : \eta_0 \cdot \sigma_0 \wedge \eta_0\ R\ \eta \wedge \neg\phi(\sigma_0)] \\
= \ & 1 - \textstyle\sum_{i=1..m} Pr[\mathbb{A} \mid \mathbb{O} : \eta_0 \cdot \sigma_0 \wedge \eta_0\ R\ \eta \wedge \sigma_0\ R\ \sigma_i] \\
= \ & 1 - \textstyle\sum_{i=1..m} Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\eta_0 \cdot \sigma_i)] \\
= \ & 1 - \textstyle\sum_{i=1..m} Pr[\mathbb{A} \mid \mathbb{O}' : \mathcal{C}_{\mathbb{O}'}(\eta_0 \cdot \sigma_i)] \qquad (2) \\
= \ & 1 - \textstyle\sum_{\sigma_0 \mid \phi(\mathcal{T}(\sigma_0))} Pr[\mathbb{A} \mid \mathbb{O}' : \eta_0 \cdot \sigma_0 \wedge \eta_0\ R\ \eta] \\
= \ & 1 - Pr[\mathbb{A} \mid \mathbb{O}' : \eta_0 \cdot \sigma_0 \wedge \eta_0\ R\ \eta \wedge \phi(\mathcal{T}(\sigma_0))] \\
= \ & Pr[\mathbb{A} \mid \mathbb{O}' : \eta_0 \cdot \sigma_0 \wedge \eta_0\ R\ \eta \wedge \neg\phi(\mathcal{T}(\sigma_0))]
\end{aligned}$$

where (2) follows from the first item of the lemma, and subsequent equalities mirror the beginning of our computation.

■

III.4.2 — Rules Using Bisimulation Up to

We have just seen that equivalence classes of traces for which steps verify ϕ have the nice property of uniting traces of $\mathbb{A} \mid \mathbb{O}$ and $\mathbb{A} \mid \mathbb{O}'$ weighing the same probability. Consequently, when considering events gathering whole classes of traces, the equality scales up. We call this type of events *compatible* with the relation R.

Definition (Compatible Events). Let E be an event, defining a subset of executions of $\mathbb{A} \mid \mathbb{O}$ and $\mathbb{A} \mid \mathbb{O}'$. We say that E is R-compatible iff for all related partial traces τ and τ' starting with initial memories (i.e. $\tau\ R\ \tau'$), if $E(\tau)$ then $E(\tau')$.

If E is an event on $\mathbb{A} \mid \mathbb{O}$ traces and E' is an event on $\mathbb{A} \mid \mathbb{O}'$ traces, E and E' are said R-compatible iff $E \cup E'$ is R-compatible. This is denoted $E\ R\ E'$. □

We are now ready to state and prove rules using bisimulation up to.

Lemma III.13. *We consider two compatible oracle systems $\mathbb{O}$ and $\mathbb{O}'$. The following rules are sound:*

$$\frac{\mathbb{O} :_\epsilon E \wedge \mathsf{G}_\varphi \quad \mathbb{O} \equiv_{R,\varphi} \mathbb{O}' \quad E\ R\ E'}{\mathbb{O}' :_\epsilon E' \wedge \mathsf{G}_\varphi}\ \textit{B-Bis}\mathsf{G}$$

$$\frac{\mathbb{O} :_\epsilon E\,\mathsf{U}\,\neg\varphi \quad \mathbb{O} \equiv_{R,\varphi} \mathbb{O}' \quad E\ R\ E'}{\mathbb{O}' :_\epsilon E'\,\mathsf{U}\,\neg\varphi}\ \textit{B-Bis}\,\mathsf{U}$$

$$\frac{\mathbb{O} :_\epsilon \mathsf{F}_{\neg\varphi} \quad \mathbb{O} \equiv_{R,\varphi} \mathbb{O}'}{\mathbb{O} \sim_\epsilon \mathbb{O}'}\ \textit{I-Bis}$$

Proof. Most of the work to prove the rules was actually done proving lemmas in the previous subsection.

For rule B-Bis$\mathsf{G}$ we start by noticing that $E \wedge \mathsf{G}_\phi$ is a compatible event. We can decompose the set of traces created by $\mathbb{A} \mid \mathbb{O}$ and $\mathbb{A} \mid \mathbb{O}'$ and verifying $E \wedge \mathsf{G}_\phi$ (resp. $E' \wedge \mathsf{G}_\phi$) into (distinct) classes of equivalences of a finite number of executions $\sigma_1, \ldots, \sigma_m$.

$$\begin{array}{rcll} Pr[\mathbb{A} \mid \mathbb{O} : E \wedge \mathsf{G}_\phi] & = & \sum_{i=1..m} Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\rho_i)] & \\ & = & \sum_{i=1..m} Pr[\mathbb{A} \mid \mathbb{O}' : \mathcal{C}_{\mathbb{O}'}(\rho_i)] & (1) \\ & = & Pr[\mathbb{A} \mid \mathbb{O}' : E' \wedge \mathsf{G}_\phi] & (2) \end{array}$$

To justify (1), we can apply first item of lemma III.12 to $\mathcal{C}_{\mathbb{O}}(\sigma_i)$. (2) follows from the compatibility property of the events: the same classes of executions verify $E \wedge \mathsf{G}_\phi$ and $E' \wedge \mathsf{G}_\phi$.

For rule B-Bis$\,\mathsf{U}$ we perform a similar kind of analysis and decompose the set of executions verifying $E\,\mathsf{U}\,(\neg\phi)$ into disjoint classes represented by $\eta_1, \ldots, \eta_m$. We then apply the second item of lemma III.12. It provides:

$$\begin{array}{rcll} Pr[\mathbb{A} \mid \mathbb{O} : E\,\mathsf{U}\,\neg\phi] & = & \sum_{i=1..m} Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\eta_i)] & \\ & = & \sum_{i=1..m} Pr[\mathbb{A} \mid \mathbb{O}' : \mathcal{C}_{\mathbb{O}'}(\eta_i)] & (1) \\ & = & Pr[\mathbb{A} \mid \mathbb{O}' : E'\,\mathsf{U}\,\neg\phi] & (2) \end{array}$$

Again, compatibility implies that the same classes of traces verify $E\,\mathsf{U}\,(\neg\phi)$ and $E'\,\mathsf{U}\,(\neg\phi)$.

Finally, the third rule follows from the equalities between probabilities shown above. Indeed,

$$\begin{aligned} & Pr[\mathbb{A} \mid \mathbb{O} : \mathrm{R} = \mathsf{true}] - Pr[\mathbb{A} \mid \mathbb{O}' : \mathrm{R} = \mathsf{true}] \\ = \; & Pr[\mathbb{A} \mid \mathbb{O} : \mathrm{R} = \mathsf{true} \wedge \mathsf{G}_\phi] + Pr[\mathbb{A} \mid \mathbb{O} : \mathrm{R} = \mathsf{true} \wedge \mathsf{F}_{\neg\phi}] \\ & \quad - Pr[\mathbb{A} \mid \mathbb{O}' : \mathrm{R} = \mathsf{true} \wedge \mathsf{G}_\phi] - Pr[\mathbb{A} \mid \mathbb{O}' : \mathrm{R} = \mathsf{true} \wedge \mathsf{F}_{\neg\phi}] \\ = \; & Pr[\mathbb{A} \mid \mathbb{O} : \mathrm{R} = \mathsf{true} \wedge \mathsf{F}_{\neg\phi}] - Pr[\mathbb{A} \mid \mathbb{O}' : \mathrm{R} = \mathsf{true} \wedge \mathsf{F}_{\neg\phi}] \end{aligned}$$

which implies $|Pr[\mathbb{A} \mid \mathbb{O} : \mathrm{R} = \mathsf{true}] - Pr[\mathbb{A} \mid \mathbb{O}' : \mathrm{R} = \mathsf{true}]| \leq Pr[\mathbb{A} \mid \mathbb{O} : \mathsf{F}_{\neg\phi}]$ and allows to conclude to I-Bis.

■

III.4.3 — Examples of Use

We provide here the justifications of steps missing in proofs of ElGamal properties which can be obtained by application of a bisimulation rule.

Confidentiality of ElGamal. We present the bisimulation argument for the first transformation of systems necessary to prove confidentiality of the ElGamal encryption scheme. We recall the initialization oracles of systems $\mathbb{CE}l\mathbb{G}amal$ and $\mathbb{CE}l\mathbb{G}amal'$ in the table below. Finalization oracles coincide.

$\mathsf{Imp}_{\mathbb{CE}l\mathbb{G}amal}(o_{\mathrm{I}})(\_,\_) =$	$\mathsf{Imp}_{\mathbb{CE}l\mathbb{G}amal'}(o_{\mathrm{I}})(\_,\_) =$
$g \leftarrow \mathcal{U}(\mathbb{G});$	$g \leftarrow \mathcal{U}(\mathbb{G});$
$\alpha \leftarrow \mathcal{U}([0..(q-1)]);$	$\alpha \leftarrow \mathcal{U}([0..(q-1)]);$
$\beta \leftarrow \mathcal{U}([0..(q-1)]);$	$\beta \leftarrow \mathcal{U}([0..(q-1)]);$
$\mu \leftarrow \mathcal{U}([0..(q-1)]);$	$\mu \leftarrow \mathcal{U}([0..(q-1)]);$
$\mathsf{return}\ (g, g^\alpha, g^\beta, g^\mu.g^{\alpha.\beta})$	$\mathsf{return}\ (g, g^\alpha, g^\beta, g^\mu)$

We define an equivalence relation $\mathcal{R}$ between states m, m' as follows:

— if $m, m' \in \mathsf{M}_{\mathbb{CE}l\mathbb{G}amal}$ or $m, m' \in \mathsf{M}_{\mathbb{CE}l\mathbb{G}amal'}$, $m \,\mathcal{R}\, m'$ iff $m = m'$

— if $m \in \mathsf{M}_{\mathbb{CE}l\mathbb{G}amal}$ and $m' \in \mathsf{M}_{\mathbb{CE}l\mathbb{G}amal'}$ $m \,\mathcal{R}\, m'$ iff $m.\alpha = m'.\alpha$, $m.\beta = m'.\beta$ and $m.\mu + m.\alpha * m.\beta = m'.\mu$.

We can then verify that $\mathbb{CE}l\mathbb{G}amal \equiv_{\mathcal{R},\mathsf{true}} \mathbb{CE}l\mathbb{G}amal'$. We let E be the event $\mathrm{R} = g^\mu$, while E' denotes $\mathrm{R} = g^\mu.(g^{\alpha.\beta})^{-1}$. To apply rule $B - Bis\mathsf{G}$, we still have to check compatibility of $E \cup E'$ with $\mathcal{R}$, i.e. that given two states $m \in \mathsf{M}_{\mathbb{CE}l\mathbb{G}amal}$ and $m' \in \mathsf{M}_{\mathbb{CE}l\mathbb{G}amal'}$ in relation by $\mathcal{R}$, E holds in state m iff E' holds in state m', which is obvious by the definition of the relation.

The rule yields the following proof, which, up to two straightforward applications of rule UR to introduce and remove conjunctions with $\mathsf{G}_{\mathsf{true}}$, fills in the gap left in the confidentiality proof of ElGamal:

$$\text{B-Bis}\mathsf{G}\ \frac{\mathbb{CE}l\mathbb{G}amal' :_\epsilon E' \wedge \mathsf{G}_{\mathsf{true}} \qquad \mathbb{CE}l\mathbb{G}amal \equiv_{\mathcal{R},\mathsf{true}} \mathbb{CE}l\mathbb{G}amal' \qquad E' \;\mathcal{R}\; E}{\mathbb{CE}l\mathbb{G}amal :_\epsilon E \wedge \mathsf{G}_{\mathsf{true}}}$$

This concludes the proof of confidentiality, for which a proof tree summing all the steps can be found in VII.1.

ROR-plaintext Security of ElGamal. The statement left aside involves systems $\mathbb{I}nterm$ and $\mathbb{RE}l\mathbb{G}amal$ whose state consist in variables α, β, r and implementaions of oracles are as follows:

$\mathbb{I}nterm$	$\mathbb{REL}\mathbb{G}amal$
$\mathsf{Imp}_{\mathbb{I}nterm}(o_{\mathrm{I}})(\_,\_)$:	$\mathsf{Imp}_{\mathbb{REL}\mathbb{G}amal}(o_{\mathrm{I}})(\_,\_)$:
$\alpha \leftarrow \mathcal{U}([0..(q-1)])$;	$\alpha \leftarrow \mathcal{U}([0..(q-1)])$;
$\beta \leftarrow \mathcal{U}([0..(q-1)])$;	$\beta \leftarrow \mathcal{U}([0..(q-1)])$;
$r := \lambda$;	$r := \lambda$;
return g^{α}	return g^{α}
$\mathcal{E}(g^{\mu}, m)$:	$\mathcal{E}(g^{\mu}, m)$:
if $r = \lambda$ then	if $r = \lambda$ then
$r \leftarrow \mathcal{U}([0..(q-1)])$;	$r \leftarrow \mathcal{U}([0..(q-1)])$;
return $(g^{\beta}, g^{r} g^{\mu})$	return $(g^{\beta}, g^{\alpha.\beta} g^{r})$
else	else
return λ	return λ

We want to establish their 0-indistinguishability. We exhibit an equivalence relation $\mathcal{R}$ between both systems based on a similar idea to the relation we use above for confidentiality. Indeed, states m, m' are in relation if:

— $m, m' \in \mathsf{M}_{\mathbb{I}nterm}$ or $m, m' \in \mathsf{M}_{\mathbb{REL}\mathbb{G}amal}$, $m \,\mathcal{R}\, m'$ iff $m = m'$

— if $m \in \mathsf{M}_{\mathbb{I}nterm}$ and $m' \in \mathsf{M}_{\mathbb{REL}\mathbb{G}amal}$ $m \,\mathcal{R}\, m'$ iff $m.\alpha = m'.\alpha$, $m.\beta = m'.\beta$ and $m.r = m'.r = \lambda$ or $m.r \neq \lambda \wedge m'.r \neq \lambda$.

Applying I-Sub results in:

$$\text{I-Sub}\ \frac{\mathbb{I}nterm :_0 \mathsf{F}_{\neg\mathsf{true}} \qquad \mathbb{I}nterm \equiv_{\mathcal{R},\mathsf{true}} \mathbb{REL}\mathbb{G}amal}{\mathbb{I}nterm \sim_0 \mathbb{REL}\mathbb{G}amal}$$

To complete the argument, one needs to apply rule Fail to obtain a proof of $\mathbb{I}nterm :_0 \mathsf{F}_{\neg\mathsf{true}}$ in CIL.

Examples of use of bisimulation rules where the bisimulation is not perfect can be found in chapter IV.

III.5 Determinization

While capturing a vast number of transformations on oracle systems, bisimulation up to relations are useless for some kind of moves involving complex modifications of the distributions yielded on traces. Indeed, the notion of bisimulation up to requires a conservation of probabilities *at each step*. Even the possibility of grouping states through an equivalence relation cannot stretch the concept enough to encompass early or belated drawing of elements. However, the frequent use of these latter arguments, commonly referred to as "lazy" or "eager sampling", demands that our system provides a formal treatment for those.

III.5.1 — Determinization of a System by Another

The concept which we introduce is inspired from an automata determinization technique. It is based on the possibility to decompose states of a system into two components, and to exhibit a distribution - say γ - allowing to obtain the second component given the first one. We consider two oracle systems $\mathbb{O}$ and $\mathbb{O}'$, and assume that states $m' \in \mathsf{M}_{\mathbb{O}'}$ can be seen as pairs $(m, m'') \in \mathsf{M}_{\mathbb{O}} \times \mathsf{M}''_{\mathbb{O}}$. We consider an exchange (o, q, a) and states m_1 and m_2 such

that $Pr[\mathsf{Imp}_{\mathbb{O}}(o)(q, m_1) = (a, m_2)] = p_1$. We must relate p_1 with $Pr[\mathsf{Imp}_{\mathbb{O}'}(o)(q, (m_1, m_1'')) = (a, (m_2, m_2''))]$ for some states m_1'' and m_2''. When m_2'' is fixed, we denote this last probability by $p_2(m_1'')$.

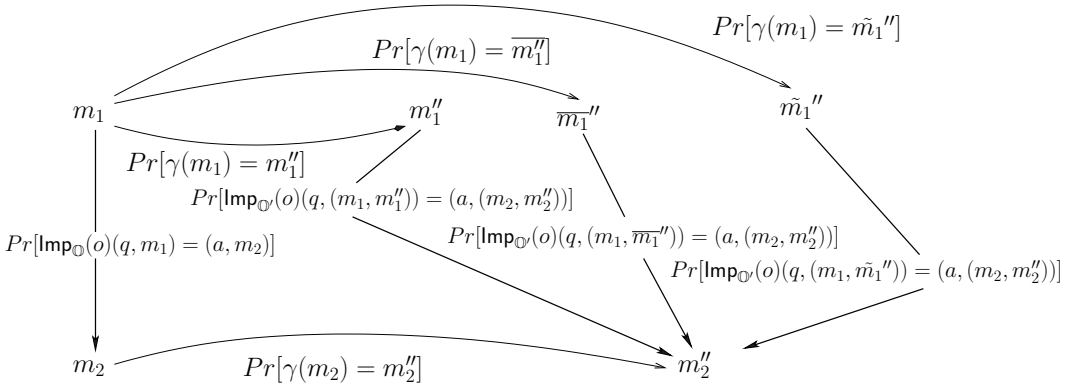


Figure III.5 – Two Ways to End Up in State (m_2, m_2'')

The intuition behind determinization is illustrated in figure III.5. The idea is that there are two ways to compute the probability to end up in state (m_2, m_2'') for a fixed m_2'' knowing that the step starts with a state of first component m_1. The first possibility is to perform the exchange in $\mathbb{O}$ and then draw m_2'' according to γ. In the figure, it corresponds to going down first and then right. A second way is to look at all possible m_1'' which γ map to m_1 and then perform the exchange in $\mathbb{O}'$. This is, in the figure, pictured by paths going from left to right and then downward. Intuitively, imposing the equality between these two ways of computing probabilities is going to compel the same equality to hold for steps, which in turn propagates to traces. The formalization of these insights is as follows.

Definition (System $\mathbb{O}$ Determinizes System $\mathbb{O}'$). Let $\mathbb{O}$ and $\mathbb{O}'$ be compatible oracle systems. $\mathbb{O}$ *determinizes* $\mathbb{O}'$ by distribution $\gamma : \mathsf{M}_{\mathbb{O}} \to \mathcal{D}(\mathsf{M}_{\mathbb{O}}'')$, written $\mathbb{O} \leq_{\mathrm{det},\gamma} \mathbb{O}'$, iff firstly $\mathsf{M}_{\mathbb{O}} \times \mathsf{M}_{\mathbb{O}}'' = \mathsf{M}_{\mathbb{O}'}$, secondly, there exists $\bar{m}_{\mathbb{O}}''$ such that $(\bar{m}_{\mathbb{O}}, \bar{m}_{\mathbb{O}}'') = \bar{m}_{\mathbb{O}}'$ and $\gamma(\bar{m}_{\mathbb{O}}) = \delta_{\bar{m}_{\mathbb{O}}''}$, and lastly for all $o \in \mathsf{N}_{\mathbb{O}}$, $q \in \mathsf{In}(o)$, $a \in \mathsf{Out}(o)$, $m_1, m_2 \in \mathsf{M}_{\mathbb{O}}$ and $m_2'' \in \mathsf{M}_{\mathbb{O}}''$:

$$\Pr[\gamma(m_2) = m_2''] \; p_1 = \sum_{m_1'' \in \mathsf{M}_{\mathbb{O}}''} \Pr[\gamma(m_1) = m_1''] \; p_2(m_1'')$$

where:

$$\begin{array}{rcl} p_1 & = & Pr[\mathsf{Imp}_{\mathbb{O}}(o)(q, m_1) = (a, m_2)] \\ p_2(m_1'') & = & Pr[\mathsf{Imp}_{\mathbb{O}'}(o)(q, (m_1, m_1'')) = (a, (m_2, m_2''))] \end{array}$$

□

Example 8. As an example, we choose the simplest possible case of eager sampling. We consider oracle systems $\mathbb{O}$ and $\mathbb{O}'$ such that states of $\mathbb{O}$ contain variable r, while those of $\mathbb{O}'$ contain variable r and r'. Both variables take values in bitstrings of length l. Initial states are $\bar{m}_{\mathbb{O}}$ for $\mathbb{O}$ and $(\bar{m}_{\mathbb{O}}, \bar{m}_{\mathbb{O}}'')$ for $\mathbb{O}'$. The oracles given in the following table:

$\mathbb{O}$:

$\mathsf{Imp}_{\mathbb{O}}(o_{\mathrm{I}})(\_,\_) =$	$\mathsf{Imp}_{\mathbb{O}}(o)(\_,m) =$
$r := \lambda$;	if $r = \lambda$ then
return λ	$r \leftarrow \mathcal{U}(l)$;
	endif
	return r

$\mathbb{O}'$:

$\mathsf{Imp}_{\mathbb{O}'}(o_{\mathrm{I}})(\_,\_) =$	$\mathsf{Imp}_{\mathbb{O}'}(o)(\_,m) =$
$r' \leftarrow \mathcal{U}(l)$;	$r := r'$;
$r := \lambda$;	return r'
return λ	

We choose distribution γ mapping $\bar{m}_{\mathbb{O}}$ to $\bar{m}''_{\mathbb{O}}$ with probability 1, states of the form $[r \mapsto \lambda]$ to $[r' \mapsto r'_0]$ with probability $\frac{1}{2^l}$, and states of the form $[r \mapsto r_0]$ to $[r' \mapsto r_0]$ with probability 1.

Let us justify that $\mathbb{O} \leq_{\mathrm{det},\gamma} \mathbb{O}'$ by checking that the equation on probabilities holds. For the initialization oracle, if we write the only possibility for left and right terms to be non-null, we obtain:

$\Pr[\gamma([r \mapsto \lambda]) = [r' \mapsto r'_0]]\ \Pr[\mathsf{Imp}_{\mathbb{O}}(o_{\mathrm{I}})(\_, \bar{m}_{\mathbb{O}}) = (\lambda, [r \mapsto \lambda])] =$
$\Pr[\gamma(\bar{m}_{\mathbb{O}}) = \bar{m}''_{\mathbb{O}}] \Pr[\mathsf{Imp}_{\mathbb{O}'}(o_{\mathrm{I}})(\_, (\bar{m}_{\mathbb{O}}, \bar{m}''_{\mathbb{O}})) = (\lambda, ([r \mapsto \lambda], [r' \mapsto r'_0]))]$

According to the definition of γ, the left term is worth $1/2^l$ and so is the right term.

As for the other oracle, we have to verify that for any r_0, r'_0 and a_0 we have:

$\Pr[\gamma([r \mapsto r_0]) = [r' \mapsto r'_0]]\ \Pr[\mathsf{Imp}_{\mathbb{O}}(o)(\_, [r \mapsto r_1]) = (a_0, [r \mapsto r_0])] =$
$\sum_{r''_0} \Pr[\gamma([r \mapsto r_1]) = [r' \mapsto r''_0]].$
$\Pr[\mathsf{Imp}_{\mathbb{O}'}(o)(\_, ([r \mapsto r_1], [r' \mapsto r''_0])) = (a_0([r \mapsto r_0], [r' \mapsto r'_0]))]$

where r_1 can be either λ or a value of positive length.

In any case, the only term in the sum is for $r''_0 = r'_0$ (otherwise the second probability is null). We now verify the equality when r_1 has null length first, and then when it is of length l. For $r_1 = \lambda$ the previous equation yields:

$\Pr[\gamma([r \mapsto r_0]) = [r' \mapsto r'_0]]\ \Pr[\mathsf{Imp}_{\mathbb{O}}(o)(\_, [r \mapsto \lambda]) = (a_0, [r \mapsto r_0])] =$
$\Pr[\gamma([r \mapsto \lambda]) = [r' \mapsto r'_0]].$
$\Pr[\mathsf{Imp}_{\mathbb{O}'}(o)(\_, ([r \mapsto \lambda], [r' \mapsto r'_0])) = (a_0([r \mapsto r_0], [r' \mapsto r'_0]))]$

Now the left term is non-null iff $r_0 = r'_0$ for the first probability and $r_0 = a_0$ for the second one. The right term requires the same conditions for the second probability to be positive. After simplification, we again obtain $1/2^l$ on both sides.

Finally, when r_1 is of positive length, we have to check

$\Pr[\gamma([r \mapsto r_0]) = [r' \mapsto r'_0]]\ \Pr[\mathsf{Imp}_{\mathbb{O}}(o)(\_, [r \mapsto r_1]) = (a_0, [r \mapsto r_0])] =$
$\Pr[\gamma([r \mapsto r_1]) = [r' \mapsto r'_0]].$
$\Pr[\mathsf{Imp}_{\mathbb{O}'}(o)(\_, ([r \mapsto r_1], [r' \mapsto r'_0])) = (a_0, ([r \mapsto r_0], [r' \mapsto r'_0]))]$

Left term still requires $r_0 = r'_0$ and $r_0 = a_0$ to be non-null, while the right term is positive iff $r_1 = r'_0$ and $a_0 = r'_0 = r_0$. In conclusion, the terms are either both null or both worth 1.

$\Diamond$

III.5.2 — Rules Using Determinization

Before stating rules, we proceed to the proof of a lemma formally linking probabilities of partial executions in both systems. We define a projection function π from $\mathbb{A} \mid \mathbb{O}'$-partial

executions to $\mathbb{A} \mid \mathbb{O}$-partial executions by extending the projection from $\mathsf{M}_{\mathbb{O}} \times \mathsf{M}''_{\mathbb{O}}$ to $\mathsf{M}_{\mathbb{O}}$ to executions.

Informally, we can foresee that if we consider a partial execution η in $\mathbb{A} \mid \mathbb{O}$ finishing with state m, we have to gather in a set all partial executions in $\mathbb{A} \mid \mathbb{O}'$ finishing with state (m, m'') for a given m'' and projecting to η. Then, from the equation imposed for one query by the definition of determinization, we can extrapolate that the set of $\mathbb{A} \mid \mathbb{O}'$-partial executions weighs the same probability as trace η multiplied by the probability that m is mapped to m'' by γ. This is proven by the lemma below.

Lemma III.14. *Let $\mathbb{O}$ and $\mathbb{O}'$ be such that $\mathbb{O} \leq_{\mathrm{det},\gamma} \mathbb{O}'$, and let η be a partial $\mathbb{O}$-execution: $\eta = (\bar{m}_{\mathbb{O}}, \bar{m}_{\mathbb{A}}) \xrightarrow{x_1} \ldots \xrightarrow{(o,q,a)} (m, m_a)$. For every $\mathbb{O}$-adversary $\mathbb{A}$ and every $m'' \in \mathsf{M}''_{\mathbb{O}}$:*

$$Pr(\mathbb{A}|\mathbb{O} : \eta) Pr[\gamma(m) = m''] = \sum_{\substack{\eta' | \pi(\eta') = \eta \\ Last(\eta') = ((m,m''),m_a)}} Pr(\mathbb{A}|\mathbb{O}' : \eta')$$

where τ' is any partial $\mathbb{O}'$-execution.

Proof. The proof is by induction on the length of partial executions.

If η is a step σ, then we can write $\sigma = (\bar{m}_{\mathbb{O}}, \bar{m}_{\mathbb{A}}) \xrightarrow{(o_{\mathrm{I}},q,a)} (m, m_a)$. Let σ' be a candidate in the sum. Then there exists m''_1 such that $\sigma' = ((\bar{m}_{\mathbb{O}}, m''_1), \bar{m}_{\mathbb{A}}) \xrightarrow{(o_{\mathrm{I}},q,a)} ((m, m''), m_a)$. The definition of determinization provides:

$$\Pr[\mathbb{A} \mid \mathbb{O} : \sigma] \;\; \Pr[\gamma(m) = m''] = \sum_{m''_1 \in \mathsf{M}''_{\mathbb{O}}} \Pr[\gamma(\bar{m}_{\mathbb{O}}) = m''_1] \; p_2(m''_1)$$

where $p_2(m''_1) = \Pr[\mathbb{A} \mid \mathbb{O}' : \sigma']$. Since $\gamma(\bar{m}_{\mathbb{O}}) = \delta_{\bar{m}''_{\mathbb{O}}}$, there is just one term in the sum, with weight 1. It yields our result:

$$\Pr[\mathbb{A} \mid \mathbb{O} : \sigma] \;\; \Pr[\gamma(m) = m''] = \Pr[((\bar{m}_{\mathbb{O}}, \bar{m}''_{\mathbb{O}}), \bar{m}_{\mathbb{A}}) \xrightarrow{(o_{\mathrm{I}},\_,a)} ((m, m''), m_a)]$$

We now continue with the induction step. We consider an execution $\eta \cdot \sigma$, which we decompose as follows: $\eta = (\bar{m}_{\mathbb{O}}, \bar{m}_{\mathbb{A}}) \longrightarrow \ldots \longrightarrow (\mu, \mu_a)$ and $(\mu, \mu_a) \xrightarrow{(o,q,a)} (m, m_a)$.

$$\begin{aligned}
& \Pr[\mathbb{A}|\mathbb{O} : \eta \cdot \sigma] \Pr[\gamma(m) = m''] \\
= \;& \Pr[\mathbb{A}|\mathbb{O} : \eta] \Pr[\mathbb{A}|\mathbb{O} : \sigma] \Pr[\gamma(m) = m''] \\
= \;& \Pr[\mathbb{A}|\mathbb{O} : \eta] \textstyle\sum_{m''_1} \Pr[\gamma(\mu) = m''_1] \Pr[\mathbb{A}|\mathbb{O}' : ((\mu, m''_1), \mu_a) \xrightarrow{(o,q,a)} ((m, m''), m_a)] \\
& \text{by applying determinization definition to last terms} \\
= \;& \textstyle\sum_{m''_1} (\Pr[\gamma(\mu) = m''_1] \Pr[\mathbb{A}|\mathbb{O} : \eta]). \Pr[\mathbb{A}|\mathbb{O}' : ((\mu, m''_1), \mu_a) \xrightarrow{(o,q,a)} ((m, m''), m_a)] \\
& \text{reorganizing terms} \\
= \;& \textstyle\sum_{m''_1} \sum_{\substack{\eta' | \pi(\eta') = \eta \\ Last(\eta') = ((\mu,m''_1),\mu_a)}} \Pr[\mathbb{A}|\mathbb{O}' : \eta'] \Pr[\mathbb{A}|\mathbb{O}' : ((\mu, m''_1), \mu_a) \xrightarrow{(o,q,a)} ((m, m''), m_a)] \\
& \text{using the induction hypothesis on } \tau \\
= \;& \textstyle\sum_{\substack{\eta' \cdot \sigma' \mid \pi(\eta' \cdot \sigma') = \eta \cdot \sigma, \\ Last(\eta' \cdot \sigma') = ((m,m''),m_a)}} \Pr[\mathbb{A} \mid \mathbb{O}' : \eta' \cdot \sigma']
\end{aligned}$$

and we can conclude.

■

We now present the set of rules formalizing the intuition provided by the previous lemma.

Given an event E defined on $\mathbb{O}$-traces, $E \circ \pi$ is the event on $\mathbb{O}'$-traces which is defined by $E \circ \pi(\tau') = \mathsf{true}$ iff $\pi(\tau')$ verifies E, where τ' is any $\mathbb{O}'$-trace.

Lemma III.15. *We consider two compatible oracle systems $\mathbb{O}$ and $\mathbb{O}'$. The following rules are sound:*

$$\frac{\mathbb{O} \leq_{\det,\gamma} \mathbb{O}' \quad \mathbb{O} :_\epsilon E}{\mathbb{O}' :_\epsilon E \circ \pi} \textit{ B-Det-Right}$$

$$\frac{\mathbb{O} \leq_{\det,\gamma} \mathbb{O}' \quad \mathbb{O}' :_\epsilon E \circ \pi}{\mathbb{O} :_\epsilon E} \textit{ B-Det-Left}$$

$$\frac{\mathbb{O} \leq_{\det,\gamma} \mathbb{O}'}{\mathbb{O} \sim_0 \mathbb{O}'} \textit{ I-Det}$$

Proof. Pretty much all the work has already been done by proving lemma III.14, which immediately results in $\Pr(\mathbb{A}|\mathbb{O} : E) = \Pr(\mathbb{A}|\mathbb{O}' : E \circ \pi)$ for every $\mathbb{O}$-event E and adversary $\mathbb{A}$. In turn, this equality results in all three rules cited above. ■

III.5.3 — Example of Use of Determinization

The leftover step of the proof of ROR-plaintext security of ElGamal cipher scheme is exactly the same as the simple determinization example developed after the definition. As a consequence, we choose to provide details for another example: we prove that we can draw results of random oracles in advance without it having any observable influence. A proof tree summarizing the ElGamal security proof can be found in VII.1.

We start by introducing a standard specification for random oracles.

Definition (Functional Random Oracle). We say that oracle $\mathcal{H}$ in $\mathbb{O}$ is a functional random oracle if oracle memories contain a list $L_\mathcal{H}$ taking values in $\mathsf{Xch}_\mathcal{H}$ initialized to the empty list by the initialization oracle, and the implementation of $\mathcal{H}$ is as follows:

$\mathsf{Imp}_\mathbb{O}(\mathcal{H})(x) =$ if $x \in \mathsf{dom}(L_\mathcal{H})$ then
 return $L_\mathcal{H}(x)$
else $h \leftarrow \mathcal{U}(n)$
 $L_\mathcal{H} := L_\mathcal{H} :: (x, h)$;
 return h
endif

where n is the length of outputs of $\mathcal{H}$. □

In our example, we consider two oracle systems. First, system $\mathbb{O}$ has two functional random oracles $\mathcal{H}$ and $\mathcal{G}$ (of respective output length n and l) in addition to its initialization and finalization oracles. This latter has input type $\mathsf{Res} = \mathsf{Bool}$ and does nothing.

Moreover, we define system $\mathbb{O}'$, compatible with $\mathbb{O}$, but with an additional list $L_{\mathcal{H}'}$ in the state, and in which oracles $\mathcal{H}$ and $\mathcal{G}$ are implemented by:

$\mathsf{Imp}_{\mathbb{O}'}(\mathcal{G})(x) =$ if $x \in \mathsf{dom}(L_{\mathcal{G}})$ then return $L_{\mathcal{G}}(x)$
else $g \leftarrow \mathcal{U}(l)$; $h \leftarrow \mathcal{U}(n)$;
$L_{\mathcal{G}} := L_{\mathcal{G}} :: (x, g)$;
$L_{\mathcal{H}'} := L_{\mathcal{H}'} :: (g, h)$;
return g
endif

$\mathsf{Imp}_{\mathbb{O}'}(\mathcal{H})(x) =$ if $x \in \mathsf{dom}(L_{\mathcal{H}})$ then return $L_{\mathcal{H}}(x)$
elsif $x \in \mathsf{dom}(L_{\mathcal{H}'})$ then
$L_{\mathcal{H}} := L_{\mathcal{H}} :: (x, L_{\mathcal{H}'}(x))$;
return $L_{\mathcal{H}'}(x)$
else $h \leftarrow \mathcal{U}(n)$;
$L_{\mathcal{H}} := L_{\mathcal{H}} :: (x, h)$;
return h
endif

System $\mathbb{O}$ is determinizing $\mathbb{O}'$ with distribution γ mapping $(L_{\mathcal{H}}, L_{\mathcal{G}})$ to $(L_{\mathcal{H}}, L_{\mathcal{G}}, L_{\mathcal{H}'})$ where $L_{\mathcal{H}'}$ has for domain the range of list $L_{\mathcal{G}}$, and for $x \in \mathsf{dom}(L_{\mathcal{H}'})$, if $x \in \mathsf{dom}(L_{\mathcal{H}})$ and images coincide (i.e. $L_{\mathcal{H}}(x) = L_{\mathcal{H}'}(x)$), otherwise images are distributed uniformly at random.

We can apply rule $I - Det$ to deduce that systems yield identical distributions:

$$\frac{\mathbb{O} \leq_{\mathrm{det},\gamma} \mathbb{O}'}{\mathbb{O} \sim_0 \mathbb{O}'} \text{ I-Det}$$

III.6 Backward Bisimulation up to Relations

The previously introduced notions of forward bisimulation up to and determinization are powerful concepts which allow us to capture a lot of reasonings. However, we meet a limitation of these tools when we try to capture arguments meant to tamper with values computed in past steps instead of changing the way we compute values in the current step or in future steps. To the cryptographer, it might evocate non-interference properties - the idea that no matter what the high values are worth, it is transparent to the adversary. To a concurrent system analysis specialist, it can bring up the typical fall of forward bisimulation, where reasoning is performed on the state in which we end up, but is useless to reason on the state from where we come. In [Buc99], Buchholz addresses this issue on stochastic automata by formalizing and studying exact performance equivalence. This latter notion has inspired the definition which we propose for backwards bisimulation up to. Our formal definition mimics the way in which we formalized forward bisimulation: we introduce a notion of stability and compatibility, but impose hypotheses on classes of states from which we start the exchange with an oracle.

Let us consider two compatible oracle systems $\mathbb{O}$ and $\mathbb{O}'$. For every oracle name, we let $\mathsf{M}_{\mathbb{O}+\mathbb{O}'}$ be $\mathsf{M}_{\mathbb{O}} + \mathsf{M}_{\mathbb{O}'}$ and for every $o \in \mathsf{N}_{\mathbb{O}}$, we let $\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)$ be the disjoint sum of $\mathsf{Imp}_{\mathbb{O}}(o)$ and $\mathsf{Imp}_{\mathbb{O}'}(o)$.

Definition (Backwards Bisimulation Up to Relation for Oracle Systems). Let $R \subseteq \mathsf{M}_{\mathbb{O}+\mathbb{O}'} \times \mathsf{M}_{\mathbb{O}+\mathbb{O}'}$ be an equivalence relation and ϕ be a predicate. $\mathbb{O}$ and $\mathbb{O}'$ are in backwards bisimulation with R up to ϕ iff the initial states are alone in their equivalence class and for all $m_1 \xrightarrow{(o,q,a)}_{>0} m_2$ and m_2' such that $m_2 \, R \, m_2'$ we have:

— stability on an equivalence class: for all $m_1' \in \mathsf{M}_{\mathbb{O}+\mathbb{O}'}$ such that $m_1' \xrightarrow{(o,q,a)}_{>0} m_2'$ and $m_1' \; R \; m_1$,

$$\phi((o,q,a), m_1, m_2) \Leftrightarrow \phi((o,q,a), m_1', m_2')$$

— backwards compatibility: if $\phi((o,q,a), m_1, m_2)$

$$\Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, \mathcal{C}(m_1)) = (a, m_2)] = \Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, \mathcal{C}(m_1)) = (a, m_2')]$$

where $\mathcal{C}(m_1)$ is the equivalence class of m_1 under R, and
$\Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, \mathcal{C}(m_1)) = (a, m_2)] = \sum_{m_1' \, R \, m_1} \Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, m_1') = (a, m_2)]$.

We write $\mathbb{O} \equiv^{b}_{R,\phi} \mathbb{O}'$. □

We define a projection $\mathcal{A}dv\mathcal{T}$ on partial executions which erases all oracle memories (only exchanges and adversarial memories are left). It defines the set of *partial adversarial traces*, for which we often use meta-variable α.

In the remaining of this subsection, we consider given systems $\mathbb{O}$ and $\mathbb{O}'$ and a backwards bisimulation up to ϕ relation, all of which satisfy the definition above. The fundamental property of backwards bisimulation is captured by the following lemma. It mostly states that the probability that a partial execution ends up in states (m, m_a) is constant on equivalence classes: it does not depend on the actual class representative m.

Lemma III.16. *Let α be a partial adversarial trace of length k $\alpha = m_a^0 \xrightarrow{x_1} m_a^1 \xrightarrow{x_2} \ldots \xrightarrow{x_k} m_a^k$. Then, for all exchange x_{k+1}, all adversary memory m_a^{k+1} and for all $m_{k+1}, m_{k+1}' \in \mathsf{M}_{\mathbb{O}}$ such that $m_{k+1} \; R \; m_{k+1}'$:*

$$\sum_{\substack{\eta \in \mathsf{PExec}(\mathbb{A}|\mathbb{O}) \;|\; \mathcal{A}dv\mathcal{T}(\eta) = \alpha \\ \mathsf{G}_\phi(\mathcal{T}(\eta \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})))}} \Pr[\mathbb{A} \mid \mathbb{O} : \eta \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})]$$
$$= \sum_{\substack{\eta \in \mathsf{PExec}(\mathbb{A}|\mathbb{O}) \;|\; \mathcal{A}dv\mathcal{T}(\eta) = \alpha \\ \mathsf{G}_\phi(\mathcal{T}(\eta \xrightarrow{x_{k+1}} (m_{k+1}', m_a^{k+1})))}} \Pr[\mathbb{A} \mid \mathbb{O} : \eta \xrightarrow{x_{k+1}} (m_{k+1}', m_a^{k+1})]$$

And if $m_{k+1} \in \mathsf{M}_{\mathbb{O}}, m_{k+1}' \in \mathsf{M}_{\mathbb{O}'}$ such that $m_{k+1} \; R \; m_{k+1}'$:

$$\sum_{\substack{\eta \in \mathsf{PExec}(\mathbb{A}|\mathbb{O}) \;|\; \mathcal{A}dv\mathcal{T}(\eta) = \alpha \\ \mathsf{G}_\phi(\mathcal{T}(\eta \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})))}} \Pr[\mathbb{A} \mid \mathbb{O} : \eta \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})]$$
$$= \sum_{\substack{\eta \in \mathsf{PExec}(\mathbb{A}|\mathbb{O}') \;|\; \mathcal{A}dv\mathcal{T}(\eta) = \alpha \\ \mathsf{G}_\phi(\mathcal{T}(\eta \xrightarrow{x_{k+1}} (m_{k+1}', m_a^{k+1})))}} \Pr[\mathbb{A} \mid \mathbb{O}' : \eta \xrightarrow{x_{k+1}} (m_{k+1}', m_a^{k+1})]$$

Proof. Proof by induction of the first equality.

Initialization (k=0):
Only partial executions starting in $(\bar{m}, \bar{m}_{\mathbb{A}})$ have a positive probability of occurrence. Hence, we look at adversarial partial traces of the form $\bar{m}_{\mathbb{A}} \xrightarrow{x_1} m_a^1$, for an exchange x_1. Given two memories m_1 and m_1' in relation (i.e. $m_1 \; R \; m_1'$) we have to prove that if $\phi(x_1, \bar{m}, m_1)$:

$\Pr[\mathbb{A} \mid \mathbb{O} : (\mathcal{C}(\bar{m}), \bar{m}_{\mathbb{A}}) \xrightarrow{x_1} (m_1, m_a^1)] = \Pr[\mathbb{A} \mid \mathbb{O} : (\mathcal{C}(\bar{m}), \bar{m}_{\mathbb{A}}) \xrightarrow{x_1} (m_1', m_a^1)]$

which is true by compatibility.

Let us proceed with the induction step:
Let $m_{k+1} \; R \; m_{k+1}'$. We let $Set(\mathbb{O})$ be the set of partial traces η and state m_k such that

trace $\eta \xrightarrow{x_k} (m_k, m_a^k) \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1}) \in \mathsf{PExec}(\mathbb{A} \mid \mathbb{O}), \mathcal{A}dv\mathcal{T}(\eta \xrightarrow{x_k} (m_k, m_a^k)) = \alpha$ and $\mathsf{G}_\phi(\mathcal{T}(\eta \xrightarrow{x_k} (m_k, m_a^k) \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})))$.

We have:

$$\sum_{\eta, m_k \in Set(\mathbb{O})} \Pr[\mathbb{A} \mid \mathbb{O} : \eta \xrightarrow{x_k} (m_k, m_a^k) \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})]$$
$$= \sum_{\eta, m_k \in Set(\mathbb{O})} \Pr[\mathbb{A} \mid \mathbb{O} : \eta \xrightarrow{x_k} (m_k, m_a^k)] \Pr[\mathbb{A} \mid \mathbb{O} : (m_k, m_a^k) \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})]$$

According to the induction hypothesis, the first term is constant on equivalence classes. As a result, if we choose m_k^i to be a representative of the equivalence class of m_k, then we can write that:

$$= \sum_{\substack{m_k^i \in \mathsf{M}_\mathbb{O}/R \\ \eta \in Set(m_k^i, \mathbb{O})}} \Pr[\mathbb{A} \mid \mathbb{O} : \eta \xrightarrow{x_k} (m_k^i, m_a^k)] \Big(\sum_{\substack{m_k \mid m_k R m_k^i \\ \phi(x_{k+1}, m_k, m_{k+1})}} \Pr[\mathbb{A} \mid \mathbb{O} : (m_k, m_a^k) \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})] \Big)$$

where $\eta \in Set(m_k^i, \mathbb{O})$ iff $\eta \xrightarrow{x_k} (m_k^i, m_a^k) \in \mathsf{PExec}(\mathbb{A} \mid \mathbb{O}), \mathcal{A}dv\mathcal{T}(\eta \xrightarrow{x_k} (m_k^i, m_a^k)) = \alpha$ and $\mathsf{G}_\phi(\mathcal{T}(\eta \xrightarrow{x_k} (m_k^i, m_a^k))$.

Consequently, if we can replace our right term by the same term featuring m'_{k+1} in place of m_{k+1}, we can conclude. We thus examine the right term, for a given equivalence class, and apply compatibility:

$$\sum_{\substack{m_k \mid m_k R m_k^i \\ \phi(x_{k+1}, m_k, m_{k+1})}} \Pr[\mathbb{A} \mid \mathbb{O} : (m_k, m_a^k) \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})]$$
$$= \Pr[\mathbb{A} \mid \mathbb{O} : (\mathcal{C}(m_k^i), m_a^k) \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})]$$
$$= \Pr[\mathbb{A} \mid \mathbb{O} : (\mathcal{C}(m_k^i), m_a^k) \xrightarrow{x_{k+1}} (m'_{k+1}, m_a^{k+1})]$$
$$= \sum_{\substack{m_k \mid m_k R m_k^i \\ \phi(x_{k+1}, m'_k, m'_{k+1})}} \Pr[\mathbb{A} \mid \mathbb{O} : (m_k, m_a^k) \xrightarrow{x_{k+1}} (m'_{k+1}, m_a^{k+1})]$$

This allows to conclude.

As for the proof of the second equality, it is very similar. Indeed, the initialization step follows from the fact that the initial states are alone in their equivalence class.

The induction step is done by performing the same factorization on equivalence classes (using the lemma we have just proved), and if we let $\tilde{m}_k^i$ be a representative of the equivalence class of m_k^i in $\mathsf{M}_{\mathbb{O}'}$,

$$\sum_{\substack{m_k^i \in \mathsf{M}_\mathbb{O}/R \\ \eta \in Set(m_k^i, \mathbb{O})}} \Pr[\mathbb{A} \mid \mathbb{O} : \eta \xrightarrow{x_k} (m_k^i, m_a^k)] \Big(\sum_{\substack{m_k \mid m_k R m_k^i \\ \phi(x_{k+1}, m_k, m_{k+1})}} \Pr[\mathbb{A} \mid \mathbb{O} : (m_k, m_a^k) \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})] \Big)$$
$$= \sum_{\substack{\tilde{m}_k^i \in \mathsf{M}_{\mathbb{O}'}/R \\ \eta' \in Set(\tilde{m}_k^i, \mathbb{O}')}} \Pr[\mathbb{A} \mid \mathbb{O}' : \eta' \xrightarrow{x_k} (\tilde{m}_k^i, m_a^k)] \Big(\sum_{\substack{m'_k \mid m'_k R \tilde{m}_k^i \\ \phi(x_{k+1}, m'_k, m'_{k+1})}} \Pr[\mathbb{A} \mid \mathbb{O}' : (m'_k, m_a^k) \xrightarrow{x_{k+1}} (m'_{k+1}, m_a^{k+1})] \Big)$$

if we apply the induction hypothesis on the right term and compatibility on the left term. The conclusion follows.

■

Definition (Compatibility with Adversarial Projection). An event is said to be *compatible with adversarial projection* if it has a constant truth value on all partial executions projecting to the same adversarial partial trace. □

These are events which only depend on what the adversary actually sees. A typical such event is $\mathrm{R} = \mathsf{true}$ used in the indistinguishability advantage. We now show how the previous lemma transfers to systems in backwards bisimulation.

Lemma III.17 (Conservation of Probability of Compatible Events). *Let E be an event compatible with adversarial projection. Then,*

$$\Pr[\mathbb{A} \mid \mathbb{O} : E \wedge \mathsf{G}_\phi] = \Pr[\mathbb{A} \mid \mathbb{O}' : E \wedge \mathsf{G}_\phi]$$

As a corollary,

$$\Pr[\mathbb{A} \mid \mathbb{O} : E \,\mathsf{U}\, \neg\phi] = \Pr[\mathbb{A} \mid \mathbb{O}' : E \,\mathsf{U}\, \neg\phi]$$

Proof. We define the set AEx of adversarial partial traces α for which one of the partial executions projecting in α satisfy $E \wedge \mathsf{G}_\phi$. The fact that E is compatible with adversarial projection allows to say that for any $\alpha \in AEx$, all partial executions projecting on α satisfy $E \wedge \mathsf{G}_\phi$. We can thus use the equality we have just proven.

$$\begin{aligned}
\Pr[\mathbb{A} \mid \mathbb{O} : E \wedge \mathsf{G}_\phi] &= \sum_{\substack{\alpha \in AEx \\ \eta \in \mathsf{Exec}(\mathbb{A}|\mathbb{O}) \mid \mathcal{A}dv\mathcal{T}(\eta) = \alpha \\ E \wedge \mathsf{G}_\phi(\mathcal{T}(\eta))}} \Pr[\mathbb{A} \mid \mathbb{O} : \eta] \\
&= \sum_{\substack{\alpha \in AEx \\ \eta \in \mathsf{Exec}(\mathbb{A}|\mathbb{O}') \mid \mathcal{A}dv\mathcal{T}(\eta) = \alpha \\ E \wedge \mathsf{G}_\phi(\mathcal{T}(\eta))}} \Pr[\mathbb{A} \mid \mathbb{O}' : \eta] \\
&= \Pr[\mathbb{A} \mid \mathbb{O}' : E \wedge \mathsf{G}_\phi]
\end{aligned}$$

As for the corollary, it mostly follows from the previous equality. Indeed, if α, m' and m'_a are fixed, and if m denotes the oracle memory in which η ends, $\Pi_1(Last(\eta))$.

$$\begin{aligned}
&\sum_{\substack{\alpha \in AEx \\ \eta \in \mathsf{Exec}(\mathbb{A}|\mathbb{O}) \mid \mathcal{A}dv\mathcal{T}(\eta) = \alpha \\ \mathsf{G}_\phi(\mathcal{T}(\eta))}} \Pr[\mathbb{A} \mid \mathbb{O} : \eta \xrightarrow{x} (m', m'_a) \wedge \neg\phi(x, m, m')] \\
= \; 1 - &\sum_{\substack{\alpha \in AEx \\ \eta \in \mathsf{Exec}(\mathbb{A}|\mathbb{O}) \mid \mathcal{A}dv\mathcal{T}(\eta) = \alpha \\ \mathsf{G}_\phi(\mathcal{T}(\eta))}} \Pr[\mathbb{A} \mid \mathbb{O} : \eta \xrightarrow{x} (m', m'_a) \wedge \phi(x, m, m')] \\
= \; 1 - &\sum_{\substack{\alpha \in AEx \\ \eta \in \mathsf{Exec}(\mathbb{A}|\mathbb{O}') \mid \mathcal{A}dv\mathcal{T}(\eta) = \alpha \\ \mathsf{G}_\phi(\mathcal{T}(\eta))}} \Pr[\mathbb{A} \mid \mathbb{O}' : \eta \xrightarrow{x} (m', m'_a) \wedge \phi(x, m, m')] \\
= &\sum_{\substack{\alpha \in AEx \\ \eta \in \mathsf{Exec}(\mathbb{A}|\mathbb{O}') \mid \mathcal{A}dv\mathcal{T}(\eta) = \alpha \\ \mathsf{G}_\phi(\mathcal{T}(\eta))}} \Pr[\mathbb{A} \mid \mathbb{O}' : \eta \xrightarrow{x} (m', m'_a) \wedge \neg\phi(x, m, m')]
\end{aligned}$$

The corollary directly follows from this equality.

■

III.6.1 — Rules Using Backwards Bisimulation

The rules presented in this section mirror those which we have proven for forward bisimulation up to.

Lemma III.18. *Let E be an event compatible with projection on adversarial partial traces. The following rules are sound:*

$$\frac{\mathbb{O} :_\epsilon E \wedge \mathsf{G}_\phi \qquad \mathbb{O} \equiv^b_{R,\phi} \mathbb{O}'}{\mathbb{O}' :_\epsilon E \wedge \mathsf{G}_\phi}\; \textit{B-BackBis}\mathsf{G}$$

$$\frac{\mathbb{O} :_\epsilon E\,\mathsf{U}\,\neg\varphi \qquad \mathbb{O} \equiv^b_{R,\phi} \mathbb{O}'}{\mathbb{O}' :_\epsilon E\,\mathsf{U}\,\neg\varphi}\ \textit{B-BackBis}\,\mathsf{U}$$

$$\frac{\mathbb{O} :_\epsilon \mathsf{F}_{\neg\phi} \qquad \mathbb{O} \equiv^b_{R,\phi} \mathbb{O}'}{\mathbb{O} \sim_\epsilon \mathbb{O}'}\ \textit{I-BackBis}$$

Proof. These rules are consequences of the previous lemma and noticing that $\textsc{r} = \mathsf{true}$ is an event compatible with projection on adversarial executions. ■

In addition to the set of rules dealing with one kind of bisimulation up to at a time, we need a rule to compose them. Indeed, if we are to use them one after the other with the same condition ϕ, the rules that we have already proven compel us to bound twice the probability that ϕ happens, once in $\mathbb{O}$ and then in its bisimilar counterpart. But both these probabilities are equal and we would count twice the same bad simulation event, augmenting artificially by a factor of two the indistinguishability bound in our conclusion. To tackle this problem, we propose the following rule.

Lemma III.19. *The following rule is sound:*

$$\frac{\mathbb{O} :_\epsilon \mathsf{F}_{\neg\phi} \qquad \mathbb{O} \equiv^b_{R,\phi} \mathbb{O}'' \qquad \mathbb{O}'' \equiv_{R',\phi} \mathbb{O}'}{\mathbb{O} \sim_\epsilon \mathbb{O}'}\ \textit{I-2-Bis}$$

Proof. The key lies in using the same predicate ϕ. From previous rules which we have shown, we have $\Pr[\mathbb{A} \mid \mathbb{O} : \textsc{r} = \mathsf{true} \wedge \mathsf{G}_\phi] = \Pr[\mathbb{A} \mid \mathbb{O}'' : \textsc{r} = \mathsf{true} \wedge \mathsf{G}_\phi]$ and $\Pr[\mathbb{A} \mid \mathbb{O}' : \textsc{r} = \mathsf{true} \wedge \mathsf{G}_\phi] = \Pr[\mathbb{A} \mid \mathbb{O}'' : \textsc{r} = \mathsf{true} \wedge \mathsf{G}_\phi]$.

These equalities allow us to write:

$$\begin{aligned}
&\Pr[\mathbb{A} \mid \mathbb{O} : \textsc{r} = \mathsf{true}] - \Pr[\mathbb{A} \mid \mathbb{O}' : \textsc{r} = \mathsf{true}] \\
&= \Pr[\mathbb{A} \mid \mathbb{O} : \textsc{r} = \mathsf{true}] - \Pr[\mathbb{A} \mid \mathbb{O}'' : \textsc{r} = \mathsf{true}] \\
&\qquad -(\Pr[\mathbb{A} \mid \mathbb{O}' : \textsc{r} = \mathsf{true}] - \Pr[\mathbb{A} \mid \mathbb{O}'' : \textsc{r} = \mathsf{true}]) \\
&= \Pr[\mathbb{A} \mid \mathbb{O} : \textsc{r} = \mathsf{true} \wedge \mathsf{F}_{\neg\phi}] - \Pr[\mathbb{A} \mid \mathbb{O}'' : \textsc{r} = \mathsf{true} \wedge \mathsf{F}_{\neg\phi}] \\
&\qquad -(\Pr[\mathbb{A} \mid \mathbb{O}' : \textsc{r} = \mathsf{true} \wedge \mathsf{F}_{\neg\phi}] - \Pr[\mathbb{A} \mid \mathbb{O}'' : \textsc{r} = \mathsf{true} \wedge \mathsf{F}_{\neg\phi}]) \\
&= \Pr[\mathbb{A} \mid \mathbb{O} : \textsc{r} = \mathsf{true} \wedge \mathsf{F}_{\neg\phi}] - \Pr[\mathbb{A} \mid \mathbb{O}' : \textsc{r} = \mathsf{true} \wedge \mathsf{F}_{\neg\phi}]
\end{aligned}$$

Consequently,

$|\Pr[\mathbb{A} \mid \mathbb{O} : \textsc{r} = \mathsf{true}] - \Pr[\mathbb{A} \mid \mathbb{O}' : \textsc{r} = \mathsf{true}]| \leq max(\Pr[\mathbb{A} \mid \mathbb{O} : \mathsf{F}_{\neg\phi}], \Pr[\mathbb{A} \mid \mathbb{O}' : \mathsf{F}_{\neg\phi}])$

Moreover, we have $\Pr[\mathbb{A} \mid \mathbb{O} : \mathsf{F}_{\neg\phi}] = \Pr[\mathbb{A} \mid \mathbb{O}'' : \mathsf{F}_{\neg\phi}]$

and $\Pr[\mathbb{A} \mid \mathbb{O}' : \mathsf{F}_{\neg\phi}] = \Pr[\mathbb{A} \mid \mathbb{O}'' : \mathsf{F}_{\neg\phi}]$

We can conclude to $|\Pr[\mathbb{A} \mid \mathbb{O} : \textsc{r} = \mathsf{true}] - \Pr[\mathbb{A} \mid \mathbb{O}' : \textsc{r} = \mathsf{true}]| \leq \Pr[\mathbb{A} \mid \mathbb{O} : \mathsf{F}_{\neg\phi}]$. ■

An illustration of the use of backwards bisimulation up to is provided in the proof of the main theorem of chapter VI. Then, backwards bisimulation is crucial to carry out our proof.

Chapter IV

Examples of Proofs in CIL

In this chapter we develop two examples of proofs carried out in CIL: unforgeability of signature schemes Full-Domain Hash (FDH) and Probabilistic Signature Scheme (PSS). We start by explaining the formalization in CIL of the cryptographic hypotheses needed to carry out the proofs, before the presentation of the proofs themselves.

IV.1 Preliminaries

IV.1.1 — Guessing an Output of a Random Oracle

Let $ROM(H,k)$ be the oracle system consisting of the hash oracle $\mathcal{H}$ implemented as a functional random oracle with outputs in $\{0,1\}^k$. More precisely, the system has a list $L_{\mathcal{H}}$ for state and implementations of oracles defined by:

$\mathsf{Imp}(o_{\mathrm{I}})(x) =$ $L_{\mathcal{H}} := [\,]$;
return **1**

$\mathsf{Imp}(\mathcal{H})(x) =$ if $x \in \mathsf{dom}(L_{\mathcal{H}})$ then
  return $L_{\mathcal{H}}(x)$
else $y \leftarrow \mathcal{U}(k)$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x,y)$;
  return y
endif

$\mathsf{Imp}(o_{\mathrm{F}})(x) =$ match $\mathrm{R}_1 : \{0,1\}^* \mid \mathrm{R}_2 : \{0,1\}^k$ with x;
return $\mathsf{Imp}(\mathcal{H})(\mathrm{R}_1)$

We want to formalize in our logic that the probability that an adversary exhibits a pair $\mathrm{R}_1, \mathrm{R}_2$ such that $\mathcal{H}(\mathrm{R}_1) = \mathrm{R}_2$, without querying for $\mathcal{H}(\mathrm{R}_1)$, is bounded by $\frac{1}{2^k}$. The guessing event is expressed as:

$\mathsf{Guess} = \lambda((o,x,\_),m,m').$ match $\mathrm{R}_1 : \{0,1\}^* \mid \mathrm{R}_2 : \{0,1\}^k$ with x;
$o = o_{\mathrm{F}} \wedge m'.L_{\mathcal{H}}(\mathrm{R}_1) = \mathrm{R}_2 \wedge \mathrm{R}_1 \notin \mathsf{dom}(m.L_{\mathcal{H}})$

Moreover, since Guess can only be satisfied when querying o_{F}, applying rule $Fail$ allows to conclude that $ROM(H,k) :_{1/2^k} \mathsf{F}_{\mathsf{Guess}}$.

IV.1.2 — Formalization of One-Wayness

We define an oracle system $OW(f)$ to capture the game played by an adversary when trying to invert a one-way function. It comprises an initialization oracle that draws matching public and secret key, an oracle $Chall$ that outputs a randomly sampled challenge for the adversary, and a finalization oracle that the adversary calls with its candidate pre-image.

$\mathsf{Imp}_{OW(f)}(o_{\mathrm{I}})(x) =$ $(pk, sk) \leftarrow \mathcal{K}$; $y := \lambda$; return pk

$\mathsf{Imp}_{OW(f)}(Chall)(x) =$ if $y = \lambda$ then let $y \leftarrow \mathcal{U}(\eta)$ in ; endif return y

$\mathsf{Imp}_{OW(f)}(o_{\mathrm{F}})(x) =$ return $\mathbf{1}$

We define the successful inversion of f_{pk} by event $\mathsf{F}_{\mathsf{Invert}}$, with predicate Invert defined as:

$\mathsf{Invert} = \lambda((o, x, \_), m, m').\quad o = o_{\mathrm{F}} \wedge f_{pk}(x) = m.y$

We then can state $OW :_{OW(t)} \mathsf{F}_{\mathsf{Invert}}$.

IV.1.3 — Macro-rule Up-to-bad

We provide here a macro-rule proving useful in the proof of PSS. It is a generalization of the "fundamental lemma" of game-playing. The rule that we prove is the following one, where E and E' are events compatible with R:

$$\frac{\mathbb{O}' :_{\epsilon} E' \qquad \mathbb{O}' :_{\epsilon'} \mathsf{F}_{\neg\varphi} \qquad \mathbb{O}' \equiv_{R,\varphi} \mathbb{O} \qquad E\ R\ E'}{\mathbb{O} :_{\epsilon+\epsilon'} E}\ UpToBad$$

We present a proof tree to derive this rule in our proof system.

$$B-Bis\mathsf{U}\ \frac{\dfrac{UR\ \dfrac{\mathbb{O}' :_{\epsilon} E'}{\mathbb{O}' :_{\epsilon} E' \wedge \mathsf{G}_{\varphi}}}{B-Bis\mathsf{G}\ \dfrac{\mathbb{O}' :_{\epsilon} E' \wedge \mathsf{G}_{\varphi} \qquad \mathbb{O}' \equiv_{R,\varphi} \mathbb{O} \qquad E\ R\ E'}{\mathbb{O} :_{\epsilon} E \wedge \mathsf{G}_{\varphi}}} \qquad \dfrac{\dfrac{\mathbb{O}' :_{\epsilon'} \mathsf{F}_{\neg\varphi} \qquad \mathbb{O}' \equiv_{R,\varphi} \mathbb{O}}{\mathbb{O} :_{\epsilon'} \mathsf{F}_{\neg\varphi}}\ UR}{\mathbb{O} :_{\epsilon'} E \wedge \mathsf{F}_{\neg\varphi}}\ UR}{\mathbb{O} :_{\epsilon+\epsilon'} E}\ UR$$

IV.2 FDH

IV.2.1 — Description of the Scheme

We define an oracle system $\mathbb{O}$ corresponding to the signature scheme Full-Domain Hash (FDH) [BR96]. The signature algorithm consists in applying the inverse of a one-way permutation to the hash value of the message to be signed. We denote η the length of bitstrings in the hash and signature range, and $\mathcal{U}(\eta)$ the uniform distribution on bitstrings of this length. In our framework, the scheme translates in a system with four oracles named $o_{\mathrm{I}}, H, \mathcal{S}, o_{\mathrm{F}}$. States contain variables pk, sk for keys corresponding to the permutation f_{pk} and its inverse f_{sk} (which we more naturally denote f and f^{-1} in the sequel) and lists $L_{\mathcal{S}}$ and L_H to store

$\mathsf{Imp}_{\mathbf{FDH}}(o_{\mathrm{I}})(x) =$ $(pk, sk) \leftarrow \mathcal{K}$;
$L_{\mathcal{S}} := [\,]$;
$L_H := [\,]$;
return pk

$\mathsf{Imp}_{\mathbf{FDH}}(H)(x) =$ if $x \in \mathsf{dom}(L_H)$ then
return $L_H(x)$
else $y \leftarrow \mathcal{U}(\eta)$;
$L_H := L_H.(x, y)$;
return y
endif

$\mathsf{Imp}_{\mathbf{FDH}}(\mathcal{S})(z) =$ $y \leftarrow \mathsf{Imp}_{\mathbf{FDH}}(H)(z)$;
$t := f^{-1}(y)$;
$L_{\mathcal{S}} := L_{\mathcal{S}}.(z, t)$;
return t

$\mathsf{Imp}_{\mathbf{FDH}}(o_{\mathrm{F}})(x) =$ match $\mathrm{R}_1 : \{0,1\}^* \mid \mathrm{R}_2 : \{0,1\}^k$ with x;
$y \leftarrow \mathsf{Imp}_{\mathbf{FDH}}(H)(\mathrm{R}_1)$
return $\mathbf{1}$

Figure IV.1 – Implementations of Oracles in **FDH**

queries and answers to corresponding oracles. The implementations of the four oracles are given in figure IV.1.

This choice of finalization oracle follows from the fact that to verify that a signature (or a forgery) for R_1 is valid, one must draw a value for $H(\mathrm{R}_1)$ whenever it has not already been done. The event corresponding to forging a signature can be captured via the following event Forge, obtained by the conjunction of two events: VSig and Fresh. The first predicate captures the validity of the signature provided ("$m'.L_H(\mathrm{R}_1) = f(\mathrm{R}_2)$") while the second predicate captures freshness of the forgery.

$$\mathsf{VSig}\ (\mathrm{R}_1, \mathrm{R}_2) = \mathsf{F}_{\lambda((o,q,\_),m,m').\ o=o_{\mathrm{F}} \wedge q=\mathrm{R}_1 || \mathrm{R}_2 \wedge m'.L_H(\mathrm{R}_1)=f(\mathrm{R}_2)}$$
$$\mathsf{Fresh}(\mathrm{R}_1, \mathrm{R}_2) = \mathsf{G}_{\lambda((o,q,a),\_,\_).\ \neg(o=\mathcal{S} \wedge q=\mathrm{R}_1)}$$

We then define Forge as the event:

$$\exists \mathrm{R}_1, \mathrm{R}_2.\ \mathsf{VSig}\ (\mathrm{R}_1, \mathrm{R}_2) \wedge\ \mathsf{Fresh}(\mathrm{R}_1, \mathrm{R}_2)$$

Theorem IV.1. *The FDH signature scheme is* $(k(H).OW(t) + \frac{1}{2^\eta})$*-secure against existential forgery:*

$$\mathbf{FDH} :_{(k(H).OW(t)+2^{-\eta})} \mathsf{Forge}$$

Proof Overview. The complete outline of the proof is illustrated by the proof tree in figure IV.2. The proof starts by splitting the event in two cases, according to whether the hash value of the message R_1 used for the forgery has been asked by the adversary. Intuitively, if this hash value is not requested by the adversary at some point of the execution, then it looks random and is thus improbable to guess. This argument is captured in left branch of the tree. In case the adversary does query H of R_1, then we can build another adversary inverting the underlying one-way permutation f out of it. This is done in the tree at the right. We have two major changes to perform: first, we have to remove occurences of f^{-1} from the

implementations, and then we must replace the answer to the hash query corresponding to the forged signature by the one-way challenge.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{ROM(\mathcal{H},\eta) :_{2^{-\eta}} \mathsf{Guess}} \quad \mathsf{Guess} \Rightarrow (\mathsf{Forge} \wedge \mathsf{G}_{\neg\mathsf{Asked}(\mathrm{R}_1)}) \circ \mathbb{C}}{ROM(\mathcal{H},\eta) :_{2^{-\eta}} (\mathsf{Forge} \wedge \mathsf{G}_{\neg\mathsf{Asked}(\mathrm{R}_1)}) \circ \mathbb{C}}\, Fail\text{-}UR}{\mathbf{FDH} = \mathbb{C}[ROM(\mathcal{H},\eta)] :_{2^{-\eta}} \mathsf{Forge} \wedge \mathsf{G}_{\neg\mathsf{Asked}(\mathrm{R}_1)}}\, B-Sub\text{-}UR \quad \mathbf{FDH} :_{k(H).OW(t)} \mathsf{Forge} \wedge \mathsf{F}_{\mathsf{Asked}(\mathrm{R}_1)}\ \text{Right tree}}{\mathbf{FDH} :_{k(H).OW(t)+2^{-\eta}} \mathsf{Forge}}$$

Right tree (1):

$$\dfrac{\dfrac{(*) \quad \dfrac{\text{Tree (2)}}{(\forall i,\ \mathbf{FDH}_2 :_{OW(t)} \mathsf{F}_{\mathsf{vsig}[i]} \wedge \mathsf{G}_{\phi_s \wedge \phi_h})}}{\mathbf{FDH}_2 :_{k(H).OW(t)} \mathsf{Forge} \wedge \mathsf{F}_{\mathsf{Asked}(\mathrm{R}_1)}(\wedge \mathsf{G}_{\mathsf{true}})}\,\mathrm{UR} \quad \mathbf{FDH}_1 \equiv_{R,\mathsf{true}} \mathbf{FDH}_2}{\mathbf{FDH}_1 :_{k(H).OW(t)} \mathsf{Forge} \wedge \mathsf{F}_{\mathsf{Asked}(\mathrm{R}_1)}(\wedge \mathsf{G}_{\mathsf{true}})}\, B-Bis\mathsf{G} \quad \mathbf{FDH} \leq_{det,\gamma} \mathbf{FDH}_1}{\mathbf{FDH} :_{k(H).OW(t)} \mathsf{Forge} \wedge \mathsf{F}_{\mathsf{Asked}(\mathrm{R}_1)}}\, B-Det-Right$$

where $(*)$ is $\bigvee_{i=1..k(H)}(\mathsf{F}_{\mathsf{vsig}[i]} \wedge \mathsf{G}_{\phi_s \wedge \phi_h}) \Rightarrow \mathsf{Forge} \wedge \mathsf{F}_{\mathsf{Asked}(\mathrm{R}_1)}$
Tree (2):

$$\dfrac{\dfrac{\dfrac{OW(f) :_{OW(t)} \mathsf{Invert} \quad (\mathsf{F}_{\mathsf{vsig}[i]} \wedge \mathsf{G}_{\phi_s \wedge \phi_h}) \circ \mathbb{C}' \Rightarrow \mathsf{Invert}}{OW(f) :_{OW(t)} (\mathsf{F}_{\mathsf{vsig}[i]} \wedge \mathsf{G}_{\phi_s \wedge \phi_h}) \circ \mathbb{C}'}\, UR}{\mathbf{FDH}_3 = \mathbb{C}'[OW(f)] :_{OW(t)} \mathsf{F}_{\mathsf{vsig}[i]} \wedge \mathsf{G}_{\phi_s \wedge \phi_h}}\, B-Sub \quad \mathbf{FDH}_3 \equiv_{R',\phi_h \wedge \phi_s} \mathbf{FDH}_2}{\mathbf{FDH}_2 :_{OW(t)} \mathsf{F}_{\mathsf{vsig}[i]} \wedge \mathsf{G}_{\phi_s \wedge \phi_h}}\, B-Bis\mathsf{G}$$

Figure IV.2 – Proof Tree For FDH

IV.2.2 — Details of the Proof

We let $\mathsf{Asked}(\mathrm{R}_1)$ denote step-predicate $\lambda((o,q,a),\_,\_).\,(o = H \wedge q = \mathrm{R}_1)$. The proof naturally starts with a case split between $\mathsf{Forge} \wedge \mathsf{G}_{\neg\mathsf{Asked}(\mathrm{R}_1)}$ and $\mathsf{Forge} \wedge \mathsf{F}_{\mathsf{Asked}(\mathrm{R}_1)}$.

When $H(\mathbf{r}_1)$ has not been queried. In this left part of the tree, the idea is to reduce success in realizing event $\mathsf{Forge} \wedge \mathsf{G}_{\neg\mathsf{Asked}(\mathrm{R}_1)}$ into success in guessing a hash value, namely realizing event Guess when interacting system $ROM(\mathcal{H},\eta)$. We can write **FDH** as a context $\mathbb{C}$ of ROM, by choosing to decompose states in $(pk, sk, L_{\mathcal{S}}) \in \mathsf{M}_{\mathbb{C}}$ and $L_H \in \mathsf{M}_{ROM}$ and choosing for applications:

$$
\begin{array}{ll}
\mathsf{C}^{\rightarrow}_{c_{\mathrm{I}}}(x): & \mathsf{return}\ (o_{\mathrm{I}}, \mathbf{1}) \\
\mathsf{C}^{\leftarrow}_{c_{\mathrm{I}}}(x,(o,q,a)): & (pk, sk) \leftarrow \mathcal{K}; \\
 & L_{\mathcal{S}} := [\,]; \\
 & \mathsf{return}\ pk \\
\mathsf{C}^{\rightarrow}_{H}(z): & \mathsf{return}\ (\mathcal{H}, z) \\
\mathsf{C}^{\leftarrow}_{H}(z,(o,q,a)): & \mathsf{return}\ a \\
\mathsf{C}^{\rightarrow}_{\mathcal{S}}(z): & \mathsf{return}\ (\mathcal{H}, z) \\
\mathsf{C}^{\leftarrow}_{\mathcal{S}}(z,(o,q,a)): & t := f^{-1}(y); \\
 & L_{\mathcal{S}} := L_{\mathcal{S}}.(z,t); \\
 & \mathsf{return}\ t \\
\mathsf{C}^{\rightarrow}_{c_{\mathrm{F}}}(x): & \mathsf{match}\ \mathrm{R}_1 : \{0,1\}^* \mid \mathrm{R}_2 : \{0,1\}^{\eta}\ \mathsf{with}\ x; \\
 & \mathsf{return}\ (o_{\mathrm{F}}, \mathrm{R}_1 || f(\mathrm{R}_2))
\end{array}
$$

We then apply rule $B - Sub$. The idea is, when interacting with the composed system, if the forgery is successful and R_1 has not been signed nor hashed, then $H(\mathrm{R}_1)$ is equal to $f(\mathrm{R}_2)$ and has been successfully guessed by the adversary. Formally, the composition $\mathsf{F}_{\mathsf{Forge}} \circ \mathbb{C}$ implies $\mathsf{F}_{\mathsf{Guess}}$.

When $H(\mathbf{r}_1)$ has been queried. To eliminate occurences of f^{-1} in the implementations of oracles, we proceed in two steps. First, we transform the hash oracle H so that it anticipates signatures $f^{-1}(H(x))$ matching the hash queries x it receives. Then, we change the method of computation of these values: we draw y and update our memory with $(f(y), y)$ rather than drawing $H(x)$ and updating our memory with $(H(x), f^{-1}(H(x)))$.

Let us provide formal details for the step anticipating the values for signatures of messages at the moment of the corresponding hash query. We introduce system $\mathbf{FDH}_1$, with memories composed of variables pk, sk and lists L_H, $L_{\mathcal{S}}$ and $L^{ant}_{\mathcal{S}}$. Implementations of oracles H and $\mathcal{S}$ are precised in IV.3. The system **FDH** determinizes system $\mathbf{FDH}_1$ for distribution γ mapping pk, sk and lists L_H, $L_{\mathcal{S}}$ to a list L^{ant}_{sig}. As this last list of anticipated signatures is completely determined by L_H and $L_{\mathcal{S}}$, distribution γ is defined as a Dirac distribution providing the list L^{ant}_{sig} such that $\mathsf{dom}(L^{ant}_{Sig}) = \mathsf{dom}(L_H) - \mathsf{dom}(L_{\mathcal{S}})$ and $\forall x \in \mathsf{dom}(L^{ant}_{Sig}), L^{ant}_{Sig}(x) = f^{-1}(L_H(x))$.

After that, we consider the system $\mathbf{FDH}_2$ obtained from $\mathbf{FDH}_1$ by adding an integer variable j in its state space and using it as a counter for H queries. Formally, we change the implementation of two oracles of $\mathbf{FDH}_1$: the initialization oracle, which now initializes j to 0, and the hash oracle which is now implemented as follows:

$\mathsf{Imp}_{\mathbf{FDH}_1}(H)(x) =$ if $x \in \mathsf{dom}(L_H)$ then
return $L_H(x)$
else $y \leftarrow \mathcal{U}(\eta)$;
$L_H := L_H.(x, y)$;
$L_{\mathcal{S}}^{ant} := L_{\mathcal{S}}^{ant}.(x, f^{-1}(y))$;
return y
endif

$\mathsf{Imp}_{\mathbf{FDH}_1}(\mathcal{S})(x) =$ if $x \in \mathsf{dom}(L_{\mathcal{S}})$ then
return $L_{\mathcal{S}}(x)$
elsif $x \in \mathsf{dom}(L_H)$ then
$t := L_{\mathcal{S}}^{ant}(x)$;
$L_{\mathcal{S}}^{ant} := L_{\mathcal{S}}^{ant} - (x, t)$;
$L_{\mathcal{S}} := L_{\mathcal{S}}.(x, t)$;
return t
else $y \leftarrow \mathcal{U}(\eta)$;
$L_H := L_H.(x, y)$;
$L_{\mathcal{S}} := L_{\mathcal{S}}.(x, f^{-1}(y))$;
return $f^{-1}(y)$
endif

Figure IV.3 – Implementations of Oracles in $\mathbf{FDH}_1$

$\mathsf{Imp}_{\mathbf{FDH}_2}(H)(x) =$ if $x \in \mathsf{dom}(L_H)$ then
return $L_H(x)$
else $j := j + 1$;
$y \leftarrow \mathcal{U}(\eta)$;
$L_H := L_H.(x, y)$;
$L_{\mathcal{S}}^{ant} := L_{\mathcal{S}}^{ant}.(x, f^{-1}(y))$;
return y
endif

System $\mathbf{FDH}_2$ is in perfect bisimulation with $\mathbf{FDH}_1$, with the equality on the commmon components of their states as a relation $R : \mathbf{FDH}_1 \equiv_{R,\mathsf{true}} \mathbf{FDH}_2$. We now have to bound $\mathsf{Forge} \wedge \mathsf{F}_{\mathsf{Asked}(\mathrm{R}_1)}$ in system $\mathbf{FDH}_2$.

We then apply UR and decompose event $\mathsf{F}_{\mathsf{Asked}(\mathrm{R}_1)}$ according to the position i in which R_1 is queried to H. More precisely, we choose a formulation of the event suiting our need at the next step of the proof. Indeed, we define predicates ϕ_h and ϕ_s capturing respectively that the i-th query performed to oracle H has never been signed (this is ϕ_h) and that it is never signed afterwards (this is ϕ_s). Moreover, we introduce event $\mathsf{VSig}[i](\mathrm{R}_1, \mathrm{R}_2)$, a modified version of $\mathsf{VSig}(\mathrm{R}_1, \mathrm{R}_2)$ where R_1 is the i-th element of L_H. Formally, we write Forge as the disjunction $\bigvee_{i=1..k(H)}(\mathsf{F}_{\mathsf{vsig}[i]} \wedge \mathsf{G}_{\phi_s \wedge \phi_h})$ where:

$\mathsf{vsig}[i]((o, q, \_), m, m') = \quad o = o_{\mathrm{F}} \wedge q = \mathrm{R}_1 || \mathrm{R}_2 \wedge m'.L_H(\mathrm{R}_1) = f(\mathrm{R}_2) \wedge \mathrm{R}_1 = \mathsf{dom}(m.L_H)[i]$
$\phi_s((o, q, a), m, m') = \quad (o = \mathcal{S}) \wedge ((j \geq i) \Rightarrow q \neq \mathsf{dom}(m.L_H)[i])$
$\phi_h((o, q, a), m, m') = \quad (o = H) \wedge ((j = i - 1 \wedge q \notin m.L_H) \Rightarrow q \notin m.L_{\mathcal{S}})$

Our next step is a bisimulation-up-to argument. We now modify the system $\mathbf{FDH}_2$ to obtain a system $\mathbf{FDH}_3$, which is in bisimulation-up-to $\phi_s \wedge \phi_h$ for the relation R' consisting

$\mathsf{Imp}_{\mathbf{FDH}_3}(\mathcal{S})(x) =$
if $x \in \mathsf{dom}(L_{\mathcal{S}})$ then
  return $L_{\mathcal{S}}(x)$
elsif $x \in \mathsf{dom}(L_H)$ then
  $t := L_{\mathcal{S}}^{ant}(x)$;
  $L_{\mathcal{S}}^{ant} := L_{\mathcal{S}}^{ant} - (x, t)$;
  $L_{\mathcal{S}} := L_{\mathcal{S}}.(x, t)$;
  return t
else $y \leftarrow \mathcal{U}(\eta)$;
  $L_H := L_H.(x, f(y))$;
  $L_{\mathcal{S}} := L_{\mathcal{S}}.(x, y)$;
  return y
endif

$\mathsf{Imp}_{\mathbf{FDH}_3}(H)(x) =$
if $x \in \mathsf{dom}(L_H)$ then
  return $L_H(x)$
else $j := j + 1$;
  $y \leftarrow \mathcal{U}(\eta)$;
  if $j = i$ then
    $L_H := L_H.(x, y)$;
    $L_{\mathcal{S}}^{ant} := L_{\mathcal{S}}^{ant}.(x, \lambda)$;
    return y
  else
    $L_H := L_H.(x, f(y))$;
    $L_{\mathcal{S}}^{ant} := L_{\mathcal{S}}^{ant}.(x, y)$;
    return y
  endif
endif

Figure IV.4 – Implementations of Oracles in $\mathbf{FDH}_3$

in imposing equality of states but on the anticipated signature for the i-th query to H. Our event $\mathsf{F}_{\mathsf{Forge}[i]}$ is compatible with this relation because it does not depend on the value of the anticipated signature.

We can write this last system $\mathbf{FDH}_3$ as a context $\mathbb{C}'$ of system $OW(f)$ as follows:

$\mathsf{C}_{c_{\mathrm{I}}}^{\rightarrow}(x)$: return $(o_{\mathrm{I}}, \mathbf{1})$
$\mathsf{C}_{c_{\mathrm{I}}}^{\leftarrow}(x, (o, q, a))$: $L_H := [\,]$;
$L_{\mathcal{S}}^{ant} := [\,]$;
$L_{\mathcal{S}} := [\,]$;
return pk
$\mathsf{C}_H^{\rightarrow}(z)$: return $(Chall, z)$
$\mathsf{C}_H^{\leftarrow}(z, (o, q, a))$: if $x \in \mathsf{dom}(L_H)$ then
  return $L_H(x)$
else $j := j + 1$;
  if $j = i$ then
    $L_H := L_H.(x, z)$;
    $L_{\mathcal{S}}^{ant} := L_{\mathcal{S}}^{ant}.(x, \lambda)$;
    return z
  else
    $y \leftarrow \mathcal{U}(\eta)$;
    $L_H := L_H.(x, f(y))$;
    $L_{\mathcal{S}}^{ant} := L_{\mathcal{S}}^{ant}.(x, y)$;
    return y
  endif
endif
$\mathsf{C}_{c_{\mathrm{F}}}^{\rightarrow}(x)$: match $\mathrm{R}_1 : \{0,1\}^* \mid \mathrm{R}_2 : \{0,1\}^{\eta}$ with x;
return $(o_{\mathrm{F}}, \mathrm{R}_2)$

Finally, we notice that $(\mathsf{F}_{\mathsf{vsig}[i]} \wedge \mathsf{G}_{\phi_s \wedge \phi_h}) \circ \mathbb{C}'$ yields $\mathsf{F}_{\mathsf{Invert}}$, which allows us to conclude.

IV.3 PSS

The Probabilistic Signature Scheme [BR96] (PSS for short) is a generic signature scheme that transforms any one-way trapdoor permutation f into a secure probabilistic signature scheme, and has been adopted as part of the PKCS standard. PSS involves three hash functions $H : \{0,1\}^* \to \{0,1\}^{k_2}$, $F : \{0,1\}^{k_2} \to \{0,1\}^{k_0}$ and $G : \{0,1\}^{k_2} \to \{0,1\}^{k_1}$. These functions are modeled as functional random oracles. In addition, PSS involves a (public) one-way permutation f_{pk} and its (private) inverse f_{sk} on bitstrings of length k with $k = k_0 + k_1 + k_2$. Keys (pk, sk) are sampled and stored in matching variables by the initialization oracle, and we refer to f_{pk} as f (resp. to f_{sk} as f^{-1}) in the sequel. Let us describe the algorithms to sign a message and verify a signature:

— The probabilistic signature oracle computes the signature of a message msg in two steps: first, it samples uniformly a random value r in $\{0,1\}^{k_1}$; then, it computes $w_1 = H(msg||r)$, $w_2 = G(w_1) \oplus r$ and $w_3 = F(w_1)$, and returns $f^{-1}(w_1||w_2||w_3)$.

— The signature verification algorithm $\mathcal{V}$ takes as input a bitstring $bs \in \{0,1\}^k$ and a message $msg \in \{0,1\}^*$ and checks whether bs is a valid signature for msg. It proceeds in two steps: first, it computes $y = f(bs)$ and parses it as $w = w_1||w_2||w_3$ with $w_1 \in \{0,1\}^{k_2}$, $w_2 \in \{0,1\}^{k_1}$ and $w_3 \in \{0,1\}^{k_0}$; then it computes $r = w_2 \oplus G(w_1)$, and checks whether $w_1 = H(msg||r)$ and $w_3 = F(w_1)$.

Henceforth, for any bitstring $bs \in \{0,1\}^{k_2+k_1+k_0}$, we denote by $r(bs, m)$ the r-bitstring computed as in the verification algorithm using hash values stored in memory m.

Formally, PSS is modeled by the oracle system $\mathbf{PSS}_0$ with memories consisting in variables pk and sk and lists L_G, L_H and L_F. As an initial memory, one can choose any memory (the output of the initialization oracle does not depend on its input memory). System $\mathbf{PSS}_0$ comprises six oracles: $o_{\mathrm{I}}, F, G, H, \mathcal{S}$ and o_{F}. The implementations of F, G and H are those of functional random oracles. The implementation of the finalization oracle consists in computing the hash values involved in the verification of the validity of the signature of a given message. We emphasize that it *does not perform* this verification, which is only checked in the event translating a successful forgery. Implementations for initialization, finalization and signature oracles are provided in figure IV.5.

Forgery is modeled by the event Forge stating that the adversary has returned a pair $(\mathrm{R}_1, \mathrm{R}_2)$ that is a valid signature, and that has not been produced by the signing oracle:

$$\exists \mathrm{R}_1, \mathrm{R}_2.\ \mathsf{VSig}\ (\mathrm{R}_1, \mathrm{R}_2) \wedge\ \mathsf{Fresh}(\mathrm{R}_1, \mathrm{R}_2)$$

where $\mathsf{VSig}\ (\mathrm{R}_1, \mathrm{R}_2)$ and $\mathsf{Fresh}(\mathrm{R}_1, \mathrm{R}_2)$ are the following events:

$$\mathsf{VSig}\ (\mathrm{R}_1, \mathrm{R}_2) = \mathsf{F}_{\lambda((o,q,\_),m,m').\ o=o_{\mathrm{F}} \wedge q=\mathrm{R}_1||\mathrm{R}_2 \wedge \mathcal{V}(R_1,R_2,m')}$$
$$\mathsf{Fresh}(\mathrm{R}_1, \mathrm{R}_2) = \mathsf{G}_{\lambda((o,q,a),\_,\_).\ \neg(o=\mathcal{S} \wedge q=\mathrm{R}_1 \wedge a=\mathrm{R}_2)}$$

Theorem IV.2. *PSSis ϵ-secure w.r.t. existential forgery against chosen message attack, where $\epsilon(k,t) = \frac{1}{2^{k_2}} + (k(\mathcal{S}) + k(H))(\frac{k(\mathcal{S})}{2^{k_1}} + \frac{k(F)+k(G)+k(H)+k(\mathcal{S})}{2^{k_2}}) + OW(t)$:*

$$\mathbf{PSS}_0 :_{\epsilon} \mathsf{Forge}$$

Proof Overview. The adversary submits a candidate to forgery by querying the finalization oracle on a bitstring we denote $\mathrm{R}_1||\mathrm{R}_2$. The proof starts with a case analysis on whether

$$
\begin{array}{ll}
\mathsf{Imp}_{\mathbf{PSS}_0}(o_{\mathrm{I}})(x) = & (pk, sk) \leftarrow \mathcal{K} \\
& L_H := [\,]; \\
& L_G := [\,]; \\
& L_F := [\,]; \\
& \mathsf{return}\ pk \\
\mathsf{Imp}_{\mathbf{PSS}_0}(\mathcal{S})(x): & r \leftarrow \{0,1\}^{k_1}; \\
& w_1 \leftarrow \mathsf{Imp}_{\mathbf{PSS}_0}(H)(x||r); \\
& w_2 \leftarrow \mathsf{Imp}_{\mathbf{PSS}_0}(G)(w_1); \\
& w_3 \leftarrow \mathsf{Imp}_{\mathbf{PSS}_0}(F)(w_1); \\
& \mathsf{return}\ f^{-1}(w_1||(w_2 \oplus r)||w_3) \\
\mathsf{Imp}_{\mathbf{PSS}_0}(o_{\mathrm{F}})(x) = & \mathsf{match}\ \mathrm{R}_1 : \{0,1\}^* \mid \mathrm{R}_2 : \{0,1\}^{\eta}\ \mathsf{with}\ x; \\
& y := f(\mathrm{R}_2); \\
& \mathsf{match}\ w_1 : \{0,1\}^{k_2} \mid w_2 : \{0,1\}^{k_1} \mid w_3 : \{0,1\}^{k_0}\ \mathsf{with}\ y; \\
& r^* := w_2 \oplus \mathsf{Imp}_{\mathbf{PSS}_0}(G)(w_1); \\
& w_1^* := \mathsf{Imp}_{\mathbf{PSS}_0}(H)(\mathrm{R}_1||r^*); \\
& w_3^* := \mathsf{Imp}_{\mathbf{PSS}_0}(F); (w_1) \\
& \mathsf{return}\ \mathbf{1}
\end{array}
$$

Figure IV.5 – Implementations of Oracles in $\mathbf{PSS}_0$

the hash query to H necessary to construct the signature R_2 of R_1 has been performed by the adversary. If this hash query has not been performed, we use that the hash function is modeled as a random oracle to bound the probability of success of the adversary. This the right tree on figure IV.6. However, if the hash query is issued, then the idea is to replace the answer to hash queries by values related to a challenge to invert the underlying one-way function. As a result, a successful forgery yields a valid pre-image.

IV.3.1 — Details of the Proof

First Step. Formally, if m is the input memory of the finalization oracle and m' its output memory, then this corresponds to verifying whether $\mathrm{R}_1||r(\mathrm{R}_2, m')$ belongs to the list $m.L_H$. This yields two events Forge_1 and Forge_2 where Forge_i is defined as

$$\exists \mathrm{R}_1, \mathrm{R}_2.\ \mathsf{VSig}_i\ (\mathrm{R}_1, \mathrm{R}_2) \wedge\ \mathsf{Fresh}(\mathrm{R}_1, \mathrm{R}_2)$$

and where $\mathsf{VSig}_i\ (\mathrm{R}_1, \mathrm{R}_2)$ are the following events:

$$
\begin{array}{l}
\mathsf{VSig}_1\ (\mathrm{R}_1, \mathrm{R}_2) = \mathsf{F}_{\lambda((o,q,\_),m,m').\ o=o_{\mathrm{F}} \wedge q=\mathrm{R}_1|\mathrm{R}_2 \wedge \mathcal{V}(R_1,R_2,m') \wedge \mathrm{R}_1||r(\mathrm{R}_2,m') \notin m.L_H} \\
\mathsf{VSig}_2\ (\mathrm{R}_1, \mathrm{R}_2) = \mathsf{F}_{\lambda((o,q,\_),m,m').\ o=o_{\mathrm{F}} \wedge q=\mathrm{R}_1|\mathrm{R}_2 \wedge \mathcal{V}(R_1,R_2,m') \wedge \mathrm{R}_1||r(\mathrm{R}_2,m') \in m.L_H}
\end{array}
$$

The first step of the proof is to apply the union rule to perform the case split.

Right Tree. To apply $B - Sub$ we define a context $\mathbb{C}$ of system $ROM(\mathcal{H}, k_2)$, and see $\mathbf{PSS}_0$ as the composition of $\mathbb{C}$ and $ROM(\mathcal{H}, k_2)$. The idea is that the inner oracle $\mathcal{H}$ plays the role that H plays in $\mathbf{PSS}_0$.

A memory of $\mathbb{C}$ has the form (pk, sk, L_G, L_F). Its initial memory is the same as the projection on these variables of the initial memory of $\mathbf{PSS}_0$. The procedures of $\mathbb{C}$ are named c_{I}, c_{F}, F, G, H and $\mathcal{S}$. The implementations of the matching functions are provided in

$$UR\ \frac{\dfrac{\text{Left Tree}}{\mathbf{PSS}_0 :_{\epsilon_1+OW(t)} \mathsf{Forge}_2} \qquad \begin{array}{c} Fail\ \dfrac{}{ROM(\mathcal{H},k_2) :_{2^{-k_2}} \mathsf{F}_{\mathsf{Guess}}} \qquad \mathsf{Forge}_1 \circ \mathbb{C} \Rightarrow \mathsf{F}_{\mathsf{Guess}} \\ UR\ \dfrac{}{B-Sub\ \dfrac{ROM(\mathcal{H},k_2) :_{2^{-k_2}} \mathsf{Forge}_1 \circ \mathbb{C}}{\mathbf{PSS}_0 = \mathbb{C}[ROM(\mathcal{H},k_2)] :_{2^{-k_2}} \mathsf{Forge}_1}} \end{array}}{\mathbf{PSS}_0 :_{\epsilon} \mathsf{Forge}}$$

Left Tree:

$$UpToBad\ \frac{\mathbf{PSS}_1 \equiv_{R,\varphi} \mathbf{PSS}_2 \qquad \dfrac{}{\mathbf{PSS}_2 :_{\epsilon_1} \mathsf{F}_{\neg\varphi}}\ Fail \qquad \dfrac{OW :_{OW(t)} \mathsf{Invert}}{\mathbf{PSS}_2 :_{OW(t)} \mathsf{Forge}_2}\ B-Sub}{B-Det-Right\ \dfrac{\mathbf{PSS}_1 :_{\epsilon_1+OW(t)} \mathsf{Forge}_2}{\mathbf{PSS}_0 :_{\epsilon_1+OW(t)} \mathsf{Forge}_2}}$$

Figure IV.6 – Proof Tree of Scheme PSS

figure IV.7 (except for those associated to F, very similar to those of G where L_F takes place of L_G and k_0 takes place of k_1). Procedures F and G do not use any oracle of $ROM(\mathcal{H}, k_2)$; they are implemented with input procedures outputting queries to the dummy oracle. However, H forwards its queries to $\mathcal{H}$, and forwards back the answer it gets. Finally, on a query x, the signature oracle draws a seed r and forwards to $\mathcal{H}$ the query $(x||r)$, instead of computing $H(x||r)$. Afterwards, on input an answer a for $\mathcal{H}(x||r)$, $\mathsf{C}_{\mathcal{S}}^{\leftarrow}$ resumes the computation of the signature of x.

We know that $ROM(\mathcal{H}, k_2) :_{2^{-k_2}} \mathsf{F}_{\mathsf{Guess}}$. The event $\mathsf{F}_{\mathsf{Guess}}$ is captured by the issue to the finalization oracle of a query of the form $\mathrm{R}_1||\mathcal{H}(\mathrm{R}_1)$. We want $\mathsf{F}_{\mathsf{Guess}}$ to be realized when Forge_1 is. If Forge_1 is verified for $\mathrm{R}||\mathrm{R}'$, then a valid signature has been issued and hence, if we denote $f(\mathrm{R}') = w_1||w_2||w_3$ then we have $H(\mathrm{R}||r(\mathrm{R}||\mathrm{R}')) = w_1$. Consequently, the finalization procedure of the context does everything the finalization oracle of $\mathbf{PSS}_0$ does *but* computing the value of H on $x||r^*$. Then, it forwards $\mathrm{R}||r(\mathrm{R}||\mathrm{R}')$ as a guess to the internal finalization oracle of $ROM(\mathcal{H}, k_2)$ to realize $\mathsf{F}_{\mathsf{Guess}}$. As a result of this choice, we have $\mathsf{Forge}_1 \circ \mathbb{C} \Rightarrow \mathsf{F}_{\mathsf{Guess}}$.

Left Tree. The first step in this left tree is to use a determinization argument to allow the hash oracle H to anticipate values for the other hash functions: every time it is queried on some fresh value x and answers a value w_1, H also samples values for $F(w_1)$ and $G(w_1)$ *if they do not already exist*, which correspond to hash queries made to F and G during a signature computation.

The formalization of this proof step requires the definition of oracle system $\mathbf{PSS}_1$, which is determinized by $\mathbf{PSS}_0$. In addition to those present in a state of $\mathbf{PSS}_0$, states of $\mathbf{PSS}_1$ use two new variables L'_F and L'_G, which have the same type as L_F and L_G. The idea is to store the "pre-computed" hash values in L'_F and L'_G, and to transfer them from L'_F (resp. L'_G) to L_F (resp. L_G) whenever the values are directly requested to F and G respectively. We stress that this transfer only takes place if there is not already values to which F or G bounds the queries that we try to anticipate. The implementations of oracles of this system are provided in figure IV.8. Finally, the signature oracle has the same implementation as in the previous system except that it naturally calls oracles H, G and F of $\mathbf{PSS}_1$.

Let us define the distribution γ such that $\mathbf{PSS}_0 \leq_{det,\gamma} \mathbf{PSS}_1$. Given a memory of $\mathbf{PSS}_0$

$\mathsf{C}^{\rightarrow}_{c_{\mathrm{I}}}(x)$: return $(o_{\mathrm{I}}, \mathbf{1})$
$\mathsf{C}^{\leftarrow}_{c_{\mathrm{I}}}(x,(o,q,a))$: $(pk, sk) \leftarrow \mathcal{K}$
$L_F := [\,]$;
$L_G := [\,]$;
return pk
$\mathsf{C}^{\rightarrow}_{G}(x)$: return $(\bot, \mathbf{1})$
$\mathsf{C}^{\leftarrow}_{G}(x,(o,q,a))$: if $x \in \mathsf{dom}\ L_G$ then return $L_G(x)$
else $g \leftarrow \{0,1\}^{k_1}$;
$L_G := L_G.(x, g)$;
return g
endif
$\mathsf{C}^{\rightarrow}_{H}(x)$: return $(\mathcal{H}, x)$
$\mathsf{C}^{\leftarrow}_{H}(x,(o,q,a))$: return a
$\mathsf{C}^{\rightarrow}_{\mathcal{S}}(x)$: $r \leftarrow \{0,1\}^{k_1}$;
return $(\mathcal{H}, x||r)$
$\mathsf{C}^{\leftarrow}_{\mathcal{S}}(x,(o,q,a))$: match $x : \{0,1\}^* \mid r : \{0,1\}^{k_1}$ with q;
$/* $ computation of $G(a) */$
if $a \in \mathsf{dom}\ L_G$ then
$g := L_G(a)$;
else $g \leftarrow \{0,1\}^{k_1}$;
$L_G := L_G.(a, g)$;
endif
$/* $ computation of $F(a) */$
if $a \in \mathsf{dom}\ L_F$ then
$w := L_F(a)$;
else $w \leftarrow \{0,1\}^{k_1}$;
$L_F := L_F.(a, w)$;
endif
return $f^{-1}(a||g \oplus r||w)$
$\mathsf{C}^{\rightarrow}_{c_{\mathrm{F}}}(x)$: match $\mathrm{R} : \{0,1\}^* \mid \mathrm{R}' : \{0,1\}^k$ with x;
match $w_1||w_2||w_3$ with $f(\mathrm{R}')$;
$/* $ computation of $G(w_1) */$
if $w_1 \in \mathsf{dom}\ L_G$ then
$g := L_G(w_1)$;
else $g \leftarrow \{0,1\}^{k_1}$;
$L_G := L_G.(w_1, g)$;
endif
$/* $ computation of $F(w_1) */$
if $w_3^* \in \mathsf{dom}\ L_F$ then
$w_3^* := L_F(w_1)$;
else $w_3^* \leftarrow \{0,1\}^{k_1}$;
$L_F := L_F.(w_1, w_3^*)$;
endif
$r^* \leftarrow w_2 \oplus g$;
return $(o_{\mathrm{F}}, \mathrm{R}||r||w_1)$

Figure IV.7 – Implementations of the Context Procedures of $\mathbb{C}$

$\mathsf{Imp}_{\mathbf{PSS}_1}(H)(x) =$ if $x \in \mathsf{dom}(L_H)$ then return $L_H(x)$
else $w_1||w_2||w_3 \leftarrow \{0,1\}^{k_2} \times \{0,1\}^{k_1} \times \{0,1\}^{k_0}$;
$L_H := L_H.(x, w_1)$;
if $w_1 \notin \mathsf{dom}(L_G)$ then
match $msg : \{0,1\}^* \mid r : \{0,1\}^{k_2}$ with x;
$L'_G := L'_G.(w_1, w_2 \oplus r)$;
endif
if $w_1 \notin \mathsf{dom}(L_F)$ then
$L'_F := L'_F.(w_1, w_3)$;
endif
return w_1
endif

$\mathsf{Imp}_{\mathbf{PSS}_1}(G)(x) =$ if $x \in \mathsf{dom}(L_G)$ then return $L_G(x)$
elsif $x \in \mathsf{dom}(L'_G)$ then
$g := L'_G(x)$;
$L_G := L_G.(x, g)$;
$L'_G := L'_G - (x, g)$;
return g
else $g \leftarrow \{0,1\}^{k_1}$;
$L_G := L_G.(x, g)$;
return g
endif

$\mathsf{Imp}_{\mathbf{PSS}_1}(F)(x) =$ if $x \in \mathsf{dom}(L_F)$ then return $L_F(x)$
elsif $x \in \mathsf{dom}(L'_F)$ then
$w := L'_F(x)$;
$L_F := L_F.(x, w)$;
$L'_F := L'_F - (x, w)$;
return w
else $w \leftarrow \{0,1\}^{k_0}$;
$L_F := L_F.(x, w)$;
return w
endif

$\mathsf{Imp}_{\mathbf{PSS}_1}(\mathcal{S})(x) :$ $r \leftarrow \{0,1\}^{k_1}$;
$w_1 \leftarrow \mathsf{Imp}_{\mathbf{PSS}_1}(H)(x||r)$;
$w_2 \leftarrow \mathsf{Imp}_{\mathbf{PSS}_1}(G)(w_1)$;
$w_3 \leftarrow \mathsf{Imp}_{\mathbf{PSS}_1}(F)(w_1)$;
return $f^{-1}(w_1||(w_2 \oplus r)||w_3)$

Figure IV.8 – Implementation of oracles in $\mathbf{PSS}_1$

$\mathsf{Imp}_{\mathbf{PSS}_2}(H)(x) =$ if $x \in \mathsf{dom}(L_H)$ then return $L_H(x)$
else
$u \leftarrow \{0,1\}^k$;
$v \leftarrow f(pk, u) \otimes y$;
match $w_1||w_2||w_3$ with v;
match $msg : \{0,1\}^* \mid r : \{0,1\}^{k_1}$ with x;
$L_H := L_H.(x, w_1)$;
$L'_F := L'_F.(w_1, w_3)$;
$L'_G := L'_G.(w_1, w_2 \oplus r)$;
$L_u := L_u.(msg, r, u, w_1)$;
return w_1
endif

$\mathsf{Imp}_{\mathbf{PSS}_2}(\mathcal{S})(x) =$ $r \leftarrow \{0,1\}^{k_1}$;
$u \leftarrow \{0,1\}^k$;
$v \leftarrow f(pk, u)$;
match $w_1||w_2||w_3$ with v;
$L_H := L_H.(x, w_1)$;
$L'_F := L'_F.(w_1, w_3)$;
$L'_G := L'_G.(w_1, w_2 \oplus r)$;
return u

Figure IV.9 – Implementations of Signature Oracle and H in $\mathbf{PSS}_2$

containing lists L_H, L_F and L_G, lists L'_G and L'_F have fully determined domains, respectively given by $\mathsf{dom}(L'_G) = \mathsf{range}(L_H) \setminus \mathsf{dom}(L_G)$ and $\mathsf{dom}(L'_F) = \mathsf{range}(L_H) \setminus \mathsf{dom}(L_F)$. Moreover, the lists L'_G and L'_F are filled in by randomly sampling a value of length k_1 (resp. k_0) for every element in their domain.

We continue with an application of the rule *UpToBad*. We define the oracle system $\mathbf{PSS}_2$. We do a number of changes w.r.t. $\mathbf{PSS}_1$:

1. We anticipate the computation of $F(h)$ and $G(h)$ *regardless of whether they have been previously computed*. This makes the new system differ from the previous $\mathbf{PSS}_1$ in case H produces a hash value that has been either produced before for a different input or directly queried by the adversary or by the signing oracle.
2. We introduce a new variable y whose value is uniformly sampled in $\{0,1\}^k$. This prepares for the one-way challenge.
3. In the implementation $\mathsf{Imp}_{\mathbf{PSS}_2}(H)$, we modify the way in which $w_1||w_2||w_3$ is determined by sampling a value u in $\{0,1\}^k$ and computing $w_1||w_2||w_3$ as $f(u) \otimes y$, where $\otimes$ is the inner law of group $\mathcal{G}$. Since f is a permutation, both ways of computing $w_1||w_2||w_3$ yield identically distributed bitstrings. In the signing oracle, we do not perform the group operation and compute $w_1||w_2||w_3$ as $f(u)$.
4. We introduce a list L_u that allows us find the value u from which originates a H-hash value computed as the k_2 prefix of $f(u) \otimes y$.

The implementations of the oracles $\mathcal{S}$ and H of $\mathbf{PSS}_2$ are given in Figure IV.9. The rest of the oracles remain implemented as in $\mathbf{PSS}_1$. Now, let the step-predicate φ be defined on

triples $((o, q, a), m, m')$ as the conjunction of the clauses:

- if $o = H \wedge q \notin m.L_H$ then
  $a \notin \mathsf{dom}\ (m.L_F \cup m.L'_F \cup m.L_G \cup m.L'_G)(*)$
- if $o = \mathcal{S}$ then
  $w_1 \notin \mathsf{dom}\ (m.L_F \cup m.L'_F \cup m.L_G \cup m.L'_G)(**)$
- if $o = \mathcal{S}$ then
  $\forall g\ s.t.\ (w_1, g) \in m'.L_G,\ \ q||(w_2 \oplus g) \notin \mathsf{dom}\ m.L_H(***)$

where $w_1 = f(a)[1, k_2]$ and $w_2 = f(a)[k_2 + 1, k_2 + k_1 + 1]$.

$\mathbf{PSS}_1$ and $\mathbf{PSS}_2$ are R-bisimilar until φ, where R is such that $m\, R\, m'$ iff m and m' coincide on their common components, namely $L_H, L_G, L'_G, L_F, L'_F, pk, sk$.

After application of the macro-rule $UpToBad$, we have two branches left to close. The left one is an application of rule $Fail$. Indeed, we can establish $\mathbf{PSS}_2 :_{\epsilon_1} \mathsf{F}_{\neg\varphi}$, with

$$\epsilon_1 = (k(\mathcal{S}) + k(H))(\frac{k(\mathcal{S})}{2^{k_1}} + \frac{k(F) + k(G) + k(H) + k(\mathcal{S})}{2^{k_2}}).$$

Indeed, a and w_1, respectively w_2, are freshly uniformly sampled value in $\{0,1\}^{k_2}$, resp. $\{0,1\}^{k_1}$. Hence, the probability of breaking property $(*)$ or $(**)$ in a single call to oracle H or $\mathcal{S}$ is $\frac{k(F)+k(G)+k(H)+k(\mathcal{S})}{2^{k_2}}$. Summing over all calls, it yields term $(k(\mathcal{S}) + k(H)).\frac{k(F)+k(G)+k(H)+k(\mathcal{S})}{2^{k_2}}$ Moreover, the probability of breaking property $(***)$ during a call of $\mathcal{S}$ is bounded by $\frac{k(\mathcal{S})+k(H)}{2^{k_1}}$. Summing over all calls to $\mathcal{S}$, it yields term $k(\mathcal{S}).\frac{k(\mathcal{S})+k(H)}{2^{k_1}}$.

Finally, the right branch is concluded by an application of rule $B - Sub$. Indeed, the oracle implementations of $\mathbf{PSS}_2$ do not use f^{-1} or the trapdoor key sk. Therefore, we can write $\mathbf{PSS}_2$ as a context $\mathbb{C}'$ applied to $OW(f)$, i.e. $\mathbf{PSS}_2 = \mathbb{C}'[OW(f)]$. Only the initialization, finalization and signature oracle are performing non-dummy queries to the inner system $OW(f)$. Procedures for H, F and G perform dummy queries and resume their computation independently of the answer they get. However, the initialization procedure queries the initialization oracle, the signature oracle queries the challenge oracle $Chall$ and uses the answer in place of the randomly sampled y in system $\mathbf{PSS}_2$. Finally, we want to choose the context $\mathbb{C}'$ such that:

$$\mathsf{Forge}_2 \circ \mathbb{C}' \Rightarrow \mathsf{Invert}$$

To this end, we define the forward implementation of the finalization procedure of $\mathbb{C}'$ as follows

(where o_{F} denotes the finalization oracle of $OW(f)$):

$\mathsf{C}^{\rightarrow}_{c_{\mathrm{F}}}(x)$: match $\mathrm{R}_1 : \{0,1\}^* | \mathrm{R}_2 : \{0,1\}^k$ with x;
match $w_1||w_2||w_3$ with $f(\mathrm{R}_2)$;
if $w_1 \in \mathsf{dom}(L'_G)$ then
$g := L'_G(w_1)$;
elsif $w_1 \in \mathsf{dom}(L_G)$ then
$g := L_G(w_1)$;
else return $(o_{\mathrm{F}}, \mathbf{1})$
endif
if $w_1 \in \mathsf{dom}(L'_F)$ then
$w := L'_G(w_1)$;
elsif $w_1 \in \mathsf{dom}(L_F)$ then
$w := L_F(w_1)$;
else return $(o_{\mathrm{F}}, \mathbf{1})$
endif
$r := w_2 \oplus g$;
$(u, w_1^*) \leftarrow L_u(\mathrm{R}_1||r)$;
return $(o_{\mathrm{F}}, \mathrm{R}_2 \oslash u)$

When Forge_2 is satisfied, we have $w_1 \in \mathsf{dom}(L_G \cup L'_G)$ and $w_1 \in \mathsf{dom}(L_G \cup L'_G)$, $w_1^* = L_H(w_2) = w_1$, $w_3 = L_F(w_1) = w$ and $w_1||w_2||w_3 = f(u) \otimes y$. Since f is homomorphic, it entails that $u \oslash \mathrm{R}_2 = f(sk, y)$, where $\oslash$ is the inverse operation of $\otimes$ in group $\mathcal{G}$. Hence, $f(\mathrm{R}_2 \oslash u) = y$ and Invert is satisfied.

Chapter V

Automated Proofs of Security for Asymmetric Encryption Schemes in the Random Oracle Model

V.1 Asymmetric-Dedicated Framework

In this section, we first present a novel framework, refered to as the asymmetric-dedicated framework, mostly capturing asymmetric encryption constructions in the random oracle model. This framework is based on programs specified using a small fixed programming language operating on what we call *constructible* distributions. After having provided a semantics for this language and defined these distributions, we introduce three predicates capturing properties of the distributions of values for given variables. These predicates are the cornerstones on which the Hoare logic which we present in the next section is built.

V.1.1 — Programming Language and Constructible Distributions

Syntactic Categories. We suppose that we have a finite set of symbols $\vec{H}$, representing a finite collection of hash functions; H is a meta-variable ranging over $\vec{H}$. Each hash function H is mapped to an integer $\ell(H)$, which is a (public) parameter of the system and represents the length of outputs of the hash function. Moreover, each element $H \in \vec{H}$ is associated to a specific list, the variable L_H. The collection of all these lists is denoted HList. In addition, we have two special variables pk and sk which are meant to store public and secret key values; symbol $\tilde{f}$ denotes an algorithm which can be instantiated for any value of pk and then yield a trapdoor permutation f. In the whole chapter, we suppose that $\tilde{f}$ is fixed and known to the adversary, and $OW(t)$ denotes an upper-bound of the probability that an adversary succeeds in inverting f on a random argument in time at most t. Additionally, $\mathcal{K}$ denotes the key generation algorithm. Another special variable, σ, takes values in bitstrings of finite length,

and denotes the variable in which the adversary can store its state. Finally, we have a finite set of variables Var to denote any other variables. We use any symbol different from those above to denote elements in Var, e.g. x, y, t, q, a. Though finite, the set Var is assumed to be large enough to contain all symbols that we introduce in programs.

Language Description. We now describe the programming language that we work with. It consists in a simple imperative language with random assignment, but does not include loops. Indeed, public-key constructions do not typically require the use of loops. Moreover, whereas the language allows the application of a trapdoor permutation f, it does not include the application of the inverse of such permutations. It remains relevant since even without loops and inverses, we can still capture a significant number of oracle systems on which to apply our results.

Our programming language is built according to the BNF described in Table V.1, where:

— $x \leftarrow \mathcal{U}(l)$ samples a value in $\mathcal{U}(l)$ and assigns it to x, with $\mathcal{U}$ the uniform distribution over the set of bitstrings of length l. Since we do not suppose that all the bitstrings are drawn in the same set $\{0,1\}^l$, we have to specify a length for each drawing command. In the sequel, we use l as a generic notation each time a length needs to be specified.

— $x := f(y)$ applies the trapdoor one-way function f to the value of y and assigns the result to x.

— $x := \alpha \oplus H(y)$, where α is a constant or a variable. This command first applies the random oracle H to the value v of y, i.e. a new hash value h is drawn whenever v has not been already hashed. As a side effect, the pair (v, h) is added to the variable L_H if it was not stored yet. After the hash computation, the bitwise exclusive or of α and the hash value is assigned to x.

— $x := y \oplus z$ applies the exclusive or operator to the values of y and z and assigns the result to x.

— $x := y||z$ represents the concatenation of the values of y and z.

— $\mathsf{c}_1; \mathsf{c}_2$ is the sequential composition of commands c_1 and c_2.

$$\begin{array}{lll} \text{Command} & \mathsf{c} & ::= \quad x \leftarrow \mathcal{U}(l) \mid x := f(y) \mid x := \alpha \oplus H(y) \mid \\ & & \qquad x := y \oplus z \mid x := y||z \mid \mathsf{c}; \mathsf{c} \end{array}$$

Table V.1 – Language Grammar.

In accordance with common practice in concrete security proofs, we disregard the execution time needed for all these operations but the application of f. We assume that command $x := f(y)$ admits an upper-bound on the time required for its execution, independent of the input, which we denote T_f. We can then define T_c for any command in the language by summing the number of applications of f involved.

Semantics. States $m \in \mathsf{M}$ map pk, sk, σ and all variables of Var to bitstrings[1], and variables HList to lists of pairs of bitstrings. The semantics of our language is specified in Table V.2. Notice that the semantic function of commands can be lifted in the usual way to a function

[1]We do not need to worry about introduction of new variables, we have assumed Var large enough to suit our needs.

from $\mathcal{D}(\mathsf{M})$ to $\mathcal{D}(\mathsf{M})$. That is, let $F : \mathsf{M} \to \mathcal{D}(\mathsf{M})$ be a function. Then, F defines a unique function $F^* : \mathcal{D}(\mathsf{M}) \to \mathcal{D}(\mathsf{M})$ such that $F^*(D) = [m \leftarrow D; m' \leftarrow F(m) : m']$. By abuse of notation we denote the semantic function of commands and their lifted counterparts by $[\![\mathsf{c}]\!]$: according to our semantics, commands denote functions that transform distributions on states into distributions on states.

$$
\begin{array}{l}
[\![x \leftarrow \mathcal{U}(l)]\!](m) = [u \leftarrow \mathcal{U}(l) : m.[x \mapsto u]] \\
[\![x := f(y)]\!](m) = \delta(m.[x \mapsto \tilde{f}(m.pk, m.y)]) \\
[\![x := \alpha \oplus H(y)]\!](m) = \\
\quad \begin{cases} \delta(m.[x \mapsto m.\alpha \oplus L_H(m.y)]) \text{ if } m.y \in \mathsf{dom}(m.L_H) \\ [v \leftarrow \mathcal{U}(\ell(H)) : m.[x \mapsto m.\alpha \oplus v, L_H \mapsto L_H :: (m.y, v)]] \text{ if } m.y \notin \mathsf{dom}(m.L_H) \end{cases} \\
[\![x := y \oplus z]\!](m) = \delta(m.[x \mapsto m.y \oplus m.z]) \\
[\![x := y||z]\!](m) = \delta(m.[x \mapsto m.y||m.z]) \\
[\![\mathsf{c}_1; \mathsf{c}_2]\!] = [\![\mathsf{c}_2]\!] \circ [\![\mathsf{c}_1]\!]
\end{array}
$$

Table V.2 – The semantics of the programming language

However, we are only interested in a subset of these distributions, namely, the constructible ones.

Definition (Constructible Distribution). Let χ be a function mapping each $H \in \vec{H}$ to a positive integer. A χ-*constructible* distribution is of the form:

$$\left[(pk_0, sk_0) \leftarrow \mathcal{K}; m \leftarrow \mathfrak{A}^{\vec{H}}(pk_0) : m[pk \mapsto pk_0, sk \mapsto sk_0]\right]$$

where $\mathfrak{A}$ is a probabilistic algorithm with oracle access to the hash functions in $\vec{H}$, making a number of calls bounded by χ. Moreover, $\mathfrak{A}$'s queries to the hash oracles are recorded with their answers in the lists of HList in m; in other words, $\mathfrak{A}$ *cannot* tamper with these lists. Finally, we require that for all hash function $H \in \vec{H}$, $\mathsf{Card}(\mathsf{dom}(L_H)) \leq \chi(H)$.

The set of χ-constructible distributions is denoted by $\mathfrak{const}\mathfrak{D}(\chi)$. A distribution X is said constructible if there exists a tuple χ such that $X \in \mathfrak{const}\mathfrak{D}(\chi)$. The set of constructible distributions is denoted $\mathfrak{const}\mathfrak{D}$. □

We emphasize that the algorithm denoted above by $\mathfrak{A}$ can, but does not necessarily represent an adversary, e.g. it can be the sequence of an adversary and a command of the language. In any case, each and every computation of a hash function value does appear in the list L_H output by $\mathfrak{A}$. Furthermore, we notice that no command in the language can modify the value stored by an adversary in σ. As a consequence, even if an adversary collects all its queries and answers to hash oracles in a list copied in σ, only the values that it *asked* can appear, which is not necessarily the whole content of lists of HList.

Since the hash command has a side effect on L_H, the distribution resulting from its application to a constructible distribution in $\mathfrak{const}\mathfrak{D}(\chi)$ may not belong to the same set, though it remains constructible. In fact, we can be more precise about this remark and write that for all atomic command c and function χ:

$$X \in \mathfrak{const}\mathfrak{D}(\chi) \Leftrightarrow [\![\mathsf{c}]\!](X) \in \mathfrak{const}\mathfrak{D}(\chi + \chi_{\mathsf{c}})$$

where χ_{c} equals either $\mathbf{1}_H$ (which denotes the function where $(\mathbf{1}_H)(H') = 0$ if $H \neq H'$

and 1 if $H = H'$) in case of a hash command, or 0 in the other cases.

We can always find a function χ such that two (or any finite number of) constructible distributions belong to the same set $\mathfrak{const}\mathfrak{D}(\chi)$, by taking the maximum bound on calls for each hash function. Therefore, we choose to simplify the statement of some subsequent results by considering that the distributions that they involve all belong to $\mathfrak{const}\mathfrak{D}(\chi)$, for a given function χ. We do not even specify its value when not needed and speak of constructible distributions then.

V.1.2 — Flavors of Indistinguishability of Constructible Distributions

Before specifying the assertion language, we give a few definitions and notations that we later use to define the predicates of the logic.

Definition (Indistinguishability). Let $\mathcal{A}$ denote a (k,t)-adversary, and let X and X' be two constructible distributions in $\mathfrak{const}\mathfrak{D}(\chi)$. Let $\varepsilon : (k,\chi,t) \mapsto \varepsilon(k,\chi,t)$ be a function ranging in $[0,1]$. The advantage of $\mathcal{A}$ in distinguishing X and X' is denoted $\mathcal{A}dv(\mathcal{A},X,X')$ and defined as

$$|\Pr[m \leftarrow X : \mathcal{A}^{\vec{H}}(m.[sk \mapsto \lambda]) = 1] - \Pr[m \leftarrow X' : \mathcal{A}^{\vec{H}}(m.[sk \mapsto \lambda]) = 1]|$$

We say that distributions X and X' are ε-indistinguishable iff for all (k,t)-adversary $\mathcal{A}$, $\mathcal{A}dv(\mathcal{A},X,X') \leq \epsilon(k,\chi,t)$. In this case, we write $X \sim_\varepsilon X'$. □

We stress that $\mathcal{A}$ is provided with the whole information stored in the state *but* the secret key value. In particular, it includes lists $(L_H)_{H\in\vec{H}}$ of hash values computed during the construction of X or X'. As a consequence, whenever $\mathcal{A}$ queries one of its oracle H on a value appearing in list L_H, it gets the same answer that is stored in L_H. We emphasize that the function k bounding the number of queries to oracles in $\vec{H}$ does not - and has no reason to - take into account queries performed during the construction of X or X'.

Definition (Distribution Restricted to Sets of Variables). Let X be a constructible distribution, let V_1 and V_2 be sets of variables such that $V_1 \subseteq \mathsf{Var} \cup \{\sigma\}$ and $V_2 \subseteq \mathsf{Var}$. By $D(X,V_1,V_2)$ we denote the following distribution:

$$D(X,V_1,V_2) = [m \leftarrow X : (m.V_1, f(m.V_2))]$$

where $m.V_1$ denotes the values of variables in V_1 in state m, and $f(m.V_2)$ denotes the point-wise application of f to the values given by state m to variables in V_2. □

Definition (Restricted Indistinguishability). Let X and X' be two constructible distributions. X and X' are $V_1;V_2;\varepsilon$-*indistinguishable*, denoted by $X \sim_{V_1;V_2;\varepsilon} X'$, iff $D(X,V_1,V_2) \sim_\varepsilon D(X',V_1,V_2)$. □

We emphasize that in the above definition, V_1 and V_2 cannot contain any list L_H, since HList and Var are disjoint. Hence, every time we use the equivalence $\sim_{V_1;V_2;\varepsilon}$, the variables $(L_H)_{H\in\vec{H}}$ *are not given to the adversary.*

Example 9. Let X_0 be the following constructible distribution: $X_0 = [(pk_0,sk_0) \leftarrow \mathcal{K} : [pk \mapsto pk_0, sk \mapsto sk_0]]$. Given an integer l, we define two other constructible distributions:

— $X_1 = [\![x \leftarrow \mathcal{U}(l); y := H(x)]\!](X_0)$,

— $X_2 = [\![x \leftarrow \mathcal{U}(l); y := H(x); y \leftarrow \mathcal{U}(\ell(H))]\!](X_0)$.

Of course, we do not have $X_1 \sim_\varepsilon X_2$ for any $\varepsilon < 1$, no matter how many queries are allowed: any adversary can compare the value given for y with that associated to x in L_H.

Neither can we state $X_1 \sim_{\{x,y\};\emptyset;\varepsilon} X_2$ for any $\varepsilon < 1$ as soon as the adversary is allowed one query to H.

However, $X_1 \sim_{\{y\};\{x\};\varepsilon} X_2$ is possible for lower values of function ε: to perform an attack based on the computation of $H(x)$ given a value for $f(x)$, where x is randomly drawn, an adversary has to invert the trapdoor one-way function. $\Diamond$

Replacing the value of a given variable by an independently uniformly drawn bitstring of the right length is an operation on constructible distributions that we often use. We introduce the notation $\nu x.X$ to denote distributions:

$$[v \leftarrow \mathcal{U}(l); m \leftarrow X : m.[x \mapsto v]] \text{ and } [m \leftarrow X; v \leftarrow \mathcal{U}(l) : m.[x \mapsto v]]$$

given any contructible distribution X and variable x of length l. These distributions coincide, so that we do not need to introduce two notations. We notice that if $X \in \mathfrak{const}\mathfrak{D}(\chi)$, then $\nu x.X$ belongs to the same set $\mathfrak{const}\mathfrak{D}(\chi)$.

V.1.3 — The Assertion Language

In the framework we propose in this chapter, we use another kind of formal method to characterize distributions of values taken by the variables: a Hoare logic. Hoare logics are inspired from the seminal works of logicians C. A. R. Hoare [Hoa69] and R. Floyd [Flo67]. They are inference systems based on Hoare triples, that is, statements of the form $\{P\}\ \mathsf{c}\ \{Q\}$, where P and Q are assertions (on a given predicate language) on the variables used in the program and c is a command. Traditionally, P is called a precondition and Q a postcondition, and a Hoare triple is valid if whenever the precondition is met, the postcondition is verified after the command execution. The inference system is usually completed with a sequence rule to allow compositionality of reasoning.

The key of the success of this kind of approaches lies in the choice of the predicates. Indeed, too precise predicates would not express general enough properties to be useful outside of their strict context of design, whereas too weak choices of predicates would not allow many preservation rules. We propose three predicates capturing three different arguments frequently appearing in proofs of constructions in the literature.

The first predicate deals with indistinguishability. We want to write that one variable x is indistinguishable from random from the adversary's point of view. This is often false if the adversary has access to the value of every variable. We thus weaken the assertion by specifying which variables the adversary can safely access (this is V_1). Some variables can be given to the adversary only under the cover of the trapdoor permutation: this is set V_2. Therefore, the predicate expresses a property of variable x, *seen through* V_1 and $f(V_2)$. The formal definition of the predicate is the following.

Definition (Indis Predicate). Let x be a variable in Var, $V_1 \subseteq \mathsf{Var} \cup \{\sigma\}$ and $V_2 \subseteq \mathsf{Var}$. If X is a χ-constructible distribution, $X \models \mathsf{Indis}(x; V_1; V_2; \varepsilon)$ iff $X \sim_{V_1;V_2;\varepsilon} \nu x.X$ $\square$

The second predicate which we define points out the inability of an adversary to compute the value of a given variable. It is denoted WS, which stands for "weak secrecy". The idea is that bounded resources do not grant enough power to an adversary to compute the value of a variable provided the knowledge of others. Yet it is weaker than Indis, since one does not need to be able to compute something to distinguish it from random. Namely, an adversary

can know how to compute the beginning of a bitstring but not the whole sequence of bits, in which case Indis is not true but WS can be. Here is the formal definition of this second predicate.

Definition (WS Predicate). Let x be a variable in Var, $V_1 \subseteq \mathsf{Var} \cup \{\sigma\}$ and $V_2 \subseteq \mathsf{Var}$, and $\varepsilon : (k, \chi, t) \mapsto \varepsilon(k, \chi, t)$ a function ranging in $[0, 1]$. For $X \in \mathfrak{const}_{\mathfrak{D}}$, $X \models \mathsf{WS}(x; V_1; V_2; \varepsilon)$ iff for any (k, t)-adversary $\mathcal{A}$, $\Pr[m \leftarrow X : \mathcal{A}^{\vec{H}}((m.V_1), f(m.V_2)) = m.x] \leq \varepsilon(k, \chi, t)$. □

For the sake of readability, sets V_1 and V_2 appearing in the predicates are enumerated as collections of variables only separated by commas. For example, in $\mathsf{Indis}(x; y; t, z; 0)$, $V_1 = \{y\}$ and $V_2 = \{t, z\}$, and in $\mathsf{WS}(x; V, t; y; \varepsilon)$, $V_1 = V \cup \{t\}$ and $V_2 = \{y\}$.

Example 10. We consider the following constructible distribution X:
$X = [(pk_0, sk_0) \leftarrow \mathcal{K}; u \leftarrow \mathcal{U}(l) : [pk \mapsto pk_0, sk \mapsto sk_0, x \mapsto u, y \mapsto f(u)]]$. We can state that $X \models \mathsf{WS}(x; y; \emptyset; OW(t))$, or even $X \models \mathsf{WS}(x; y; x; OW(t))$ (here we provide an adversary with the same value twice). However, we notice that $X \not\models \mathsf{Indis}(x; y; x; OW(t))$: here, according to the definitions, a new value v for x is drawn *independently* the sampling of a state in X, but *before* the evaluation of adversarial inputs. It is extremely likely to result in $y \neq f(x)$ (i.e. $f(u) \neq f(v)$). ◊

The two predicates we have yet defined do not express anything about arguments of hash oracles. Consequently, nothing in our assertion language is designed to take advantage of the fact that we work in the random oracle model. True randomness of hash functions allows to consider that the link between an input and its hash value is so thin that, except for the case in which the same argument has already been hashed, the hash value is seemingly random. It is precisely the exceptional case that we want to capture and to rule out. Thus we need a predicate to capture that an expression has not been hashed yet, except with bounded probability. Notice that this predicate is independent of any adversarial intervention, the bound only depends on function χ.

Definition ($\mathsf{H}(H; .)$ Predicate). Let $\varepsilon : \chi \mapsto \varepsilon(\chi)$ be a function ranging in $[0, 1]$. Let e be an expression in the language constructed out of variables of Var stored in state m. Then, $X \models \mathsf{H}(H; e; \varepsilon)$ iff $\Pr[m \leftarrow X : m.e \in \mathsf{dom}(m.L_H)] \leq \varepsilon(\chi)$ where $m.e$ is the evaluation of expression e in state m. □

The predicates cited above are meant to be combined using rules of the Hoare logic we provide in section V.3. Our assertion language is defined by the following grammar, where ψ defines the set of atomic assertions:

$$\psi ::= \mathsf{Indis}(x; V_1; V_2; \varepsilon) \mid \mathsf{WS}(x; V_1; V_2; \varepsilon) \mid \mathsf{H}(H; e; \varepsilon)$$
$$\varphi ::= \mathsf{true} \mid \psi \mid \varphi \wedge \varphi$$

Formally, the meaning of the assertion language is defined by a satisfaction relation $X \models \varphi$, already defined for all atomic assertions, and that we complete with:

— $X \models \mathsf{true}$ and

— $X \models \varphi \wedge \varphi'$ iff $X \models \varphi$ and $X \models \varphi'$.

We combine predicates of our assertion language to prove validity of Hoare triples $\{\varphi\}\ \mathsf{c}\ \{\varphi'\}$, whose meaning is that for all $X \in \mathfrak{const}_{\mathfrak{D}}$, $X \models \varphi$ implies that $[\![\mathsf{c}]\!](X) \models \varphi'$.

V.2 Preliminary Results

V.2.1 — Preservation Results

When two distributions are indistinguishable, it seems fair to expect that predicates holding for one of them are transmitted to the other, maybe up to some additional term. This intuition is formalized by the following lemma.

Lemma V.1 (Compatibility Lemma). *Let $X, X' \in \mathfrak{const}_{\mathfrak{D}}(\chi)$. For any sets of variables $V_1 \subseteq \mathsf{Var} \cup \{\sigma\}$ and $V_2 \subseteq \mathsf{Var}$, and any variable $x \in \mathsf{Var}$:*

1. *if $X \sim_{V_1;V_2;\varepsilon} X'$ then $X \models \mathsf{Indis}(x; V_1; V_2; \varepsilon') \Rightarrow X' \models \mathsf{Indis}(x; V_1; V_2; \varepsilon'')$, where $\varepsilon''(k, \chi, t) = 2.\varepsilon(k, t) + \varepsilon'(k, \chi, t)$.*
2. *if $X \sim_{V_1;V_2,x;\varepsilon} X'$ then $X \models \mathsf{WS}(x; V_1; V_2; \varepsilon') \Rightarrow X' \models \mathsf{WS}(x; V_1; V_2; \varepsilon'')$, where $\varepsilon''(k, \chi, t) = \varepsilon(k, t + \mathsf{T}_f) + \varepsilon'(k, \chi, t)$.*
3. *if $X \sim_{\varepsilon} X'$ then $X \models \mathsf{H}(H; e; \varepsilon') \Rightarrow X' \models \mathsf{H}(H; e; \varepsilon'')$, where $\varepsilon''(\chi) = \varepsilon(k_e, t_e) + \varepsilon'(\chi)$.*

Proof. 1. The following sequence of inequalities justifies the result, with $\mathcal{A}^{\vec{H}}$ a (k, t)-adversary,

$$
\begin{aligned}
&|\Pr[m \leftarrow X' : \mathcal{A}^{\vec{H}}(m.V_1, f(m.V_2)) = \mathsf{true}] - \\
&\qquad \Pr[m \leftarrow \nu x.X' : \mathcal{A}^{\vec{H}}(m.V_1, f(m.V_2)) = \mathsf{true}]| \\
&\leq |\Pr[m \leftarrow X' : \mathcal{A}^{\vec{H}}(m.V_1, f(m.V_2)) = \mathsf{true}] - \\
&\qquad |\Pr[m \leftarrow X : \mathcal{A}^{\vec{H}}(m.V_1, f(m.V_2)) = \mathsf{true}]| \\
&+ |\Pr[m \leftarrow X : \mathcal{A}^{\vec{H}}(m.V_1, f(m.V_2)) = \mathsf{true}] - \\
&\qquad \Pr[m \leftarrow \nu x.X : \mathcal{A}^{\vec{H}}(m.V_1, f(m.V_2)) = \mathsf{true}]| \\
&+ |\Pr[m \leftarrow \nu x.X : \mathcal{A}^{\vec{H}}(m.V_1, f(m.V_2)) = \mathsf{true}] - \\
&\qquad \Pr[m \leftarrow \nu x.X' : \mathcal{A}^{\vec{H}}(m.V_1, f(m.V_2)) = \mathsf{true}]| \\
&\leq \varepsilon(k, t) + \varepsilon'(k, \chi, t) + |\Pr[m \leftarrow X : \mathcal{B}^{\vec{H}}(m.V_1, f(m.V_2)) = \mathsf{true}] \\
&\qquad - \Pr[m \leftarrow X' : \mathcal{B}^{\vec{H}}(m.V_1, f(m.V_2)) = \mathsf{true}]|
\end{aligned}
$$

where $\mathcal{B}$ is the adversary executing the assignment to x of a newly sampled random value and then running $\mathcal{A}$ before forwarding this latter's output. Its advantage is thus bounded by that of $\mathcal{A}$, in turn bounded by $\varepsilon(k, t)$. We can conclude from here.

2. Let $\mathcal{A}^{\vec{H}}$ be a (k, t)-adversary against $\mathsf{WS}(x; V_1; V_2; \varepsilon'')$. We can build an (X, X')-distinguisher $\mathcal{B}^{\vec{H}}$ as follows:

$$
\begin{aligned}
\mathcal{B}^{\vec{H}}(V_1; f(V_2 \cup \{x\})) = \quad & x_0 \leftarrow \mathcal{A}^{\vec{H}}(V_1; f(V_2 \cup \{x\})) \\
& \mathsf{if}\ f(x_0) = f(x)\ \mathsf{then\ return\ true} \\
& \mathsf{else\ return\ false}
\end{aligned}
$$

From $X \sim_{V_1;V_2,x;\varepsilon} X'$ we can deduce:

$$
\begin{aligned}
&|\Pr[m \leftarrow X' : \mathcal{B}^{\vec{H}}(m.V_1, f(m.(V_2 \cup \{x\})) = \mathsf{true}] - \\
&\Pr[m \leftarrow X : \mathcal{B}^{\vec{H}}(m.V_1, f(m.V_2 \cup \{x\})) = \mathsf{true}]| \leq \varepsilon(k, t + \mathsf{T}_f)
\end{aligned}
$$

Besides, the code of $\mathcal{B}$ allows to write:

$$
\begin{aligned}
& \Pr[m \leftarrow X : \mathcal{B}^{\vec{H}}(m.V_1, f(m.(V_2 \cup \{x\})) = \mathsf{true}] \\
= \ & \Pr[m \leftarrow X : f(\mathcal{A}^{\vec{H}}(m.V_1, f(m.V_2))) = f(m.x)] \\
= \ & \Pr[m \leftarrow X : \mathcal{A}^{\vec{H}}(m.V_1, f(m.V_2)) = m.x] \\
\leq \ & \varepsilon'(k, \chi, t)
\end{aligned}
$$

which we can write because f is a bijection.

We can now use a similar equality and conclude using a triangle inequality:

$$
\begin{aligned}
& |\Pr[m \leftarrow X' : \mathcal{A}^{\vec{H}}(m.V_1, f(m.V_2)) = m.x]| \\
= \; & |\Pr[m \leftarrow X' : \mathcal{B}^{\vec{H}}(m.V_1, f(m.(V_2 \cup \{x\})) = \mathsf{true}]| \\
\leq \; & |\Pr[m \leftarrow X' : \mathcal{B}^{\vec{H}}(m.V_1, f(m.(V_2 \cup \{x\})) = \mathsf{true}] - \\
& \qquad \Pr[m \leftarrow X : \mathcal{B}^{\vec{H}}(m.V_1, f(m.(V_2 \cup \{x\})) = \mathsf{true}]| + \\
& \qquad |\Pr[m \leftarrow X : \mathcal{B}^{\vec{H}}(m.V_1, f(m.(V_2 \cup \{x\})) = \mathsf{true}]| \\
\leq \; & \varepsilon(k, t + \mathsf{T}_f) + \varepsilon'(k, \chi, t)
\end{aligned}
$$

3. The proof of the last statement is based on an idea similar to the previous one. We can build an (X, X')-distinguisher $\mathcal{B}^{\vec{H}}$ as follows:

   $\mathcal{B}^{\vec{H}}(m) =$ if $m.e \in \mathsf{dom}(m.L_H)$ then return true
   else return false

   From $X \sim_\varepsilon X'$ we can deduce:
   $|\Pr[m \leftarrow X' : \mathcal{B}^{\vec{H}}(m) = \mathsf{true}] - \Pr[m \leftarrow X : \mathcal{B}^{\vec{H}}(m) = \mathsf{true}]| \leq \varepsilon(k_e, t_e)$
   where k_e and t_e denote the number of queries and the time necessary to compute e.
   Besides, the code of $\mathcal{B}$ allows to write:
   $\Pr[m \leftarrow X : \mathcal{B}^{\vec{H}}(m) = \mathsf{true}] = \Pr[m \leftarrow X : m.e \in \mathsf{dom}(m.L_H)]$,
   which is bounded by $\varepsilon'(\chi)$.
   We can now use a similar equality and conclude using a triangle inequality:

$$
\begin{aligned}
& |\Pr[m \leftarrow X' : m.e \in \mathsf{dom}(m.L_H)| \\
= \; & |\Pr[m \leftarrow X' : \mathcal{B}^{\vec{H}}(m) = \mathsf{true}]| \\
\leq \; & |\Pr[m \leftarrow X' : \mathcal{B}^{\vec{H}}(m) = \mathsf{true}] - \Pr[m \leftarrow X : \mathcal{B}^{\vec{H}}(m) = \mathsf{true}]| + \\
& \qquad |\Pr[m \leftarrow X : \mathcal{B}^{\vec{H}}(m) = \mathsf{true}]| \\
\leq \; & \varepsilon(k_e, t_e) + \varepsilon'(\chi)
\end{aligned}
$$

■

Similar proofs can be performed to prove the following lemma.

Lemma V.2 (Conservation Lemma). *Let $X, X' \in \mathfrak{const}\mathfrak{D}$ such that $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$ iff $X' \in \mathfrak{const}\mathfrak{D}(\chi_0 + \chi')$, with $\chi' \geq 0$. For any sets of variables $V_1 \subseteq \mathsf{Var} \cup \{\sigma\}$ and $V_2 \subseteq \mathsf{Var}$, and any variable $x \in \mathsf{Var}$:*

1. *if $X \sim_{V_1;V_2;0} X'$ then $X \models \mathsf{Indis}(x; V_1; V_2; \varepsilon) \Rightarrow X' \models \mathsf{Indis}(x; V_1; V_2; \varepsilon')$, where $\varepsilon'(k, \chi, t) = \varepsilon(k, \chi - \chi', t)$.*
2. *if $X \sim_{V_1;V_2,x;0} X'$ then $X \models \mathsf{WS}(x; V_1; V_2; \varepsilon) \Rightarrow X' \models \mathsf{WS}(x; V_1; V_2; \varepsilon')$, where $\varepsilon'(k, \chi, t) = \varepsilon(k, \chi - \chi', t + \mathsf{T}_f)$.*

V.2.2 — Weakening Lemmas

When building a Hoare logic, it is in our interest to require the weakest possible premises and show the strongest possible conclusion. As a result, we want to be able to weaken predicates so as to be able to obtain premises of rules from too strong hypotheses. This is the role of the two lemmas presented below.

Lemma V.3 (Weakening Lemma). *Let X be a χ-constructible distribution.*

1. *If $X \models \mathsf{Indis}(x; V_1; V_2; \varepsilon)$, $V_1' \subseteq V_1$ and $V_2' \subseteq V_1 \cup V_2$ then $X \models \mathsf{Indis}(x; V_1'; V_2'; \varepsilon')$, where $\varepsilon'(k, \chi, t) = \varepsilon(k, \chi, t + \mathsf{Card}((V_1 \setminus V_2) \cap V_2').\mathsf{T}_f)$.*
2. *If $X \models \mathsf{WS}(x; V_1; V_2; \varepsilon)$, $V_1' \subseteq V_1$ and $V_2' \subseteq V_1 \cup V_2$ then $X \models \mathsf{WS}(x; V_1'; V_2'; \varepsilon)$, where $\varepsilon'(k, \chi, t) = \varepsilon(k, \chi, t + \mathsf{Card}((V_1 \setminus V_2) \cap V_2').\mathsf{T}_f)$.*

The intuition behind the proofs is the simple fact that the adversary cannot have a better chance to win when given less information. Formally, this translates in the fact that any adversary against the predicate in conclusion of the lemma can be used as an adversary against the predicate in its hypothesis up to the application of f to arguments in $(V_1 \setminus V_2) \cap V_2'$.

Lemma V.4 (Hybrid Weakening Lemma). *Let X be a constructible distribution. If $X \models \mathit{Indis}(x; V_1; V_2, x; \varepsilon)$ and $x \notin V_1$ then $X \models \mathit{WS}(x; V_1; V_2, x; \varepsilon')$, with $\varepsilon'(k, \chi, t) = \varepsilon(k, \chi, t + \mathsf{T}_f) + OW(t)$.*

Proof. Let $\mathcal{A}$ be a (k,t)-adversary against $\mathsf{WS}(x; V_1; V_2, x; \varepsilon')$. We build the following $(k, t+\mathsf{T}_f)$-adversary $\mathcal{B}$ against $\mathsf{Indis}(x; V_1; V_2, x; \varepsilon)$:

$$\begin{aligned}\mathcal{B}^{\vec{H}}(V_1; f(V_2 \cup \{x\})) = \quad & x_0 \leftarrow \mathcal{A}^{\vec{H}}(V_1; f(V_2 \cup \{x\})) \\ & \mathsf{if}\ f(x_0) = f(x)\ \mathsf{then\ return\ true} \\ & \mathsf{else\ return\ false}\end{aligned}$$

The hypothesis on X yields:

$$|\Pr[m \leftarrow X : \mathcal{B}(m.V_1, f(m.(V_2 \cup \{x\})) = \mathsf{true}] - \Pr[m \leftarrow \nu x.X : \mathcal{B}(m.V_1, f(m.(V_2 \cup \{x\})) = \mathsf{true}]| \leq \varepsilon(k, \chi, t + \mathsf{T}_f)$$

We also notice that if $\mathcal{B}$ answers true on a random input $f(x)$, then the value x_0 is a pre-image of f applied on a random value, entailing:

$$\begin{aligned} & |\Pr[m \leftarrow \nu x.X : \mathcal{B}(m.V_1, f(m.(V_2 \cup \{x\}))) = \mathsf{true}]| \\ = \ & |\Pr[m \leftarrow \nu x.X : x_0 \leftarrow \mathcal{A}(m.V_1, f(m.(V_2 \cup \{x\}))) : f(x_0) = f(x)]| \\ \leq \ & OW(t)\end{aligned}$$

Moreover,

$$\begin{aligned} & \Pr[m \leftarrow X; x_0 \leftarrow \mathcal{A}(m.V_1, f(m.(V_2 \cup \{x\}))) : x_0 = m.x] \\ = \ & \Pr[m \leftarrow X; x_0 \leftarrow \mathcal{A}(m.V_1, f(m.(V_2 \cup \{x\}))) : f(x_0) = f(m.x))] \\ = \ & \Pr[m \leftarrow X : \mathcal{B}(m.V_1, f(m.(V_2 \cup \{x\}))) = \mathsf{true}] \\ \leq \ & |\Pr[m \leftarrow X : \mathcal{B}(m.V_1, f(m.(V_2 \cup \{x\}))) = \mathsf{true}] \\ & \qquad - \Pr[m \leftarrow \nu x.X : \mathcal{B}(m.V_1, f(m.(V_2 \cup \{x\}))) = \mathsf{true}]| \\ & \qquad\quad + |\Pr[m \leftarrow \nu x.X : \mathcal{B}(m.V_1, f(m.(V_2 \cup \{x\}))) = \mathsf{true}]| \\ \leq \ & \varepsilon(k, \chi, t + \mathsf{T}_f) + OW(t)\end{aligned}$$

■

V.2.3 — About Expressions

We first define the set of subvariables of an expression, which are variables appearing as a substrings of an expression.

Definition (Subvariable Set $\mathsf{subvar}(e)$). The set of variables used as substring of an expression e is denoted $\mathsf{subvar}(e)$: $x \in \mathsf{subvar}(e)$ iff $e = x$ or $e = e_1||e_2$ and $x \in \mathsf{subvar}(e_1) \cup \mathsf{subvar}(e_2)$, for some expressions e_1 and e_2. Each subvariable x of expression e is associated to an extraction function g, such for all state m, $g(m.e) = m.x$. The execution time of g is disregarded. □

Example 11. For example, we assume that we consider the following expression: $e = (R||q||f(R||r)) \,||\, (h \oplus G(R))$. Here, $\mathsf{subvar}(e) = \{R, q\}$, but $r, h \notin \mathsf{subvar}(e)$. ♢

Moreover, given sets $(V_1; V_2)$, some commands of the type $x := e$ can be executed by the adversary on its own. These expressions e are called constructible. We emphasize that we talk about constructibility in one command application, not constructibility in a finite number of them.

Definition (Constructible Expressions). Let $V_1, V_2 \subseteq \mathsf{Var}$. Then, the set of expressions constructible from $(V_1; V_2)$ are:

— $f(x)$ for all variables $x \in V_1 \cup V_2$,

— $\alpha \oplus H(x)$ for all variables $x \in V_1$ and $\alpha \in V_1$, or for any constant value of α,

— exclusive or and concatenation of all pairs of variables in V_1.

□

Lemma V.5. *We let $X, X' \in \mathfrak{const}\mathfrak{D}(\chi_0)$ such that $X \sim_{V_1;V_2;\varepsilon} X'$, e be an expression constructible from $(V_1 \setminus \{\sigma\}; V_2)$, and $\mathsf{c} \equiv x := e$. Then,*

$[\![x := e]\!](X) \sim_{V_1,x;V_2;\varepsilon'} [\![x := e]\!](X')$, *where* $\varepsilon'(k, \chi, t) = \varepsilon(k + \chi_{\mathsf{c}}, \chi - \chi_{\mathsf{c}}, t + \mathsf{T}_{\mathsf{c}})$.

Proof. We recall that $X, X' \in \mathfrak{const}\mathfrak{D}(\chi_0)$ iff $[\![x := e]\!](X), [\![x := e]\!](X') \in \mathfrak{const}\mathfrak{D}(\chi_0 + \chi_{\mathsf{c}})$. To any (k, t)-adversary $\mathcal{A}$ trying to distinguish between $[\![x := e]\!](X)$ and $[\![x := e]\!](X')$ we can associate a counterpart $\mathcal{B}$ trying to distinguish between X and X' by prepending $\mathcal{A}$'s execution by the construction of e. This latter adversary distinguishes between distributions constructible in at most χ_0 requests, performs at most $k + \chi_c$ queries, and takes a time bounded by $t + \mathsf{T}_{\mathsf{c}}$ to compute $\mathcal{A}$'s inputs. The conclusion follows. ■

The same ideas allow to prove the following statements. We notice that in the case of the indistinguishability statement, we need to impose that z does not appear at all in e, even under any function symbol. Indeed, the idea of the proof above is that the adversary is able to construct e. If to do so, an adversary needs z and is provided with a random value in place of it, then our argument does not hold.

Lemma V.6. *Let $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$, let e be an expression constructible from $(V_1 \setminus \{\sigma\}; V_2)$, and $z \neq x$. Let $\mathsf{c} \equiv x := e$.*

1. *If $X \models \mathsf{Indis}(z; V_1; V_2; \varepsilon)$ and z does not (syntactically) appear in e, then $[\![x := e]\!](X) \models \mathsf{Indis}(z; V_1, x; V_2; \varepsilon')$*
2. *If $X \models \mathsf{WS}(z; V_1; V_2; \varepsilon)$ then $[\![x := e]\!](X) \models \mathsf{WS}(z; V_1, x; V_2; \varepsilon')$*

where $\varepsilon'(k, \chi, t) = \varepsilon(k + \chi_{\mathsf{c}}, \chi - \chi_{\mathsf{c}}, t + \mathsf{T}_{\mathsf{c}}))$.

V.2.4 — Hash-Related Lemmas

As hash functions are implemented by random oracles, the images that they associate to different inputs are completely independent from one another. Therefore, while a hash value has not been queried, then one can redraw it without this changing anything from the adversary's point of view. We can even go a little further in our reasoning: to execute the command $x := \alpha \oplus H(y)$, we can either draw a value for $H(y)$ at random and bind it by storing it in L_H, or draw x at random and bind $H(y)$ to be worth $x \oplus \alpha$. Of course, we shall carefully take into account the side effects of the command on L_H. To deal with rebinding matters, we introduce a new operator on states.

Definition (Rebinding Operator). For any state m, variable $y \in \mathsf{Var}$, expression e and function $H \in \vec{H}$,

$$\mathsf{rebind}_H^{y \mapsto e}(m) = m[L_H \mapsto m.L_H \bullet (m.y, m.e)],$$

where $L \bullet (a,b) = (L - [(a, L(a))]) :: (a,b)$, i.e. the association list L where a is mapped to b, no matter whether it did already appear in $\mathsf{dom}(L)$. This definition can easily be lifted to distributions in $\mathfrak{const}\mathfrak{D}$ by setting $\mathsf{rebind}_H^{y\mapsto e}(X) = [m \leftarrow X : \mathsf{rebind}_H^{y\mapsto e}(m)]$. $\square$

We stress that, given a constructible distribution X, while $[\![x := \alpha \oplus H(y)]\!](X)$ associates to X another constructible distribution, it is not the case of the rebinding operator: $\mathsf{rebind}_H^{y\mapsto e}(X) \notin \mathfrak{const}\mathfrak{D}$.

The following lemma is the formalization of the intuition that we can freely rebind the hash value associated to a variable y to a random value of our choice as long as y does not appear in L_H already.

Lemma V.7 (Rebinding Lemma). *For any $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$, any hash function H, any variables x and y in Var, if $X \models \mathsf{H}(H; y; \varepsilon)$, then*

$$[\![x := \alpha \oplus H(y)]\!](X) \sim_\varepsilon \mathsf{rebind}_H^{y\mapsto \alpha\oplus x}(\nu x.X),$$

where α is either a constant or a variable.[2]

Proof. We start by developping the terms involved in the distance computation. Firstly,

$$\begin{aligned}
&\Pr[m \leftarrow [\![x := \alpha \oplus H(y)]\!](X) : m = m_0] \\
&= \Pr[m \leftarrow X : m.y \in \mathsf{dom}(m.L_H) \wedge m_0 = m.[x \mapsto m.\alpha \oplus L_H(m.y)]] \\
&\quad + \Pr[m \leftarrow X; v \leftarrow \mathcal{U}(\ell(H)) : m.y \notin \mathsf{dom}(m.L_H) \\
&\qquad\qquad \wedge m_0 = m.[x \mapsto v \oplus m.\alpha, L_H \mapsto L_H :: (m.y, v)]]
\end{aligned}$$

by applying the definition provided in the semantics for the command.

In addition to that,

$$\begin{aligned}
&\Pr[m \leftarrow \mathsf{rebind}_H^{y\mapsto\alpha\oplus x}(\nu x.X) : m = m_0] \\
&= \Pr[m \leftarrow X; v \leftarrow \mathcal{U}(\ell(H)) : m_0 = m.[x \mapsto v, L_H \mapsto L_H \bullet (m.y, m.\alpha \oplus v)]]
\end{aligned}$$

by definition of the rebind operator,

$$\begin{aligned}
&= \Pr[m \leftarrow X; v \leftarrow \mathcal{U}(\ell(H)) : m_0 = m.[x \mapsto v \oplus m.\alpha, L_H \mapsto L_H \bullet (m.y, v)]]
\end{aligned}$$

since x is uniformly distributed,

$$\begin{aligned}
&= \Pr[m \leftarrow X; v \leftarrow \mathcal{U}(\ell(H)) : m.y \notin \mathsf{dom}(m.L_H) \\
&\qquad\qquad \wedge m_0 = m.[x \mapsto v \oplus m.\alpha, L_H \mapsto L_H :: (m.y, v)]] \\
&\quad + \Pr[m \leftarrow X; v \leftarrow \mathcal{U}(\ell(H)) : m.y \in \mathsf{dom}(m.L_H) \\
&\qquad\qquad \wedge m_0 = m.[x \mapsto v \oplus m.\alpha, L_H \mapsto L_H :: (m.y, v)]]
\end{aligned}$$

Using the results of these computations, we can compute the statistical distance between distributions:

$$\begin{aligned}
&|\Pr[[\![x := \alpha \oplus H(y)]\!](X) = m_0] - \Pr[\mathsf{rebind}_H^{y\mapsto\alpha\oplus x}(\nu x.X) = m_0]| \\
&= |\Pr[m \leftarrow X : m.y \in \mathsf{dom}(m.L_H) \wedge m_0 = m.[x \mapsto L_H(m.y)]] \\
&\qquad - \Pr[m \leftarrow X; v \leftarrow \mathcal{U}(\ell(H)) : m.y \in \mathsf{dom}(m.L_H) \\
&\qquad\qquad \wedge m_0 = m.[x \mapsto v \oplus m.\alpha, L_H \mapsto L_H :: (m.y, v)]]|
\end{aligned}$$

Both probabilities are bounded by $\Pr[m \leftarrow X : m.y \in \mathsf{dom}(m.L_H)]$, which is in turn worth less than $\varepsilon(\chi_0)$. Consequently, both quantities belong to $[0, \varepsilon(\chi_0)]$, so that we can deduce that $\varepsilon(\chi_0)$ is a uniform bound on the statistical distance between the distributions, which allows us to conclude.

■

Now we are interested in formally proving the useful and intuitive following lemma, which states that to distinguish between a distribution and its "rebound" version, an adversary must

[2]The indistinguishability statement here does not involve a pair of constructible distributions. We consider the function ε to depend on (k,t) bounding an adversary's resources, and see χ as a constant.

be able to compute the argument y whose hash value has been rebound. Indeed, even though V_1 can contain the adversary variable σ in which a previously asked value for $H(y)$ might be stored, there is no distinguishing between both distributions without querying once more for $H(y)$ to acknowledge a possible change. Therefore a distinguishing adversary is only as good as it is to compute y given V_1 and V_2.

Lemma V.8 (Hash vs. Rebind Lemma). *For any $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$, any two variables x and y in Var, any two finite sets of variables $V_1 \subseteq \mathsf{Var} \cup \{\sigma\}$ and $V_2 \subseteq \mathsf{Var}$, and any hash function H, if $X \models \mathsf{WS}(y; V_1; V_2; \varepsilon)$, then*

$$X \sim_{V_1;V_2;\varepsilon'} \mathsf{rebind}_H^{y \mapsto \alpha \oplus x}(X).$$

where α is either a constant or a variable in Var, and $\varepsilon'(k,t) = k(H) * \varepsilon(k, \chi_0, t)$.[3]

Proof. This result follows from the fact that the distributions coincide on *everything* but the value to which y is mapped by L_H. Indeed, the inputs to the adversary are identically distributed:

$$\begin{array}{ll} & \Pr[m \leftarrow \mathsf{rebind}_H^{y \mapsto \alpha \oplus x}(X) : (m.V_1, f(m.V_2))] \\ = & \Pr[m \leftarrow X; m' = m.[L_H \mapsto L_H \bullet (m.y, m.\alpha \oplus v)] : (m'.V_1, f(m'.V_2))] \\ = & \Pr[m \leftarrow X; m' = m.[L_H \mapsto L_H \bullet (m.y, m.\alpha \oplus v)] : (m.V_1, f(m.V_2))] \\ & \text{since } L_H \notin \mathsf{Var} \text{ then we have } L_H \notin V_1 \cup V_2 \\ = & \Pr[m \leftarrow X : (m.V_1, f(m.V_2))] \end{array}$$

Moreover, for all hash queries performed to any function different from H, distribution of answers obviously coincide. In addition to that, except for a query on the value of y, answers of queries to H are distributed identically in both settings, whether or not hash values have already been computed.

As a result, by splitting the probabilities according to whether the value of y has been queried to H, we can bound the advantage of a distinguisher by the probability that y is queried. In turn, out of a distinguisher issuing a query worth y amongst the $k(H)$ which it performs, we can build an adversary computing y by picking one of the $k(H)$ queries at random. The conclusion follows. ■

From the last part of the previous proof, we can notice that if we have a way to identify amongst queries performed the one corresponding to y, we can build a more performant adversary to compute a value for y. Namely, if an adversary has a value for $f(y)$, it can determine which of the values from a set is y (f is a permutation so $f(y)$ has only one pre-image). The additional time taken by this adversary is the number of values to test multiplied by the time of a test, i.e. $k(H) * \mathsf{T}_f$. With these updates in the proof, we can state the following weakened version of the lemma above.

Lemma V.9 (Weak Hash vs. Rebind). *For any $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$, any two variables $x, y \in \mathsf{Var}$, any two finite sets of variables $V_1 \subseteq \mathsf{Var} \cup \{\sigma\}$ and $V_2 \subseteq \mathsf{Var}$, and any hash function H, if $X \models \mathsf{WS}(y; V_1; V_2, y; \varepsilon)$, then*

$$X \sim_{V_1;V_2;\varepsilon'} \mathsf{rebind}_H^{y \mapsto \alpha \oplus x}(X),$$

[3] As for the previous result, ε' can only be a function of (k,t).

where α is either a constant or a variable in Var and with $\varepsilon'(k,t) = \varepsilon(k, \chi_0, t + k(H) * \mathsf{T}_f)$.

V.3 The Hoare Logic

Now that basic predicates and properties are set, we must provide the rules that ensure their conservation or sound transformation when commands are applied. Rules fall into two categories: preservation rules that express how a property is modified after the command, and creation rules that state a property of a variable newly assigned. The rules are given along with some inequality restrictions, which are meant to be understood syntactically, e.g. $x \neq t, y$ means that x and t, y are not the same variable name. In all the rules we state, and as their definitions require, predicates take as arguments a variable in Var, a set V_1 included in $\mathsf{Var} \cup \{\sigma\}$ and a set V_2 is included in Var.

Hereafter, except for the first set of general preservation cases, the rules are sorted according to the command which they deal with.

Generic Preservation Rules. The first two rules deal with predicates on a variable z different from variables appearing in the command c that is applied. The idea is that the predicates Indis and WS are quite intuitively preserved as soon as the newly assigned variable (here x) does not appear in sets V_1, V_2.

Lemma V.10 (Rules (G1) and (G2)). *The following rules are sound, when $z \neq x$, and c is $x \leftarrow \mathcal{U}$ or of the form $x := e'$ with e' being either $w||y$, $w \oplus y$, $f(y)$ or $\alpha \oplus H(y)$, provided $x \notin V_1 \cup V_2$:*

(G1) $\{\mathsf{Indis}(z; V_1; V_2; \varepsilon)\}\ \mathsf{c}\ \{\mathsf{Indis}(z; V_1; V_2; \varepsilon')\}$

(G2) $\{\mathsf{WS}(z; V_1; V_2; \varepsilon)\}\ \mathsf{c}\ \{\mathsf{WS}(z; V_1; V_2; \varepsilon')\}$
where $\varepsilon'(k, \chi, t) = \varepsilon(k, \chi - \chi_{\mathsf{c}}, t)$

Proof. Let $X \in \mathfrak{const}\mathfrak{D}(\chi)$. Let us notice that for any of these commands c, $[\![\mathsf{c}]\!]$ affects at most x and L_H. It follows that for any sets V_1 and V_2 not containing x, $D(X, V_1, V_2) = D([\![\mathsf{c}]\!](X), V_1, V_2)$. We know that $[\![\mathsf{c}]\!](X)$ belongs to $\mathfrak{const}\mathfrak{D}(\chi + \chi_{\mathsf{c}})$. The above equality between distributions can be stated as $X \sim_{V_1;V_2;0} [\![c]\!](X)$. Applying the conservation lemma V.2 with this hypothesis and $X \models \mathsf{Indis}(z; V_1; V_2; \varepsilon)$ provides $[\![c]\!](X) \models \mathsf{Indis}(z; V_1; V_2; \varepsilon')$, where $\varepsilon'(k, \chi, t) = \varepsilon(k, \chi - \chi_{\mathsf{c}}, t)$. Similarly, $[\![c]\!](X) \models \mathsf{WS}(z; V_1; V_2; \varepsilon')$ ensues from $X \models \mathsf{WS}(z; V_1; V_2; \varepsilon)$ and the second item of the conservation lemma. ■

The same kind of statement can be made when executing a command concerning an expression which is constructible out of the pair of sets V_1 and V_2.

Lemma V.11 (Rules (G1') and (G2')). *The following rules are sound, when $z \neq x$, and c is of the form $x := e'$ with e' being either $w||y$, $w \oplus y$, $f(y)$ or $\alpha \oplus H(y)$:*

(G1') $\{\mathsf{Indis}(z; V_1; V_2; \varepsilon)\}\ \mathsf{c}\ \{\mathsf{Indis}(z; V_1, x; V_2; \varepsilon')\}$, *if e' is constructible from $(V_1 \setminus \{z\}; V_2 \setminus \{z\})$.*

(G2') $\{\mathsf{WS}(z; V_1; V_2; \varepsilon)\}\ \mathsf{c}\ \{\mathsf{WS}(z; V_1, x; V_2; \varepsilon')\}$, *if e' is constructible from $(V_1; V_2)$,*
where $\varepsilon'(k, \chi, t) = \varepsilon(k + \chi_{\mathsf{c}}, \chi - \chi_{\mathsf{c}}, t + \mathsf{T}_{\mathsf{c}}))$.

Proof. As e' is constructible from $(V_1 \setminus \{z\}; V_2 \setminus \{z\})$, we can apply lemma V.6 which yields $[\![x := e']\!](X) \models \mathsf{Indis}(z; V_1, x; V_2; \varepsilon')$, where $\varepsilon'(k, \chi, t) = \varepsilon(k + \chi_{\mathsf{c}}, \chi - \chi_{\mathsf{c}}, t + \mathsf{T}_{\mathsf{c}}))$.

The same lemma allows to conclude to weak secrecy. ■

We also propose rules covering the preservation of predicate $\mathsf{H}(H'; e)$. Firstly, no matter what the command assigning a new value to x is, if it does not use H' and x does not appear in e, the probability of e appearing in $L_{H'}$ is the same before and after the command execution. Secondly, the probability of finding in $L_{H'}$ the value of e where x is replaced by e' remains unchanged if e does contain x and we execute $x := e'$. Hereafter, $e[e'/x]$ denotes the expression obtained from e by replacing x by e'.

Lemma V.12 (Rules (G3) and (G3')). *The following rules are sound, when $H' \neq H$ and c is of the form $x := e'$ with e' being either $t||y$, $t \oplus y$, $f(y)$ or $\alpha \oplus H(y)$:*

(G3) $\{\mathsf{H}(H'; e[e'/x]; \varepsilon)\}\ \mathsf{c}\ \{\mathsf{H}(H'; e; \varepsilon')\}$, *where* $\varepsilon'(k, \chi, t) = \varepsilon(k, \chi - \chi_{\mathsf{c}}, t)$.

(G3') $\{\mathsf{H}(H'; e; \varepsilon)\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{H}(H'; e; \varepsilon)\}$, *if x does not appear in e.*

Proof. Let $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$. As mentioned previously, side effects of c can only appear on x and L_H. As a result, probability of belonging to $L_{H'}$ is conserved:

$$\begin{array}{rl} & \Pr[m \leftarrow X; m' \leftarrow [\![\mathsf{c}]\!](m) : m'.e \in \mathsf{dom}(m'.L_{H'})] \\ = & \Pr[m \leftarrow X; m' \leftarrow [\![\mathsf{c}]\!](m) : m.(e[e'/x]) \in \mathsf{dom}(m'.L_{H'})] \\ & \text{because } \mathsf{c} \equiv x := e' \\ = & \Pr[m \leftarrow X; m' \leftarrow [\![\mathsf{c}]\!](m) : m.(e[e'/x]) \in \mathsf{dom}(m.L_{H'})] \\ & \text{since } m.L_{H'} = m'.L_{H'} \\ = & \Pr[m \leftarrow X : m.(e[e'/x]) \in \mathsf{dom}(m.L_{H'})] \\ \leq & \varepsilon(k, \chi_0, t) \end{array}$$

As a conclusion, we can choose $\varepsilon'(k, \chi, t) = \varepsilon(k, \chi - \chi_{\mathsf{c}}, t)$.

In a similar manner, if x does not appear in e,

$$\begin{array}{rl} & \Pr[m \leftarrow X; m' \leftarrow [\![x \leftarrow \mathcal{U}(l)]\!](m) : m'.e \in \mathsf{dom}(m'.L_{H'})] \\ = & \Pr[m \leftarrow X; m' \leftarrow [\![x \leftarrow \mathcal{U}(l)]\!](m) : m.e \in \mathsf{dom}(m.L_{H'})] \end{array}$$

which justifies the perfect conservation of the bound. ■

Rules for Random Assignment. After the random draw of a bitstring for x, we can of course state that x is perfectly indistinguishable from random given any set of variables: this is what is captured by the first rule. Furthermore, for any expression containing x as a subvariable, we can bound the probability that e appears in a given set if x is distributed uniformly at random. This idea accounts for rule $(R2)$.

Lemma V.13 (Rules (R1) and (R2)). *The following rules are sound:*

(R1) $\{\mathsf{true}\}\ x \leftarrow \mathcal{U}(l)\ \{\mathit{Indis}(x; \mathit{Var} \cup \{\sigma\}; \emptyset; 0)\}$

(R2) $\{\mathsf{true}\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{H}(H; e; \frac{\chi(H)}{2^l})\}$ *if* $x \in \mathit{subvar}(e)$.

Proof. Only the second rule needs justification. Since $x \in \mathsf{subvar}(e)$, there exists a function g to extract x such that $g(m.e) = m.x$ for any state m. We are interested in evaluating, for

$X \in \mathfrak{const}\mathfrak{D}(\chi)$:

$$\begin{array}{ll} & Pr[m \leftarrow [\![x \leftarrow \mathcal{U}(l)]\!](X) : m.e \in \mathsf{dom}(m.L_H)] \\ = & Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l); m' := m.[x \mapsto u] : m'.e \in \mathsf{dom}(m'.L_H)] \\ = & Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l); m' := m.[x \mapsto u] : m'.e \in \mathsf{dom}(m.L_H)] \\ & \text{because } m.L_H = m'.L_H \\ \leq & Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l) : u \in g(\mathsf{dom}(m.L_H))] \\ & \text{because } g(m'.e) = g(m'.x) = u \\ \leq & \frac{\chi(H)}{2^l} \\ & \text{since } \mathsf{Card}(g(\mathsf{dom}(m.L_H))) \leq \mathsf{Card}(\mathsf{dom}(m.L_H)) \leq \chi(H). \end{array}$$

■

We then present a pair of rules providing a little more than sheer preservation of predicates. They are based on the consideration that if a predicate holds for sets V_1, V_2 before the execution of $x \leftarrow \mathcal{U}(l)$, then it cannot help to provide an adversary with the value of x after the execution of the command: the same predicate should hold for sets $V_1 \cup \{x\}$ and V_2.

Lemma V.14 (Rules (R3) and (R4)). *Additionally, we have the following preservation rules, where we assume $x \neq y$:*

(R3) $\{\mathsf{Indis}(y; V_1; V_2; \varepsilon)\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{Indis}(y; V_1, x; V_2; \varepsilon)\}$

(R4) $\{\mathsf{WS}(y; V_1; V_2; \varepsilon)\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{WS}(y; V_1, x; V_2; \varepsilon)\}$

Proof. We prove these rules by reduction. Let $\mathcal{A}$ be a (k, t) adversary against the conclusion predicate. Firstly, let us assume $x \notin V_1 \cup V_2$. Let $\mathcal{B}$ be the adversary obtained out of $\mathcal{A}$ by prepending the execution of $x \leftarrow \mathcal{U}(l)$ to the execution of $\mathcal{A}$ on the same arguments given to $\mathcal{B}$. Then the adversaries have the same output distribution. Besides, $\mathcal{B}$'s execution time is bounded by that of $\mathcal{A}$. As a result, $\mathcal{A}$'s advantage is bounded by $\varepsilon(k, \chi, t)$.

Secondly, if $x \in V_1 \cup V_2$, the same arguments hold except that $\mathcal{A}$ has to be provided inputs where x's value is updated to its newly drawn random value (and/or its image by f if $x \in V_2$). ■

Rules for Hash Commands. In this subsection, we present rules for hash commands. Let $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$, we know that $[\![x := \alpha \oplus \mathsf{H}(y)]\!](X) \in \mathfrak{const}\mathfrak{D}(\chi_0 + \mathbf{1}_H)$ is equivalent to $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$.

The idea behind the first two creation rules is that if a variable has not been mapped to a hash value yet, then once it is, the only way to differentiate its hash value from random is to know how to compute the variable and perform the adequate query to the hash oracle.

Lemma V.15 (Rules (H1) and (H2)). *The following rules are sound when $x \neq y$:*

(H1) $\{\mathsf{WS}(y; V_1; V_2; \varepsilon) \wedge \mathsf{H}(H; y; \varepsilon')\}\ x := \alpha \oplus \mathsf{H}(y)\ \{\mathsf{Indis}(x; V_1, x; V_2; \varepsilon'')\}$, *where* $\varepsilon''(k, \chi, t) = \varepsilon'(\chi - \mathbf{1}_H) + k(H) * \varepsilon(k, \chi - \mathbf{1}_H, t)$.

(H2) $\{\mathsf{WS}(y; V_1; V_2, y; \varepsilon) \wedge \mathsf{H}(H; y; \varepsilon')\}\ x := \alpha \oplus \mathsf{H}(y)\ \{\mathsf{Indis}(x; V_1, x; V_2, y; \varepsilon'')\}$ *if* $y \notin V_1$, *where* $\varepsilon''(k, \chi, t) = \varepsilon'(\chi - \mathbf{1}_H) + \varepsilon(k, \chi - \mathbf{1}_H, t + k(H) * \mathsf{T}_f)$

Proof. Let $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$ be a distribution satisfying both our hypotheses. We prove rule (H1) by proving three indistinguishability statements and then using transitivity.

Firstly, $X \models \mathsf{WS}(y; V_1; V_2; \varepsilon)$ and $x \neq y$ results in $\nu x.X \models \mathsf{WS}(y; V_1, x; V_2; \varepsilon_1)$ using rule (R4), where $\varepsilon_1(k, \chi, t) \leq \varepsilon(k, \chi_0, t)$. Now, applying the hash vs. rebind lemma V.8, $\nu x.X \sim_{V_1,x;V_2;\varepsilon_2} \mathsf{rebind}_H^{y \mapsto \alpha \oplus x}(\nu x.X)$ with $\varepsilon_2(k,t) = k(H) * \varepsilon(k, \chi_0, t)$.

In addition to that, predicate $\mathsf{H}(H; y; \varepsilon')$ allows to use the rebinding lemma V.7, providing:

$[\![x := \alpha \oplus H(y)]\!](X) \sim_{\varepsilon'} \mathsf{rebind}_H^{y \mapsto \alpha \oplus x}(\nu x.X)$,

which of course implies the following restricted indistinguishability statement $[\![x := \alpha \oplus H(y)]\!](X) \sim_{V_1,x;V_2;\varepsilon'} \mathsf{rebind}_H^{y \mapsto \alpha \oplus x}(\nu x.X)$.

Finally, we observe that $\nu x.X \sim_{V_1,x;V_2;0} \nu x.[\![x := \alpha \oplus H(y)]\!](X)$. Indeed, the command $x := \alpha \oplus H(y)$ affects x and L_H only, and x is resampled in both distributions so that obviously,

$D(\nu x.X, V_1 \cup \{x\}, V_2) = D(\nu x.[\![x := \alpha \oplus H(y)]\!](X), V_1 \cup \{x\}, V_2)$.

In addition, answers of queries to hash functions are distributed identically in both settings. Therefore, provided with identical input distribution and interacting with oracles yielding identical answer distributions, distinguishers output equal distributions.

In conclusion, the transitivity of the indistinguishability relation allows to deduce $\nu x.[\![x := \alpha \oplus H(y)]\!](X) \sim_{V_1,x;V_2;\varepsilon_3} [\![x := \alpha \oplus H(y)]\!](X)$ where $\varepsilon_3(k,t) \leq \varepsilon'(k, \chi_0, t) + k(H) * \varepsilon(k, \chi_0, t)$. In addition to the fact that $[\![x := \alpha \oplus H(y)]\!](X)$ is an element of $\mathfrak{const}\mathfrak{D}(\chi_0 + \mathbf{1}_H)$, this justifies our choice of $\varepsilon''(k, \chi, t) = \varepsilon'(k, \chi - \mathbf{1}_H, t) + k(H) * \varepsilon(k, \chi - \mathbf{1}_H, t)$ for a bound in our conclusion predicate.

As for rule (H2), the same proof as above can be carried out, but using the weak version of the hash vs rebind lemma V.9 to end up with the desired result. ■

The next rule takes advantage of randomness of $H(y)$ when y does not belong to L_H. Indeed, if $H(y)$ looks random to the adversary, then so does x after we execute $x := \alpha \oplus H(y)$. If x appears as a subvariable of an expression e, we can use its seemingly randomness to bound the probability that e belongs to a list $L_{H'}$, for a different hash function H'.

Lemma V.16 (Rule (H3)). *The following rule is sound when $x \neq y$, and $x \in \mathsf{subvar}(e)$:*

(H3) $\{\mathsf{H}(H; y; \varepsilon)\}\ x := \alpha \oplus H(y)\ \{\mathsf{H}(H'; e; \varepsilon')\}$

where $\varepsilon'(\chi) = \varepsilon(\chi - \mathbf{1}_H) + \frac{\chi(H)}{2^{\ell(H)}}$.

Proof. Let $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$. The hypothesis that $\mathsf{H}(H; y; \varepsilon)$ enables to use the rebinding lemma V.7, providing:

$[\![x := \alpha \oplus H(y)]\!](X) \sim_{\varepsilon(\chi_0)} \mathsf{rebind}_H^{y \mapsto \alpha \oplus x}(\nu x.X)$.

Let e be an expression containing x as a subvariable. We show below that $\mathsf{rebind}_H^{y \mapsto \alpha \oplus x}(\nu x.X) \models \mathsf{H}(H; e; \frac{\chi_0(H)+1}{2^{\ell(H)}})$. We let g be an extracting function for $x \in \mathsf{subvar}(e)$.

$$
\begin{array}{rl}
& \Pr[m \leftarrow \mathsf{rebind}_H^{y\mapsto\alpha\oplus x}(\nu x.X) : m.e \in \mathsf{dom}(m.L_{H'})] \\
\leq & \Pr[m \leftarrow \nu x.X; m' \leftarrow \mathsf{rebind}_H^{y\mapsto\alpha\oplus x}(m) : m'.e \in \mathsf{dom}(m.L_{H'}) \cup \{m.y\}] \\
& \text{since } \mathsf{dom}(m'.L_{H'}) \subseteq \mathsf{dom}(m.L_{H'}) \cup \{m.y\}, \\
= & \Pr[m \leftarrow \nu x.X; m' \leftarrow \mathsf{rebind}_H^{y\mapsto\alpha\oplus x}(m) : m.e \in \mathsf{dom}(m.L_{H'}) \cup \{m.y\}] \\
& \text{using } m.e = m'.e \text{ by definition of the rebinding} \\
= & \Pr[m \leftarrow \nu x.X : m.e \in \mathsf{dom}(m.L_{H'}) \cup \{m.y\}] \\
& \text{because } m' \text{ does not appear in the event} \\
= & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(\ell(H)); m' := m.[x \mapsto u] : m'.e \in \mathsf{dom}(m.L_{H'}) \cup \{m.y\}] \\
& \text{since we imposed } x \neq y \\
\leq & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(\ell(H)); m' := m.[x \mapsto u] : \\
& \qquad\qquad g(m'.e) \in g(\mathsf{dom}(m.L_{H'})) \cup \{g(m.y)\}] \\
\leq & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(\ell(H)); m' := m.[x \mapsto u] : \\
& \qquad\qquad u \in g(\mathsf{dom}(m.L_{H'})) \cup \{g(m.y)\}] \\
& \text{since } g(m'.e) = m'.x \text{ by definition of g} \\
\leq & \frac{\chi_0(H)+1}{2^{\ell(H)}} \\
& \text{because } \mathsf{Card}(g(\mathsf{dom}(m.L_{H'}))) \leq \mathsf{Card}(\mathsf{dom}(m.L_{H'})) \leq \chi_0(H)
\end{array}
$$

Now, though we cannot directly use the compatibility lemma since it has only been proven for constructible distributions, and the rebinding operator does not yield a constructible distribution, a very similar demonstration justifies that $[\![x := \alpha \oplus H(y)]\!](X) \models \mathsf{H}(H; e; \varepsilon_1)$, where $\varepsilon_1(k, \chi, t) \leq \frac{\chi_0(H)+1}{2^{\ell(H)}} + \varepsilon(\chi_0)$. Since $[\![x := \alpha \oplus H(y)]\!](X) \in \mathfrak{const}\mathfrak{D}(\chi_0 + \mathbf{1}_H)$, we have our conclusion. ■

The next rule investigates the condition of improvement of a weak secrecy predicate $\mathsf{WS}(z; V_1; V_2; .)$ into $\mathsf{WS}(z; V_1, x; V_2; .)$. The intuition is that providing the adversary with knowledge of variable x is not of much help to compute z as long as x looks random.

Lemma V.17 (Rule (H4)). *The following preservation rule is sound provided that $x \neq y, z$:*

(H4) $\{\mathsf{WS}(y; V_1; V_2; \varepsilon_1) \wedge \mathsf{WS}(z; V_1; V_2; \varepsilon_2) \wedge \mathsf{H}(H; y; \varepsilon_3)\}\ x := \alpha \oplus H(y)$
$\{\mathsf{WS}(z; V_1, x; V_2; \varepsilon_4)\}$
where $\varepsilon_4(k, \chi, t) = k(H) * \varepsilon_1(k, \chi - \mathbf{1}_H, t) + \varepsilon_2(k, \chi - \mathbf{1}_H, t) + \varepsilon_3(\chi - \mathbf{1}_H)$.

Proof. Let $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$ satisfying our assumptions. First, we use rule (R4) and $x \neq y$, to deduce $\nu x.X \models \mathsf{WS}(y; V_1, x; V_2; \varepsilon_1')$ from $X \models \mathsf{WS}(y; V_1; V_2; \varepsilon_1)$, with $\varepsilon_1'(k, \chi, t) \leq \varepsilon_1(k, \chi_0, t)$. Then, from the hash vs. rebind lemma V.8 applied on $\nu x.X$, we obtain that $\nu x.X \sim_{V_1,x;V_2;\varepsilon_1''} \mathsf{rebind}_H^{y\mapsto\alpha\oplus x}(\nu x.X)$, where $\varepsilon_1''(k, t) = k(H) * \varepsilon_1(k, \chi_0, t)$.

Now, using the assumption $X \models \mathsf{H}(H; y; \varepsilon_3)$ and the rebinding lemma V.7, we obtain $\mathsf{rebind}_H^{y\mapsto\alpha\oplus x}(\nu x.X) \sim_{\varepsilon_3(\chi_0)} [\![x := \alpha \oplus H(y)]\!](X)$. In turn, this statement yields that $\mathsf{rebind}_H^{y\mapsto\alpha\oplus x}(\nu x.X) \sim_{V_1,x;V_2;\varepsilon_3(\chi_0)} [\![x := \alpha \oplus H(y)]\!](X)$.

Hence, by transitivity, $\nu x.X \sim_{V_1,x;V_2;\varepsilon_1''(k,t)+\varepsilon_3(\chi_0)} [\![x := \alpha \oplus H(y)]\!](X)$. Besides, as $X \models \mathsf{WS}(z; V_1; V_2; \varepsilon_2)$, rule (R4) along with $x \neq z$ allows to deduce that $\nu x.X \models \mathsf{WS}(z; V_1, x; V_2; \varepsilon_2')$ where $\varepsilon_2'(k, \chi, t) \leq \varepsilon_2(k, \chi_0, t)$. Thanks to the compatibility lemma V.1, we can conclude that $[\![x := \alpha \oplus H(y)]\!](X) \models \mathsf{WS}(z; V_1, x; V_2; \varepsilon_4')$, where $\varepsilon_4'(k, \chi, t) \leq k(H) * \varepsilon_1(k, \chi_0, t) + \varepsilon_2(k, \chi_0, t) + \varepsilon_3(\chi_0)$.

Since $[\![x := \alpha \oplus H(y)]\!](X)$ belongs to $\mathfrak{const}\mathfrak{D}(\chi_0 + \mathbf{1}_H)$, we have $[\![x := \alpha \oplus H(y)]\!](X) \models \mathsf{WS}(z; V_1, x; V_2; \varepsilon_4)$. ■

Rule (H5) looks into the preservation of predicate $\mathsf{H}(H; e; .)$ after the execution of a hash command using H on a variable y. We have to consider the possibility that L_H is extended

with y during the execution, which can potentially increase the probability that $e \in L_H$. The idea is that, if there is in e a subvariable z such that the probability to retrieve z from y is bounded by ε', then we can use it to bound the increase of probability that $e \in L_H$.

Lemma V.18 (Rule (H5)). *The following preservation rule is sound provided that* $z \in \mathsf{subvar}(e) \wedge x \notin \mathsf{subvar}(e)$*:*

(H5) $\{\mathsf{H}(H; e; \varepsilon) \wedge \mathsf{WS}(z; y; \emptyset; \varepsilon')\}\ x := \alpha \oplus H(y)\ \{\mathsf{H}(H; e; \varepsilon'')\}$
where $\varepsilon''(\chi) = \varepsilon(\chi - \mathbf{1}_H) + \varepsilon'(0, \chi - \mathbf{1}_H, 0)$.

Proof. Since $z \in \mathsf{subvar}(e)$, there is a function g such that for every m, $g(m.e) = m.z$. Let $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$.

$$\begin{array}{rl}
& \Pr[m \leftarrow X; m' \leftarrow [\![x := \alpha \oplus H(y)]\!](m) : m'.e \in \mathsf{dom}(m'.L_H)] \\
= & \Pr[m \leftarrow X; m' \leftarrow [\![x := \alpha \oplus H(y)]\!](m) : m'.e \in \mathsf{dom}(m.L_H) \cup \{m.y\}] \\
& \text{using that } \mathsf{dom}(m'.L_H) = \mathsf{dom}(m.L_H) \cup \{m.y\}, \\
= & \Pr[m \leftarrow X; m' \leftarrow [\![x := \alpha \oplus H(y)]\!](m) : m.e \in \mathsf{dom}(m.L_H) \cup \{m.y\}] \\
& m.e = m'.e \text{ because of the rebinding definition and } x \notin \mathsf{subvar}(e) \\
= & \Pr[m \leftarrow X : m.e \in \mathsf{dom}(m.L_H) \cup \{m.y\}] \\
& \text{since } m' \text{ does not appear in the event} \\
\leq & \Pr[m \leftarrow X : m.e \in \mathsf{dom}(m.L_H)] + \Pr[m \leftarrow X : m.e = m.y] \\
\leq & \varepsilon(\chi_0) + \Pr[m \leftarrow X : m.e = m.y]
\end{array}$$

The second term can be bounded as follows:

$$\begin{array}{rl}
\Pr[m \leftarrow X : m.e = m.y] & \leq \Pr[m \leftarrow X : g(m.e) = g(m.y)] \\
& = \Pr[m \leftarrow X : m.z = g(m.y)] \\
& \text{by definition of } g \\
& \leq \varepsilon'(0, \chi_0, 0) \\
& \text{seeing } g \text{ as an adversarial function.}
\end{array}$$

Consequently, we obtain that $\varepsilon''(\chi) \leq \varepsilon(\chi_0) + \varepsilon'(0, \chi_0, 0)$. With $[\![x := \alpha \oplus H(y)]\!](X) \in \mathfrak{const}\mathfrak{D}(\chi_0 + \mathbf{1}_H)$, we obtain $\varepsilon''(\chi) \leq \varepsilon(\chi - \mathbf{1}_H) + \varepsilon'(0, \chi - \mathbf{1}_H, 0)$, our result. ■

The last two of rules are more instances of slightly improved preservation rules, which, again, are based on the idea that providing a random-looking x should not help any adversary to solve a challenge.

Lemma V.19 (Rule (H6)). *The following preservation rule is sound provided that* $x \neq y$*:*

(H6) $\{\mathsf{WS}(y; V_1; V_2, y; \varepsilon) \wedge \mathsf{H}(H; y; \varepsilon')\}\ x := \alpha \oplus H(y)\ \{\mathsf{WS}(y; V_1, x; V_2, y; \varepsilon'')\}$
where $\varepsilon''(k, \chi, t) = \varepsilon(k, \chi - \mathbf{1}_H, t + (k(H) + 1) * \mathsf{T}_f) + \varepsilon'(\chi - \mathbf{1}_H) + \varepsilon(k, \chi - \mathbf{1}_H, t)$.

Proof. Let $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$ satisfy our assumptions. From the first hypothesis, the fact that $x \neq y$ and rule (R3), we get that $\nu x.X \models \mathsf{WS}(y; V_1, x; V_2, y; \varepsilon_1)$, where $\varepsilon_1(k, \chi, t) \leq \varepsilon(k, \chi_0, t)$. Using the weak hash vs. rebind lemma V.9, we deduce that $\nu x.X \sim_{V_1, x; V_2, y; \varepsilon_1'} \mathsf{rebind}_H^{y \mapsto \alpha \oplus x}(\nu x.X)$, where $\varepsilon_1'(k, t) = \varepsilon(k, \chi_0, t + k(H) * \mathsf{T}_f)$. Also, the rebinding lemma V.7 (with $\mathsf{H}(H; y; \varepsilon')$) implies that $\mathsf{rebind}_H^{y \mapsto \alpha \oplus x}(\nu x.X) \sim_{\varepsilon'(\chi_0)} [\![x := \alpha \oplus H(y)]\!](X)$. If we restrict this last statement, and use transitivity, we can conclude that $\nu x.X \sim_{V_1, x; V_2, y; \varepsilon_1''} [\![x := \alpha \oplus H(y)]\!](X)$ with $\varepsilon_1''(k, t) = \varepsilon(k, \chi_0, t + k(H) * \mathsf{T}_f) + \varepsilon'(\chi_0)$.

Finally, we apply the compatibility lemma V.1 to $\nu x.X$ and $[\![x := \alpha \oplus H(y)]\!](X)$, which result in $[\![x := \alpha \oplus H(y)]\!](X) \models \mathsf{WS}(y; V_1, x; V_2, y; \varepsilon'')$ with $\varepsilon''(k, \chi, t) \leq \varepsilon_1''(k, t + \mathsf{T}_f) + \varepsilon_1(k, \chi, t)$.

In other words, we can choose $\varepsilon''(k, \chi, t) = \varepsilon(k, \chi - \mathbf{1}_H, t + (k(H) + 1) * \mathsf{T}_f) + \varepsilon'(\chi - \mathbf{1}_H) + \varepsilon(k, \chi - \mathbf{1}_H, t)$. ■

Lemma V.20 (Rule (H7)). *The following preservation rule is sound provided that $x \neq y, z$:*

(H7) $\{\mathit{Indis}(z; V_1, z; V_2; \varepsilon_1) \wedge \mathit{WS}(y; V_1, z; V_2; \varepsilon_2) \wedge \mathit{H}(H; y; \varepsilon_3)\}\ x := \alpha \oplus H(y)\ \{\mathit{Indis}(z; V_1, z, x; V_2; \varepsilon_4)\}$
where $\varepsilon_4(k, \chi, t) = \varepsilon_1(k, \chi - \mathbf{1}_H, t) + 2.k(H) * \varepsilon_2(k, \chi - \mathbf{1}_H, t) + 2.\varepsilon_3(\chi - \mathbf{1}_H)$.

Proof. Consider any $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$ satisfying the three predicates. From the first two predicates and rules (R3) and (R4) (which we can use since $x \neq z, y$), we get that $\nu x.X \models \mathsf{Indis}(z; V_1, z, x; V_2; \varepsilon_1') \wedge \mathsf{WS}(y; V_1, z, x; V_2; \varepsilon_2')$, with $\varepsilon_1'(k, \chi, t) \leq \varepsilon_1(k, \chi_0, t)$ and $\varepsilon_2'(k, \chi, t) \leq \varepsilon_2(k, \chi_0, t)$. Therefore, the weak secrecy predicate combined with the hash vs rebind lemma V.8 allows to conclude that

$$\mathsf{rebind}_H^{y \mapsto \alpha \oplus x}(\nu x.X) \sim_{V_1, z, x; V_2; \varepsilon} \nu x.X$$

with $\varepsilon(k, t) = k(H) * \varepsilon_2(k, \chi_0, t)$. Besides, the rebinding lemma V.7 and $\mathsf{H}(H; y; \varepsilon_3)$ entails $[\![x := \alpha \oplus H(y)]\!](X) \sim_{\varepsilon_3(\chi_0)} \mathsf{rebind}_H^{y \mapsto \alpha \oplus x}(\nu x.X)$. By transitivity, we thus can state $[\![x := \alpha \oplus H(y)]\!](X) \sim_{V_1, z, x; V_2; \varepsilon'} \nu x.X$, where $\varepsilon'(k, t) = \varepsilon(k, t) + \varepsilon_3(\chi_0)$. Remembering that $\nu x.X \models \mathsf{Indis}(z; V_1, z, x; V_2; \varepsilon_1')$, the compatibility lemma V.1 provides us with $[\![x := \alpha \oplus H(y)]\!](X) \models \mathsf{Indis}(z; V_1, z, x; V_2; \varepsilon_4')$ where $\varepsilon_4'(k, \chi, t) \leq \varepsilon_1'(k, \chi, t) + 2.\varepsilon'(k, t)$, which allows us to conclude by replacing terms to: $\varepsilon_4'(k, \chi, t) \leq \varepsilon_1(k, \chi_0, t) + 2.\varepsilon(k, t) + 2.\varepsilon_3(\chi_0)$. This justifies our choice for ε_4.

■

One-Way Permutation Rules. We provide four rules concerning the command $x := f(y)$, amongst which two are creation rules and the others deal with preservation of predicates. The first rule captures the fact that distinguishing argument of f or image by f from randomness amounts to the same thing.

Lemma V.21 (Rule (P1)). *The following rule is sound, if $x, y \notin V_1 \cup V_2$:*

(P1) $\{\mathit{Indis}(y; V_1; V_2, y; \varepsilon)\}\ x := f(y)\ \{\mathit{Indis}(x; V_1, x; V_2; \varepsilon)\}$

Proof. Let $X \in \mathfrak{const}\mathfrak{D}(\chi)$ be a distribution satisfying our hypothesis. We notice that

$$\begin{aligned}
& D(\nu y.X, V_1, V_2 \cup \{y\}) \\
= \ & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l); m' := m.[y \mapsto u] : ((m'.V_1), (f(m'.V_2), f(u)))] \\
= \ & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l); m' := m.[y \mapsto u, x \mapsto f(u)] : ((m'.V_1), (f(m'.V_2), m'.x)] \\
& \text{because } x \notin V_1 \cup V_2 \\
= \ & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l); m' := m.[y \mapsto f^{-1}(u), x \mapsto u] : \\
& \qquad\qquad (m'.V_1, (f(m'.V_2), m'.x))] \\
& \text{if we denote } f^{-1} \text{ the inverse of permutation } f \\
= \ & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l); m' := m.[x \mapsto u] : (m'.V_1, (f(m'.V_2), m'.x))] \\
& \text{because } y \notin V_1 \cup V_2 \\
= \ & \Pr[m \leftarrow X; m' := m.[x \mapsto f(m.y)]; u \leftarrow \mathcal{U}(l); m'' := m'.[x \mapsto u] : \\
& \qquad\qquad (m''.V_1, (f(m''.V_2), m''.x)] \\
= \ & D(\nu x.[\![x := f(y)]\!](X), V_1 \cup \{x\}, V_2)
\end{aligned}$$

Consequently, up to a reorganization of its inputs, any adversary against $[\![x := f(y)]\!](X) \models \mathsf{Indis}(x; V_1, x; V_2; \varepsilon)$ can be turned into an adversary attacking $X \models \mathsf{Indis}(y; V_1; V_2, y; \varepsilon)$ with the same advantage.

■

The second rule that we present formalizes in our logic one-wayness of the function. Indeed, it states that given the image by f of a random-looking input as a challenge, an adversary's probability to compute a value for the input is bounded by that of inverting f.

Lemma V.22 (Rule (P2)). *The following rule is sound if $y \notin V_1 \cup \{x\}$:*

(P2) $\{\mathit{Indis}(y; V_1; V_2, y; \varepsilon)\}\ x := f(y)\ \{\mathit{WS}(y; V_1, x; V_2, y; \varepsilon')\}$
where $\varepsilon'(k, \chi, t) = \varepsilon(k, \chi, t + \mathsf{T}_f) + OW(t)$.

Proof. Let $X \in \mathfrak{const}\mathfrak{D}$, such that $X \models \mathsf{Indis}(y; V_1; V_2, y; \varepsilon)$. Since $y \notin V_1$, applying the hybrid weakening lemma V.4 allows to deduce that $X \models \mathsf{WS}(y; V_1; V_2, y; \varepsilon')$ with $\varepsilon'(k, \chi, t) = \varepsilon(k, \chi, t + \mathsf{T}_f) + OW(t)$. Moreover, as $f(y)$ is a constructible from $(V_1; V_2 \cup \{y\})$ in no time and $y \neq x$, then lemma V.6 provides us with $X \models \mathsf{WS}(y; V_1, x; V_2, y; \varepsilon')$. ■

Our third rule expresses a slightly improved preservation of indistinguishability from random: x appears in the conclusion.

Lemma V.23 (Rule (P3)). *The following rule is sound if $z \neq x, y$:*

(P3) $\{\mathit{Indis}(z; V_1, z; V_2, y; \varepsilon)\}\ x := f(y)\ \{\mathit{Indis}(z; V_1, z, x; V_2, y; \varepsilon)\}$.

Proof. This is a direct application of lemma V.6: we have $z \neq x$, $z \neq y$ implies that z does not syntactically appear in e and and $f(y)$ is constructible from $(V_1, z; V_2, y)$ in no time. ■

Last but not least, the fourth preservation rule allows to preserve and possibly improve a weak secrecy predicate. Indeed, the capacity to compute the value of a variable z given some sets V_1 and V_2 should not fundamentally change when additionally provided with x and $f(y)$ as long as y looks random w.r.t. V_1 and V_2.

Lemma V.24 (Rule (P4)). *The following rule is sound if $z \neq x, y$:*

(P4) $\{\mathit{WS}(z; V_1; V_2; \varepsilon) \wedge \mathit{Indis}(y; V_1; V_2, y, z; \varepsilon')\}\ x := f(y)\ \{\mathit{WS}(z; V_1, x; V_2, y; \varepsilon'')\}$
where $\varepsilon''(k, \chi, t) = \varepsilon(k, \chi, t + \mathsf{T}_f) + \varepsilon'(k, \chi, t + \mathsf{T}_f)$.

Proof. Notice that when $z = y$ this rule is actually an instance of (P2), so that we can impose $z \neq y$ without regret. Let $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$ be a distribution satisfying our two hypotheses. We start by rewriting $X \models \mathsf{Indis}(y; V_1; V_2, y, z; \varepsilon')\}$ as $X \sim_{V_1; V_2, y, z; \varepsilon'_1} \nu y.X$ with $\varepsilon'_1(k, t) = \varepsilon'(k, \chi_0, t)$. Lemma V.5 provides us with statement $[\![x := f(y)]\!](X) \sim_{V_1, x; V_2, y, z; \varepsilon'_1} [\![x := f(y)]\!](\nu y.X)$ (*).

We then propose ε_1 such that $[\![x := f(y)]\!](\nu y.X) \models \mathsf{WS}(z; V_1, x; V_2, y; \varepsilon_1)$. Let $\mathcal{A}^{\vec{H}}$ be a adversary against this last statement. We construct the following $(k, t + \mathsf{T}_f)$-adversary against our hypothesis predicate $X \models \mathsf{WS}(z; V_1; V_2; \varepsilon)$.

$$\begin{aligned} \mathcal{B}^{\vec{H}}(V_1; f(V_2)) = \quad & u \leftarrow \mathcal{U}(l) \\ & z_0 \leftarrow \mathcal{A}^{\vec{H}}(V_1, f(u); f(V_2), f(u)) \\ & \mathsf{return}\ z_0 \end{aligned}$$

Then, we show that the advantages of both adversaries coincide:

$$\begin{aligned}
& \Pr[m \leftarrow X : \mathcal{B}^{\vec{H}}(m.V_1; f(m.V_2)) = m.z] \\
= \; & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l) : \mathcal{A}^{\vec{H}}(m.V_1, f(u); f(m.V_2), f(u)) = m.z] \\
= \; & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l); m' = m.[y \mapsto u, x \mapsto f(u)] : \\
& \qquad \mathcal{A}^{\vec{H}}(m'.(V_1 \cup \{x\}); f(m'.(V_2 \cup \{y\}))) = m.z] \\
= \; & \Pr[m \leftarrow [\![x := f(y)]\!](\nu y.X) : \mathcal{A}^{\vec{H}}(m'.(V_1 \cup \{x\}); f(m'.(V_2 \cup \{y\}))) = m.z]
\end{aligned}$$

As $X \models \mathsf{WS}(z; V_1; V_2; \varepsilon)$, the probability that $\mathcal{B}$ wins is bounded by $\varepsilon(k, \chi, t + \mathsf{T}_f)$. Then, so is the advantage of $\mathcal{A}$. Therefore, we can choose $\varepsilon_1(k, \chi, t) = \varepsilon(k, \chi, t + \mathsf{T}_f)$.

The compatibility lemma V.1 applied to (*) and this latter statement enables us to deduce that $[\![x := f(y)]\!](X) \models \mathsf{WS}(z; V_1, x; V_2, y; \varepsilon_1'')$ where $\varepsilon_1''(k, \chi, t) \leq \varepsilon_1'(k, t + \mathsf{T}_f) + \varepsilon_1(k, \chi, t)$.

In conclusion, by replacing terms and remarking that $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$ iff $[\![x := f(y)]\!](X)$ belongs to it too, our function $\varepsilon''(k, \chi, t)$ can be chosen equal to $\varepsilon(k, \chi, t + \mathsf{T}_f) + \varepsilon'(k, \chi, t + \mathsf{T}_f)$. ■

Exclusive Or Rules. The following creation rule captures the properties of the execution of $x := y \oplus z$, assuming y and z are syntactically distinct. It can remind the reader of the properties of a one-time pad encryption of z with key y: even given x and z, if key y looks random enough, then so does x.

Lemma V.25 (Rule (X1)). *The following rule is sound if $y \notin V_1 \cup V_2$, $y \neq x, z$:*

(X1) $\{\mathsf{Indis}(y; V_1, y, z; V_2; \varepsilon)\} \;\; x := y \oplus z \;\; \{\mathsf{Indis}(x; V_1, x, z; V_2; \varepsilon)\}$

Proof. Let $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$ such that $X \models \mathsf{Indis}(y; V_1, y, z; V_2; \varepsilon)$. It can be rewritten as $X \sim_{V_1,y,z;V_2;\varepsilon(k,\chi_0,t)} \nu y.X$. With the fact that $y \oplus z$ is constructible from y and z and lemma V.5, we get $[\![x := y \oplus z]\!](X) \sim_{V_1,x,y,z;V_2;\varepsilon(k,\chi_0,t)} [\![x := y \oplus z]\!](\nu y.X)$, which we weaken into $[\![x := y \oplus z]\!](X) \sim_{V_1,x,z;V_2;\varepsilon(k,\chi_0,t)} [\![x := y \oplus z]\!](\nu y.X)$. Now we study this last distribution.

$$\begin{aligned}
& D([\![x := y \oplus z]\!](\nu y.X), V_1 \cup \{x, z\}, V_2) \\
= \; & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l); m' := m.[y \mapsto u, x \mapsto u \oplus m.z] : \\
& \qquad\qquad ((m'.V_1, m'.x, m'.z), f(m'.V_2))] \\
= \; & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l); m' := m.[y \mapsto u \oplus m.z, x \mapsto u] : \\
& \qquad\qquad ((m'.V_1, m'.x, m'.z), f(m'.V_2))] \\
& \text{since } \oplus \text{ is idempotent} \\
= \; & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l); m' := m.[x \mapsto u] : ((m'.V_1, m'.x, m'.z), f(m'.V_2))] \\
& \text{because } y \notin V_1 \cup V_2 \cup \{x, z\} \\
= \; & \Pr[m \leftarrow X; m' := m.[x \mapsto m.y \oplus m.z]; u \leftarrow \mathcal{U}(l); m'' := m'[x \mapsto u] : \\
& \qquad\qquad ((m''.V_1, m''.x, m''.z), f(m''.V_2))] \\
= \; & D(\nu x.[\![x := y \oplus z]\!](X), V_1 \cup \{x, z\}, V_2)
\end{aligned}$$

As a result of this and transitivity, $[\![x := y \oplus z]\!](X) \sim_{V_1,x,z;V_2;\varepsilon(k,\chi_0,t)} \nu x.[\![x := y \oplus z]\!](X)$, which in turn can be rewritten $[\![x := y \oplus z]\!](X) \models \mathsf{Indis}(x; V_1, x, z; V_2; \varepsilon(k, \chi_0, t))$. Considering that $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$ iff $[\![x := y \oplus z]\!](X) \in \mathfrak{const}\mathfrak{D}(\chi_0)$, we can conclude. ■

Preservation rules for exclusive or take advantage of the constructibility of $y \oplus z$ given y and z, and are simple applications of lemma V.6.

Lemma V.26 (Rules (X2) and (X3)). *The following rules are sound:*

(X2) $\{\mathsf{Indis}(w; V_1, y, z; V_2; \varepsilon)\}\ x := y \oplus z\ \{\mathsf{Indis}(w; V_1, x, y, z; V_2; \varepsilon)\}$, *if* $w \neq x, y, z$,

(X3) $\{\mathsf{WS}(w; V_1, y, z; V_2; \varepsilon)\}\ x := y \oplus z\ \{\mathsf{WS}(w; V_1, x, y, z; V_2; \varepsilon)\}$, *if* $w \neq x$,

Concatenation Rules. We have two creation rules to deal with concatenation command $x := y||z$. The first rule, (C1), states that computing a value for x is at least as difficult as computing a substring of x.

Lemma V.27 (Rule (C1)). *The following rule is sound if* $x \notin V_1 \cup V_2$*:*

(C1) $\{\mathsf{WS}(y; V_1; V_2; \varepsilon)\}\ x := y||z\ \{\mathsf{WS}(x; V_1; V_2; \varepsilon)\}$

A dual rule applies for z*.*

Proof. The proof captures the idea that out of any adversary $\mathcal{A}^{\vec{H}}$ computing x one can obtain an adversary $\mathcal{B}^{\vec{H}}$ computing y by extracting the head of the bitstring computed for x. We disregard the time necessary to the extraction. The bounding of $\mathcal{A}^{\vec{H}}$'s advantage is justified by the following computation:

$$\begin{aligned}
& \Pr[m \leftarrow [\![x := y||z]\!](X) : \mathcal{A}^{\vec{H}}(m.V_1, m.V_2) = m.x] \\
= \ & \Pr[m \leftarrow X; m' := m.[x \mapsto m.y||m.z] : \mathcal{A}^{\vec{H}}(m'.V_1, m'.V_2) = m'.x] \\
= \ & \Pr[m \leftarrow X; m' := m.[x \mapsto m.y||m.z] : \mathcal{A}^{\vec{H}}(m'.V_1, m'.V_2) = m.y||m.z] \\
= \ & \Pr[m \leftarrow X; m' := m.[x \mapsto m.y||m.z] : \mathcal{A}^{\vec{H}}(m.V_1, m.V_2) = m.y||m.z] \\
& \text{because } x \notin V_1 \cup V_2 \\
= \ & \Pr[m \leftarrow X : \mathcal{A}^{\vec{H}}(m.V_1, m.V_2) = m.y||m.z] \\
= \ & \Pr[m \leftarrow X : \mathcal{B}^{\vec{H}}(m.V_1, m.V_2) = m.y] \\
\leq \ & \varepsilon(k, \chi, t)
\end{aligned}$$

■

The idea behind (C2), our second creation rule, is that y and z being random implies randomness of x, with respect to V_1 and V_2. Of course, y has to be random given y *and* z and not just only y; otherwise, there might exist a dependency between both substrings of x that allows an adversary to distinguish this latter from a random value. Obviously the same goes for z.

Lemma V.28 (Rule (C2)). *The following rule is sound if* $y, z \notin V_1 \cup V_2 \cup \{x\}$*:*

(C2) $\{\mathsf{Indis}(y; V_1, y, z; V_2; \varepsilon) \wedge \mathsf{Indis}(z; V_1, y, z; V_2; \varepsilon')\}\ x := y||z\ \{\mathsf{Indis}(x; V_1, x; V_2; \varepsilon + \varepsilon')\}$,

Proof. Let $X \in \mathfrak{const}_{\mathfrak{D}}(\chi_0)$ be a distribution satisfying both premises. On the one hand, $X \models \mathsf{Indis}(y; V_1, y, z; V_2; \varepsilon)$ can be written as $X \sim_{V_1,y,z;V_2;\varepsilon_1} \nu y.X$, where $\varepsilon_1(k,t) = \varepsilon(k, \chi_0, t)$. Therefore, $\nu z.X \sim_{V_1,y,z;V_2;\varepsilon_2} \nu z.\nu y.X$, where $\varepsilon_2(k,t) = \varepsilon(k, \chi_0, t)$. On the other hand, $X \models \mathsf{Indis}(z; V_1, y, z; V_2; \varepsilon')$ rewrites as $X \sim_{V_1,y,z;V_2;\varepsilon'_1} \nu z.X$ where $\varepsilon'_1(k,t) = \varepsilon'(k, \chi_0, t)$. By transitivity, we deduce that $X \sim_{V_1,y,z;V_2;\varepsilon_2+\varepsilon'_1} \nu z.\nu y.X$. As $y||z$ is constructible from y and z, lemma V.5 provides us with $[\![x := y||z]\!](X) \sim_{V_1,x,y,z;V_2;\varepsilon_3} [\![x := y||z]\!](\nu z.\nu y.X)$, where $\varepsilon_3(k,t) = \varepsilon_2(k,t) + \varepsilon'_1(k,t)$. This statement can be weakened as $[\![x := y||z]\!](X) \sim_{V_1,x;V_2;\varepsilon_3} [\![x := y||z]\!](\nu z.\nu y.X)$

Now, we show that $D([\![x := y||z]\!](\nu z.\nu y.X), V_1 \cup \{x\}, V_2) = D(\nu x.[\![x := y||z]\!](X), V_1 \cup \{x\}, V_2)$:

$$\begin{aligned}
& D([\![x := y||z]\!](\nu z.\nu y.X), V_1 \cup \{x\}, V_2) \\
= \;& D([m \leftarrow \nu z.\nu y.X : m.[x \mapsto m.y||m.z]], V_1 \cup \{x\}, V_2) \\
= \;& D([m \leftarrow X; u \leftarrow \mathcal{U}(l); v \leftarrow \mathcal{U}(l') : \\
& \qquad\qquad m[y \mapsto u, z \mapsto v, x \mapsto u||v]], V_1 \cup \{x\}, V_2) \\
= \;& D([m \leftarrow X; w \leftarrow \mathcal{U}(l + l') : m.[x \mapsto w]], V_1 \cup \{x\}, V_2) \\
& \text{because } y, z \notin V_1 \cup V_2 \cup \{x\} \\
= \;& D(m \leftarrow \nu x.X, V_1 \cup \{x\}, V_2)
\end{aligned}$$

In conclusion, $[\![x := y||z]\!](X) \sim_{V_1,x;V_2;\varepsilon_3} \nu x.[\![x := y||z]\!](X)$ by transitivity. Rewritten as $[\![x := y||z]\!](X) \models \mathsf{Indis}(x; V_1, x; V_2; \varepsilon_3)$, together with $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$ iff $[\![x := y||z]\!](X) \in \mathfrak{const}\mathfrak{D}(\chi_0)$, it yields our conclusion. ■

Finally, the following preservation rules follow directly from lemma V.6: capacity to distinguish t from random or to compute it are not improved by the additional knowledge of $y||z$ when one already is provided with y and z.

Lemma V.29 (Rules (C3) and (C4)). *The following rules are sound:*

(C3) $\{\mathsf{Indis}(w; V_1, y, z; V_2; \varepsilon)\}\ x := y||z\ \{\mathsf{Indis}(w; V_1, x, y, z; V_2; \varepsilon)\}$, *if* $w \neq x, y, z$,

(C4) $\{\mathsf{WS}(w; V_1, y, z; V_2; \varepsilon)\}\ x := y||z\ \{\mathsf{WS}(w; V_1, y, z, x; V_2; \varepsilon)\}$, *if* $w \neq x$.

Additional General Rules. To reason on programs built according to the language grammar described in Table V.1, we additionally need the two following rules.

Lemma V.30. *Let $\varphi_0, \varphi_1, \varphi_2, \varphi_3$ be assertions from our language, and $\mathsf{c}, \mathsf{c}_1, \mathsf{c}_2$ be any commands. The following rules are sound:*

— *(Csq) if* $\varphi_0 \Rightarrow \varphi_1$ *and* $\{\varphi_1\}\ \mathsf{c}\ \{\varphi_2\}$ *and* $\varphi_2 \Rightarrow \varphi_3$ *then* $\{\varphi_0\}\ \mathsf{c}\ \{\varphi_3\}$.

— *(Seq) if* $\{\varphi_0\}\ \mathsf{c}_1\ \{\varphi_1\}$ *and* $\{\varphi_1\}\ \mathsf{c}_2\ \{\varphi_2\}$ *then* $\{\varphi_0\}\ \mathsf{c}_1; \mathsf{c}_2\ \{\varphi_2\}$.

— *(Conj) if* $\{\varphi_0\}\ \mathsf{c}\ \{\varphi_1\}$ *and* $\{\varphi_0\}\ \mathsf{c}\ \{\varphi_2\}$, *then* $\{\varphi_0\}\ \mathsf{c}\ \{\varphi_1 \wedge \varphi_2\}$.

We omit the proofs of these classical rules. The soundness of the Hoare logic follows by induction from the soundness of each rule.

V.4 Verification Procedure and Interface to CIL

V.4.1 — Public-Key Oracle Systems

The asymmetric-dedicated framework can be related quite easily to a subset of oracle systems. Namely, constructible distributions call to mind oracle systems containing only random oracles in addition to one other oracle whose implementation can be written in the programming language defined in this chapter. We consider fixed a trapdoor algorithm and a matching key-generation procedure, together with a finite collection of hash functions $\vec{H}$.

Definition (Oracle Declaration). An oracle declaration is of the form $o(q, a) : \mathbf{var}\ V; \mathsf{c}$, where o is an oracle name, c is a command in our language, q and a are respectively the input and output variables, and variables of $V \subseteq \mathsf{Var}$ are those used in the procedure in addition to q and a. We also require that no assignment to variable q is performed by command c. □

Definition (Public-Key Oracle System). We let $o(q,a)$: **var** V; c be an oracle declaration from the previous programming language. The public-key oracle system $\mathbb{O}(\mathsf{c})$ based on this declaration is given by:

— memories $\mathsf{M}_{\mathbb{O}}$ containing variables pk, sk, one list L_H for every element in $\vec{H}$ and a list L_o.

— in addition to the initialization and finalization oracles, the system contains an oracle for each element in $\vec{H}$ and oracle o. They are implemented as follows:

1. for the initialization oracle:
$\mathsf{Imp}(o_{\mathrm{I}})(\_,\_) = (pk_0, sk_0) \leftarrow \mathcal{K};$
$\mathsf{return}\ (pk_0, [pk \mapsto pk_0, sk \mapsto sk_0, (L_H \mapsto [\,])_{H\in\vec{\mathcal{H}}}, L_o \mapsto [\,]])$
2. every $H \in \vec{H}$ is implemented as the functional random oracle with outputs of length $\ell(H)$[4],
3. for oracle o:
$\mathsf{Imp}(o)(q,m) =$ $\mathsf{if}\ q \in \mathsf{dom}(m.L_o)\ \mathsf{then}$
$\mathsf{return}\ (L_o(q), m)$
else
$\mathsf{let}\ m' \leftarrow [\![\mathsf{c}]\!](m)\ \mathsf{in}$
$\mathsf{return}\ (m'.a, m'.[L_o \mapsto L_o :: (q, m'.a, m'.V), V \mapsto m.V])$
where $V \mapsto m.V$ is short for the assignment to every variable in V of its value according to m.
4. finally, the finalization oracle is chosen to have a trivial idle implementation, but we do not fix its input type, which can be set according to the context of use of the system (e.g. $\mathsf{Res} = \mathsf{Bool}$ to use the system in an indistinguishability statement).

This defines the *deterministic* public-key oracle system matching a declaration. The *probabilistic* version of the system is identical except for the implementation of o, which does not test for belonging in list L_o and directly applies c.

□

The list L_o stores values of every query, along with final values of variables in V used to compute an answer, and the answer itself. We reset the values of variables in V after every call to the oracle o. As a result, variables of V are local variables of the oracle implementation. In the remainder, we come to use a restriction of the list L_o containing only the pairs of query and answer, which is the part of the list visible to the adversary. This restricted list is denoted by $(L_o)_{|(q,a)}$.

V.4.2 — Plug-In Theorems

Theorem V.31. *Let $\mathbb{O}(\mathsf{c})$ be the public-key oracle system based on declaration $o(q,a)$: **var** V; c, with the finalization oracle taking boolean as inputs. The following rule is sound:*

$$\frac{\{\mathsf{true}\}\ \mathsf{c}\ \{\mathit{Indis}(a; q, a, \sigma; \emptyset; \varepsilon)\}}{\mathbb{O}(\mathsf{c}) \sim_{k(o)*\varepsilon(k,k,t)} \mathbb{O}(\mathsf{c}; a \leftarrow \mathcal{U}(l))}\ \mathit{Indis}$$

Before starting with the proof, we introduce the following notation. For a given integer i, we denote $\mathbb{O}(\mathsf{c}/\mathsf{c}; a \leftarrow \mathcal{U}(l))[i]$ the oracle system whose oracle o is implemented as in $\mathbb{O}(\mathsf{c})$ until it has been queried on i fresh queries, and as $\mathbb{O}(\mathsf{c}; a \leftarrow \mathcal{U}(l))$ for the remaining queries. As an example, $\mathbb{O}(\mathsf{c}/\mathsf{c}; a \leftarrow \mathcal{U}(l))[0]$ actually is $\mathbb{O}(\mathsf{c}; a \leftarrow \mathcal{U}(l))$.

[4] See definition in subsection III.5.3 of the previous chapter.

Proof. Let $\mathbb{A}$ be an $\mathbb{O}(\mathsf{c})$-adversary whose resources are bounded by (k,t). To prove the theorem, we have to show that we have:

$$|Pr(\mathbb{A}|\mathbb{O}(\mathsf{c}) : \mathrm{R} = \mathsf{true}) - Pr(\mathbb{A}|\mathbb{O}(\mathsf{c}; a \leftarrow \mathcal{U}(l)) : \mathrm{R} = \mathsf{true})| \leq k(o) * \varepsilon(k,k,t).$$

In fact, we notice that system $\mathbb{O}(\mathsf{c})$ coincides with $\mathbb{O}(\mathsf{c}/\mathsf{c}; a \leftarrow \mathcal{U}(l))[k(o)]$, while $\mathbb{O}(\mathsf{c}; a \leftarrow \mathcal{U}(l))$ corresponds to $\mathbb{O}(\mathsf{c}/\mathsf{c}; a \leftarrow \mathcal{U}(l))[0]$. Consequently, it is sufficient to show the inequality for those systems.

In order to obtain such a statement, we apply what is seemingly a hybrid argument: we prove that for any i in $[0..(k(o)-1)]$,

$$\begin{aligned}|&Pr(\mathbb{A}|\mathbb{O}(\mathsf{c}/\mathsf{c}; a \leftarrow \mathcal{U}(l))[i] : \mathrm{R} = \mathsf{true}) \\ &\qquad -Pr(\mathbb{A}|\mathbb{O}(\mathsf{c}/\mathsf{c}; a \leftarrow \mathcal{U}(l))[i+1] : \mathrm{R} = \mathsf{true})| \leq \varepsilon(k,k,t)\end{aligned}$$

and can then conclude using triangle inequality.

We let i be a fixed integer in $[0..(k(o)-1)]$, and $\mathbb{A}$ an $\mathbb{O}(\mathsf{c})$-adversary trying to distinguish between $\mathbb{O}(\mathsf{c}/\mathsf{c}; a \leftarrow \mathcal{U}(l))[i]$ and $\mathbb{O}(\mathsf{c}/\mathsf{c}; a \leftarrow \mathcal{U}(l))[i+1]$. In other words, $\mathbb{A}$ has to find out whether the $(i+1)$-th fresh query it performs to oracle o is followed by a redraw of a in the end. The idea is to relate the advantage of $\mathbb{A}$ to that of an adversary $\mathcal{A}$ trying to break our indistinguishability premise. To do so, we are going to decompose the interaction of $\mathbb{A}$ with its oracle system into the three parts appearing in the statement of our hypothesis: the first one corresponds to a constructible distribution, the second one to the issue of a challenge obtained by applying the command and the third and last one is the execution of an adversary $\mathcal{A}$.

In the remainder of this proof, we write $\mathbb{O}$ when the oracle system we consider can either be $\mathbb{O}(\mathsf{c}/\mathsf{c}; a \leftarrow \mathcal{U}(l))[i]$ or $\mathbb{O}(\mathsf{c}/\mathsf{c}; a \leftarrow \mathcal{U}(l))[i+1]$. Firstly, we let $(\mathbb{A} \mid \mathbb{O})[i]$ denote the transition system which can be obtained by running $\mathbb{A} \mid \mathbb{O}$ until reaching the step during which the i-th fresh query is performed to o, and outputting result of this step, or outputting the last step's result if $\mathbb{A} \mid \mathbb{O}$ never reaches such a step. This transition system $(\mathbb{A} \mid \mathbb{O})[i]$ yields a distribution on $\mathsf{Xch} \times \mathsf{M}_{\mathbb{O}} \times \mathsf{M}_{\mathbb{A}}$. We assume the existence of a function $\mathsf{Encode}(.)$ which takes as input a state and encodes it as a bitstring of finite length. Furthermore, this function is assumed to admit an inverse that we denote $\mathsf{Decode}(.)$. We consider the distribution X obtained as follows:

$$\begin{aligned}X = \;& \mathsf{let}\ ((o', q', \_), m_{\mathbb{O}}, m_{\mathbb{A}}) \leftarrow (\mathbb{A} \mid \mathbb{O})[i]\ \mathsf{in} \\ & \mathsf{if}\ o' = o\ \mathsf{then} \\ & \mathsf{let}\ ((o, q_{i+1}), m'_{\mathbb{A}}) \leftarrow \mathsf{A}(m_{\mathbb{A}})\ \mathsf{in} \\ & \mathsf{return}\ [(pk, sk, L_H) \mapsto m_{\mathbb{O}}, \sigma \mapsto \mathsf{Encode}((m_{\mathbb{A}}, (m_{\mathbb{O}}.L_o)_{|(q,a)})), q \mapsto q_{i+1}] \\ & \mathsf{else}\ \ /* \text{ here, we have } o' = o_{\mathrm{F}} \ */ \\ & \mathsf{return}\ [(pk, sk, L_H) \mapsto m_{\mathbb{O}}, \sigma \mapsto \mathsf{Encode}([(o', q')]), q \mapsto \lambda]\end{aligned}$$

where $(pk, sk, L_H) \mapsto m_{\mathbb{O}}$ is short for "variables pk, sk and all lists L_H receive the values stored for them in state $m_{\mathbb{O}}$". This distribution is constructible for some function χ such that $\chi(H) \leq k(H)$. Indeed, $(\mathbb{A} \mid \mathbb{O})[i]$ is a sequence of alternating adversarial querying and update functions A and $\mathsf{A}_{\downarrow}$, calls to hash oracles and calls to oracle o. In addition to this, these latter calls consist in the execution of the implementation of o in $\mathbb{O}(\mathsf{c})$. As far as calls to hash oracles are concerned, we have no knowledge of when $\mathbb{A}$ places them, so that we can only say that their number is bounded by function k.

Our premise provides us with $\mathsf{c}(X) \models \mathsf{Indis}(a; q, a, \sigma; \emptyset; \varepsilon)$.

We now build a specific adversary $\mathcal{A}^{\vec{H}}$ against this statement. It takes as inputs values q, a and σ, and first runs the decoding algorithm $\mathsf{Decode}(\sigma)$ to retrieve $m'_{\mathbb{A}}$ and a list L or $[(o_{\mathrm{F}}, q')]$ from σ. We are sure that for any state output by X, $\mathsf{c}(X)(m).\sigma = m.\sigma$ and $(\mathsf{c}; a \leftarrow \mathcal{U}(l))(X)(m).\sigma = m.\sigma$, because commands of the language are designed to leave the adversarial variable σ unchanged. If the decoding results in $[o_{\mathrm{F}}, q']$, then $\mathcal{A}$ forwards q' as its answer. Otherwise, $\mathcal{A}$ goes on with updating L with the new pair (q, a).

After that, the idea is that $\mathcal{A}$ runs $\mathbb{A}$ as a subroutine and forwards this latter's answer. To do so, $\mathcal{A}$ runs adversary $\mathbb{A}$'s update function $\mathsf{A}_{\downarrow}$ on $((o, q, a), m'_{\mathbb{A}})$, and then goes on running $\mathbb{A}$. The oracle calls that $\mathbb{A}$ places to hash functions are forwarded by $\mathcal{A}$ to its own hash oracles. As previously, we have no way of providing a better bound to the number of hash function calls performed by $\mathbb{A}$ *after* its i-th fresh call to o than by means of function k. Of course, for every execution, we know that the *total* number of hash calls is bounded by k. However, since $\mathbb{A}$ is probabilistic, there may not exist a tighter function bounding calls performed before (resp. after) $\mathbb{A}$'s i-th fresh call to o uniformly over all the executions.

The oracle calls to o are simulated by $\mathcal{A}$ as follows. $\mathcal{A}$ knows the list of calls previously performed by $\mathbb{A}$ (they appear in L), so that it can answer consistently to potential queries performed twice, and remaining fresh calls are answered by executing the command $\mathsf{c}; a \leftarrow \mathcal{U}(l)$ and updating L.

We have built X and $\mathcal{A}^{\vec{H}}$ such that they perfectly simulate the execution of $\mathbb{A} \mid \mathbb{O}$. Therefore,

$$\begin{aligned} & |\Pr(\mathbb{A}|\mathbb{O}(\mathsf{c}/\mathsf{c}; a \leftarrow \mathcal{U}(l))[i] : \mathrm{R} = \mathsf{true}) \\ & \qquad - \Pr(\mathbb{A}|\mathbb{O}(\mathsf{c}/\mathsf{c}; a \leftarrow \mathcal{U}(l))[i+1] : \mathrm{R} = \mathsf{true})| \\ = \; & |\Pr[m \leftarrow [\![\mathsf{c}]\!](X) : \mathcal{A}^{\vec{H}}(m.q, m.a, m.\sigma) = \mathsf{true}] \\ & - \Pr[m \leftarrow \nu a.[\![\mathsf{c}]\!](X) : \mathcal{A}^{\vec{H}}(m.q, m.a, m.\sigma) = \mathsf{true}]| \\ \leq \; & \varepsilon(k, k, t) \end{aligned}$$

■

A good example of use of this first theorem is when the oracle declaration is an encryption algorithm. Then, if we manage to derive the premise $\mathsf{Indis}(a; q, a, \sigma; \emptyset; \varepsilon)$ for some function ε, we can use the theorem to conclude to $\mathbb{O}(\mathsf{c}) \sim_{k(o)*\varepsilon(k,k,t)} \mathbb{O}(\mathsf{c}; a \leftarrow \mathcal{U}(l))$. This is nearly the statement in CIL of ROR-ciphertext security of the encryption algorithm, which would be $\mathbb{O}(\mathsf{c}) \sim_{k(o)*\varepsilon(k,k,t)} \mathbb{O}(a \leftarrow \mathcal{U}(l))$. An application of bisimulation rule can bridge the gap by providing $\mathbb{O}(\mathsf{c}; a \leftarrow \mathcal{U}(l)) \sim_0 \mathbb{O}(a \leftarrow \mathcal{U}(l))$, using as a relation the equality on common components of states.

Theorem V.32. *Let $\mathbb{O}(\mathsf{c})$ be the public-key oracle system based on the declaration $o(q, a)$: **var** V; c, with the finalization oracle taking bitstrings of length l as inputs. Let x be one of the local variables used in c, taking values in $\{0,1\}^l$. The operator Π_x maps list L_o of tuples of the form (q, a, V) to the corresponding list of values taken by x. The following rule is sound:*

$$\frac{\{\mathsf{true}\}\ \mathsf{c}\ \{\mathsf{WS}(x; q, a, \sigma; \emptyset; \varepsilon)\}}{\mathbb{O}(\mathsf{c}) :_{k(o)*\varepsilon(k,k,t)} \mathrm{R} \in \Pi_x(L_o)}\ \mathsf{WS}$$

Proof. Let i be an integer between 1 and $k(o)$, and $\mathbb{A}$ be an $\mathbb{O}(\mathsf{c})$-adversary. We reuse elements of the proof of theorem V.31 to prove that:

$$\mathbb{O}(\mathsf{c}) :_{\varepsilon(k,k,t)} \mathrm{R} = \Pi_x(L_o[i])$$

where we recall that $L_o[i]$ is the i-th element of list L_o.

We consider distribution X defined from $(\mathbb{A} \mid \mathbb{O})[i]$ as the previous proof.

Our premise provides us with $\mathsf{c}(X) \models \mathsf{WS}(x; q, a, \sigma; \emptyset; \varepsilon)$.

We now build a specific adversary $\mathcal{A}^{\vec{H}}$ trying to compute a value for x. It takes as inputs values q, a and σ, and first runs the decoding algorithm $\mathsf{Decode}(\sigma)$ to retrieve $m'_{\mathbb{A}}$ and a list of queries and answers L or $[(o_{\mathrm{F}}, q')]$ from σ. If the decoding results in $[o_{\mathrm{F}}, q']$, then $\mathcal{A}$ forwards q' as its answer. Otherwise, $\mathcal{A}$ goes on with updating L with the new pair (q, a).

After that, $\mathcal{A}$ runs $\mathbb{A}$ like a subroutine and forwards this latter's answer. To do so, $\mathcal{A}$ runs adversary $\mathbb{A}$'s update function $\mathsf{A}_{\downarrow}$ on $((o, q, a), m'_{\mathbb{A}})$, and then goes on running $\mathbb{A}$. The oracle calls that $\mathbb{A}$ places to hash functions are forwarded by $\mathcal{A}$ to its own hash oracles. Queries to o are answered by $\mathcal{A}$ in the following way: the list of calls L_o previously performed by $\mathbb{A}$ in known to $\mathcal{A}$, so that it can answer consistently redundant queries, and remaining fresh calls to o are answered by executing the command c and updating L accordingly.

The oracle system interacting with $\mathbb{A}$ is perfectly simulated by this transformation. As a consequence, if $\mathbb{A}$ outputs a result worth $\Pi_x(L_o[i])$, then $\mathcal{A}$ has computed a satisfactory value for its challenge.

$$\begin{array}{ll} & \Pr[\mathbb{A}|\mathbb{O}(\mathsf{c}) : \mathrm{R} = \Pi_x(L_o[i])] \\ \leq & \Pr[m \leftarrow [\![\mathsf{c}]\!](X) : \mathcal{A}^{\vec{H}}(m.q, m.a, m.\sigma) = m.x] \\ \leq & \varepsilon(k, k, t) \end{array}$$

where we have bounded the number of hash function calls performed for the construction of X and by $\mathcal{A}$ by function k. The details concerning the reasons why we cannot be tighter are the same as in the proof of rule Indis.

An application of the CIL rule UR allows to conclude:

$$\frac{\mathbb{O}(\mathsf{c}) :_{\varepsilon(k,k,t)} \mathrm{R} = \Pi_x(L_o[i]) \qquad \mathrm{R} \in \Pi_x(L_o) \Rightarrow \bigvee_{i=1..k(o)} \mathrm{R} = \Pi_x(L_o[i])}{\mathbb{O}(\mathsf{c}) :_{k(o)*\varepsilon(k,k,t)} \mathrm{R} \in \Pi_x(L_o)}\ \mathrm{UR}$$

■

V.4.3 — Generalization to Systems with Multiple Oracles

The theorems we have presented before only capture systems with one non-random oracle. We may want to use rules dealing with oracle systems containing multiple oracles. To this end, we propose the following definition.

Definition ((Generalized) Public-Key Oracle System). We let $(o_p(q_p, a_p) : \mathbf{var}\ V_p; \mathsf{c}_p)_{p=1..P}$ be a finite family of oracle declarations from the previous programming language. The public-key oracle system $\mathbb{O}(\mathsf{c}_1, \ldots, \mathsf{c}_p)$ based on these declarations is given by:

— memories $\mathsf{M}_{\mathbb{O}}$ containing variables pk, sk, one list L_H for every element in $\vec{H}$ and a list L_{o_p} for each declaration of the form o_p.

— in addition to the initialization and finalization oracles, the system contains an oracle for each element in $\vec{H}$ and oracles o_p. They are implemented as follows:

1. for the initialization oracle:
   $\mathsf{Imp}(o_{\mathrm{I}})(\_, \_) = (pk_0, sk_0) \leftarrow \mathcal{K};$
   $\mathsf{return}\ (pk_0, [pk \mapsto pk_0, sk \mapsto sk_0, (L_H \mapsto [\,])_{H \in \vec{\mathcal{H}}}, L_o \mapsto [\,]])$

2. every $H \in \vec{H}$ is implemented as the functional random oracle with outputs of length $\ell(H)$,[5]
3. for oracle o_p:

$$\begin{array}{rl} \mathsf{Imp}(o_p)(q_p, m) = & \mathsf{if}\ q_p \in \mathsf{dom}(m.L_o)\ \mathsf{then} \\ & \mathsf{return}\ (L_o(q_p), m) \\ & \mathsf{else} \\ & \mathsf{let}\ m' \leftarrow [\![\mathsf{c}]\!](m)\ \mathsf{in} \\ & \mathsf{return}\ (m'.a_p, m'.[L_o \mapsto L_o :: (m'.q_p, m'.a_p, m'.V_p), V \mapsto m.V_p]) \end{array}$$

4. finally, the finalization oracle just outputs its input memory. Its input type can be set according to the context of use of the system.

□

Rule Indis can be generalized quite easily. Indeed, since the variable taken as an argument of the predicate is either a local or the output variable, other oracle implementations are blind to a possible redraw. Thus, the proof performed for rule Indis can be adapted by considering that calls to oracles different from the one (say, o) that we want to change are simulated using their code together with a list of queries and answers. Of course, these lists have to be encoded and forwarded to adversary $\mathcal{A}$ constructed in the proof, along with the restricted list $(L_o)_{|(q,a)}$.

Theorem V.33. *The following rule is sound, where $x_i \in V_i \cup \{a_i\}$:*

$$\frac{\{\mathsf{true}\}\ \mathsf{c}\ \{\mathit{Indis}(x_i; q_i, a_i, \sigma; \emptyset; \varepsilon_i)\}}{\mathbb{O}(\mathsf{c}_1, \ldots, \mathsf{c}_p) \sim_{(k(o_i)*\varepsilon_i(k,k,t))} \mathbb{O}(\mathsf{c}'_1, \ldots, \mathsf{c}'_p)}\ \mathit{Indis}_{mult}$$

where $\mathsf{c}'_j = \mathsf{c}_i; x_i \leftarrow \mathcal{U}(l)$ if $j = i$ and $\mathsf{c}'_j = \mathsf{c}_j$ otherwise.

By changing one oracle at a time, we can use several Hoare logic statements on the same system.

The generalization of the weak secrecy plug-in theorem is done similarly to its Indis counterpart.

Theorem V.34. *The following rule is sound, where $x \in V_i$:*

$$\frac{\{\mathsf{true}\}\ \mathsf{c}\ \{\mathit{WS}(x; q_i, a_i, \sigma; \emptyset; \varepsilon_i)\}}{\mathbb{O}(\mathsf{c}_1, \ldots, \mathsf{c}_p) :_{k(o_i)*\varepsilon_i(k,k,t)} R \in \Pi_x(L_{o_i})}\ \mathit{WS}_{mult}$$

V.4.4 — Using the Verification Procedure as a Proof Strategy

The Hoare logic developed in the previous section offers an opportunity to perform automatic search for a proof of statements expressed in the form of conclusions of rules Indis, WS, Indis_{mult} or WS_{mult}.

A search algorithm for a given program and conclusion can be implemented in two steps: roughly, the first step provides a pattern for a satisfactory derivation, which the second step fills in.

Concerning the first step, we notice that, though our logic is presented to derive concrete security statements, we can obtain an asymptotic version of the rules by forgetting about

[5] See definition in III.5.3.

expressing a bound function for every predicate. By doing so, we conceptually replace being bounded by a specific function ε by just being bounded by a negligible function. With these new versions of the rules, we can compute the invariants by going through the program backwards, starting from the conclusion we want to establish. A difficulty is that several rules can lead to the same postcondition. As a result, given a postcondition, we compute a set of sufficient conditions before the execution of each command. For each set (of postconditions) $\{\mathsf{Inv}_1, \ldots, \mathsf{Inv}_n\}$ and each command c, we can compute a set of preconditions $\{\mathsf{Inv}'_1, \ldots, \mathsf{Inv}'_m\}$ such that, for each $i = 1, \ldots, n$, there exists a subset $J \subseteq [1, \ldots, m]$ such that $\left\{\bigwedge_{j\in J} \mathsf{Inv}'_j\right\}\ \mathsf{c}\ \mathsf{Inv}_i$ can be derived using the rules (with a possible use of the consequence rule ($Cons$) and the weakening lemmas). To compute the precondition set $\{\mathsf{Inv}'_1, \ldots, \mathsf{Inv}'_m\}$, we successively apply all possible Hoare rules to get Inv_i on the right side of command c. This provides us with an intermediate finite set of preconditions $\mathsf{PreCond}(\mathsf{Inv}_i)$, for each assertion Inv_i in the post-conditions. Then, the consequence rule is applied along with the weakening lemmas, which results in replacing $\mathsf{PreCond}(\mathsf{Inv}_i)$ by stronger assertions, leading to the finite set $\{\mathsf{Inv}'_1, \ldots, \mathsf{Inv}'_m\}$. Since the commands we consider do not include loops and the set of invariants we compute for each command is finite, our verification procedure always terminates. However, this verification is potentially exponential in the number of instructions in the program which we examine, as each postcondition may potentially have several preconditions. Nevertheless, this technique has been implemented and outputs a derivation for the examples which we present in section V.5 quasi-instantaneously.

Once we have obtained an asymptotic version of a derivation allowing to conclude, the last step to obtain a concrete security version of it is to go through the proof tree from top to bottom to actually compute the functions which are the last arguments of predicates. This verification procedure is not guaranteed to derive the best possible bound. Indeed, to do that, we ought to compute all possible asymptotic proofs and the bounds corresponding to their concrete version. Practice has led us to believe that if there was a proof derivable for a predicate, it was usually the only possible one, modulo the version of the preservation rule used or some weakenings. However, formalization or exploitation of the consequences of these observations have not been investigated. In a nutshell, the only result we know for certain with this verification procedure is that if a proof exists, we find it. However, in case no proof is found, it does not mean that the CIL statement is invalid.

V.5 Examples and Extensions

V.5.1 — Example of Application

We illustrate our proposition with Bellare & Rogaway's generic construction [BR93], which can be shortly described as $f(r)||(q \oplus G(r))||H(q||r)$, where q is a plaintext to be ciphered. A description of the algorithm in our framework, under the form of an oracle declaration, is the following:

$$o(q, a) : \textbf{var}\ \{r, g, b, c, d, s, t\};$$
$$(r \leftarrow \mathcal{U}(l); b := f(r); g := G(r); c := q \oplus g; s := q||r; d := H(s); t := b||c; a := t||d)$$

We propose a detailed proof of the construction by applying the rules of our logic. The table below provides a synoptic description of the proof for which we detail which rule is

applied and what the bound is worth one predicate at a time afterwards. In the sequel, $V = \{q, a, \sigma, r, g, b, c, d, s, t\}$:

$$
\begin{array}{l}
1)\ r \leftarrow \mathcal{U}(l) \\
\mathsf{Indis}(r; V; \emptyset; 0) \wedge \mathsf{H}(G; r; \frac{\chi_0(G)}{2^l}) \wedge \mathsf{H}(H; q||r; \frac{\chi_0(H)}{2^l}) \\
2)\ b := f(r) \\
\mathsf{Indis}(b; V \setminus \{r\}; \emptyset; 0) \wedge \mathsf{WS}(r; V \setminus \{r\}; r; OW(t)) \\
\wedge \mathsf{H}(G; r; \frac{\chi_0(G)}{2^l}) \wedge \mathsf{H}(H; q||r; \frac{\chi_0(H)}{2^l}) \\
3)\ g := G(r) \\
\mathsf{Indis}(b; V \setminus \{r\}; \emptyset; \varepsilon) \wedge \mathsf{Indis}(g; V \setminus \{r\}; r; \varepsilon) \wedge \\
\mathsf{WS}(r; V \setminus \{r\}; r; \varepsilon/2) \wedge \mathsf{H}(H; q||r; \frac{\chi_0(H)}{2^l}) \\
4)\ c := q \oplus g \\
\mathsf{Indis}(b; V \setminus \{r\}; \emptyset; \varepsilon) \wedge \mathsf{Indis}(c; V \setminus \{g, r\}; r; \varepsilon) \wedge \\
\mathsf{WS}(r; V \setminus \{r\}; r; \varepsilon/2) \wedge \mathsf{H}(H; q||r; \frac{\chi_0(H)}{2^l}) \\
5)\ s := q||r \\
\mathsf{Indis}(b; V \setminus \{r, s\}; \emptyset; \varepsilon) \wedge \mathsf{Indis}(c; V \setminus \{g, r, s\}; r; \varepsilon) \wedge \\
\mathsf{WS}(s; V \setminus \{r, s\}; r; \varepsilon/2) \wedge \mathsf{H}(H; s; \frac{\chi_0(H)}{2^l}) \\
6)\ d := H(s) \\
\mathsf{Indis}(b; V \setminus \{r, s\}; \emptyset; \varepsilon') \wedge \mathsf{Indis}(c; V \setminus \{r, g, s\}; r; \varepsilon') \wedge \\
\mathsf{Indis}(d; V \setminus \{r, s\}; r; \varepsilon'') \\
7)\ t := b||c \\
\mathsf{Indis}(t; V \setminus \{b, c, r, g, s\}; \emptyset; 2.\varepsilon') \\
\mathsf{Indis}(d; V \setminus \{b, c, r, s\}; r; \varepsilon'') \\
8)\ a := t||d \\
\mathsf{Indis}(a; q, a, \sigma; \emptyset; 2.\varepsilon' + \varepsilon'')
\end{array}
$$

We let $X \in \mathfrak{const}\mathfrak{D}(\chi_0)$ be a distribution to which we successively apply the commands above. We notice that χ is preserved by the first couple of commands; namely $[\![r \leftarrow \mathcal{U}(l); b := f(r)]\!](X) \in \mathfrak{const}\mathfrak{D}(\chi_0)$. The third command transforms the distribution into a $\chi_0 + \mathbf{1}_G$ constructible distribution. Fourth and fifth commands preserve this value, sixth command turns it into a $\chi_0 + \mathbf{1}_G + \mathbf{1}_H$ constructible distribution, which it remains until the end.

First Command. Predicate $\mathsf{Indis}(r; V; \emptyset; 0)$ is a consequence of rule (R1). Rule (R2) provides both $\mathsf{H}(G; r; \frac{\chi_0(G)}{2^l})$ and $\mathsf{H}(H; q||r; \frac{\chi_0(H)}{2^l})$. We then use $(Conj)$ to end up with our conjunction. In the remaining of the proof, we do not precise use of $(Conj)$ anymore. Neither do we specify uses of (Seq) or (Csq) after weakening.

Second Command. To obtain $\mathsf{Indis}(b; V \setminus \{r\}; r; 0)$, we first weaken $\mathsf{Indis}(r; V; \emptyset; 0)$ into $\mathsf{Indis}(r; V \setminus \{r, b\}; r; 0)$, and then apply $(P1)$ (we do have $r, b \notin V \setminus \{r, b\}$). Weakened predicate $\mathsf{Indis}(r; V \setminus \{r\}; r; 0)$ is used in $(P2)$ to derive $\mathsf{WS}(r; V \setminus \{r\}; r; OW(t))$. Indeed, we can do this because $r \notin (V \setminus \{r\}) \cup \{b\}$. The preservation of both other predicates is an aplication of $(G3)$.

Third Command. Using $\mathsf{Indis}(b; V \setminus \{r\}; \emptyset; 0) \wedge \mathsf{WS}(r; V \setminus \{r\}; r; OW(t)) \wedge \mathsf{H}(G; r; \frac{\chi_0(G)}{2^l})$ and rule (H7) (since $g \neq b, r$) allows us to deduce that $\mathsf{Indis}(b; V \setminus \{r\}; \emptyset; \varepsilon)$ where $\varepsilon(k, \chi, t) = 2.k(G) * OW(t) + 2.\frac{\chi_0(G)}{2^l}$. Moreover, $\mathsf{WS}(r; V \setminus \{r\}; r; OW(t)) \wedge \mathsf{H}(G; r; \frac{\chi_0(G)}{2^l})$ can be used

as the premises of rule $(H1)$ (since $g \neq r$) to derive $\mathsf{Indis}(g; V \setminus \{r\}; r; \varepsilon)$. On top of that, the same hypotheses used with rule $(H4)$ (and $g \neq r$) yield $\mathsf{WS}(r; V \setminus \{r\}; r; \varepsilon/2)$. Rule $(G3)$ and $H \neq G$ account for the preservation of the last predicate.

Fourth Command. Predicate $\mathsf{Indis}(b; V \setminus \{r\}; \emptyset; \varepsilon)$ follows from the application of rule $(X2)$, along with $b \neq c, q, g$. Now, we apply rule $(X1)$ to $\mathsf{Indis}(g; V \setminus \{g, r\}, g; r; \varepsilon)$, with $g \notin V \setminus \{g, r\} \cup \{r\}$ and $g \neq q, c$, to obtain $\mathsf{Indis}(c; V \setminus \{g, r\}; r; \varepsilon)$. Furthermore, rule $(X3)$ is applied with predicate $\mathsf{WS}(r; V \setminus \{r\}; r; \varepsilon/2)$ and $r \neq c$ to preserve it. Eventually, rule $(G3)$ justifies the preservation of the remaining predicate.

Fifth Command. Firstly, $\mathsf{Indis}(b; V \setminus \{r\}; \emptyset; \varepsilon)$ can be weakened in $\mathsf{Indis}(b; V \setminus \{r, s\}; \emptyset; \varepsilon)$. Then, rule $(G1)$, $s \notin (V \setminus \{r, s\}) \cup \{r\}$ and $b \neq s$ provide its conservation. The same reasoning can be done to obtain $\mathsf{Indis}(c; V \setminus \{g, r, s\}; r; \varepsilon)$. Weakening $\mathsf{WS}(r; V \setminus \{r\}; r; \varepsilon/2)$ into $\mathsf{WS}(r; V \setminus \{r, s\}; r; \varepsilon/2)$ enables to use $(C1)$ and get $\mathsf{WS}(s; V \setminus \{r, s\}; r; \varepsilon/2)$: we do have $s \notin (V \setminus \{r, s\}) \cup \{r\}$. Finally, rule $(G3)$ and $\mathsf{H}(H; q || r; \frac{\chi_0(H)}{2^l})$ allow us to write $\mathsf{H}(H; s; \frac{\chi_0(H)}{2^l})$.

Sixth Command. First, we apply $(H7)$ using that $d \neq s, b$, $\mathsf{Indis}(b; V \setminus \{r\}; \emptyset; \varepsilon)$, $\mathsf{WS}(s; V \setminus \{r, s\}; r; \varepsilon/2)$ and $\mathsf{H}(H; s; \frac{\chi_0(H)}{2^l})$. We get $\mathsf{Indis}(b; V \setminus \{r\}; \emptyset; \varepsilon')$ where $\varepsilon'(k, \chi, t) = (1 + k(H))\varepsilon(k, \chi - \mathbf{1}_H, t) + 2.\frac{\chi_0(H)}{2^l} = 2.(1 + k(H))(k(G) * OW(t) + \frac{\chi_0(G)}{2^l}) + 2\frac{\chi_0(H)}{2^l}$. The same rule applied with $d \neq s, c$, $\mathsf{Indis}(c; V \setminus \{g, r, s\}; r; \varepsilon)$ and $\mathsf{WS}(s; V \setminus \{r, s\}; r; \varepsilon/2)$ and $\mathsf{H}(H; s; \frac{\chi_0(H)}{2^l})$ provides us with $\mathsf{Indis}(c; V \setminus \{g, r, s\}; r; \varepsilon')$. Moreover, rule $(H1)$ taking as premises $\mathsf{WS}(s; V \setminus \{r, s\}; r; \varepsilon/2)$ and $\mathsf{H}(H; s; \frac{\chi_0(H)}{2^l})$, along with $d \neq s$, yield $\mathsf{Indis}(d; V \setminus \{r, s\}; r; \varepsilon'')$, where $\varepsilon''(k, \chi, t) = \frac{\chi_0(H)}{2^l} + k(H) * (k(G) * OW(t) + \frac{\chi_0(G)}{2^l})$.

Seventh Command. First, $\mathsf{Indis}(b; V \setminus \{r, s\}; \emptyset; \varepsilon')$ is rewritten as $\mathsf{Indis}(b; V \setminus \{r, b, c, s\}, b, c; \emptyset; \varepsilon')$ and $\mathsf{Indis}(c; V \setminus \{g, r, s\}; r; \varepsilon')$ is weakened in $\mathsf{Indis}(c; V \setminus \{g, r, s, b, c\}, b, c; \emptyset; \varepsilon')$. From rule $(C2)$, $\mathsf{Indis}(b; V \setminus \{r, b, c, s\}, b, c; \emptyset; \varepsilon')$, $\mathsf{Indis}(c; V \setminus \{g, r, s, b, c\}, b, c; \emptyset; \varepsilon')$ and $b, c \notin (V \setminus \{r, b, c, s\}) \cup \{r\}$, we get $\mathsf{Indis}(t; V \setminus \{g, r, s, b, c\}; \emptyset; 2.\varepsilon')$. In addition to that, we can use rule $(C3)$, $d \neq t, b, c$ and $\mathsf{Indis}(d; V \setminus \{r, s\}; r; \varepsilon'')$ to get $\mathsf{Indis}(d; V \setminus \{r, s\}; r; \varepsilon'')$.

Eighth Command. We start by weakening predicate $\mathsf{Indis}(d; V \setminus \{r, s\}; r; \varepsilon'')$ into predicate $\mathsf{Indis}(d; V \setminus \{r, s, g, b, c\}; \emptyset; \varepsilon'')$ Using rule $(C2)$ with $\mathsf{Indis}(t; V \setminus \{g, r, s, b, c, t, d\}, t, d; \emptyset; 2.\varepsilon')$, $\mathsf{Indis}(d; V \setminus \{r, s, g, b, c, t, d\}, t, d; \emptyset; \varepsilon'')$ and $t, d \notin (V \setminus \{r, s, g, b, c, t, d\}) \cup \{r, a\}$, we get $\mathsf{Indis}(a; V \setminus \{r, s, g, b, c, t, d\}; \emptyset; 2.\varepsilon' + \varepsilon'')$. A last weakening provides $\mathsf{Indis}(a; q, a, \sigma; \emptyset; 2.\varepsilon' + \varepsilon'')$.

V.5.2 — Extensions of the Logic

We have developped rules for a command $x := f(y)$ where f is a one-way permutation. However, bijectivity is a strong requirement for a cryptographic primitive. In this subsection, we show how we can relax this assumption and comment on which rules it invalidates, before exploring what we can do with functions with a stronger property than one-wayness.

Let us first deal with the case when function f is not surjective. It sometimes makes the Indis predicate too strong to still hold. Indeed, the output values of an algorithm of the form $f(.)$, with f an only injective function, can never be uniformly distributed among the set of all bitstrings of the right length, but they may be uniformly distributed among f's range. To capture this new notion, we introduce another predicate.

Definition (Indis_f Predicate). Let X be a constructible distribution, ε be a function mapping (k, χ, t) to $[0, 1]$, x be a variable in Var, and sets of variables $V_1 \subseteq \mathsf{Var} \cup \{\sigma\}$, $V_2 \subseteq (\mathsf{Var} \setminus \{x\})$. $X \models \mathsf{Indis}_f(x; V_1; V_2; \varepsilon)$ iff
$X \sim_{V_1;V_2;\varepsilon} [u \leftarrow \mathcal{U}(l); m \leftarrow X : m.[x \mapsto f(u)]]$ □

If f is bijective, then Indis_f is strictly the same predicate as Indis.

We can derive new versions of compatibility and weakening lemmas. We omit the proofs, which are very similar, when not exactly identical, to those of their counterpart lemmas of section V.2.

Lemma V.35 (Adapted Lemmas). *Let $X, X' \in \mathfrak{const}\mathfrak{D}(\chi)$. For any sets of variables $V_1 \subseteq \mathsf{Var} \cup \{\sigma\}$ and $V_2 \subseteq \mathsf{Var}$, and any variable $x \in \mathsf{Var} \setminus V_2$:*

1. *Compatibility: if $X \sim_{V_1;V_2;\varepsilon} X'$ then $X \models \mathsf{Indis}_f(x; V_1; V_2; \varepsilon') \Rightarrow X' \models \mathsf{Indis}_f(x; V_1; V_2; \varepsilon'')$, where $\varepsilon''(k, \chi, t) = 2.\varepsilon(k, \chi, t) + \varepsilon'(k, \chi, t)$. The conservation version of this lemma is verified too.*
2. *Weakening: if $X \models \mathsf{Indis}_f(x; V_1; V_2; \varepsilon)$, $V_1' \subseteq V_1$ and $V_2' \subseteq (V_1 \cup V_2 \setminus \{x\})$ then $X \models \mathsf{Indis}_f(x; V_1'; V_2'; \varepsilon')$, where $\varepsilon'(k, \chi, t) = \varepsilon(k, \chi, t + \mathsf{Card}(V_1 \cap V_2').\mathsf{T}_f)$.*
3. *Hybrid weakening: if $X \models \mathsf{Indis}_f(x; V_1, x; V_2; \varepsilon)$ and $x \notin V_1 \cup V_2$ then $X \models \mathsf{WS}(x; V_1; V_2, x; \varepsilon')$, with $\varepsilon'(k, \chi, t) = \varepsilon(k, \chi, t + \mathsf{T}_f) + OW(t)$*
4. *Constructible expression: let e be an expression constructible from $V_1 \setminus \{\sigma\}$ and V_2, and $z \neq x$. Let $\mathsf{c} \equiv x := e$. If $X \models \mathsf{Indis}_f(z; V_1; V_2; \varepsilon)$ and z does not (syntactically) appear in e, then $[\![x := e]\!](X) \models \mathsf{Indis}_f(z; V_1, x; V_2; \varepsilon')$, where $\varepsilon'(k, \chi, t) = \varepsilon(k + \chi_{\mathsf{c}}, \chi - \chi_{\mathsf{c}}, t + \mathsf{T}_{\mathsf{c}}))$.*

The rule concerning one-way permutations ($P1$) must be replaced by the following one.

Lemma V.36 (Rule for One-Way Function). *The following rule is sound if $x, y \notin V_1 \cup V_2$:*

$(P1)^f$ $\{\mathsf{Indis}(y; V_1; V_2, y; \varepsilon)\}\ x := f(y)\ \{\mathsf{Indis}_f(x; V_1, x; V_2; \varepsilon)\}$.

Proof. To prove this rule, we start by noticing that:

$$\begin{aligned}
& \Pr[m \leftarrow \nu y.X : (m.V_1, f(m.y), f(m.V_2))] \\
= \ & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l); m' := m[y \mapsto u] : (m'.V_1, f(m'.y), f(m'.V_2))] \\
= \ & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l) : (m.V_1, f(u), f(m.V_2))] \\
& \quad \text{because } y \notin V_1 \cup V_2 \\
= \ & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l); m' := m[x \mapsto f(u)] : (m'.V_1, m'.x, f(m'.V_2))] \\
& \quad \text{since } x \notin V_1 \cup V_2
\end{aligned}$$

From this equality and the fact that no hash oracle is modified by the command, we deduce that any adversary against one of the predicate can be used to attack the other one up to a reorganization of its inputs and succeeds with the same advantage. The conclusion follows. ■

As for the other rules, here are the new instances that we can obtain by generalizing their counterparts presented in section V.3.

Lemma V.37 (Adapted Preservation Rules). *c is $x \leftarrow \mathcal{U}$ or of the form $x := e'$ with e' being either $w||y$, $w \oplus y$, $f(y)$ or $\alpha \oplus H(y)$,*

$(G1)^f$ *$\{\mathsf{Indis}_f(z; V_1; V_2; \varepsilon)\}\ \mathsf{c}\ \{\mathsf{Indis}_f(z; V_1; V_2; \varepsilon')\}$, when $z \neq x$, $x \notin V_1 \cup V_2$, and where $\varepsilon'(k, \chi, t) = \varepsilon(k, \chi - \chi_{\mathsf{c}}, t)$.*

$(G1')^f$ *$\{\mathsf{Indis}_f(z; V_1; V_2; \varepsilon)\}\ \mathsf{c}\ \{\mathsf{Indis}_f(z; V_1, x; V_2; \varepsilon')\}$, when $z \neq x$, if e' is constructible from $(V_1 \setminus \{z\}; V_2 \setminus \{z\})$ and where $\varepsilon'(k, \chi, t) = \varepsilon(k + \chi_{\mathsf{c}}, \chi - \chi_{\mathsf{c}}, t + \mathsf{T}_{\mathsf{c}}))$.*

$(R3)^f$ $\{\mathsf{Indis}_f(y;V_1;V_2;\varepsilon)\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{Indis}_f(y;V_1,x;V_2;\varepsilon)\}$, *if* $x \neq y$.

$(H7)^f$ $\{\mathsf{Indis}_f(z;V_1,z;V_2;\varepsilon_1)\wedge \mathsf{WS}(y;V_1,z;V_2;\varepsilon_2)\wedge \mathsf{H}(H;y;\varepsilon_3)\}\ x := \alpha\oplus H(y)\ \{\mathsf{Indis}_f(z;V_1,z,x;V_2;\varepsilon_4)\}$ *if* $x \neq y, z$ *and where* $\varepsilon_4(k,\chi,t) = \varepsilon_1(k,\chi-\mathbf{1}_H,t)+2.k(H)*\varepsilon_2(k,\chi-\mathbf{1}_H,t)+2.\varepsilon_3(\chi-\mathbf{1}_H)$.

$(P3)^f$ $\{\mathsf{Indis}_f(z;V_1,z;V_2,y;\varepsilon)\}\ x := f(y)\ \{\mathsf{Indis}_f(z;V_1,z,x;V_2,y;\varepsilon)\}$ *if* $z \neq x, y$.

$(X2)^f$ $\{\mathsf{Indis}_f(w;V_1,y,z;V_2;\varepsilon)\}\ x := y \oplus z\ \{\mathsf{Indis}_f(w;V_1,x,y,z;V_2;\varepsilon)\}$, *if* $w \neq x, y, z$.

$(C3)^f$ $\{\mathsf{Indis}_f(w;V_1,y,z;V_2;\varepsilon)\}\ x := y||z\ \{\mathsf{Indis}_f(w;V_1,x,y,z;V_2;\varepsilon)\}$, *if* $w \neq x, y, z$.

We have cited adapted versions of all the results on which the proofs of the original counterparts of these rules rely. Proofs of adapted rules are the same reasonings as original rules but put to use adapted lemmas. As for the rest of the rules, they remain sound as they are cited in section V.3, since nothing in their proofs depends on the fact that f is a permutation. We have now at our disposal a four predicate assertion language and a more complete Hoare logic, to deal with the case when f is not a bijection.

Let us consider an oracle declaration $o(q,a)$: **var** V; c associated with an encryption scheme. We have seen that one of the major application of our three predicate Hoare logic is to deduce ROR-ciphertext security of the encryption scheme from predicate $\mathsf{Indis}(a;q,a,\sigma;\emptyset;\_)$. Nevertheless, we notice that in the definition of the ROR-ciphertext security notion, a ciphertext is sampled *in the range* of the encryption scheme. This does not raise any issue as long as this range is the whole set of bitstrings of a given length. However, in case this range is strictly included in the set of bitstrings of a given length, then a predicate such as $\mathsf{Indis}(a;\dots)$ is too demanding: either it is not possible to prove such a statement, or the function ε one gets is too high to be interesting. To tackle this problem, we decompose the output of the encryption into $a := f_1(a_1)||\dots||f_n(a_n)$, where f_i is a function defined on $\{0,1\}^{l_i}$. Such a decomposition is always possible. The relevance of a decomposition can be evaluated in terms of the bound it results in. Usually, the most natural decomposition is the best. To transpose the notion of ROR-ciphertext security, we propose to show that an adversary cannot distinguish between a real ciphertext and $f_1(a_1)||\dots||f_n(a_n)$ computed using random values for a_i's. Yet another plug-in theorem is needed to be able to use our Hoare logic to derive such conclusions in CIL.

Theorem V.38. *Let* $\mathbb{O}(\mathsf{c}; a := f_1(a_1)||\dots||f_n(a_n))$ *be the public-key oracle system based on declaration* $o(q,a)$: **var** $V \cup \{a_1,\dots a_n\}$; c; $a := f_1(a_1)||\dots||f_n(a_n)$, *with the finalization oracle taking boolean as inputs. The following rule is sound:*

$$\frac{\{\mathsf{true}\}\ \mathsf{c}\ \mathsf{Indis}_{f_j}(a_j;q,a_1,\dots,a_n,\sigma;\emptyset;\varepsilon_j)\}}{\mathbb{O}(\mathsf{c}) \sim_{k(o)*\varepsilon_j(k,k,t)} \mathbb{O}(\mathsf{c}; a_j \leftarrow \mathcal{U}(l_j); a := f_1(a_1)||\dots||f_n(a_n))}\ \mathsf{Indis}_f$$

This theorem can be proven by the same hybrid argument and simulation as the first plug-in theorem. However, this does not yield exactly the ROR-ciphertext security of the oracle in CIL, which reads:

$$\mathbb{O}(\mathsf{c}) \sim_{\varepsilon(k,k,t)} \mathbb{O}(\mathsf{c}; a_1 \leftarrow \mathcal{U}(l_1); \dots; a_n \leftarrow \mathcal{U}(l_n); a := f_1(a_1)||\dots||f_n(a_n))$$

If we can derive $\{\mathsf{true}\}\ \mathsf{c}\ \bigwedge_{j=1..n} \mathsf{Indis}_{f_j}(a_j;q,a_1,\dots,a_n,\sigma;\emptyset;\varepsilon_j)\}$, such a statement can be obtained by iteratively applying rule Indis_f as follows. Let us first remark that conjunction $\bigwedge_{j=1..n} \mathsf{Indis}_{f_j}(a_j;q,a_1,\dots,a_n,\sigma;\emptyset;\varepsilon_j)\}$ implies $\mathsf{Indis}_{f_1}(a_1;q,a_1,\dots,a_n,\sigma;\emptyset;\varepsilon_1)$. We apply our rule and get $\mathbb{O}(\mathsf{c}) \sim_{k(o)*\varepsilon_1(k,k,t)} \mathbb{O}(\mathsf{c}; a_1 \leftarrow \mathcal{U}(l_1); a := f_1(a_1)||\dots||f_n(a_n))$. Now, we denote

c_1 the command $\mathsf{c}; a_1 \leftarrow \mathcal{U}(l_1)$. From $\{\mathsf{true}\}\ \mathsf{c}\ \bigwedge_{j=2..n} \mathsf{Indis}_{f_j}(a_j; q, a_1, \ldots, a_n, \sigma; \emptyset; \varepsilon_j)\}$ and $(R3)^f$, we get $\{\mathsf{true}\}\ \mathsf{c}_1\ \bigwedge_{j=2..n} \mathsf{Indis}_{f_j}(a_j; q, a_1, \ldots, a_n, \sigma; \emptyset; \varepsilon_j)\}$. In particular, $\mathsf{Indis}_{f_2}(a_2; \ldots)$ is verified. Therefore, we can apply rule Indis_f to $\mathbb{O}(\mathsf{c}_1)$. We get $\mathbb{O}(\mathsf{c}_1) \sim_{k(o)*\varepsilon_2(k,k,t)} \mathbb{O}(\mathsf{c}_1; a_2 \leftarrow \mathcal{U}(l_2); a := f_1(a_1)||\ldots||f_n(a_n))$. The iteration of this reasoning until we reach $j = n$, combined with transitivity of relation $\sim$ allows us to conclude to:

$$\mathbb{O}(\mathsf{c}) \sim_{k(o)*(\sum_j \varepsilon_j(k,k,t))} \mathbb{O}(\mathsf{c}; a_1 \leftarrow \mathcal{U}(l_1); \ldots; a_n \leftarrow \mathcal{U}(l_n); a := f_1(a_1)||\ldots||f_n(a_n))$$

Now that we have shown how to relax the hypothesis of bijectivity of f, let us elaborate a little on another axis of extension of our logic, to capture cases when f is not a one-way function but something stronger, namely an injective partially trapdoor one-way function. Hereafter, we give the definition of partially trapdoor one-way function as is formalized in [Poi00].

Definition (Partially Trapdoor One-Way Function).
A function $f : X \times Y \to Z$ is said to be *partially trapdoor one-way* iff

1. from any $z = f(x, y)$, for all adversary $\mathcal{A}$,
   $\Pr[x \leftarrow \mathcal{U}(X), y \leftarrow \mathcal{U}(Y) : \exists y' \in Y s.t. f(\mathcal{A}(f(x||y))||y') = f(x||y)] \leq POW(t)$
2. for any $z \in Z$, there exists a partial trapdoor t such that given t one can easily compute a partial preimage x, that is, such that $\exists y \in Y\ s.t.\ z = f(x||y)$. Notice that the trapdoor does not necessarily provide a value for y (hence its name).

□

The fact that we impose our primitive to be injective allows to state the first item of partial one-wayness as:

$$\Pr[x \leftarrow \mathcal{U}(X), y \leftarrow \mathcal{U}(Y) : \mathcal{A}(f(x||y)) = x] \leq POW(t)$$

Dealing with a stronger primitive, all the rules that we have cited are sound. We reformulate rules for $x := f(y)$ into rules for $z := f(x||y)$ so that they take into account the new properties of f.

Lemma V.39 (Rules for Injective Partially Trapdoor One-Way Functions). *The following rules are sound if $x, y, z \notin (V_1 \cup V_2)$:*

$(IPO1)^f$ $\{\mathsf{Indis}(x; V_1, x, y; V_2; \varepsilon) \wedge \mathsf{Indis}(y; V_1, x, y; V_2; \varepsilon')\}\ z := f(x||y)$
$\{\mathsf{Indis}_f(z; V_1, z; V_2; \varepsilon + \varepsilon')\}$.

$(IPO2)$ $\{\mathsf{Indis}(x; V_1, x, y; V_2; \varepsilon) \wedge \mathsf{Indis}(y; V_1, x, y; V_2; \varepsilon')\}\ z := f(x||y)$
$\{\mathsf{WS}(x; V_1, z; V_2; \varepsilon'')\}$. *where* $\varepsilon''(k, \chi, t) = POW(t) + \varepsilon(k, \chi, t + \mathsf{T}_f) + \varepsilon'(k, \chi, t + \mathsf{T}_f)$.

Proof. First, the couple of hypothesis allow us to write:

$$\mathsf{Indis}(x; V_1, x, y; V_2; \varepsilon) \wedge \mathsf{Indis}(y; V_1, x, y; V_2; \varepsilon')\ w := x||y\ \mathsf{Indis}(w; V_1, w; V_2; \varepsilon + \varepsilon')$$

by applying $(C2)$ with $x, y \notin V_1 \cup V_2$.

Then, to prove $(IPO1)^f$, we can weaken this predicate into $\mathsf{Indis}(w; V_1; V_2, w; \varepsilon + \varepsilon')$ and use $(P1)^f$ (and $w, z \notin V_1 \cup V_2$) to get $\mathsf{Indis}_f(z; V_1; V_2; \varepsilon + \varepsilon')$.

As for rule $(IPO2)$, given a (k, t)-adversary $\mathcal{A}$ for $\mathsf{WS}(x; V_1, z; V_2; \_)$, we can build a $(k, t + \mathsf{T}_f)$-adversary $\mathcal{B}$ against $\mathsf{Indis}(w; V_1, w; V_2; \varepsilon + \varepsilon')$. We suppose that x is of length l and y of length l'.

$\mathcal{B}(V_1, w; V_2) =$ $x_0 \leftarrow \mathcal{A}(V_1, f(w); V_2)$;
$\mathsf{if}\ [w]_1^l = x_0\ \mathsf{then\ return\ true}$
$\mathsf{else\ return\ false}$

On the one hand, given a constructible distribution X, $|\Pr[m \leftarrow X; \mathcal{B}(m.V_1, m.w; f(m.V_2)) = \mathsf{true}] - \Pr[m \leftarrow \nu w.X; \mathcal{B}(m.V_1, m.w; f(m.V_2)) = \mathsf{true}]| \leq \varepsilon(k, \chi, t + \mathsf{T}_f) + \varepsilon'(k, \chi, t + \mathsf{T}_f)$.

On the other hand,

$$\begin{aligned} & \Pr[m \leftarrow \nu w.X; \mathcal{B}(m.V_1, m.w; f(m.V_2)) = \mathsf{true}] \\ = \ & \Pr[m \leftarrow \nu w.X : \mathcal{A}(m.V_1, f(m.w); f(m.V_2)) = [w]_1^l] \\ = \ & \Pr[m \leftarrow X; u \leftarrow \mathcal{U}(l); v \leftarrow \mathcal{U}(l') : \mathcal{A}(m.V_1, f(u||v); f(m.V_2)) = [w]_1^l] \\ \leq \ & POW(t) \end{aligned}$$

As a result, a triangle inequality allows us to conclude to

$\Pr[m \leftarrow [\![z := f(x||y)]\!](X) : \mathcal{A}(m.V_1, m.z; f(m.V_2)) = x] \leq POW(t) + \varepsilon(k, \chi, t + \mathsf{T}_f) + \varepsilon'(k, \chi, t + \mathsf{T}_f)$. ■

With these extensions, we can prove Pointcheval's transformer [Poi00], which can be written shortly as $f(r||H(q||s))||(q||s) \oplus G(r)$. where q is a plaintext to be ciphered. A description of the algorithm in our framework, under the form of an oracle declaration, is the following:

$o(q, a)$: **var** $\{r, s, w, h, t, u\}$;
$r \leftarrow \mathcal{U}(l); s \leftarrow \mathcal{U}(l'); w := q||s; h := H(w); t := f(r||h); u := w \oplus G(r); a := t||u$

We let $V = \{q, a, \sigma, r, s, w, h, t, u\}$.

true
1) $r \leftarrow \mathcal{U}(l)$
$\mathsf{Indis}(r; V; \emptyset; 0) \wedge \mathsf{H}(G; r; \frac{\chi(G)}{2^l})$
2) $s \leftarrow \mathcal{U}(l')$
$\mathsf{Indis}(s; V; \emptyset; 0) \wedge \mathsf{Indis}(r; V; \emptyset; 0) \wedge$
$\mathsf{H}(G; r; \frac{\chi(G)}{2^l}) \wedge \mathsf{H}(H; q||s; \frac{\chi(H)}{2^{l'}})$
3) $w := q||s$
$\mathsf{WS}(w; V \setminus \{s, w\}; \emptyset; POW(t)) \wedge \mathsf{Indis}(r; V \setminus \{s, w\}; \emptyset; 0) \wedge$
$\mathsf{H}(G; r; \frac{\chi(G)}{2^l}) \wedge \mathsf{H}(H; w; \frac{\chi(H)}{2^{l'}})$
4) $h := H(w)$
$\mathsf{Indis}(h; V \setminus \{w, s\}; \emptyset; \varepsilon) \wedge \mathsf{Indis}(r; V \setminus \{w, s\}; \emptyset; 2.\varepsilon)$
$\wedge \mathsf{H}(G; r; \frac{\chi(G)}{2^l})$
5) $t := f(r||h)$
$\mathsf{Indis}_f(t; V \setminus \{r, s, w, h\}; \varepsilon) \wedge$
$\mathsf{WS}(r; V \setminus \{r, s, w, h\}; \varepsilon') \wedge \mathsf{H}(G; r; \frac{\chi(G)}{2^l})$
6) $u := w \oplus G(r)$
$\mathsf{Indis}(u; V \setminus \{r, s, w, h\}; \varepsilon'') \wedge \mathsf{Indis}_f(t; V \setminus \{r, s, w, h\}; \varepsilon^3)$
7) $a := t||u$
$\mathsf{Indis}_f(t; V \setminus \{r, s, w, h, a\}; \varepsilon^3)$
$\mathsf{Indis}(u; V \setminus \{r, s, w, h, a\}; \varepsilon'')$

First Command. Using rule $(R1)$, we generate $\mathsf{Indis}(r; V; \emptyset; 0)$ and rule $(R2)$ provides us with $\mathsf{H}(G; r; \frac{\chi(G)}{2^l})$.

Second Command. Rule $(R1)$ allows to obtain $\mathsf{Indis}(s; V; \emptyset; 0)$. Predicate $\mathsf{Indis}(r; V; \emptyset; 0)$ follows from preservation rule $(R3)$, together with $s \neq r$. Moreover, $\mathsf{H}(G; r; \frac{\chi(G)}{2^l})$ is preserved thanks to $(G3')$ and $s \neq r$. Eventually, $\mathsf{H}(H; q||s; \frac{\chi(H)}{2^l})$ is created by an application of rule $(R2)$.

Third Command. We start by weakening $\mathsf{Indis}(s; V; \emptyset; 0)$ into $\mathsf{WS}(s; V \setminus \{s, w\}; \emptyset; POW(t))$. Then, we check that $w \notin V \setminus \{s, w\}$ and apply rule $(C1)$ to get $\mathsf{WS}(w; V \setminus \{s, w\}; \emptyset; POW(t))$. $\mathsf{Indis}(r; V; \emptyset; 0)$ is weakened into $\mathsf{Indis}(r; V \setminus \{s, w\}; \emptyset; 0)$, which is preserved thanks to $(G1)$, $w \neq r$ and $w \notin V \setminus \{s, w\}$. As for $\mathsf{H}(G; r; \frac{\chi(G)}{2^l})$, it is preserved thanks to $(G3)$, while this same rule applied with premise $\mathsf{H}(H; q||s; \frac{\chi(H)}{2^l})$ provides $\mathsf{H}(H; w; \frac{\chi(H)}{2^l})$.

Fourth Command. We apply $(H1)$ on $\mathsf{WS}(w; V \setminus \{s, w\}; \emptyset; POW(t))$ and $\mathsf{H}(H; w; \frac{\chi(H)}{2^l})$ and $h \neq w$, to obtain $\mathsf{Indis}(h; V \setminus \{s, w\}; \emptyset; \varepsilon)$ with $\varepsilon(k, \chi, t) = \frac{\chi(H)}{2^{l'}} + k(H) * POW(t)$. Furthermore, we can apply rule $(H7)$ on $\mathsf{Indis}(r; V \setminus \{s, w\}; \emptyset; 0)$, $\mathsf{WS}(w; V \setminus \{s, w\}V; \emptyset; POW(t))$ and $\mathsf{H}(H; w; \frac{\chi(H)}{2^{l'}})$ and $h \neq w, r$. It provides us with $\mathsf{Indis}(r; V \setminus \{s, w\}; \emptyset; 2.\varepsilon)$. Eventually, $\mathsf{H}(G; r; \frac{\chi(G)}{2^l})$ is preserved thanks to $(G3)$.

Fifth Command. $\mathsf{Indis}(h; V \setminus \{w, s, h, r, t\}, h, r; \emptyset; \varepsilon) \wedge \mathsf{Indis}(r; V \setminus \{w, s, h, r, t\}, h, r; \emptyset; 2.\varepsilon)$ provides with rule $(IPO1)^f$ and $h, r, t \notin V \setminus \{w, s, h, r, t\}$ the predicate $\mathsf{Indis}_f(t; V \setminus \{w, s, h, r\}; \emptyset; 3.\varepsilon)$. $\mathsf{WS}(r; V \setminus \{r, s, w, h\}; \varepsilon')$ follows from the same premises and rule $(IPO2)$, where we let $\varepsilon'(k, \chi, t) = 3.\varepsilon(k, \chi, t + \mathsf{T}_f) + POW(t)$. Finally, an application of rule $(G3)$ allows to preserve $\mathsf{H}(G; r; \frac{\chi(G)}{2^l})$.

Sixth Command. We apply $(H1)$ with premises $\mathsf{WS}(r; V \setminus \{r, s, w, h\}; \varepsilon')$ and $\mathsf{H}(G; r; \frac{\chi(G)}{2^l})$ (and of course $u \neq r$). This generates predicate $\mathsf{Indis}(u; V \setminus \{r, s, w, h\}; \varepsilon'')$ with $\varepsilon''(k, \chi, t) = k(G) * \varepsilon'(k, \chi, t) + \frac{\chi(G)}{2^l}$. We apply $(H7)^f$ with premises $\mathsf{Indis}_f(t; V \setminus \{r, s, w, h\}; 3.\varepsilon)$ $\mathsf{WS}(r; V \setminus \{r, s, w, h\}; \varepsilon')$ and $\mathsf{H}(G; r; \frac{\chi(G)}{2^l})$ (and of course $u \neq r, t$). $\mathsf{Indis}_f(t; V \setminus \{r, s, w, h\}; \varepsilon^3)$ where $\varepsilon^3 = \varepsilon + 2.k(G) * \varepsilon' + 2.\frac{\chi(G)}{2^l}$.

Seventh Command. Applying $(G1)^f$ and $(G1)$ after having weakened the predicates by removing a from the first set of variables.

Chapter VI

A Reduction Theorem for Hash Constructions

VI.1 Semantic Extensions of Our Framework

VI.1.1 — Our Motivation for a New Definition

In this chapter, we investigate problems raised by a particular way of modifying dependencies which can exist between oracles. In the definition of oracle systems, we specify that they are stateful, thus allowing oracles of the system to share a state and modify it at will without imposing any restriction. That being said, we remark that we can draw useful conclusions whenever an oracle uses another as a black-box, namely, when an oracle o depends on an oracle o' in such a way that the implementations of o and o' can be written without a shared memory component, but the implementation of o contains calls to o'. In fact, for each oracle in a system, we can specify the set of other oracles on which it depends in a black-box manner, leaving the other forms of dependency (including independence) for oracles outside of this set. This is captured by the following definition.

Definition (Notation for Black-Box Dependency). We consider an oracle system $\mathbb{O}$ featuring oracles $\{o_1, \ldots, o_n\}$ in addition to initialization and finalization oracles, with a memory set $\mathsf{M}_{\mathbb{O}} = \mathsf{M}_1 \times \cdots \times \mathsf{M}_n$.

The notation $\mathsf{Imp}(o_i)^{o_{j_1},\cdots,o_{j_i}}$ means that the implementation $\mathsf{Imp}(o_i)$ of o_i has exclusive access to memories in $\prod_{k\in[1..n]-\{j_1,\ldots,j_i\}} \mathsf{M}_k$. However, while executing $\mathsf{Imp}(o_i)$, the oracles $o_{j_1}, \cdots, o_{j_i}$ may be called, which causes reading and writing in $\mathsf{M}_{j_1}, \cdots, \mathsf{M}_{j_i}$. We allow ourselves to write either o_i or use the more explicit name $o_i^{o_{j_1},\cdots,o_{j_i}}$ for oracle o_i.

Given a function $k : \mathsf{N}_{\mathbb{O}} \to (\mathsf{N}_{\mathbb{O}} \to \mathbb{N})$ and $t \in \mathbb{N}$, we say that $\mathsf{Imp}(o_i)^{o_{j_1},\cdots,o_{j_i}}$ is $(k(o_i), t)$-bounded, if *one* execution makes at most $k(o_i)(o_{j_m})$ calls to o_{j_m} and takes at most time t.

We write $\{o_1, \ldots, o_n\}^{o_{j_1},\cdots,o_{j_i}}$ instead of $\{o_1^{o_{j_1},\cdots,o_{j_i}}, \ldots, o_n^{o_{j_1},\cdots,o_{j_i}}\}$.

□

In the sequel of this chapter, the oracles systems which we specify are written according to these conventions, in particular with respect to the memory decomposition.

Observing cryptographic primitives at the level of abstract constructions, we notice that

they are classically constructed on top of a set of probabilistic inner primitives, e.g. hash functions or block-ciphers, which are applied successively to the input message. In the definition of the security notions associated with these constructions, an adversary has the possibility to query both the global construction and inner primitives. In these systems, the only link between the inner primitives lies in the executions of the global construction. In this section, we propose a new notion to capture the particular architecture of the resulting oracle system in our framework.

VI.1.2 — Definition of Overlayers and Their Application to a System

We notice that two things are quite common in existing designs. Firstly, it is frequent that the order of the application of the inner primitives in the algorithm does not depend on the input. Secondly, it is often the case that during an execution of the algorithm, from one point on, inner primitive results appear in the computation of the final output. This separation in two phases of the implementation of primitives is what is later captured by the notion of pivot. These remarks are particularly well illustrated by iterative hash constructions. In the sequel, we restrict our scope and comments to hash constructions in particular.

Hash functions deal with input messages of any length and produce a hash of a fixed length, whereas the inner primitives on top of which they are built have fixed input/output length. Therefore, hash functions are based on *domain extenders*, which specify how the input message is split into blocks and how the inner primitives are applied to one block and the previous inner primitive outputs. A widely used domain extender was proposed by Merkle [Mer89] and Damgård [Dam90].

In [BMN09], Bhattacharyya et. al. present a formal definition for domain extenders. Though applicable to several known constructions, this definition is limited in that it does not capture constructions that include a post-processing function. Such a function is used to compute the global hash result out of the multiple inner primitive outputs; *it cannot contain any application of an inner primitive*. Another limitation is that it does not deal with the case of multiple inner primitives. For instance, neither the ChopMD [CDMP05] nor the Grøstl [GKM$^+$11] constructions fall in the scope of this definition. These limitations motivate the introduction of the new notion of *overlayer*.

Intuitively, the set of inner primitives are gathered in an oracle system $\mathbb{O}$. An $\mathbb{O}$-overlayer h provides everything needed to specify an oracle $\mathcal{H}$ calling successively oracles in $\mathbb{O}$. Moreover, the definition of overlayers exploits that the order in which the inner primitives are called does not depend on the input. In other words, the sequence of calls of inner primitives generated by every hash input is the prefix of a statically known finite sequence $[o^1, \ldots, o^{\mathfrak{L}}]$ of inner primitives.

Definition (Overlayer). Consider an oracle system $\mathbb{O}$ with oracle names in $\mathsf{N}_{\mathbb{O}}$. An $\mathbb{O}$-overlayer h is a tuple $(\mathsf{In}_{\mathcal{H}}, \mathsf{Out}_{\mathcal{H}}, [o^1, \ldots, o^{\mathfrak{L}}], \mathsf{init}, \mathsf{piv}, \Theta, (\mathsf{H}_j)_{j\in\{1..\mathfrak{L}\}}, \mathsf{H}_{post})$, where:

— $\mathsf{In}_{\mathcal{H}}$ and $\mathsf{Out}_{\mathcal{H}}$ are finite sets of bitstrings defining query and answer domain of the hash design.

— $[o^1, \ldots, o^{\mathfrak{L}}]$ is the statically known sequence of oracles in $\mathsf{N}_{\mathbb{O}}$, which describes the order in which the oracles are queried.

— $\mathsf{init} : \mathsf{In}_{\mathcal{H}} \to [1, \mathfrak{L}]$ outputs the number $\mathsf{init}(x)$ of oracle calls necessary for computing the hash of x. For computing the hash of an input x, the sequence of oracle calls is the prefix

of $[o^1, \ldots, o^{\mathfrak{L}}]$ of length $\mathsf{init}(x)$.

— $\mathsf{piv} : \mathsf{In}_{\mathcal{H}} \to [1, \mathfrak{L}]$ outputs a pivot index, which we require to be less than the number of calls to compute the hash of the input (i.e. $\forall x, \mathsf{piv}(x) \leq \mathsf{init}(x)$). We require that for any x, x', the pivot oracle for x is the same as the pivot oracle for x', namely, that $o^{\mathsf{piv}(x)} = o^{\mathsf{piv}(x')}$. This common oracle is referred to as the pivot oracle, and denoted by o_{piv}.

— Input transformation

$$\Theta \ : \left| \begin{array}{lll} \mathsf{In}_{\mathcal{H}} & \to & (\{0,1\}^{\leq r})^+ \\ x & \mapsto & \Theta(x) = (\theta_1(x), \ldots, \theta_{\mathsf{init}(x)}(x)) \end{array} \right.$$

where $\theta_j(x)$ is the function of the input used to compute the j-th query to oracle o^j. It usually consists in a block of length r of the padded input x. We suppose that Θ is injective.

— functions $\mathsf{H}_1 : \{0,1\}^{\leq r} \to \mathsf{In}(o^1)$ and $\mathsf{H}_j : \{0,1\}^{\leq r} \times \mathsf{Xch} \to \mathsf{In}(o^j)$ for $j \geq 2$ compute the j-th query performed by $\mathcal{H}$ using $\theta_j(x)$ and when $j \geq 2$ the previous step exchange with oracle o^{j-1}.

— Post-processing function

$$\mathsf{H}_{post} \ : \left| \begin{array}{lll} \mathsf{In}_{\mathcal{H}} \times \mathsf{Out}_{o_{\mathsf{piv}}} \times \mathsf{Xch}^* & \to & \mathsf{Out}_{\mathcal{H}} \\ (x, y, Q) & \mapsto & \mathsf{H}_{post}(x, y, Q) \end{array} \right.$$

computing the hash of x, if $Q = (o^k, q^k, a^k)_{k \in [1, \mathsf{init}(x)]}$ is the list of exchanges generated by the H_j functions for x. We impose that H_{post} only depends on pivot and post-pivot queries. Namely, for all lists Q and Q' of exchanges that coincide on post-pivot exchanges, outputs of H_{post} coincide: for all x, y, if $[Q]_{k > \mathsf{piv}(x)} = [Q']_{k > \mathsf{piv}(x)}$ then $\mathsf{H}_{post}(x, y, Q) = \mathsf{H}_{post}(x, y, Q')$.

□

We emphasize that the post-processing function H_{post} is defined as a deterministic function and *does not perform any oracle call*. The set of $\mathbb{O}$-overlayers is denoted by $\mathbb{O}$-OverL. We can now proceed with the definition of the oracle $\mathcal{H}$ resulting from the composition of $\mathbb{O}$-overlayer h with $\mathbb{O}$.

Definition (Composition of an Overlayer with an Oracle System). The composition of an $\mathbb{O}$-overlayer h with $\mathbb{O}$ defines an oracle system which contains the oracles of $\mathbb{O}$ augmented with the overlayer oracle $\mathcal{H}$ given by:

— the memory $L_{\mathcal{H}}$ of oracle $\mathcal{H}$ is a mapping from $\mathsf{In}_{\mathcal{H}}$ to $\mathsf{Out}_{\mathcal{H}} \times \mathsf{Xch}^*$; its initial value is the empty mapping.

— The implementation of oracle $\mathcal{H}$ is:

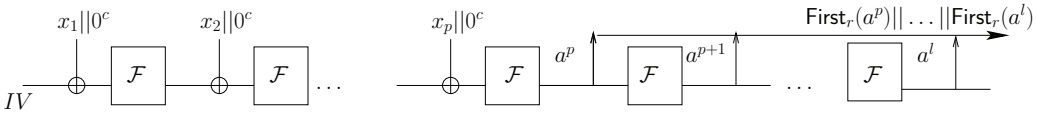


Figure VI.1 – The Sponge Construction

$\mathsf{Imp}(\mathcal{H})^{o^1,\ldots,o^{\mathfrak{L}}}(x) = \mathsf{if}\ x \in \mathsf{dom}(L_{\mathcal{H}})\ \mathsf{then}$
 $\mathsf{return}\ L_{\mathcal{H}}(x)$
else
 $l := \mathsf{init}(x);$
 $(x_1, \ldots, x_l) := \Theta(x);$
 $q^1 := \mathsf{H}_1(x_1);$
 $\mathsf{let}\ a^1 \leftarrow \mathsf{Imp}(o^1)(q^1)\ \mathsf{in}$
 $Q := [(o^1, q^1, a^1)];$
 $\mathsf{for}\ j = 2\ \mathsf{to}\ l\ \mathsf{do}$
  $q^j := \mathsf{H}_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}));$
  $\mathsf{let}\ a^j \leftarrow \mathsf{Imp}(o^j)(q^j)\ \mathsf{in}$
  $Q := Q :: (o^j, q^j, a^j);$
 endfor
 $a^f := \mathsf{H}_{post}(x, a^{\mathsf{piv}(x)}, Q);$
 $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q);$
 $\mathsf{return}\ a^f$
endif

□

In the remainder, we slightly abusive use notations and write $\mathsf{H}_{post}(x, a^{\mathsf{piv}(x)}, [Q]_{k>\mathsf{piv}(x)})$ instead of $\mathsf{H}_{post}(x, a^{\mathsf{piv}(x)}, Q)$ appearing in the implementation. It is allowed by the assumption that H_{post} only depends on post-pivot queries imposed on the post-processing function in the definition of overlayer.

Example 12. The sponge construction [BDPA07] relies on an inner primitive $\mathcal{F}$, which is a random function from $\{0,1\}^{r+c}$ into $\{0,1\}^{r+c}$, where r is the length of blocks parsed during preprocessing. The output size is parameterized by an integer we denote K. While the general design deals with any possible K, in the sequel we assume for sake of simplicity that $K = kr$, and refer the readers to [BDPA07] for more details. The sponge algorithm comprises two phases: in a first phase, the input is padded using Pad_{sp}, an injective, easily computable and invertible padding function that outputs a bitstring $x_1||\ldots||x_p$ of length $p * r$. Then, the algorithm iteratively applies a bitwise xor operation to $(x_j||0^c)$ and the previous answer from $\mathcal{F}$ to compute its next query. In a second phase, it queries $(k-1)$ more times $\mathcal{F}$ to get a collection of answers $(a^p, \ldots, a^l)$. The final output is then obtained by concatenation of the first r bits of each a^j: $\mathsf{First}_r(a^p)||\ldots||\mathsf{First}_r(a^l)$. The implementation is provided below and illustrated in figure VI.1.

The sponge hash oracle results from the application of an overlayer to $\mathcal{F}$. As a bound $\mathfrak{L}_{sp}$ on the number of oracle calls it is possible to perform during an execution of the hash oracle, we choose $\lceil \frac{2^{64}}{r} \rceil$. [1] The sequence of oracles is the list $[\mathcal{F}, \ldots, \mathcal{F}]$ (of length $\mathfrak{L}_{sp}$). Function init_{sp} is given by $\lceil |x|/r \rceil + \mathbf{1}_{\mathsf{Last}_r(x)=0^r} + k - 1$, which corresponds to number of

[1] This value is arbitrary; our choice is motivated by the fact that it is quite classic to choose a maximum length of 2^{64} for hash inputs in the literature.

r-blocks in the padding of input x and the $(k-1)$ calls necessary to finish the computation. The algorithm naturally falls into two parts, which allow us to specify a pivot value worth $\mathsf{piv}_{sp}(x) = \mathsf{init}_{sp}(x) - k + 1$. Indeed, after having gone through the $\mathsf{piv}_{sp}(x)$ blocks of the padded message, all $(k-1)$ calls performed are used in the computation of the final output.

In addition to that, for $j = 1..p$, we define $\mathsf{H}_j(\alpha, (\mathcal{F}, q^{j-1}, a^{j-1})) = (\alpha||0^c) \oplus a^{j-1}$ and $\mathsf{H}_{post}(x, a^{\mathsf{piv}_{sp}(x)}, [(\mathcal{F}, q^j, a^j)]_{j>\mathsf{piv}_{sp}(x)}) = \mathsf{First}_r(a^{\mathsf{piv}_{sp}(x)})|| \ldots ||\mathsf{First}_r(a^{\mathsf{init}_{sp}(x)})$. Moreover, we let $\theta_j(x) = x_j$ for $j \in [1..\mathsf{piv}_{sp}(x)]$, and $\theta_j(x) = 0^r$ for $j \in [\mathsf{piv}_{sp}(x) + 1..\mathsf{init}(x)_{sp}]$.

$\mathsf{In}(Sponge) = \{0,1\}^{\leq 2^{64}}$, $\mathsf{Out}(Sponge) = \{0,1\}^K$
$\mathsf{Imp}(Sponge)(x) =$
if $x \in \mathsf{dom}(L_{sp})$ then
  return $L_{sp}(x)$
else
  $l := \mathsf{init}_{sp}(x)$;
  $w := \mathsf{Pad}_{sp}(x)$;
  $p := \mathsf{piv}_{sp}(x)$;
  $(x_1, \ldots, x_p) := (w[1, r], \ldots, w[r * (p-1) + 1, r * p])$;
  for $j = 1$ to p do
    $q^j := (x_j||0^c) \oplus a^{j-1}$;
    let $a^j \leftarrow \mathcal{F}(q^j)$ in
    $Q := Q :: (\mathcal{F}, q^j, a^j)$;
  endfor
  for $j = p + 1$ to l do
    $q^j := a^{j-1}$;
    let $a^j \leftarrow \mathcal{F}(q^j)$ in
    $Q := Q :: (\mathcal{F}, q^j, a^j)$;
  endfor
  $a^f := \mathsf{First}_r(a^p)|| \ldots ||\mathsf{First}_r(a^l)$;
  $L_{sp} := L_{sp}.(x, a^f, Q)$;
  return a^f
endif

$\Diamond$

VI.1.3 — Security of Layered Systems : Indifferentiability

In the remaining of this chapter, the systems which we consider are all consisting of a set of inner primitives and an overlayer. Now that we have a formal definition of this kind of systems, we want to express a security notion for them in our framework: indifferentiability of a random oracle [MRH04, CDMP05].

To describe the security of hash functions, the idea is to assess their deviation from randomness. As stand-alone constructions, it is generally not really difficult to imagine that hash constructions look utterly random to any system or adversary interacting with them. Nevertheless, the task is more delicate to handle when taking into account the inner primitives. Unfortunately, in most of the systems we model, the adversary is granted access to inner primitives, so that we have to allow for these latter in the definition of a relevant security notion.

In a nutshell, we have on one side the layered system as defined by the overlayer designers, and on the other side a system which we call the idealized system, in which we want to implement the overlayer oracle as a completely independent random oracle. The problem is that, in this latter system, we still have to provide an implementation for inner primitives. If we leave them as in the original system, it would be easy to distinguish between original and idealized systems. Indeed, any adversary with access to the inner primitives can potentially verify the answers provided by the overlayer oracle by computing them on its own, querying inner primitives adequately.

However, whereas the idealized overlayer oracle cannot query inner primitives in the idealized system, inner primitives can query the idealized oracle. So here is the intuition behind the security notion we choose : find a way to implement the set of inner primitives in the idealized system so that it makes up for the total independence and randomness of the idealized overlayer oracle, possibly by querying this latter.

More formally, let us denote by $\mathcal{U}(\mathcal{H})$ the oracle system where $\mathcal{H}$ is the functional oracle distributed as the uniform distribution on $\mathsf{Out}_{\mathcal{H}}$. Given a system $\mathbb{S}$ implementing inner primitives, which we call the simulator, we estimate the advantage of adversaries in distinguishing the original system $(\mathcal{H}^{\mathbb{O}}, \mathbb{O})$ and its idealization $(\mathcal{U}(\mathcal{H}), \mathbb{S}^{\mathcal{U}(\mathcal{H})})$. It brings forth the following definition, which captures in our framework the classic notion of indifferentiability.

Definition (Indifferentiability). Consider an oracle $\mathbb{O}$ and an $\mathbb{O}$-OverL h. The system $(\mathcal{H}^{\mathbb{O}}, \mathbb{O})$ defined by the composition of h with $\mathbb{O}$ is said to be (k_s, t_s, ϵ)-*indifferentiable from its idealization* $\mathcal{U}(\mathcal{H})$, if there is an oracle set $\mathbb{S}^{\mathcal{U}(\mathcal{H})}$ that is (k_s, t_s)-bounded and such that the oracle systems $(\mathcal{H}^{\mathbb{O}}, \mathbb{O})$ and $(\mathcal{U}(\mathcal{H}), \mathbb{S}^{\mathcal{U}(\mathcal{H})})$ are compatible and for any adversary $\mathbb{A} \in \mathsf{Adv}(k, t)$, $\mathsf{Indiff}(\mathcal{H}, \mathbb{O}, \mathbb{S}) \leq \epsilon(k, t)$, where $\mathsf{Indiff}(\mathcal{H}, \mathbb{O}, \mathbb{S})$ denotes

$$|Pr[\mathbb{A}|(\mathcal{H}^{\mathbb{O}}, \mathbb{O}) : \mathsf{true}] - Pr[\mathbb{A}|(\mathcal{U}(\mathcal{H}), \mathbb{S}^{\mathcal{U}(\mathcal{H})}) : \mathsf{true}]|$$

□

Notice that the oracle set $\mathbb{S}$ in this definition is usually not a stand-alone oracle system, since it requires access to $\mathcal{U}(\mathcal{H})$ to compute its outputs. If we remember the discussion about dependencies at the beginning of the chapter, we can remark that what we try to do here is to covertly invert dependencies between one set of oracles and another.

VI.1.4 — Aim of This Chapter

If we aim to establish formal proofs of indifferentiability, it seems that the first thing to do is to find a proper way of defining the simulator. Indeed, it must be conceived keeping in mind that its outputs must look coherent with the outputs of the overlayer oracle. The main problem lies in the direct access that the adversary has to the overlayer oracle. Indeed, the simulator, while able to query the overlayer oracle, does not know the list of queries and answers performed to this latter by the adversary. As a consequence, coherence relies on the ability of the simulator to query the random oracle soon enough and on the right value. These two issues are illustrated by two examples below.

This first example is strongly inspired from the article [CDMP05] by Coron et. al. and illustrates the possible difficulty raised by computing the value of the relevant query to ask to the random oracle. We consider an oracle system composed of two oracles (in addition to initialization and finalization): a random oracle o_2 and an oracle o_1 which, on input x, queries

o_2 on $f(x)$, with f a one-way permutation drawn during initialization[2]. In this system, the overlayer oracle is o_1, it is this oracle which we replace with a random oracle. To design an interesting simulator, we have to find a way to implement o_2 to simulate the global behavior of the original system. Therefore, the equation $o_1(x) = o_2(f(x))$ must remain true, meaning that to a query y, o_2 must answer $o_2(y) = o_1(f^{-1}(y))$. Otherwise, an efficient attack of the adversary would be to query o_1 on a random x, query o_2 on $f(x)$ and check whether answers match. Of course, unless the time bound given to the simulator version of o_2 to compute an answer to a query y is large enough, it shall be very difficult for this oracle to compute $f^{-1}(y)$. In conclusion, the indifferentiability between our system and its idealization is lower bounded by the one-way bound of f.

A second issue is raised by the timing in which the overlayer oracle should be queried. Here is another example of system that illustrates this problem. Now, we have three oracles in addition to initialization and finalization. The overlayer oracle o_1 calls o_2 and then o_3 on its input: $o_1(x) = o_3(o_2(x))$. We assume that o_3 is a functional random oracle. If we want to replace o_1 by a random oracle and find a suitable simulator for o_2 and o_3, we have to take into account the following attack. An adversary can compute $o_3(y)$, and then try to find x such that $o_2(x) = y$. In the real setting, the answers to $o_3(y)$ and $o_1(x)$ should coincide. In the simulated setting, the problem is that a simulator potentially becomes aware that the adversary can build the answer to $o_1(x)$ at the moment $o_2(x)$ is queried. It is fine as long as $o_3(o_2(x))$ has not been queried already - because then the simulator can query $o_1(x)$ and impose o_3 to output $o_1(x)$ when queried on $o_2(x)$ - but it is a problem if the adversary has already asked for the value $o_3(o_2(x))$. The conclusion that we can draw from this example is that a simulator should be able to answer with very good probability *at the moment it is queried on a value* whether this value can be used in the construction of an output for the overlayer oracle.

The problem we aim to solve in this chapter is to build well-suited simulators, in that they should yield a reasonable indifferentiability bound. In the sequel, we present a generic process to construct such simulators. However, finding a simulator does not complete the work, since it remains to bound the advantage of indifferentiability adversaries. Thus, we subsequently develop a proof in the logic CIL of a theorem providing a generic way to compute the indifferentiability bound for these simulators.

VI.2 A Generic Theorem for Independent Inner Primitives

VI.2.1 — The Setting: Restriction on the Set of Inner Primitives

In the remainder of the chapter, we consider a given hash construction that uses a set of inner primitives which we assume *independent*. It is modeled by an overlayer h applied to an oracle system $\mathbb{O}$. Our goal is to provide a proof that the overlayer oracle $\mathcal{H}$ is (k_s, t_s, ε)-indifferentiable from oracle $\mathcal{U}(\mathcal{H})$ when $\mathbb{O}$ implements independent random functions for a specific function ε we shall detail further. Formally, we denote $\mathcal{U}(\mathbb{O})$ the oracle system compatible with $\mathbb{O}$, such that any oracle o_i of the system is functional and distributed as $\mathcal{U}(\mathsf{Out}_i)$. We must then provide an implementation for a generic simulator, which has the same set of oracles as $\mathbb{O}$. It could be the case that some of the oracles in $\mathbb{O}$ do not appear in the overlayer static sequence

[2]Notice that f is *not* an oracle of the system here.

of oracles to call. In such a case, it is clear that the oracle in the real and simulated world can be implemented in the same way. Therefore, we suppose that all oracles in $\mathbb{O}$ do appear in the overlayer sequence.

VI.2.2 — Construction of the Generic Simulator

We have seen in subsection VI.1.4 that one of the essential preoccupations when developing a simulator must be to preserve dependencies existing between oracle outputs in the real setting. Indeed, any inconsistency may allow the attacker to distinguish between the real and simulated world. In particular, if an equality holds in the real world and can be efficiently checked by the adversary, then the simulator has to contrive it to hold as well. Consequently, we get down to designing a way to represent these dependencies which the simulator can effectively use.

We are interested in depicting what we call *request chains*, that is, lists of exchanges with oracles in $\mathbb{O}$ that correspond to sequences of requests appearing during the computation of some hash value. To do that, graphs appear to be well-suited data structures, justifying our choice to define simulator graphs. They represent all potential chains that can be constructed out of a given list of exchanges. The idea is to represent these latter by vertices, and to draw an edge between two vertices if they can be two subsequent queries during the computation of a hash value. Edges of the form $((o, q, a), x_j, (o', q', a'))$ are labeled with the function of hash input x_j which would be necessary to compute q' out of (o, q, a) during the j-th step of computation of a hash value, i.e. $q' = \mathsf{H}_j(x_j, (o, q, a))$. To identify potential first queries, we choose to link them to a particular vertex that we name the root. It is the only vertex which is not related to an exchange, and which cannot be the target of an edge. Formally, this translates in the following definition.

Definition (Simulator Graph). A *simulator graph* $SG = (v_{root}, V, E)$ is given by:

— a root v_{root},

— a finite set of vertices $V \subseteq \mathsf{Xch}$,

— a set $E \subseteq (V \cup v_{root}) \times \{0,1\}^{\leq r} \times V$ of labeled edges such that:

1. for all $(o^{j-1}, q, a), (o^j, q', a') \in V$ (with $j \geq 2$), for all $x_j \in \{0,1\}^{\leq r}$, $((o^{j-1}, q, a), x_j, (o^j, q', a')) \in E$ if and only if $q' = \mathsf{H}_j(x_j, (o^{j-1}, q, a))$,
2. for all $(o^1, q, a) \in V$, $(v_{root}, x_1, (o^1, q, a)) \in E$ if and only if $q = \mathsf{H}_1(x_1)$.

□

The set of simulator graphs is denoted by $\mathcal{SG}$, and we define the initial simulator graph $SG_{init} = (v_{root}, \{v_{root}\}, \emptyset)$. The way in which we have defined these graphs calls for the fact that paths are evocative of the existence of request chains. The idea is that vertices represent control points through which the execution of the computation of a hash value has to go. The existence of a path between them represents their belonging to the execution of the hash oracle on a same input. Consequently, the simulator can identify which hash values the adversary can compute out of the queries it has asked: it corresponds to the existence of rooted and complete paths. This leads us to the following useful definitions.

Definition (Paths Properties in a Simulator Graph).

— Given a graph, a *path* is a chain $v_0 \xrightarrow{l_1} v_1 \xrightarrow{l_2} \ldots \xrightarrow{l_n} v_n$ of vertices v_i such that for all i, edge (v_i, l_i, v_{i+1}) belongs to the graph.

— A *rooted path* is a path starting with vertex v_{root}. A vertex is *rooted* whenever it belongs to a rooted path.

— A *meaningful path* is a rooted path such that if $[x_1, \ldots, x_L]$ is the list of labels on the sequence of edges, then there exists x such that $\forall j = 1..L$, $\theta_j(x) = x_j$.

— A meaningful path is said to be *determined* when $L = \mathsf{piv}(x)$. A meaningful path is said to be *complete* when $L = \mathsf{init}(x)$. In such cases, bitstring x is then said to *label* the meaningful path to which it corresponds.

□

The definition of edges in a simulator graph implies that such a graph is completely determined by its vertex set. We can thus define an application mapping a vertex set to the corresponding set of edges. Moreover, we provide an update function for the graph, to be applied by the simulator at every query it receives.

Definition (Edge Functions for Simulator Graphs). A simulator edge function $SEdge$ maps a vertex set to the matching set of edges appearing in the corresponding simulator graph. Namely, if V is a vertex set, then $SEdge(V)$ contains all edges such that:

1. for all $(o^{j-1}, q, a), (o^j, q', a') \in V$ (with $j \geq 2$), for all $x_j \in \{0,1\}^{\leq r}$, $((o^{j-1}, q, a), x_j, (o^j, q', a')) \in E$ if and only if $q' = \mathsf{H}_j(x_j, (o^{j-1}, q, a))$,
2. for all $(o^1, q, a) \in V$, $(v_{root}, x_1, (o^1, q, a)) \in E$ if and only if $q = \mathsf{H}_1(x_1)$.

A graph update function UpSG is a function which, on input an exchange (o, q, y) and a simulator graph $SG = (v_{root}, V, E)$, outputs the simulator graph SG' given by:

— $V' = V \cup \{(o, q, y)\}$;

— $E' = SEdge(V')$.

□

We assume that there exists a function $t_{\mathsf{UpSG}}(\alpha)$ bounding the execution time of the update process independently of the added vertex, in function of the number α of vertices in the graph[3].

At this point, we have enough elements at our disposal to sketch a simulating strategy. We must determine when the simulator has enough information to figure out which hash value can be anticipated. Indeed, if the simulator can do that, so does a smart adversary. Therefore, in such a situation, the simulator can outsmart the adversary by first querying the hash oracle itself and then imposing the values of the remaining exchanges necessary to complete a meaningful path, before the adversary has asked for them.

To pinpoint this situation, we have made two observations about the hash constructions we have come across. First, let us recall that in our definition of overlayer, the hash computation falls into two parts, namely before and after the pivot query is performed. The expression "pivot query of a hash value" is a shorthand that we use to name the pivot query to be performed to o_{piv} during the computation of a hash value in the real setting. Pre-pivot queries are those performed before the pivot query in a real execution, post-pivot queries are those performed after. The first observation is that in most hash constructions, it is difficult for an adversary to compute a hash value without querying for its matching pivot query, and before the pivot query, every pre-pivot query. Furthermore, we have observed that it is difficult for an adversary to perform a post-pivot query before the matching pivot query.

[3]This is a sound assumption in the sense that the number of edges that can exist between two vertices is bounded.

Therefore, a strategy to outsmart the adversary consists in trying to identify potential pivot queries when they are asked to $\mathbb{O}$, compute a hash input to which it is associated, query the random oracle on this input to get an answer t, and impose adequate values to post-pivot queries so that they are coherent with t. We move on to the formalization of the hypotheses required to put to work this strategy.

Obviously, we are going to use determined paths to achieve the two first steps of our strategy. As a result, we need to impose that given a prefix to a determined path and a query q which can make this path determined, the labels $x_1, \ldots, x_L$ appearing on the edges of the resulting determined path allow to compute a value $x \in \mathsf{In}_{\mathcal{H}}$ such that $\forall j \in [1..L]$, $\theta_j(x) = x_j$.

Afterwards, we assume the existence of an algorithm which, on input a graph and a query, identifies a rooted path rendered determined by this query when it exists, and in such a case outputs it along with a hash input labeling it. Such an algorithm is called a *path-finder*. Intuitively, it should have a non-trivial output as soon as there exists a satisfactory path, and any non-trivial output should correspond to a satisfying answer. This is captured by the following definition.

Definition (Path-Finder Algorithm). A *path-finder algorithm* $\mathsf{PathFinder}$ takes as input a query $q \in \mathsf{In}_{o_{\mathsf{piv}}}$ and a simulator graph SG. Its output is either the triple $(\mathsf{false}, \lambda, [\,])$, or a triple of the form $(\mathsf{true}, x, List)$, with $(x, List) \in \mathsf{In}_{\mathcal{H}} \times V^*$ such that:

1. if, for any answer y to q, there exists in the updated simulator graph $\mathsf{UpSG}((o_{\mathsf{piv}}, q, y), SG)$ a determined meaningful path then $\mathsf{PathFinder}$ outputs $(\mathsf{true}, \_, \_)$
2. if $\mathsf{PathFinder}(q, SG) = (x, [(o^1, q_1, y_1), \ldots, (o^{p-1}, q_{p-1}, y_{p-1})])$ then for any answer y to q, there exists in the updated graph $\mathsf{UpSG}((o_{\mathsf{piv}}, q, y)\ , SG)$ a determined meaningful path $v_{root} \xrightarrow{x_1} (o^1, q_1, y_1) \xrightarrow{x_2} \ldots \xrightarrow{x_p} (o_{\mathsf{piv}}, q, y)$ which is labeled by x (i.e. $p = \mathsf{piv}(x)$ and $\forall j \in [1..p]$ $\theta_j(x) = x_j$).

□

We assume that the execution time of the path-finder algorithm is bounded by a function $t_{\mathsf{PathFinder}}(\alpha)$ of the number α of vertices in the input simulator graph.

In case $\mathsf{PathFinder}$ outputs some bitstring and request chain $(x, [v_1, \ldots, v_{\mathsf{piv}(x)-1}])$, our simulating strategy imposes on the answer y provided by the simulator to match a query q and the values of the vertices $[v_{\mathsf{piv}(x)+1}, \ldots, v_{\mathsf{init}(x)}]$ to be coherent with $\mathcal{H}(x)$. More precisely, we would like to guarantee $\mathsf{H}_{post}(x, y, [v_j]_{j>\mathsf{piv}(x)}) = \mathcal{H}(x)$ in the simulated world. When $\mathcal{H}$ is implemented by $\mathcal{U}(\mathcal{H})$, the only way to compute the value for $t = \mathcal{H}(x)$ is to query the random oracle on x. After that, we would have to provide the simulator with a sampling algorithm to find a value for y and $[v_{\mathsf{piv}(x)+1}, \ldots, v_{\mathsf{init}(x)}]$ such that the equation holds.

For given values of x and t^0, there is a set of lists of vertices $[v_j]_{j>\mathsf{piv}(x)}$ and values for y such that $\mathsf{H}_{post}(x, y, [v_j]_{j>\mathsf{piv}(x)}) = t^0$, which we denote $PreIm(t^0)$. Notice here that if we have specified the overlayer such that $\mathsf{Out}_{\mathcal{H}}$ is larger than the actual range of $\mathcal{H}$, then $PreIm(t^0)$ may be empty. In the sequel we suppose that $\mathsf{Out}_{\mathcal{H}}$ *is the range of* $\mathcal{H}$. We can define an algorithm sampling a solution and preserving the original distribution on $((y, [v_j]_{j>\mathsf{piv}(x)}), t)$ as follows. Notice that the very existence of this algorithm is conditioned by the fact that for all (x, t^0), $PreIm(t^0)$ contains at least one element.

Definition (Forward Sampler Algorithm). A forward sampler algorithm $\mathsf{FwdSplr}$ is an algorithm which, on any input (x, t^0) outputs a pair $(y^0, [v_j^0]_{j>\mathsf{piv}(x)}) \neq (\lambda, [\,])$ sampled according to:

$$Pr[(y,[v_j]_{j>\mathsf{piv}(x)}) \leftarrow \mathsf{FwdSplr}(x,t^0) : (y,[v_j]_{j>\mathsf{piv}(x)}) = (y^0,[v_j^0]_{j>\mathsf{piv}(x)})] =$$
$$\frac{1}{Pr[\mathcal{U}(\mathcal{H})=t^0]} \times Pr\left[\begin{array}{l} y \leftarrow \mathcal{U}(o_{\mathsf{piv}}); v_{\mathsf{piv}(x)} := (o_{\mathsf{piv}},q,y); \\ (y_j \leftarrow \mathcal{U}(o^j); v_j = (o^j, \mathsf{H}_j(\theta_j(x), v_{j-1}), y_j);)_{j>\mathsf{piv}(x)} : \\ \quad (y,[v_j]_{j>\mathsf{piv}(x)}) = (y^0,[v_j^0]_{j>\mathsf{piv}(x)}) \wedge \mathsf{H}_{post}(x,y^0,[v_j^0]_{j>\mathsf{piv}(x)}) = t^0 \end{array}\right]$$

□

We assume that the execution time of this algorithm is upper-bounded by time t_{FwdSplr}, independently of possible inputs.

We can now provide implementations of oracles in the generic simulator. It uses a table $L_{\mathcal{S}}$ shared by all oracles in $\mathbb{O}$, which stores all triples of the form (o_i,q,y). We recall that $L_{\mathcal{S}}(o_i)$ denotes the list of all tuples starting with o_i appearing in $L_{\mathcal{S}}$ and that if (o_i,q,y) is one of these tuples, $L_{\mathcal{S}}(o_i,q)$ denotes the value y[4].

Definition (Generic Simulator Definition). Under the assumption that the choice of the pivot of our overlayer oracle allows to define both path-finder and forward sampler algorithms with execution times bounded respectively by $t_{\mathsf{PathFinder}}(\alpha)$ and t_{FwdSplr}, we let the generic simulator $\mathbb{S}$ be the following set of oracles compatible with $\mathbb{O}$. They have memories $(L_{\mathcal{S}},SG) \in \mathsf{Xch}^* \times \mathcal{SG}$, and the initial memory $\bar{m}$ of a system containing the simulator is chosen so that $\bar{m}.(L_{\mathcal{S}},SG) = ([\,],SG_{init})$. Moreover o_{piv} is implemented as follows:

$\mathsf{Imp}_{\mathbb{S}}(o_{\mathsf{piv}})^{\mathcal{H}}(q) =$ **if** $q \in \mathsf{dom}(L_{\mathcal{S}}(o_{\mathsf{piv}}))$ **then**
  return $L_{\mathcal{S}}(o_{\mathsf{piv}},q)$
elsif $\mathsf{PathFinder}(q,SG) = (\mathsf{true},x,List)$ **then**
  let $t \leftarrow \mathcal{H}(x)$ **in**
  $(y,L) := \mathsf{FwdSplr}(x,t)$;
  $L_{\mathcal{S}} := L_{\mathcal{S}}.((o_{\mathsf{piv}},q,y) :: L)$;
else let $y \leftarrow \mathcal{U}(o_{\mathsf{piv}})$ **in**
  $L_{\mathcal{S}} := L_{\mathcal{S}}.(o_{\mathsf{piv}},q,y)$;
endif
$SG := \mathsf{UpSG}((o_{\mathsf{piv}},q,y),SG)$;
return y

For any $o \neq o_{\mathsf{piv}}$ in $\mathsf{N}_{\mathbb{O}}$, the simulator implementation is:

$\mathsf{Imp}_{\mathbb{S}}(o)^{\mathcal{H}}(q) =$ **if** $q \in \mathsf{dom}(L_{\mathcal{S}}(o))$ **then**
  return $L_{\mathcal{S}}(o,q)$
else let $y \leftarrow \mathcal{U}(o)$ **in**
endif
$G := \mathsf{UpSG}((o,q,y),G)$;
$L_{\mathcal{S}} := L_{\mathcal{S}}.(o,q,y)$;
return y

The number of vertices in the simulator graph is bounded by the total number K of calls to oracles in $\mathbb{O}$. The implementation of o_{piv} is (k_s,t_s)-bounded, where $k_s(o) = \mathbf{1}_{o=\mathcal{H}}$, and $t_s = t_{\mathsf{UpSG}}(K) + t_{\mathsf{FwdSplr}} + t_{\mathsf{PathFinder}}(K)$. The implementation of the other oracles is $(0,t'_s)$-bounded, where $t'_s = t_{\mathsf{UpSG}}(K)$.

□

[4]This value is unique by construction.

This simulator works completely independently of the fact that an update might result in creating a great number of edges between two given vertice of the simulator graph, or that there can exist multiple determined paths from which the path-finder has to choose. However, we notice that if it is possible that the path-finder can answer two distinct hash inputs x, x' corresponding to meaningful determined paths, the simulator can only anticipate the adversary queries for one of these inputs to $\mathcal{H}$. If the adversary can easily uncover such values, our simulation strategy is flawed and should yield a large indifferentiability bound.

VI.2.3 — A Generic Way to Bound Indifferentiability

Even though the path-finder and forward sampler can prevent some obvious inconsistencies with respect to the idealized system, there are still cases in which they are not sufficient. We try to provide a little intuition about what can go wrong before presenting the formal definitions capturing sources of incoherence. When a pivot query is made to the simulator, consistency can only be enforced if, first, the path-finder can detect that it *is* a pivot query, and secondly, the pivot and post-pivot queries are not already bound to an answer.

Concerning the first point, given the hypotheses we have imposed on the path-finder, we can only expect that it detects pivot queries in case all pre-pivot queries have been performed before by the adversary. In this case, the meaningful path does exist in the simulator graph. On the contrary, if the adversary manages to anticipate hash values without asking all pre-pivot queries and then the matching pivot query, our path-based simulating strategy does not work: there is probably no path to identify in the graph when the pivot query is performed. This situation highlights a first possible source of incoherence between idealized and real systems.

Furthermore, concerning the eventuality that a pivot or post-pivot query corresponding to a hash input x is already bound to an answer, we notice that pivot or post-pivot queries can be determined in two ways : either as an adversary query to an oracle in $\mathbb{O}$, or during the construction of a hash output for a hash input $x' \neq x$. In both cases, when the simulator detects that it is asked the pivot query corresponding to x, it carries on running the forward sampler, but when updating the simulator graph and list of queries, it stumbles upon a preexisting vertex.

This discussion allows to foresee that some kind of graph structure can become handy to capture our inconsistency events. The simulator graph deals with direct $\mathbb{O}$ queries and represents all possible vertices existing between them. What we need is more than that: we want to capture dependencies enforced in the real setting by intermediate queries (performed by $\mathcal{H}$ to oracles in $\mathbb{O}$) in addition to direct and anticipated queries. To this end, we introduce an intermediate system, the *anticipating system* $\mathbb{O}_{ant}$, mostly consisting of the real system augmented with the anticipation of the post-pivot queries by oracle o_{piv}, and a matching graph construction.

We start by introducing the concept of visibility to help categorizing vertices in function of what the adversary knows about them and in function of the order in which oracles are queried for them. If a vertex appears in the graph as a result of a direct query to oracles in $\mathbb{O}$, it is considered visible to the adversary. Concerning vertices appearing in the graph on behalf of intermediate queries involved in the computation of the output for a query to $\mathcal{H}$, the pivot vertex and post-pivot vertices are considered partially visible, whereas pre-pivot queries are considered invisible. Finally, the last class of vertices we have to deal with is anticipated queries, which we consider visible. We take these elements into account in our definition for a

characteristic graph below.

Definition (Characteristic Graph). A *characteristic graph* CG is defined by a tuple $(v_{root}, CV, CE, \mathcal{V})$ where :

— v_{root}, CV and CE are such that for all edge $(v, l, (o, q, y))$, either $v = v_{root}$ and $q = \mathsf{H}_1(l)$ or $v \neq v_{root}$ and there exists $j \geq 2$ s.t. $q = \mathsf{H}_j(l, v)$.

— $\mathcal{V}$ is a visibility map, which associates to every vertex in CV a value in $\{Inv, PVis, Vis\}$ (standing for invisible, partially visible and visible and are ordered this way).

— The edge set of the simulator graph one can build out of the visible vertices is included in CE. Formally, $SEdge(\mathcal{V}^{-1}(Vis)) \subseteq CE$.

□

The set of characteristic graphs is denoted by $\mathcal{CG}$. We distinguish a particular graph $CG_{init} = (v_{root}, [\,], \emptyset, \mathcal{V}_{init})$ with $\mathsf{dom}(\mathcal{V}_{init}) = \emptyset$ which we call the initial characteristic graph. We use the term *non-visible* to refer to vertices which are either partially visible or invisible. Moreover, we talk about visibility of *queries*: the visibility of a query (o, q) is the same as that of the (unique) vertex v in a characteristic graph such that $v = (o, q, \_)$. We will construct the characteristic graph so that a pair of non-visible vertices are linked if an execution of $\mathcal{H}$ has successively been through them, while an edge will link two visible vertices as soon as it is possible: we have imposed that the simulator graph on visible queries is included in the characteristic graph.

We say that a vertex (or even a query) is visibly (resp. non-visibly) rooted when there exists a path linking it to the root containing only visible vertices (resp. containing at least one non-visible vertex).

We can now provide the precise implementations of the anticipating system $\mathbb{O}_{ant}$, which anticipates values of post-pivot queries when pivot queries are detected by a path-finder algorithm, and computes visibility labels dynamically. To this end, we substitute the code of o_i to every call to another oracle o_i in the implementation of $\mathcal{H}$. We also create a table $L_{\mathcal{S}}$ shared and updated by all oracles, containing all tuples of the form (o_i, q, y, lbl) ever computed, in order to ensure coherence between direct and indirect calls to o_i. As previously, $L_{\mathcal{S}}(o_i)$ denotes the list of exchanges starting with o_i and belonging to $L_{\mathcal{S}}$; if (o_i, q, y, lbl) is one of these tuples, then $L_{\mathcal{S}}(o_i, q)$ denotes the pair (y, lbl) (which is unique by construction). We denote (L, lbl) a list of exchanges consisting in L except that all visibility labels are replaced by lbl, and $(L|lbl)$ denotes the restriction of list L to the elements of label lbl. The implementations are provided in figure VI.2.

Let us describe what the inconsistency events outlined above correspond to in terms of a characteristic graph updated at each direct or indirect query to an oracle in $\mathbb{O}$. Intuitively, many situations which we have depicted above result in some sort of collision. We formalize what we mean by collision vertex as follows.

Definition (Collision Vertex). Given a characteristic graph CG, v is a collision vertex, denoted $v \in \mathsf{CollVertex}(CG)$, if there exist at least two distinct edges having v as a target (i.e. there exists $(v', l') \neq (v'', l'')$ such that edges (v', l', v) and (v'', l'', v) appear in the graph), and one of them belongs to a meaningfully rooted path going through v. □

A careful examination reveals that all inconsistencies result in the creation of a connection between a preexisting vertex and a rooted meaningful path. The configurations in which this can happen fall into three categories. First, it can result in a collision vertex: this first event is named $\mathsf{Collide}$. Second, it can happen due to the creation of a visible vertex linking a

$\mathsf{Imp}_{\mathbb{O}_{ant}}(\mathcal{H})(x) =$
if $x \in \mathsf{dom}(L_{\mathcal{H}})$ then
  $(a^f, Q) := L_{\mathcal{H}}(x)$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q)$;
  return a^f
else
  $l := \mathsf{init}(x)$;
  $p := \mathsf{piv}(x)$;
  $(x_1, \ldots, x_l) := \Theta(x)$;
  $(o^1, q^1) := \mathsf{H}_1(x_1)$;
  if $q^1 \in \mathsf{dom}(L_{\mathcal{S}}(o^1))$ then
    $(a^1, lbl) := L_{\mathcal{S}}(o^1, q^1)$;
    $Q := [(o^1, q^1, a^1, lbl)]$;
  else let $a^1 \leftarrow \mathcal{U}(o^1)$ in
    $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^1, q^1, a^1, Inv)$;
    $Q := [(o^1, q^1, a^1, Inv)]$;
  endif
  for $j = 2$ to $p - 1$ do
    $(o^j, q^j) := \mathsf{H}_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$;
    if $q^j \in \mathsf{dom}(L_{\mathcal{S}}(o^j))$ then
      $(a^j, lbl) := L_{\mathcal{S}}(o^j, q^j)$;
      $Q := Q :: (o^j, q^j, a^j, lbl)$;
    else let $a^j \leftarrow \mathcal{U}(o^j)$ in
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, Inv)$;
      $Q := Q :: (o^j, q^j, a^j, Inv)$;
    endif
  endfor
  for $j = p$ to l do
    $(o^j, q^j) := \mathsf{H}_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$;
    if $q^j \in \mathsf{dom}(L_{\mathcal{S}}(o^j))$ then
      $(a^j, lbl) := L_{\mathcal{S}}(o^j, q^j)$;
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, max(PVis, lbl))$;
      $Q := Q :: (o^j, q^j, a^j, max(PVis, lbl))$;
    else let $a^j \leftarrow \mathcal{U}(o^j)$ in
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, PVis)$;
      $Q := Q :: (o^j, q^j, a^j, PVis)$;
    endif
  endfor
  $a^f := \mathsf{H}_{post}(x, a^p, [Q]_{j>p})$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q)$;
  return a^f
endif

If $o_i \neq o_{\mathsf{piv}}$:
$\mathsf{Imp}_{\mathbb{O}_{ant}}(o_i)(q) =$
if $q \in \mathsf{dom}(L_{\mathcal{S}}(o_i))$ then
  $(y, \_) := L_{\mathcal{S}}(o_i, q)$;
else let $y \leftarrow \mathcal{U}(o_i)$ in
endif
$L_{\mathcal{S}} := L_{\mathcal{S}}.(o_i, q, y, Vis)$;
$SG := \mathsf{UpSG}((o_i, q, y), SG)$;
return y

$\mathsf{Imp}_{\mathbb{O}_{ant}}(o_{\mathsf{piv}})^{\mathcal{H}}(q) =$
if $q \in \mathsf{dom}(L_{\mathcal{S}}(o_{\mathsf{piv}})|Vis)$ then
  $(y, Vis) := L_{\mathcal{S}}(o_{\mathsf{piv}}, q)$;
elsif $\mathsf{PathFinder}(q, SG) = (\mathsf{true}, x, List)$ then
  let $t \leftarrow \mathcal{H}(x)$ in
  $(o_{\mathsf{piv}}, q, y) :: L := \Pi_3(L_{\mathcal{H}}(x))_{j \geq \mathsf{piv}(x)}$;
  $L_{\mathcal{S}} := L_{\mathcal{S}}.((o_{\mathsf{piv}}, q, y, Vis) :: (L, Vis))$;
elsif $q \in \mathsf{dom}(L_{\mathcal{S}}(o_{\mathsf{piv}})|PVis, Inv)$ then
  $(y, \_) := L_{\mathcal{S}}(o_{\mathsf{piv}}, q)$;
  $L_{\mathcal{S}} := L_{\mathcal{S}}.(o_{\mathsf{piv}}, q, y, Vis)$;
else let $y \leftarrow \mathcal{U}(o_{\mathsf{piv}})$ in
  $L_{\mathcal{S}} := L_{\mathcal{S}}.(o_{\mathsf{piv}}, q, y, Vis)$;
endif
return y
  $SG := \mathsf{UpSG}((o_{\mathsf{piv}}, q, y), SG)$;

Figure VI.2 – Implementations of the Oracles in the Anticipating System $\mathbb{O}_{ant}$

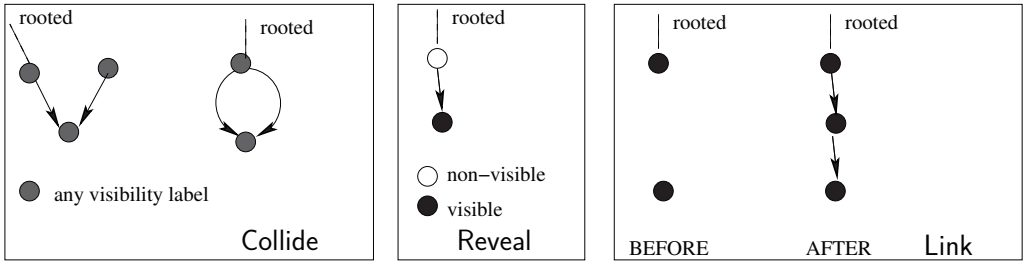


Figure VI.3 – Inconsistency Events

preexisting meaningfully and visibly rooted path and a preexisting visible vertex. Namely, if the adversary manages to ask all the direct queries necessary to compute a hash value in another order than that imposed by the implementation of $\mathcal{H}$, this configuration happens. This event is named Link. Third, a meaningful non-visibly rooted path can be connected with a preexisting visible vertex. Then, there exists an edge between a non-visible and a visible vertex. This event is called Reveal. The events are pictured in figure VI.3.

We now define a way to map traces to graph sequences, to be able to capture the occurrence of our inconsistency events. Let us start by the definition of a mapping from memories of the system to characteristic graphs.

Definition (Memory to Characteristic Graph Map). We let Γ be the function mapping memories m containing lists $L_{\mathcal{S}}$ and $L_{\mathcal{H}}$ to a characteristic graph $CG = (v_{root}, CV, CE, \mathcal{V})$ as follows:

— For all $(o, q, y, lbl) \in L_{\mathcal{S}}$, $(o, q, y) \in CV$ and $\mathcal{V}((o, q, y)) = lbl$.

— The set of edges CE contains all edges in $SEdge(\mathcal{V}^{-1}(Vis))$, and for all $(x, t, Q) \in L_{\mathcal{H}}$, edge $(v_{root}, \theta_1(x), Q[1])$ and edges $(Q[j-1], \theta_j(x), Q[j])$ for $2 \leq j \leq \mathsf{init}(x)$ belong to CE.

□

We notice that the blunt solution to map traces to sequences of graphs by changing each memory into a graph using Γ turns out to be unsound, because traces only take into account exchanges performed with the adversary. Indeed, let us assume that Reveal happens during a query to $\mathcal{H}$ as a result of an adversarial call to o_{piv}. If we look at the graph before the call to o_{piv}, it certainly shows a visible and meaningfully rooted path to the query made by the adversary. Now looking at the graph after the call to o_{piv} shows that the latter path has been completed and is fully visible. One cannot know whether Reveal has occured by looking at this pair of graphs: the usual granularity of a trace does not allow us to properly capture our inconsistency events.

To address this issue, we need to deduce from input and output memories of each step whether an internal $\mathcal{H}$ call has been performed and in such a case the value of the intermediate memory, before all vertices are colored visible. These ideas are formalized by the following mapping, called a step transformer.

Definition (Step Transformer). Given a step $m \xrightarrow{(o,q,a)} m'$ and an input characteristic graph, the step transformer $\mathsf{StTr} : \mathsf{M}_{\mathbb{O}_{ant}} \times \mathsf{M}_{\mathbb{O}_{ant}} \times \mathsf{Xch} \to \mathcal{CG}^*$ maps it to a graph sequence according to the following conditions:

$$\mathsf{StTr}(m, m', (o, q, a)) = \begin{cases} \Gamma(m) \xrightarrow{(o,q,a)} \Gamma(m') \text{ if } o \neq o_{\mathsf{piv}} \text{ or } q \text{ is not} \\ \quad \text{visibly rooted in } m.SG \\ \Gamma(m) \xrightarrow{(\mathcal{H},x,t)} \Gamma(m'') \xrightarrow{(o_{\mathsf{piv}},q,a)} \Gamma(m') \\ \quad \text{otherwise, where } (x,t,Q) = \mathsf{Last}(m'.L_{\mathcal{H}}), \\ \quad m'' = m.[L_{\mathcal{S}} \mapsto L_{\mathcal{S}}.Q, L_{\mathcal{H}} \mapsto L_{\mathcal{H}}.(x,t,Q)] \end{cases}$$

□

We call *graph stages* the graph sequences of the form $CG \xrightarrow{(o,q,a)} CG'$. Now that we know how to transform a step into a graph stages, we can provide a definition for a function transforming a partial trace starting with the initial state $\bar{m}$ into graph stages.

Definition (Trace Transformer). The *trace transformer* $\mathsf{GraphTr}$ is an application that maps a partial trace of interaction with $\mathbb{O}_{ant}$ given by

$$\bar{m} \xrightarrow{xch_1} m_1 \xrightarrow{xch_2} \ldots m_{k-1} \xrightarrow{xch_k} m_k$$

to a *partial graph trace* defined as

$$\mathsf{StTr}(\bar{m} \xrightarrow{xch_1} m_1) \cdot \ldots \cdot \mathsf{StTr}(m_{k-1} \xrightarrow{xch_k} m_k),$$

where $\cdot$ denotes the concatenation of sequences. □

A *graph trace* is the image by the transformer of a trace. A *graph event* is a predicate E_{CG} over partial graph traces. We choose to abusively denote E_{CG} too the event $E_{CG} \circ \mathsf{GraphTr}$ on partial traces.

Events on traces can be defined by means of a temporal operator and a step-predicate. Similarly, we can use predicates on graph stages and temporal operators to define graph events. Such predicates are called graph stage predicates, they are mappings of type $(\mathcal{CG} \times CV \times \mathcal{CG}) \to \mathsf{Bool}$. Our problematic events $\mathsf{Collide}$, Reveal and Link which we have introduced before are formally defined as graph stage predicates as follows.

Definition (Inconsistency Predicates). Let $CG = (v_{root}, CV, CE, \mathcal{V})$ and $CG' = (v_{root}, CV', CE', \mathcal{V}')$ be two characteristic graphs such that $CG \xrightarrow{(o,q,y)} CG'$ is a graph stage.

— $\mathsf{Collide}$ is true at graph stage $CG \xrightarrow{(o,q,y)} CG'$ iff a collision vertex is created at this step, i.e. $\mathsf{CollVertex}(CG') - \mathsf{CollVertex}(CG) \neq \emptyset$.

— Reveal is true at graph stage $CG \xrightarrow{(o,q,y)} CG'$ if CG' contains a vertex which is the first visible vertex of a meaningful non-visibly rooted path: there exists $v, v' \in CV'$ s.t. firstly, $\mathcal{V}'(v) = Vis$ and $\mathcal{V}'(v') \neq Vis$, secondly, a meaningful path goes through v' in CG' and lastly edge $(v', \_, v)$ is in CE'.

— Link is true at graph stage $CG \xrightarrow{(o,q,y)} CG'$ if (o, q, y) is a visible vertex of CG' not belonging to CG and there exists a visible vertex (o', q', y') in CV such that a visibly meaningfully rooted path of CG' goes through (o, q, y) and (o', q', y').

□

We have now defined all the elements needed to state the expected theorem, according to which the advantage of an adversary in distinguishing real from simulated setting (with the generic simulator) is bounded by the probability that either $\mathsf{Collide}$, Reveal or Link become true at some point of the execution of the anticipating system.

Theorem VI.1. *Let h be an overlayer using as inner-primitives the oracle system idealized as $\mathcal{U}(\mathbb{O})$. The composition of the overlayer and $\mathcal{U}(\mathbb{O})$ yields oracle system $(\mathcal{H}^{\mathcal{U}(\mathbb{O})},\mathcal{U}(\mathbb{O}))$,*

We denote by $\mathbb{S}$ the generic simulator and $\mathbb{O}_{ant}$ the anticipating system, which we have defined above. Then, for all adversaries $\mathbb{A} \in \mathsf{Adv}(k,t)$,

— *$\mathbb{S}$ is (k_s,t_s)-bounded with $k_s(\mathcal{H}) = 1$ and k_s is null otherwise, and $t_s = t_{\mathsf{UpSG}}(k'+1) + t_{post} + t_{\mathsf{PathFinder}}(k'+1)$, where $k' = \sum_{o\in\mathbb{O}} k(o)$.*

— *the indifferentiability advantage is bounded by the probability of inconsistency events: $|Pr[\mathbb{A}|(\mathcal{H}^{\mathcal{U}(\mathbb{O})},\mathcal{U}(\mathbb{O})) : \mathsf{true}] - Pr[\mathbb{A}|(\mathcal{U}(\mathcal{H}),\mathbb{S}^{\mathcal{U}(\mathcal{H})}) : \mathsf{true}]| \leq Pr[\mathbb{A}|\mathbb{O}_{ant} : \mathsf{F}_{\mathsf{Collide}\vee\mathsf{Reveal}\vee\mathsf{Link}}]$*

where t_{UpSG}, t_{FwdSplr} and $t_{\mathsf{PathFinder}}$ respectively bound of the execution time of UpSG*, forward sampler* FwdSplr *and path-finder* PathFinder *used in the simulator.*

VI.3 Proof of the Theorem

In this section we provide a detailed proof in CIL for the generic theorem VI.1. The trees summing up the proof can be found in figure VI.4. Here is the outline of our reasoning. The proof starts with a layered oracle system implemented as in the definition, which we must relate to the anticipating system $\mathbb{O}_{ant}$. The formal relation between the real setting and the anticipating system is mostly one of determinization, though it seems easier to introduce intermediate systems to write the underlying distribution properly. This is developped in VI.3.1 and corresponds to the left tree in figure VI.4.

Then, the anticipating system is transformed into a system $\mathbb{Q}_5$ closer to the simulated setting. We show that the probability to distinguish between $\mathbb{O}_{ant}$ and $\mathbb{Q}_5$ is bounded by the same bound as our theorem: $\Pr[\mathbb{A}|\mathbb{O}_{ant} : \mathsf{F}_{\mathsf{Collide}\vee\mathsf{Reveal}\vee\mathsf{Link}}]$. To justify this, we successively present a series of modified systems, from $\mathbb{Q}_2$ to $\mathbb{Q}_5$, and the formal link existing between one and the next, before being able to conclude in VI.3.6. This corresponds to the middle tree in figure VI.4.

Eventually, we argue that $\mathbb{Q}_5$ is determinized by the simulated setting. The global conclusion finally follows from transitivity of the indistinguishabiltiy relation.

VI.3.1 — Relation Between $(\mathcal{H}^{\mathbb{O}},\mathbb{O})$ and $\mathbb{O}_{ant}$: Left Tree

We provide the specification of the intermediate system $\mathbb{Q}_0$ in figure VI.5. It mostly consists in the anticipating system but the anticipation part. Namely the visibility labels are added and computed dynamically, and the branching is modified in o_{piv}, but $\mathcal{H}$ is not called by o_{piv} to anticipate post-pivot queries. Moreover, a list *Pivot* is added to the memory, to collect detected pivot queries, their answers and the value of x output by the path-finder PathFinder. Memories of $(\mathcal{H}^{\mathbb{O}},\mathbb{O})$ contain lists L_{o_i}, $L_{\mathcal{H}}$. Memories of $\mathbb{Q}_0$ contain a shared table $L_{\mathcal{S}}$ collecting all tuples of the form (o_i,q,y,lbl), a simulator graph SG, a list $L_{\mathcal{H}}$ and list *Pivot*. This system is in bisimulation up to with the real setting, for relation R defined as follows. Memories m and m' are in relation iff they are equal when they belong to the same memory space and if $m \in \mathsf{M}_{(\mathcal{H}^{\mathbb{O}},\mathbb{O})}$ and $m' \in \mathsf{M}_{\mathbb{Q}_0}$:

— $(x,a^f,Q) \in m.L_{\mathcal{H}}$ iff $(x,a^f,Q) \in m'.L_{\mathcal{H}}$, but the order of appearance might not be the same,

$$\dfrac{\dfrac{\text{Left tree}}{(\mathcal{H}^{\mathbb{O}},\mathbb{O}) \sim_0 \mathbb{O}_{ant}} \quad \dfrac{\text{Middle tree}}{\mathbb{O}_{ant} \sim_\epsilon \mathbb{Q}_5} \quad \dfrac{(\mathcal{U}(\mathcal{H}),\mathbb{S}^{\mathcal{U}(\mathcal{H})}) \leq_{\det,\gamma'} \mathbb{Q}_5}{(\mathcal{U}(\mathcal{H}),\mathbb{S}^{\mathcal{U}(\mathcal{H})}) \sim_0 \mathbb{Q}_5}\ \text{I-Det}}{(\mathcal{H}^{\mathbb{O}},\mathbb{O}) \sim_\epsilon (\mathcal{U}(\mathcal{H}),\mathbb{S}^{\mathcal{U}(\mathcal{H})})}$$

Left tree:

$$\text{I-Bis}\ \dfrac{(\mathcal{H}^{\mathbb{O}},\mathbb{O}) \equiv_{R,\mathsf{true}} \mathbb{Q}_0}{(\mathcal{H}^{\mathbb{O}},\mathbb{O}) \sim_0 \mathbb{Q}_0} \quad \dfrac{\dfrac{\mathbb{Q}_0 \leq_{\det,\gamma} \mathbb{Q}_1}{\mathbb{Q}_0 \sim_0 \mathbb{Q}_1}\ \text{I-Det} \quad \dfrac{\mathbb{Q}_1 \equiv_{R',\mathsf{true}} \mathbb{O}_{ant}}{\mathbb{Q}_1 \sim_0 \mathbb{O}_{ant}}\ \text{I-Bis}}{(\mathcal{H}^{\mathbb{O}},\mathbb{O}) \sim_0 \mathbb{O}_{ant}}$$

Middle tree:

$$\text{UR}\ \dfrac{\mathbb{O}_{ant} :_\epsilon E \qquad \mathsf{F}_{\neg\phi} \Rightarrow E}{\mathbb{O}_{ant} :_\epsilon \mathsf{F}_{\neg\phi}} \quad \dfrac{\mathbb{O}_{ant} \equiv^b_{R'',\phi} \mathbb{Q}_2 \qquad \mathbb{Q}_2 \equiv_{=,\phi} \mathbb{Q}_5}{\mathbb{O}_{ant} \sim_\epsilon \mathbb{Q}_5}\ \text{I-2-Bis}$$

where $E = \mathsf{F}_{\mathsf{Collide}\vee\mathsf{Reveal}\vee\mathsf{Link}}$

Figure VI.4 – Trees Of The Proof Of The Generic Theorem

— lists $(m.L_{o_i})_i$ and list $m'.L_{\mathcal{S}}$ contain the same queries and answers, which we formalize as:
(1.) $\forall (q,y) \in m.L_{o_i}$, there exists a label lbl such that $(o_i,q,y,lbl) \in m'.L_{\mathcal{S}}$;
(2.) $\forall (o_i,q,y,lbl) \in m'.L_{\mathcal{S}}$, $(q,y) \in m.L_{o_i}$.

We then have $(\mathcal{H}^{\mathbb{O}},\mathbb{O}) \equiv_{R,\mathsf{true}} \mathbb{Q}_0$.

We now define a second intermediate system, $\mathbb{Q}_1$, which is similar to $\mathbb{Q}_0$ but for the four lines starting with † in the implementation of o_{piv}, which are replaced by:

$\mathsf{let}\ t \leftarrow \mathcal{H}(x)\ \mathsf{in}$
$(o_{\mathsf{piv}},q,y) :: L := \Pi_3(L_{\mathcal{H}}(x))_{j \geq \mathsf{piv}(x)}$;
$L_{\mathcal{S}} := L_{\mathcal{S}}.(L,Vis)$;

To apply a determinization rule, we should separate $L_{\mathcal{S}}$ into two tables $L_{\mathcal{S}}$ and $L^{ant}_{\mathcal{S}}$. However, we bypass this step and just provide the distribution γ induced by a memory on $L^{ant}_{\mathcal{S}}$, table of anticipated queries. System $\mathbb{Q}_0$ determinizes $\mathbb{Q}_1$ for distribution γ for which we provide a constructive definition:

$\gamma(m) =$ $L^{ant}_{\mathcal{S}} := [\,]$;
$\mathsf{for}\ q\ \mathsf{in}\ \mathsf{dom}(Pivot)\ \mathsf{do}$
$\mathsf{let}\ (y,x) \leftarrow Pivot(q)\ \mathsf{in}$
$\mathsf{let}\ t \leftarrow \mathsf{Imp}_{\mathbb{Q}_0}(\mathcal{H})(x)\ \mathsf{in}$
$L := \Pi_3(\mathsf{Last}(L_{\mathcal{H}}(x)))$;
$L_{\mathcal{S}} := L_{\mathcal{S}}.([L]_{j>\mathsf{piv}(x)},Vis)$;
$L^{ant}_{\mathcal{S}} := L^{ant}_{\mathcal{S}}.([L]_{j>\mathsf{piv}(x)},Vis)$;
endfor
$\mathsf{return}\ L^{ant}_{\mathcal{S}} - m.L_{\mathcal{S}}$

Finally, the justification of the step from $\mathbb{Q}_1$ to $\mathbb{O}_{ant}$ is again a perfect bisimulation relation R' induced by equality on lists $L_{\mathcal{H}}$, $L_{\mathcal{S}}$ and graph SG.

$\mathsf{Imp}_{\mathbb{O}_{ant}}(\mathcal{H})(x) =$
if $x \in \mathsf{dom}(L_{\mathcal{H}})$ then
  $(a^f, Q) := L_{\mathcal{H}}(x)$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q)$;
  return a^f
else
  $l := \mathsf{init}(x)$;
  $p := \mathsf{piv}(x)$;
  $(x_1, \ldots, x_l) := \Theta(x)$;
  $(o^1, q^1) := \mathsf{H}_1(x_1)$;
  if $q^1 \in \mathsf{dom}(L_{\mathcal{S}}(o^1))$ then
    $(a^1, lbl) := L_{\mathcal{S}}(o^1, q^1)$;
    $Q := [(o^1, q^1, a^1, lbl)]$;
  else let $a^1 \leftarrow \mathcal{U}(o^1)$ in
    $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^1, q^1, a^1, Inv)$;
    $Q := [(o^1, q^1, a^1, Inv)]$;
  endif
  for $j = 2$ to $p - 1$ do
    $q^j := \mathsf{H}_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$;
    if $q^j \in \mathsf{dom}(L_{\mathcal{S}}(o^j))$ then
      $(a^j, lbl) := L_{\mathcal{S}}(o^j, q^j)$;
      $Q := Q :: (o^j, q^j, a^j, lbl)$;
    else let $a^j \leftarrow \mathcal{U}(o^j)$ in
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, Inv)$;
      $Q := Q :: (o^j, q^j, a^j, Inv)$;
    endif
  endfor
  for $j = p$ to l do
    $q^j := \mathsf{H}_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$;
    if $q^j \in \mathsf{dom}(L_{\mathcal{S}}(o^j))$ then
      $(a^j, lbl) := L_{\mathcal{S}}(o^j, q^j)$;
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, max(PVis, lbl))$;
      $Q := Q :: (o^j, q^j, a^j, max(PVis, lbl))$;
    else let $a^j \leftarrow \mathcal{U}(o^j)$ in
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, PVis)$;
      $Q := Q :: (o^j, q^j, a^j, PVis)$;
    endif
  endfor
  $a^f := \mathsf{H}_{post}(x, a^p, [Q]_{j>p})$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q)$;
  return a^f
endif

If $o_i \neq o_{\mathsf{piv}}$:

$\mathsf{Imp}_{\mathbb{O}_{ant}}(o_i)(q) =$
if $q \in \mathsf{dom}(L_{\mathcal{S}}(o_i))$ then
  $(y, \_) := L_{\mathcal{S}}(o_i, q)$;
else let $y \leftarrow \mathcal{U}(o_i)$ in
endif
$L_{\mathcal{S}} := L_{\mathcal{S}}.(o_i, q, y, Vis)$;
$SG := \mathsf{UpSG}((o_i, q, y), SG)$;
return y

$\mathsf{Imp}_{\mathbb{Q}_0}(o_{\mathsf{piv}})(q) =$
if $q \in \mathsf{dom}(L_{\mathcal{S}}(o_{\mathsf{piv}})|Vis)$ then
  $(y, Vis) := L_{\mathcal{S}}(o_{\mathsf{piv}}, q)$;
elsif $\mathsf{PathFinder}(q, SG) = (\mathsf{true}, x, List)$ then
  † if $q \in \mathsf{dom}(L_{\mathcal{S}}(o_{\mathsf{piv}})|PVis, Inv)$ then
  † $(y, \_) := L_{\mathcal{S}}(o_{\mathsf{piv}}, q)$;
  † else let $y \leftarrow \mathcal{U}(o_{\mathsf{piv}})$ in
  † endif
  $Pivot := Pivot.(q, y, x)$;
elsif $q \in \mathsf{dom}(L_{\mathcal{S}}(o_{\mathsf{piv}})|PVis, Inv)$ then
  $(y, \_) := L_{\mathcal{S}}(o_{\mathsf{piv}}, q)$;
else let $y \leftarrow \mathcal{U}(o_{\mathsf{piv}})$ in
endif
return y
  $L_{\mathcal{S}} := L_{\mathcal{S}}.(o_{\mathsf{piv}}, q, y, Vis)$;
  $SG := \mathsf{UpSG}((o_{\mathsf{piv}}, q, y), SG)$;

Figure VI.5 – Implementations Of $\mathbb{Q}_0$

VI.3.2 — Redrawing Some Invisible Vertices

The idea behind this step is to allow the oracles to redraw new images for values of which the image has already been used, or in other words to resample some vertices. When can such a resampling be a problem for coherence of the simulation? The idea is to preserve the structure of the input characteristic graph during oracle calls. Of course, if the image we consider is visible or partially visible, we do not redraw it. Furthermore, even when the vertex we want to modify is invisible, we have to be careful. The idea is that we have to preserve paths existing in the input graph. To this end, we introduce a new terminology: a vertex v' is one of the *next neighbors* of a vertex v in graph CG iff there exists an edge $(v, \_, v')$ between v and v'. The set of next neighbors of v in graph CG is denoted $\mathsf{Next}(v, CG)$. Every time we change an invisible vertex into another vertex, we want to modify its next neighbors so that the same edges still exist between them. This is doable only if such neighbors are non-visible. Besides, in case one of the next neighbors is a collision vertex, redrawing suppresses the collision and changes the structure of the graph. This is also a case we want to exclude.

Formally, we define a function ReSamp taking as input a query (o, q) and a memory m such that query (o, q) corresponds to a vertex (o, q, y) in m. The function outputs a boolean corresponding to whether we can redraw vertex (o, q, y) in memory m. The characteristic graph associated to m is $\Gamma(m) = (v_{root}, CV, CE, \mathcal{V})$.

$$\mathsf{ReSamp} : \left| \begin{array}{lcl} \mathsf{Que} \times \mathsf{M}_{\mathbb{O}_{ant}} & \to & \mathsf{Bool} \\ & & \\ ((o,q),m) & \mapsto & \begin{cases} \mathsf{true} & \text{if } \mathcal{V}((o,q,y)) = Inv, \\ & \mathsf{Next}((o,q,y),\Gamma(m)) \cap \mathcal{V}^{-1}(Vis) = \emptyset \\ & \mathsf{Next}((o,q,y),\Gamma(m)) \cap \mathsf{CollVertex}(\Gamma(m)) = \emptyset \\ \mathsf{false} & \text{otherwise.} \end{cases} \end{array} \right.$$

In particular, we emphasize that for all values corresponding to partially visible and visible vertices, ReSamp outputs false.

To form up again the paths existing in the input graph, we define a function named $\mathsf{Stitch} : \mathsf{Xch} \times \{Inv, PVis\} \times \mathsf{M}_{\mathbb{O}_{ant}} \to \mathsf{M}_{\mathbb{O}_{ant}}$, which takes as input a (possibly resampled) vertex $(o, q, \tilde{y})$, a visibility label for this latter and a memory m and outputs a new memory m'. If (o, q, y) appears in the memory m for some y, Stitch modifies the memory so that $(o, q, \tilde{y})$ replaces (o, q, y) with the visibility label given in input of Stitch and next neighbors of the vertex (o, q, y) in $\Gamma(m)$ become next neighbors of the new vertex $(o, q, \tilde{y})$ in $\Gamma(m')$ (with the same edges). Thus, paths existing in the input graph exist in the output graph too.

Formally, if (o, q, y) appears in the memory m and $(o, q, \tilde{y})$ is the new vertex, Stitch outputs m' computed as follows. To build $m'.L_{\mathcal{S}}$, we start with $m'.L_{\mathcal{S}} = m.L_{\mathcal{S}}$ and then proceed in the following way. For all edges $((o, q, y), l, (o', q', y'))$ in $\Gamma(m)$ where $(o, q', y') \in \mathsf{Next}((o, q, y), \Gamma(m))$, if j is an index such that $q' = \mathsf{H}_j(l, (o, q, y))$, then we let $\tilde{q} = \mathsf{H}_j(l, (o, q, \tilde{y}))$. Then, if $(o', \tilde{q})$ does not appear in $m'.L_S$ yet, (o', q', y', lbl) is removed from $m'.L_{\mathcal{S}}$ and $(o', \tilde{q}, y', lbl)$ is added to $m'.L_{\mathcal{S}}$. If $(o', \tilde{q})$ already appears in $m'.L_{\mathcal{S}}$, we do not modify it. Finally, we remove $(o, q, y, \_)$ from $m'.L_{\mathcal{S}}$ and replace it by $(o, q, \tilde{y}, lbl)$, where lbl is given in input of Stitch. List $m'.L_{\mathcal{H}}$ is then built out of $m.L_{\mathcal{H}}$ by rebuilding the third component of every triple it contains: given $(x, a^f, Q) \in m.L_{\mathcal{H}}$, (x, a^f, Q') is put in $m'.L_{\mathcal{H}}$, where Q' is the list of calls necessary to compute $\mathcal{H}(x)$ in $m'.L_{\mathcal{S}}$.

Notice that we cannot turn a vertex into a collision vertex when we resample it: the fact that a collision occurs in a vertex depends only on its query part and we only change the

answer. However, there is a possibility when we apply Stitch that we change the structure of the characteristic graph. Namely, we can stumble upon a preexisting vertex by computing a value $\tilde{q}$ which already corresponds to a vertex.

The set of values $\tilde{y}$ such that it happens is:

$$\mathsf{PbSet}((o,q),m) = \{\tilde{y} \in \mathsf{Out}(o) \mid \exists j \ s.t. \ \tilde{q} = \mathsf{H}_j(l,(o,q,\tilde{y})) \in \mathsf{dom}(m.L_{\mathcal{S}}(o^j))\}$$

To write our new system $\mathbb{Q}_2$, we introduce an auxiliary procedure Adjust. It takes as input a query (o,q) and a visibility label lbl, resamples the vertex if it is possible and modifies the lists with Stitch, which adds the query and answer to list $L_{\mathcal{S}}(o)$ with the desired visibility label.

```
Adjust ((o,q), lbl, m) =
       if q ∈ dom(L_S(o)) then
        if ReSamp((o,q), m) then
         let a ← U(o) in
         m' := Stitch((o,q,a), lbl, m);
        else (a, lbl') := L_o(q);
         L_S := L_S.(o,q,a,max(lbl,lbl'));
        endif
       else let a ← U(o) in
         L_S := L_S.(o,q,a,lbl);
       endif
       return (o,q,L_S(o,q))
```

Then, we can define the implementation of oracle $\mathcal{H}$ in the adjusted system as in figure VI.6, while both other oracles remain implemented as in $\mathbb{O}_{ant}$. The claim proven above justifies the existence and unicity of related adjusted states when ϕ holds.

To formalize our proof step, we use a relation of backwards bisimulation. Two states are in relation R'' iff they yield graphs with the same structure. As the number of neighbors of a resampled vertex can potentially be modified by stitching, we impose that it is equal in two states in relation. Formally, we impose the conditions:

$m \ R'' \ m'$ iff there exist $n \geq 0$ and a list $[(o_1,q_1,a_1),\ldots,(o_n,q_n,a_n)]$ of distinct vertices and labels, such that, if we denote $m^0 = m$ and $m^n = m'$:

— For all $i = 1..n$, $m^i = \mathsf{Stitch}((o_i,q_i,a_i), Inv, m^{i-1})$.

— For all $i = 1..n$, $\mathsf{ReSamp}((o_i,q_i),m^{i-1})$ or $q_i \notin m^{i-1}.L_{\mathcal{S}}(o_i)$.

— For all $i = 1..n$, $a_i \notin \mathsf{PbSet}((o_i,q_i),m^{i-1})$.

— For all i, if y_i is the image of q_i by o_i in state m^{i-1}, then $\mathsf{Card}(\mathsf{Next}((o_i,q_i,y_i),\Gamma(m^{i-1}))) = \mathsf{Card}(\mathsf{Next}((o_i,q_i,a_i),\Gamma(m^i)))$, i.e. the stitch operation conserves the number of neighbors of the resampled vertex.

To be able to apply rule $I-2-Bis$, we need a common set of conditions ϕ for backward and forward bisimulation relations. Therefore, we choose for ϕ the conjunction of every condition that we need to require in the next steps determining $\mathbb{Q}_5$. To do so, we express two conditions on the execution of an exchange $m_1 \xrightarrow{xch} m_2$, one is a condition on the characteristic graph from which we start (this is ϕ_1) and one is a condition on what happens during the exchange execution (this is ϕ_2).

The first condition expresses that the input characteristic graph exhibits no collision or

$\mathsf{Imp}_{\mathbb{Q}_2}(\mathcal{H})(x) =$ if $x \in \mathsf{dom}(L_{\mathcal{H}})$ then
  $(a^f, Q) := L_{\mathcal{H}}(x)$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q)$;
  return a^f
else
  $l := \mathsf{init}(x)$;
  $p := \mathsf{piv}(x)$;
  $(x_1, \ldots, x_l) := \Theta(x)$;
  $q^1 := \mathsf{H}_1(x_1)$;
  if $q^1 \in \mathsf{dom}(L_{\mathcal{S}}(o^1))$ then
    $(a^1, lbl) := L_{\mathcal{S}}(o^1, q^1)$;
    $Q := [(o^1, q^1, a^1, lbl)]$;
  else let $a^1 \leftarrow \mathcal{U}(o^1)$ in
    $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^1, q^1, a^1, Inv)$;
    $Q := [(o^1, q^1, a^1, Inv)]$;
  endif
  for $j = 2$ to $p - 1$ do
    $q^j := \mathsf{H}_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$;
    if $q^j \in \mathsf{dom}(L_{\mathcal{S}}(o^j))$ then
      $(a^j, lbl) := L_{\mathcal{S}}(o^j, q^j)$;
      $Q := Q :: (o^j, q^j, a^j, lbl)$;
    else let $a^j \leftarrow \mathcal{U}(o^j)$ in
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, Inv)$;
      $Q := Q :: (o^j, q^j, a^j, Inv)$;
    endif
  endfor
  for $j = p$ to l do
    $q^j := \mathsf{H}_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$;
    let $(o^j, q^j, a^j, lbl) \leftarrow \mathsf{Adjust}((o^j, q^j), PVis)$ in
    $Q := Q : (o^j, q^j, a^j, lbl)$;
  endfor
  $a^f := \mathsf{H}_{post}(x, a^p, [Q]_{j>p})$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q)$;
  return a^f
endif

Figure VI.6 – Implementation of $\mathcal{H}$ in System $\mathbb{Q}_2$

non-resamplable vertex. This is naturally formalized as

$$\phi_1(m) = \begin{cases} \mathsf{CollVertex}(\Gamma(m)) = \emptyset \\ \forall i, \forall (o,q,a) \in (m.L_{\mathcal{S}}(o_i)|Inv), \mathsf{ReSamp}((o,q),\Gamma(m)) = \mathsf{true} \end{cases}$$

The second condition captures that neither Collide nor Reveal happen during the execution of the exchange, using the function mapping execution of exchanges to graph sequences defined in the previous section. We also impose that no query to oracles $o_i \neq o_{\mathsf{piv}}$ is labeled partially visible (this is P_1) and that all hash queries, if they have a matching pivot query that is visible, are not fresh hash queries (this is P_2).

$$\phi_2(xch, m_1, m_2) = \mathsf{G}_{\neg\mathsf{Collide}\wedge\neg\mathsf{Reveal}}(\mathsf{StTr}(m_1 \xrightarrow{xch} m_2)) \wedge P_1(m_1) \wedge P_2(m_1, m_2)$$

where

$$\begin{cases} P_1(m) \quad (xch = (o_i, q, y) \wedge o_i \neq o_{\mathsf{piv}} \wedge q \in \mathsf{dom}(m.L_{\mathcal{S}}(o_i))) \Rightarrow \mathcal{V}(q) \neq PVis \\ P_2(m, \tilde{m}) \quad (xch = (\mathcal{H}, x, a^f) \wedge \\ \qquad \Pi_3(\tilde{m}.L_{\mathcal{H}})[\mathsf{piv}(x)] \in \mathsf{dom}(m.L_{\mathcal{S}}(o_{\mathsf{piv}})|Vis)) \Rightarrow x \in \mathsf{dom}(m.L_{\mathcal{H}}) \end{cases}$$

We now let $\phi(xch, m_1, m_2) = \phi_1(m_1) \wedge \phi_2(xch, m_1, m_2)$. We must show that R'' is a relation of backwards bisimulation up to ϕ for our oracle system. We start by showing the following useful claim.

Claim. Given $m_1 \xrightarrow{xch}_{>0} m_2$, and a state m_2' such that $m_2' \; R'' \, m_2$, if $\phi(xch, m_1, m_2)$, there exists a unique state m_1' such that $m_1 \; R'' \; m_1'$ and $m_1' \xrightarrow{xch}_{>0} m_2'$. Moreover, the same number of vertices are added in the graph $\Gamma(m_2)$ w.r.t. $\Gamma(m_1)$ and in the graph $\Gamma(m_2')$ w.r.t. $\Gamma(m_1')$.

Proof. We know that $m_2 \; R'' \, m_2'$. Hence there exist $n \geq 0$ and a list $[(o_1, q_1, a_1), \ldots, (o_n, q_n, a_n)]$ of distinct vertices such that, if we denote $m^0 = m_2$ and $m^n = m_2'$, we have:

— For all $i = 1..n$, $m^i = \mathsf{Stitch}((o_i, q_i, a_i), Inv, m^{i-1})$.

— For all i, $\mathsf{ReSamp}((o_i, q_i), m^{i-1})$ or $q_i \notin m^{i-1}.L_{\mathcal{S}}(o_i)$.

— For all i, $a_i \notin \mathsf{PbSet}((o_i, q_i), m^{i-1})$.

— For all i, if y_i is the image of q_i by o_i in state m^{i-1}, then $\mathsf{Card}(\mathsf{Next}((o_i, q_i, y_i), \Gamma(m^{i-1}))) = \mathsf{Card}(\mathsf{Next}((o_i, q_i, a_i), \Gamma(m^i)))$.

Let us define the following candidate for m_1':

$$m_1' = \mathsf{Stitch}((o_1, q_1, a_1), Inv, \ldots \mathsf{Stitch}((o_n, q_n, a_n), Inv, m_1) \ldots)$$

The state m_1' defined satisfies $m_1 \; R'' \, m_1'$. Indeed, without loss of generality, we can assume that the (o_i, q_i) are distinct. The stitching application has no effect on a state m if its first argument (o_i, q_i, a_i) is such that (o_i, q_i) does not satisfy $q_i \in \mathsf{dom}(m.L_{\mathcal{S}}(o_i))$.

Let us show now that every time a new vertex is added to $m_1.L_{\mathcal{S}}$ during the execution leading to m_2, it is added in any state in relation with m_1 leading to m_2' too.

Suppose that we reason about an exchange xch with an oracle o_i. First, we argue that related states coincide on visible vertices, so in particular on visible parts of the domain of list $L_{\mathcal{S}}(o_i)$. Moreover, the only invisible queries that can be asked without realizing Reveal are

vertices directly linked to the root. If ϕ_1 holds, none of these vertices can be a collision vertex. Therefore, there is no possibility that their query part be resampled as neighbors of another vertex. Consequently, if an invisible query is asked and ϕ holds, it is in the domain of $L_{\mathcal{S}}(o_i)$ for all related memories. Furthermore, in case we ask a partially visible query, either $o_i \neq o_{\mathsf{piv}}$ and it breaks P_1, or it has to be visibly rooted, otherwise Reveal becomes true. Hence, since it has a visible (previous) neighbor and no other previous neighbor (otherwise it is a collision vertex), it cannot be resampled as a next neighbor of some vertex. As a result, it is in the domain of all related memories.

Suppose now that we reason on an exchange xch with $\mathcal{H}$. The trick is to notice that our equivalence relation is built so that the same paths exist in related states. Consequently, if at step j, we meet the first query resulting in the addition of a new vertex in $\Gamma(m_1)$, then it is also the first query resulting in the addition of a vertex in any related memory, or there would exist a rooted path in one graph and not the other. Furthermore, once we start adding vertices during the execution, we have to draw new vertices until the end, or we contradict ϕ by either creating a collision vertex or realizing Reveal. The conclusion follows. ■

Let us first check stability, i.e. that given $m_1 \xrightarrow{xch} m_2$, and m_2' such that $m_2' \; R'' \; m_2$, all states $\tilde{m_1}$ in relation with m_1 such that $\tilde{m_1} \xrightarrow{xch} m_2'$ are such that $\phi(xch, m_1, m_2)$ iff $\phi(xch, \tilde{m_1}, m_2')$. This follows from the claim: if $\phi(xch, m_1, m_2)$, then there is one possibility of state $\tilde{m_1}$, it is m_1'. Moreover, $\phi_1(m_1')$ holds: no collision vertex or non-resamplable vertex can be created. This allows us to say that $\phi_1(m_2)$ holds iff $\phi_1(m_2')$ holds. Therefore, if Reveal happens or a collision vertex is created, then it is in both cases. This justifies stability of $\mathsf{G}_{\neg\mathsf{Collide}\wedge\neg\mathsf{Reveal}}$. Concerning P_1, it only deals with input states. Visible vertices are equal in related states, so we only need to justify that there cannot exist a vertex which is partially visible in one state and invisible in the other. In fact, P_1 is not a stable property, but $\neg\mathsf{Reveal} \wedge \phi_1 \wedge P_1$ is. If $\neg\mathsf{Reveal} \wedge \phi_1$ holds for an exchange, then the only invisible queries that an adversary can perform are directly linked to the root, otherwise Reveal happens, and linked only to the root, since ϕ_1 holds. Since we do not resample the root, the set of invisible queries not breaking $\neg\mathsf{Reveal} \wedge \phi_1$ coincide in related states. Therefore, if visible and invisible queriable vertices coincide, P_1 holds for all or none of the states in relation. Finally, stability of P_2 follows from the visibility property imposed on the pivot: it has the same value in m_1 and m_1', so does $L_{\mathcal{H}}$. Stability follows.

We have to verify compatibility. We consider states m_1, m_2 and m_2' and an exchange $xch = (o, q, a)$ such that $m_1 \xrightarrow{xch}_{>0} m_2$ and $\phi(xch, m_1, m_2)$. The claim proves that there is only one state m_1' such that $m_1' \xrightarrow{xch}_{>0} m_2'$ and that executions starting in states m_1 and m_1' lead to the same number of draws. It yields the equality between probabilities:

$$\Pr[\mathbb{A} \mid \mathbb{O}_{ant} : m_1 \xrightarrow{xch}_{>0} m_2] = \Pr[\mathbb{Q}_3 : m_1' \xrightarrow{xch}_{>0} m_2']$$

Then, we deduce from the one-to-one mapping between m_1 and m_1' that it yields:

$$\Pr[\mathbb{A} \mid \mathbb{O}_{ant} : \mathcal{C}(m_1) \xrightarrow{xch}_{>0} m_2] = \Pr[\mathbb{Q}_2 : \mathcal{C}(m_1') \xrightarrow{xch}_{>0} m_2']$$

VI.3.3 — Replacing Adjust by Simple Sampling

We keep the same overall implementations but change the implementation of Adjust into:

Adjust' $((o,q),lbl) =$
$\mathsf{let}\ a \leftarrow \mathcal{U}(o)\ \mathsf{in}$
$\mathsf{if}\ q \in \mathsf{dom}(L_{\mathcal{S}}(o))\ \mathsf{then}$
$m' := \mathsf{Stitch}((o,q,a),lbl,m);$
$\mathsf{else}\ L_{\mathcal{S}} := L_{\mathcal{S}}.(o,q,a,lbl);$
endif
$\mathsf{return}\ (o,q,a)$

In other words, we redraw a value for q, no matter whether it is resamplable. This yields a system we name $\mathbb{Q}_3$.

This step is formalized using a bisimulation up to ϕ, with as a relation the equality of states. ϕ is obviously stable for this relation. Now let us check compatibility. Given that only the implementation of $\mathcal{H}$ possibly resamples vertices, the simulation is imperfect during an execution of $\mathcal{H}(x)$ (not necessarily called directly). It can happen if we resample a non-resamplable vertex.

Let v be the first vertex posing a simulation problem during an execution of $\mathcal{H}$.

— If v has been resampled whereas it was partially visible, it means v belongs to the pivot and post-pivot queries of another hash input x'. Necessarily the paths of x and x' meet in some vertex v' (not necessarily distinct of v), which is a collision vertex. The execution of $\mathcal{H}(x)$ realizes $\mathsf{Collide}$ at the moment of the query for v'.

— If v has been resampled whereas it was visible and v is not the pivot then Reveal happens: the visibility label of the pivot is partially visible, so that sequence of labels has to increase.

— If v has been resampled whereas it was a visible pivot query matching x, then P_2 is broken.

We conclude that $\mathbb{Q}_2 \equiv_{=,\phi} \mathbb{Q}_3$.

VI.3.4 — Changing Oracles in $\mathsf{N}_{\mathbb{O}}$

In this step, we modify the implementation of the oracles in $\mathsf{N}_{\mathbb{O}}$ assuming that pivot queries are on the one hand always detected when queried directly, and on the other hand always asked before any of their matching post-pivot queries. It gives us a new system $\mathbb{Q}_4$, for which the implementations are provided in figure VI.7.

If the first assumption holds, we can safely simplify the end of the implementation of o_{piv} by replacing the test of belonging to $(L_{\mathcal{S}}(o_{\mathsf{piv}})|PVis,Inv)$ by that of belonging to $(L_{\mathcal{S}}(o_{\mathsf{piv}})|Inv)$. If the second assumption holds, no partially visible query should be directly asked to an oracle $o_i \neq o_{\mathsf{piv}}$. Indeed, the pivot query being queried on before implies that all post-pivot queries become visible vertices. We thus modify the implementation of $o_i \neq o_{\mathsf{piv}}$ by just checking if a query already belongs to $\mathsf{dom}(L_{\mathcal{S}}(o_i)|Inv,Vis)$ before drawing an answer.

The formal justification of this step is that $\mathbb{Q}_3 \equiv_{=,\phi} \mathbb{Q}_4$. Indeed, the simulation is perfect except when:

— o_{piv} is queried on a partially visible vertex, but does not branch in the path-finder branch, meaning the vertex is non-visibly meaningfully rooted. Yet, it is meaningfully rooted since it is partially visible. This is captured by Reveal.

— During an execution of o_i, if we redraw a new answer to a partially visible query, but then P_1 is broken.

If $o_i \neq o_{\mathsf{piv}}$:
$\mathsf{Imp}_{\mathbb{Q}_4}(o_i)(q) =$
if $q \in \mathsf{dom}(L_{\mathcal{S}}(o_i)|Inv, Vis)$ then
$(y, \_) := L_{\mathcal{S}}(o_i, q)$;
else let $y \leftarrow \mathcal{U}(o_i)$ in
endif
$L_{\mathcal{S}} := L_{\mathcal{S}}.(o_i, q, y, Vis)$;
$SG := \mathsf{UpSG}((o_i, q, y), SG)$;
return y

$\mathsf{Imp}_{\mathbb{Q}_4}(o_{\mathsf{piv}})^{\mathcal{H}}(q) =$
if $q \in \mathsf{dom}(L_{\mathcal{S}}(o_{\mathsf{piv}})|Vis)$ then
$(y, Vis) := L_{\mathcal{S}}(o_{\mathsf{piv}}, q)$;
elsif $\mathsf{PathFinder}(q, SG) = (\mathsf{true}, x, List)$ then
let $t \leftarrow \mathcal{H}(x)$ in
$(o_{\mathsf{piv}}, q, y) :: L := \Pi_3(L_{\mathcal{H}}(x))_{j \geq \mathsf{piv}(x)}$;
$L_{\mathcal{S}} := L_{\mathcal{S}}.((o_{\mathsf{piv}}, q, y, Vis) :: (L, Vis))$;
elsif $q \in \mathsf{dom}(L_{\mathcal{S}}(o_{\mathsf{piv}})|Inv)$ then
$(y, Inv) := L_{\mathcal{S}}(o_{\mathsf{piv}}, q)$;
else let $y \leftarrow \mathcal{U}(o_{\mathsf{piv}})$ in
endif
$L_{\mathcal{S}} := L_{\mathcal{S}}.(o_{\mathsf{piv}}, q, y, Vis)$;
$SG := \mathsf{UpSG}((o_{\mathsf{piv}}, q, y), SG)$;
return y

Figure VI.7 – Implementations of Oracles in $\mathbb{Q}_4$

VI.3.5 — Changing $\mathcal{H}$

In this last step, we define a system $\mathbb{Q}_5$ (see in figure VI.8) and replace the series of uniform sampling of the pivot and post-pivot vertices, followed by the computation of a^f, by the sampling of a^f and the execution of the forward sampler algorithm. According to the hypotheses we have formulated on this latter, both implementations yield equal distributions on the lists $(L_{\mathcal{H}}, L_{\mathcal{S}})$ as soon as the forward sampler does not sample elements for which there are yet vertices in the characteristic graph. This last condition is captured by the fact that neither Collide nor Reveal happen. It follows that $\mathbb{Q}_4 \equiv_{=,\phi} \mathbb{Q}_5$

VI.3.6 — Conclusion of the Tree in the Middle

We start by providing details about the application of rule $I-2-Bis$. In the last three transformations, we have created systems $\mathbb{Q}_2$ to $\mathbb{Q}_5$, and such that $\mathbb{Q}_2 \equiv_{=,\phi} \mathbb{Q}_3$, $\mathbb{Q}_3 \equiv_{=,\phi} \mathbb{Q}_4$ and $\mathbb{Q}_4 \equiv_{=,\phi} \mathbb{Q}_5$. From these statements, we can deduce that $\mathbb{Q}_2 \equiv_{=,\phi} \mathbb{Q}_5$. As $\mathbb{Q}_2$ is expressed as an adjusted system of $\mathbb{O}_{ant}$, we can apply rule $I-2-Bis$.

Furthermore, we want to justify that $\mathsf{F}_{\neg\phi}$ yields that eventually, Collide, Reveal or Link happens, i.e. $\mathsf{F}_{\neg\phi} \Rightarrow \mathsf{F}_{\mathsf{Collide}\vee\mathsf{Reveal}\vee\mathsf{Link}}$. To do so, we prove that $\neg P_1$ and $\neg P_2$ imply that Reveal or Link have happened. Concerning P_1, if when querying $o_i \neq o_{\mathsf{piv}}$ on q, $q \in \mathsf{dom}(m_1.L_{\mathcal{S}}(o_i))$ is part of a partially visible vertex, then this latter is meaningfully rooted. If it is not visibly meaningfully rooted, then we can conclude that Reveal has happened. Otherwise, if all queries on the path from the root to our queried vertex are visible, since it is a post-pivot query, but still tagged with a partially visible label, it means that the matching pivot was not visibly meaningfully rooted at the time of its query. Consequently, we are sure that at some point, a query was issued to one of the o_i's to link two chains of visible vertices, i.e. Link has happened.

Finally, for property P_2, if a fresh query on x is issued to $\mathcal{H}$ with a pivot already visible, it means that the pivot has been directly queried for, but that at the time of query, it was not visibly rooted (otherwise $\mathcal{H}(x)$ would have been called). Similarly to the previous event, we can show that either all queries before the pivot are visible, and at some point Link has happened, or there exists an invisible query on the path from the root to the pivot, and Reveal holds.

This concludes the discussion about the middle tree.

VI.3.7 — Determinization of $\mathbb{Q}_5$ to Obtain the Simulated System

As we did previously, we abuse a little the determinization rule and only provide the distribution yielded by a memory on anticipated queries in $L_{\mathcal{S}}$, which we name $L_{\mathcal{S}}^{ant}$. To build possible anticipated components of state out of a state $m = (L_{\mathcal{S}}, L_{\mathcal{H}})$ of the simulated system, we have to generate the list of queries matching every pair (x, a^f) in $L_{\mathcal{H}}$ and to tag them with visibility labels. Given the first pair (x, a^f), this can be done by executing the implementation $\mathsf{Imp}_{\mathbb{Q}_5}(\mathcal{H})$ given as input x and the list $L_{\mathcal{S}}$ where every vertex has been deemed visible. It provides us with a new table $L_{\mathcal{S}}$, on which to iterate what we have just done with the following pairs in list $L_{\mathcal{H}}$. This provides us with a constructive definition for a distribution γ' such that $(\mathcal{U}(\mathcal{H}), \mathbb{S}^{\mathcal{U}(\mathcal{H})}) \leq_{\mathrm{det},\gamma'} \mathbb{Q}_5$:

$\mathsf{Imp}_{\mathbb{Q}_5}(\mathcal{H})(x) =$ if $x \in \mathsf{dom}(L_{\mathcal{H}})$ then
  $(a^f, Q) := L_{\mathcal{H}}(x)$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q)$;
  return a^f
else
  $l := \mathsf{init}(x)$;
  $p := \mathsf{piv}(x)$;
  $(x_1, \ldots, x_l) := \Theta(x)$;
  $(o^1, q^1) := \mathsf{H}_1(x_1)$;
  if $q^1 \in \mathsf{dom}(L_{\mathcal{S}}(o^1))$ then
    $(a^1, lbl) := L_{\mathcal{S}}(o^1, q^1)$;
    $Q := [(o^1, q^1, a^1, lbl)]$;
  else let $a^1 \leftarrow \mathcal{U}(o^1)$ in
    $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^1, q^1, a^1, Inv)$;
    $Q := [(o^1, q^1, a^1, Inv)]$;
  endif
  for $j = 2$ to $p - 1$ do
    $q^j := \mathsf{H}_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$;
    if $q^j \in \mathsf{dom}(L_{\mathcal{S}}(o^j))$ then
      $(a^j, lbl) := L_{\mathcal{S}}(o^j, q^j)$;
      $Q := Q :: (o^j, q^j, a^j, lbl)$;
    else let $a^j \leftarrow \mathcal{U}(o^j)$ in
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, Inv)$;
      $Q := Q :: (o^j, q^j, a^j, Inv)$;
    endif
  endfor
  $q^p := \mathsf{H}_p(x_p, (o^{p-1}, q^{p-1}, a^{p-1}))$;
  let $a^f \leftarrow \mathcal{U}_{\mathcal{H}}$ in
  let $(a^p, Q') \leftarrow \mathsf{FwdSplr}(x, a^f)$ in
  $L_{\mathcal{S}} := L_{\mathcal{S}}.((o_{\mathsf{piv}}, q^p, a^p, PVis) :: (Q', PVis))$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q :: (o_{\mathsf{piv}}, q^p, a^p) :: Q')$;
  return a^f
endif

Figure VI.8 – Implementation Of $\mathcal{H}$ In System $\mathbb{Q}_5$

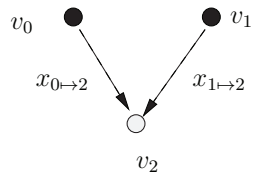


Figure VI.9 – Vertices Involved in $\mathsf{F}_{\mathsf{Collide}}$

$$
\begin{array}{ll}
\gamma'(m.L_{\mathcal{S}}, m.L_{\mathcal{H}}) = & m'.L_{\mathcal{S}} := (m.L_{\mathcal{S}}, Vis); \\
& L_{\mathcal{S}}^{ant}, m'.L_{\mathcal{H}} := [\,]; \\
& \mathsf{for}\ x\ \mathsf{in}\ m.L_{\mathcal{H}}\ \mathsf{do} \\
& \ \mathsf{let}\ a^f \leftarrow \mathsf{Imp}_{\mathbb{Q}_5}(\mathcal{H})(x, m')\ \mathsf{in} \\
& \ L_{\mathcal{S}}^{ant} := L_{\mathcal{S}}^{ant}.\Pi_3(\mathsf{Last}(m'.L_{\mathcal{H}})); \\
& \mathsf{endfor} \\
& \mathsf{return}\ L_{\mathcal{S}}^{ant} - m.L_{\mathcal{S}}
\end{array}
$$

VI.4 Examples of Application

We first want to underline that the generic theorem proven in the previous section reminds hash construction designers of the importance of preventing length-extension attacks. A length-extension attack is an attack in which the adversary is able to produce two hash inputs x_1 and x_2, x_1 being a strict prefix of x_2 such that the value for $\mathcal{H}(x_2)$ can be computed out of the result of a query of $\mathcal{H}(x_1)$, that is, without asking any intermediate query necessary to compute $\mathcal{H}(x_1)$. In other words, this adversarial capacity is a threat to indistinguishability of real and simulated worlds because the adversary can then predict things about one or several inner-primitive queries: those needed to compute $\mathcal{H}(x_2)$ out of $\mathcal{H}(x_1)$. In particular, length-extension attacks are a well-known weakness of the Merkle-Dåmgard extension technique. In our result, the indifferentiability bound we obtain for this construction is 1. Indeed, length extension allows to realize Reveal with probability 1.

We show next the results provided by our theorem on two constructions: the Sponge construction, introduced in section VI.1, and the ChopMD construction. We do not provide implementation for a path-finder algorithm (though to obtain an instantiated bound on the execution time we should), we only specify forward sampler algorithms.

As the events $\mathsf{F}_{\mathsf{Collide}}$, $\mathsf{F}_{\mathsf{Reveal}}$ and $\mathsf{F}_{\mathsf{Link}}$ can intersect, we take care to evaluate slightly weaker events which partly avoid that some overlapping artificially increasing the bound. The following decomposition which proves useful in both examples we develop[5]. In our examples, the hash constructions are such that there is only one possible label for an edge between two vertices. Consequently, when event $\mathsf{F}_{\mathsf{Collide}}$ happens and results in the creation of a collision vertex v_2, then it necessarily involves vertices v_0, v_1 linked to v_2 such that $v_0 \neq v_1$. Without loss of generality, we suppose that v_0 is created before v_1. Our reasoning is illustrated in figure VI.9; we reason on conditions neccessarily filled by v_0 and v_1 for v_2 to exist. We denote $v_1 = (\mathcal{F}, q_1, a_1)$. We let $\mathsf{RootCollide}$ be the predicate capturing the event

[5]Without any specific argument about the relevance of this decomposition for other constructions, we have chosen to cite the theorem with events 'naturally' appearing in the proof, leaving the choice of a decomposition up to the users.

$\mathsf{FwdSplr}(x,t) =$
$(t_0,\ldots,t_{k-1}) := (t[1..r],\ldots,t[(k-1)r+1,r]);$
$t' \leftarrow \mathcal{U}(c)$
$y^0 := t_0||t';$
$q^1 := y^0;$
for $j = 1$ to $k-1$ do
$t'_j \leftarrow \mathcal{U}(c)$
$y^j := t_j||t'_j;$
$v^j := (\mathcal{F}, q^j, y^j);$
$q^{j+1} := y^j;$
endfor
return $(y^0, [v^j]_{j=(\mathsf{piv}(x)+1)..(\mathsf{piv}(x)+k-1)})$

Figure VI.10 – Forward Sampler Algorithm for the Sponge Construction

that $v_0 = v_{root}$ and v_1 is created such that the collision happens. Then necessarily, there exist $j, x_{0\mapsto 2}, x_{1\mapsto 2}$ such that $\mathsf{H}_1(x_{0\mapsto 2}) = \mathsf{H}_j(x_{1\mapsto 2}, (\mathcal{F}, q_1, a_1))$ since they both equal the query part of v_2. Furthermore, we let $\mathsf{WkCollide}$ be the predicate verified when $v_0 \neq v_{root}$, denoted by $v_0 = (\mathcal{F}, q_0, a_0)$, and vertex v_1 is created: namely, if there exist $x_{1\mapsto 2}$ and $x_{0\mapsto 2}$ such that $\mathsf{H}_j(x_{1\mapsto 2}, (\mathcal{F}, q_1, a_1)) = \mathsf{H}_{j'}(x_{0\mapsto 2}, (\mathcal{F}, q_0, a_0))$, since they both equal the query part of v_2. We reason *on the appearance* of v_1 in the graph, which can indifferently occur before or after creation of v_2 and edge $(v_1, \_v_2)$. It allows us to state that $\mathsf{F}_{\mathsf{Collide}}$ implies $\mathsf{F}_{\mathsf{RootCollide}\vee\mathsf{WkCollide}}$. We define WkLink as the event in which a vertex v_1 is created and gets linked to a preexisting vertex v_2, without imposing any visibility constraint on the vertices v_1 and v_2. We can see that $\mathsf{F}_{\mathsf{Collide}\vee\mathsf{Reveal}\vee\mathsf{Link}}$ is implied by $\mathsf{F}_{\mathsf{WkLink}} \vee \mathsf{F}_{\mathsf{RootCollide}\vee\mathsf{WkCollide}} \vee \mathsf{F}_{\mathsf{Reveal}\wedge\neg\mathsf{WkLink}\wedge\neg\mathsf{Collide}}$.

VI.4.1 — The Sponge Construction

As a forward sampler $\mathsf{FwdSplr}$, we choose the algorithm described in figure VI.10. Intuitively, it parses a hash output t into k blocks of r bits and draws iteratively the c missing bits of the answers to pivot and post-pivot queries.

We start by the computation of an upper-bound of event $\mathsf{F}_{\mathsf{WkCollide}}$. Let us assume it happens at the ℓ-th fresh query, when vertex v_1 is created. There exist $x_{1\mapsto 2}$ and $x_{0\mapsto 2}$ such that $\mathsf{H}_j(x_{1\mapsto 2}, (\mathcal{F}, q_1, a_1)) = \mathsf{H}_{j'}(x_{0\mapsto 2}, (\mathcal{F}, q_0, a_0))$, which imposes $\mathsf{Last}_c(a_0) = \mathsf{Last}_c(a_1)$. Since the answer a_1 is drawn uniformly at random, the probability that it satisfies $\mathsf{Last}_c(a_0) = \mathsf{Last}_c(a_1)$ is bounded by $\frac{\ell-1}{2^c}$. Summing on ℓ, it results in a bound of $\frac{k_{tot}(k_{tot}-1)}{2^{c+1}}$.

We now turn to bound the probability of $\mathsf{F}_{\mathsf{RootCollide}}$. If at some point of the execution, there exists v_1 such that $\mathsf{H}_1(x_{0\mapsto 2}) = \mathsf{H}_j(x_{1\mapsto 2}, (\mathcal{F}, q_1, a_1))$ for some labels $x_{0\mapsto 2}$ and $x_{1\mapsto 2}$ and index j, then it translates in imposing the last bits of a_1 to be worth 0^c. The probability that this happens is bounded by $\frac{k_{tot}}{2^c}$.

Let now bound the probability that $\mathsf{F}_{\mathsf{WkLink}}$. If it happens at the ℓ-th direct query, when a vertex v_1 is created, then there exists a vertex v_2 to which v_1 gets linked by an edge labeled by x_j. Since the answer a_1 is drawn uniformly at random, the probability that there is a v_2 such that $\mathsf{H}_j(x_j, (\mathcal{F}, q_1, a_1)) = q_2$ is bounded by $\frac{\ell-1}{2^c}$. Summing on ℓ, it results in a bound of $\frac{k_{tot}(k_{tot}-1)}{2^{c+1}}$.

Finally, a bound to $\mathsf{F}_{\mathsf{Reveal}\wedge\neg\mathsf{WkLink}\wedge\neg\mathsf{Collide}}$ remains to be computed. When the event

occurs, there exists a vertex $v_1 = (\mathcal{F}, q_1, a_1)$, non-visible, which gets linked to $v_2 = (\mathcal{F}, q_2, a_2)$, visible, by an edge labeled by x_j. Again, we reason in terms of which vertex is the first to appear in the graph. Since we assume $\neg\mathsf{WkLink}$, necessarily, v_1 is created before v_2. Realizing Reveal means that $\mathsf{H}_j(x_j, (\mathcal{F}, q_1, a_1)) = q_2$. Since we assume $\neg\mathsf{Collide}$, there is only one vertex v_1 which can satisfy this equation. Since v_1 is at most partially visible, we know that the probability to issue a satisfactory query q_2 is bounded by $\frac{1}{2^c}$. We then have to sum over the total number of direct queries issued, which results in a bound worth $\frac{k(\mathcal{F})}{2^c}$.

We can now collect the bounds and cite the following theorem.

Theorem VI.2. *We consider the Sponge construction and simulator* $\mathbb{S}$ *implemented as in section VI.2. For an adversary* $\mathbb{A} \in \mathsf{Adv}(k, t)$*,*

$$\mathsf{Indiff}(Sponge, \mathcal{F}, \mathbb{S}) \leq \frac{k_{tot}^2}{2^c} + \frac{k(\mathcal{F})}{2^c}$$

where $k_{tot} = k(\mathcal{F}) + \mathfrak{L}_{sp} * k(\mathcal{H})$.

In [BDPA08], Bertoni et. al. present a clever proof of the indifferentiability the sponge construction concluding to a bound of $\frac{k_{tot}(k_{tot}+1)}{2^{c+1}}$. We obtain a greater bound, containing terms which are omitted in their final bound computation, as was first suggested in [BCCM$^+$08]. The missing term corresponds to the probability that length-extension attacks can be carried out, which, even though the authors propose a simulator different from ours, should not be overlooked in their computation. Nevertheless, it does not alter the merits of their proof which mainly lie in the graph construction and simulator they propose.

VI.4.2 — The ChopMD Construction

We consider the hash function ChopMD introduced in [CDMP05] and inspired of [DGH$^+$04]. It is obtained from the Merkle-Damgård construction by chopping off the last s bits of the output in order to prevent extension attacks. This construction can be described as an oracle system that contains two oracles: $ChopMD_s$ and $\mathcal{F}$. The memory of $ChopMD_s$ consists of a mapping L_{chop} and that of $\mathcal{F}$ of a mapping $L_{\mathcal{F}}$. Their implementations are described in figure VI.11.

The oracle system chopMD can be seen as the application of an overlayer to $\mathcal{F}$. The statically known list of oracles is the list of length $\lceil \frac{2^{64}}{r} \rceil$ whose elements are the oracle $\mathcal{F}$, $\mathsf{init}(x) = \lceil |x|/r \rceil$, $\Theta(x)$ is the function padding x into w and then cutting it into r-blocks, $\mathsf{H}_j(x_j, (\mathcal{F}, q^{j-1}, a^{j-1})) = a^{j-1}||x_j$, and finally $\mathsf{H}_{post}(x, (\mathcal{F}, q, a)) = \mathsf{First}_{n-s}(a)$. Lastly, $\mathsf{piv}(x) = \mathsf{init}(x)$, since the only exchange used by H_{post} is the last one performed during an execution of $ChopMD_s$.

The forward sampler algorithm $\mathsf{FwdSplr}$, on input (x, t), samples uniformly the s missing bits to compute the result of the pivot query $y^{\mathsf{piv}(x)}$, and outputs the concatenation of t with these bits as a value for $y^{\mathsf{piv}(x)}$.

We start by bounding the probability that $\mathsf{WkCollide}$ happens, at the ℓ-th fresh query, when vertex v_1 is created. The equation between v_0 and v_1 imposes $a_0 = a_1$. Since the answer a_1 is drawn uniformly at random, the probability that it satisfies $a_1 = a_0$ is bounded by $\frac{\ell-1}{2^n}$. Summing on ℓ, it results in a bound of $\frac{k_{tot}(k_{tot}-1)}{2^{n+1}}$.

Moreover, to bound the probability that $\mathsf{F}_{\mathsf{RootCollide}}$ holds at some point of the execution, at some point v_1 is created such that there exist $j, x_{0\mapsto 2}, x_{1\mapsto 2}$ such that $\mathsf{H}_1(x_{0\mapsto 2}) =$

Oracle $ChopMD_s$
$\mathsf{In}(ChopMD_s) = \{0,1\}^{\leq 2^{64}}$, $\mathsf{Out}(ChopMD_s) = \{0,1\}^{n-s}$
$\mathsf{Imp}(ChopMD_s)^{\mathcal{F}}(x) =$
if $x \in \mathsf{dom}(L_{chop})$ then
 return $L_{chop}(x)$
else
 $l := \lceil |x|/r \rceil$;
 $w := x||10^{l*r-|x|-1}$;
 $(w_1, \ldots, w_l) := (w[1,r], \ldots, w[r*(l-1)+1, r*l])$;
 $a^0 := 0^n$;
 for $j = 1$ to l do
  $q^j := a^{j-1}||w_j$;
  let $a^j \leftarrow \mathcal{F}(q^j)$ in
 endfor
 $a^f := \mathsf{First}_{n-s}(a^l)$;
 $L_{chop} := L_{chop}.(x, a^f)$;
 return a^f
endif

Oracle $\mathcal{F}$
$\mathsf{In}(\mathcal{F}) = \{0,1\}^{n+r}$, $\mathsf{Out}(\mathcal{F}) = \{0,1\}^n$
$\mathcal{F}(q) =$
if $q \in \mathsf{dom}(L_{\mathcal{F}})$ then
 return $L_{\mathcal{F}}(q)$
else
 let $a \leftarrow \mathcal{U}(\{0,1\}^n)$ in
 $L_{\mathcal{F}} := L_{\mathcal{F}}.(q, a)$;
 return a
endif

Figure VI.11 – ChopMD Implementation

$\mathsf{H}_j(x_{1\mapsto 2}, (\mathcal{F}, q_1, a_1))$. We notice that necessarily, $j > 1$, otherwise $v_1 = v_2$. It translates in imposing a_1 to be worth 0^n. The probability that this happens is bounded by $\frac{k_{tot}}{2^n}$.

Then we upper-bound of the probability that event $\mathsf{F}_{\mathsf{WkLink}}$ occurs. If it happens at the ℓ-th fresh computation of an image by $\mathcal{F}$, when a vertex v_1 is created, then there exists a vertex v_2 to which v_1 gets linked by an edge labeled bu x_j. Since the answer a_1 is drawn uniformly at random, the probability that there is a v_2 such that $\mathsf{H}_j(x_j, (\mathcal{F}, q_1, a_1)) = q_2$ is bounded by $\frac{\ell-1}{2^n}$. Summing on ℓ, it results in a bound of $\frac{k_{tot}(k_{tot}-1)}{2^{n+1}}$.

Finally, we have to bound $\mathsf{F}_{\mathsf{Reveal}\wedge\neg\mathsf{WkLink}\wedge\neg\mathsf{Collide}}$. When the event occurs, there exists a vertex $v_1 = (\mathcal{F}, q_1, a_1)$, non-visible, which gets linked to $v_2 = (\mathcal{F}, q_2, a_2)$, visible, by an edge labeled by x_j. Again, we reason in terms of which vertex is the first to appear in the graph. Since we assume $\neg\mathsf{WkLink}$, necessarily, v_1 is created before v_2. The fact that Reveal happens means that $\mathsf{H}_j(x_j, (\mathcal{F}, q_1, a_1)) = q_2$. Since we assume $\neg\mathsf{Collide}$, there is only one vertex v_1 which can satisfy this equation. Since v_1 is at most partially visible, we know that the probability to issue a satisfactory query q_2 is bounded by $\frac{1}{2^s}$. We then have to sum over

the total number of direct queries issued, which results in a bound worth $\frac{k(\mathcal{F})}{2^s}$.

These three bounds result in a global bound of $\frac{k_{tot}^2}{2^n} + \frac{k(\mathcal{F})}{2^s}$.

Theorem VI.3. *We consider the $ChopMD_s$ construction and simulator $\mathbb{S}$ implemented as in section VI.2.2. For an adversary $\mathbb{A} \in \mathsf{Adv}(k,t)$,*

$$\mathsf{Indiff}(ChopMD_s, \mathcal{F}, \mathbb{S}) \leq \frac{k_{tot}^2}{2^n} + \frac{k(\mathcal{F})}{2^s}$$

where $k_{tot} = k(\mathcal{F}) + \mathfrak{L} * k(\mathcal{H})$.

Results concerning indifferentiability of the Chop construction already appear in various works. In [CDMP05], Coron et. al. determine a bound for this construction considering a random permutation in place of $\mathcal{F}$. We only have a result for random functions, but we however notice that their proof results in a bound of $\mathcal{O}(\frac{(\mathfrak{L}*k_{tot})^2}{2^s})$, which is the same magnitude.

Later, Maurer and Tessaro show in [MT07] that using a prefix-free padding function allows to conclude to a bound of $\mathcal{O}(\frac{(\mathfrak{L}*k_{tot})^2}{2^n})$. This result is particularly interesting, since it beats the usual birthday bound: indeed, $n-s$ bits are output by the hash function, and n is the output-length of the inner primitive. We notice assuming prefix-free padding, we obtain the same bound. Since no *meaningful* path can be obtained as an extension of a meaningful path, Reveal can only happen when $y' = \mathsf{First}_n(q)$ belongs to an *invisible* vertex. As a consequence, the adversary has to guess all n bits of y' and its probability of success is bounded by $\frac{1}{2^n}$. Our second term is turned into $\frac{k(\mathcal{F})}{2^n}$.

Eventually, in [CN08], Chang and Nandi provide a very refined computation for $ChopMD_s$ without assuming prefix-free padding. It leads to a bound of

$$\frac{k_{tot}^2}{2^{n+1}} + \frac{(3(n-s)+1) * k(\mathcal{F}) + (n-s) * k(\mathcal{H})}{2^s} + \frac{k(\mathcal{F}) + k(\mathcal{H})}{2^{n-s-1}}$$

or $\mathcal{O}(\frac{3(n-s)(k(\mathcal{F})+k(\mathcal{H}))}{2^s})$. This improves the result given by [CDMP05], quadratic in the number of queries.

Chapter VII

Conclusion

First, we have introduced the logic CIL, a general proof system enabling concrete proofs directly in the computational model. The framework is designed independently of any particular cryptographic hypothesis. Indeed, while it allows to formalize and use hypotheses such as one-wayness of a function or working in the Random Oracle Model, no built-in assumption comes as a limiting feature of our proof system. Along with the independence of any particular formalism or programming language to describe oracle systems and adversaries, this versatility is one of the strong advantages of CIL.

We have also presented a framework dedicated to the asymmetric setting. Different from the approach developed in CIL, this system is based on the derivation of invariants via a compositional Hoare logic. To design such a Hoare logic, we consider a programming language enriched with primitives that enable the description of asymmetric encryption schemes. The presented Hoare Logic can be used to automate proofs in CIL via proof rules.

Using CIL, we proved a reduction theorem for iterative hash constructions that we use to analyze many of the constructions presented in the SHA-3 competition. For example, we point out and correct a mistake in the indifferentiability proof of Sponge construction associated to the SHA-3 finalist Keccak. This reduction theorem provides a bound for the indifferentiability of the considered hash construction in terms of the bounds of pre-defined simple events.

Perspectives. There are many ways in which we shall pursue the development of our reasoning framework. Let us start the discussion with the critical need for automation of the verification of the proofs and for the design of a suitable user interface. Halevi makes a point in [Hal05] writing that "Just like any other software product, the usefulness of a tool will depend crucially on the willingness of the customers to use it". We are aware that the framework in its current status is yet another pencil-and-paper logic framework and that it is not to be adopted by a significant number of cryptographers as is. However, in the context of the ANR projects SCALP and PROSE, a formalization in Coq of the semantics and set of rules of CIL is near completion, along with a specific module for probabilities. In addition to that, we propose to develop a security-dedicated proof assistant on top of this formalization, maybe with cross-overs with Certicrypt. We strongly believe that the intermediate level of

abstraction of our framework is the right one to develop such a tool. Indeed, looking at a proof tree in CIL already allows to separate the "mundane" parts of the proofs from the fundamental cryptographic arguments.

To obtain a relevant level of automatic verification, we think that a logic dedicated to the derivation of proofs of bisimulation relations would bring a significant amount of confidence in these judgments. Indeed, forward and backward bisimulation are in the present work external judgments to the CIL framework. Therefore, they have to be established outside the system, thus constituting non-verified inputs. Besides, we want to exploit further the possible connections with our Hoare logic (and even other variants of the same kind of approaches) to further automate the derivation of CIL statements. Indeed, we think that the plug-in rules proposed in this work are only capturing the most basic possibility of interaction between the two approaches. Namely, present rules exclusively allow for the use of internal variables in oracle implementations, thus forbidding dependencies of implementations of one another. We want to investigate further the possibility of invariant preservation by oracle implementations exhibiting more complex use of variables.

Last constribution but not the least, our first example of a proof strategy, the reduction theorem for hash functions, has to be generalized to more general constructions. Namely, the overlayer definition imposes a restriction on the constructions that it captures: the query issued to an inner-primitive at the j-th step only depends on the $(j-1)$-th exchange. This restriction, in addition to the independence of inner-primitives from one another, limits our possible applications. In practice, two examples come to mind: hash functions are generally built on top of block-ciphers, and proofs in the Ideal Cipher Model assume that the adversary is provided access to the cipher and its inverse, which are obviously not independent from each other. Moreover, if we take the example of the Grøstl construction, it is not readily the composition of an overlayer with innner-primitives; e.g. , some queries would have to depend on the two previous exchanges. The problem is then to find a suitable trade-off between hypotheses on the dependency between inner-primitives and a generalization of the definition of the overlayer definition, which still allow to derive interesting bounds on the indifferentiability of the constructions.

Finally, we strongly believe that this strategy is a first step to a more fruitful use of simulation-based arguments in security proofs. More precisely, we want to investigate formulations of other security properties in terms of the existence of a simulator. We also want to allow multiple oracles in the overlayer. In particular, we are looking at plaintext-awareness and believe that it can allow modular proofs of IND-CCA security of encryption schemes. The idea underlying these research directions is that since cryptographic proofs are essentially reduction arguments, a cryptosystem has one way or another to be decomposed into a context and an inner-system to allow for a reduction. In CIL, this corresponds to rule $I-Sub$. However, it is often the case that such a decomposition is not possible because of dependencies between oracles that prevent the split of the state space in two independent state spaces. In such a case, we believe that simulation-based rules would provide relevant strategies to obtain intermediate systems with which to carry out the reduction, in addition to a bound on the indistinguishability between these intermediate systems and the original cryptosystems.

Bibliography

[AHMP10] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. Sha-3 proposal blake. Submission to NIST (Round 3), 2010. 12

[AMP10] Elena Andreeva, Bart Mennink, and Bart Preneel. On the indifferentiability of the grøstl hash function. In Juan A. Garay and Roberto De Prisco, editors, *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, volume 6280 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2010. 9

[AR00] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000)*, Sendai, Japan, 2000. Springer-Verlag, Berlin Germany. 9, 3

[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, February–March 2008. 8, 2

[BBM00] A. Boldyreva, M. Bellare, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Advances in Cryptology-EUROCRYPT 2000, Lecture Notes in Comput. Sci., Vol. 1807*, pages 259–274. Springer, 2000. 15

[BBU08] Michael Backes, Matthias Berg, and Dominique Unruh. A formal language for cryptographic pseudocode. In *LPAR 2008*, pages 353–376. Springer, November 2008. 10

[BCCM+08] Emmanuel Bresson, Anne Canteaut, Benoit Chevallier-Mames, Christophe Clavier, Thomas Fuhr, Aline Gouget, Thomas Icart, Jean-Francois Misarsky, M. Naya-Plasencia, Pascal Paillier, Thomas Pornin, Jean-René Reinhard, Céline Thuillet, and Marion Videau. Shabal, a submission to nists cryptographic hash algorithm competition. Submission to NIST, 2008. i, 9, 11, 12, 139

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. In *Proceedings of the 37th Symposium on Foundations of Computer Science, IEEE*, pages 514–523. IEEE, 1996. 12

[BCT04] Gilles Barthe, Jan Cederquist, and Sabrina Tarento. A Machine-Checked Formalization of the Generic Model and the Random Oracle Model. In D. Basin and M. Rusinowitch, editors, *Proceedings of IJCAR'04*, volume 3097 of *LNCS*, pages 385–399, 2004. 10

[BDD+06] Michael Backes, Anupam Datta, Ante Derek, John C. Mitchell, and Mathieu Turuani. Compositional analysis of contract-signing protocols. *Theor. Comput. Sci.*, 367(1-2):33–56, 2006. 10

[BDJR97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *FOCS*, pages 394–403, 1997. 15

[BDPA07] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions. Ecrypt Hash Workshop, may 2007. 112

[BDPA08] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008. 9, 139

[BDPA11] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The keccak sha-3 submission. Submission to NIST (Round 3), 2011. 12

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, pages 26–45, London, UK, 1998. Springer-Verlag. 15

[BGHB11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *CRYPTO*, pages 71–90, 2011. 11

[BGLB11] Gilles Barthe, Benjamin Grégoire, Yassine Lakhnech, and Santiago Zanella Béguelin. Beyond provable security verifiable ind-cca security of oaep. In *CT-RSA*, pages 180–196, 2011. 11

[BGZB09] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *POPL '09: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 90–101, New York, NY, USA, 2009. ACM. 11

[BH97] Christel Baier and Holger Hermanns. Weak bisimulation for fully probabilistic processes. In *CAV*, pages 119–130, 1997. 40

[BHK09] Mihir Bellare, Dennis Hofheinz, and Eike Kiltz. Subtleties in the definition of ind-cca: When and how should challenge-decryption be disallowed? Cryptology ePrint Archive, Report 2009/418, 2009. 15

[BJST08] Bruno Blanchet, Aaron D. Jaggard, Andre Scedrov, and Joe-Kai Tsay. Computationally sound mechanized proofs for basic and public-key Kerberos. In *Proceedings of ASIACCS'08*, pages 87–99. ACM, 2008. 11

[Bla08] Bruno Blanchet. *Vérification automatique de protocoles cryptographiques : modèle formel et modèle calculatoire. Automatic verification of security protocols: formal model and computational model.* Mémoire d'habilitation à diriger des recherches, Université Paris-Dauphine, November 2008. En français avec publications en anglais en annexe. In French with publications in English in appendix. 11

[Bla11] Bruno Blanchet. A second look at shoup's lemma. In *Workshop on Formal and Computational Cryptography (FCC 2011)*, Paris, France, June 2011. 11

[BMN09] Rishiraj Bhattacharyya, Avradip Mandal, and Mridul Nandi. Indifferentiability characterization of hash functions and optimal bounds of popular domain extensions. In Bimal Roy and Nicolas Sendrier, editors, *Progress in Cryptology - INDOCRYPT 2009*, volume 5922 of *Lecture Notes in Computer Science*, pages 199–218. Springer Berlin / Heidelberg, 2009. 11, 12, 110

[BMN10] Rishiraj Bhattacharyya, Avradip Mandal, and Mridul Nandi. Security analysis of the mode of jh hash function. In Seokhie Hong and Tetsu Iwata, editors, *Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers*, volume 6147 of *Lecture Notes in Computer Science*, pages 168–191. Springer, 2010. 9, 11

[BMU10] Michael Backes, Matteo Maffei, and Dominique Unruh. Computationally sound verification of source code. In *ACM CCS 2010*, pages 387–398. ACM Press, October 2010. Preprint on IACR ePrint 2010/416. 10

[BP06] Bruno Blanchet and David Pointcheval. Automated security proofs with sequences of games. In *Advances in Cryptology – CRYPTO'06*, volume 4117 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 2006. 11

[BP10] Bruno Blanchet and David Pointcheval. The computational and decisional Diffie-Hellman assumptions in CryptoVerif. In *Workshop on Formal and Computational Cryptography (FCC 2010)*, Edimburgh, United Kingdom, July 2010. 11

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993. i, 6, 17, 101

[BR94] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In *EUROCRYPT*, pages 92–111, 1994. 10, 4

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with rsa and rabin. In *EUROCRYPT*, pages 399–416, 1996. 58, 65

[BR04] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331, 2004. 10, 4

[BR06] Mihir Bellare and Thomas Ristenpart. Multi-property-preserving hash domain extension and the emd transform. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 299–314. Springer Berlin / Heidelberg, 2006. 12

[BT04] Gilles Barthe and Sabrina Tarento. A machine-checked formalization of the random oracle model. In Jean-Christophe Filliâtre, Christine Paulin-Mohring, and Benjamin Werner, editors, *Proceedings of TYPES'04*, volume 3839 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2004. 10

[Buc99] Peter Buchholz. Exact performance equivalence: An equivalence relation for stochastic automata. *Theor. Comput. Sci.*, 215(1-2):263–287, 1999. 51

[CdH06] Ricardo Corin and Jerry den Hartog. A probabilistic Hoare-style logic for game-based cryptographic proofs. In *Proceedings of ICALP'06*, volume 4052 of *LNCS*, pages 252–263, 2006. 10

[CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-damgård revisited: How to construct a hash function. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, 2005. i, 7, 9, 110, 113, 114, 139, 141

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004. 7

[CKW11] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *J. Autom. Reasoning*, 46(3-4):225–259, 2011. 9, 3

[CLNY06] Donghoon Chang, Sangjin Lee, Mridul Nandi, and Moti Yung. Indifferentiable security analysis of popular hash functions with prefix-free padding. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *Lecture Notes in Computer Science*, pages 283–298. Springer, 2006. 11

[CN08] Donghoon Chang and Mridul Nandi. Improved indifferentiability security analysis of chopmd hash function. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 429–443. Springer, 2008. 9, 141

[Dam90] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, volume 435 of *Lecture*

Notes in Computer Science, pages 416–427. Springer Berlin / Heidelberg, 1990. 6, 110

[DDMR07] Anupam Datta, Ante Derek, John C. Mitchell, and Arnab Roy. Protocol composition logic (PCL). *Electr. Notes Theor. Comput. Sci.*, 172:311–358, 2007. 10

[DDMW06] Anupam Datta, Ante Derek, John C. Mitchell, and Bogdan Warinschi. Computationally sound compositional logic for key exchange protocols. In *CSFW '06: Proceedings of the 19th IEEE workshop on Computer Security Foundations*, pages 321–334, Washington, DC, USA, 2006. IEEE Computer Society. 10

[DGH+04] Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the cbc, cascade and hmac modes. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2004. 139

[DRS09] Yevgeniy Dodis, Thomas Ristenpart, and Thomas Shrimpton. Salvaging merkle-damgård for practical applications. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 371–388. Springer Berlin / Heidelberg, 2009. 11

[DY83] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983. 7, 1

[FGL10] Ewan Fleischmann, Michael Gorski, and Stefan Lucks. Some observations on indifferentiability. In Ron Steinfeld and Philip Hawkes, editors, *Information Security and Privacy - 15th Australasian Conference, ACISP 2010, Sydney, Australia, July 5-7, 2010. Proceedings*, volume 6168 of *Lecture Notes in Computer Science*, pages 117–134. Springer, 2010. 11

[Flo67] Robert W. Floyd. Assigning meanings to programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967. 77

[FLS+10] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The skein hash function family. Submission to NIST (Round 3), 2010. 12

[FOPS04] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. Rsa-oaep is secure under the rsa assumption. *J. Cryptology*, 17(2):81–104, 2004. 10, 4

[GKM+11] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl – a sha-3 candidate. Submission to NIST (Round 3), 2011. 12, 110

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. 7, 1, 14

[GMR88] Shafi Goldwasser, Silvio Micali, and Ron L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988. 16

[Hal05] Shai Halevi. A plausible approach to computer-aided cryptographic proofs. `http://theory.lcs.mit.edu/~shaih/pubs.html`, 2005. 10, 4, 143

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12:576–580, October 1969. 77

[IK06] Russell Impagliazzo and Bruce M. Kapron. Logics for reasoning about cryptographic constructions. *Journal of Computer and Systems Sciences*, 72(2):286–320, 2006. 9

[Low95] G. Lowe. An attack on the Needham-Schroeder public-key authentification protocol. *Information Processing Letters*, 56(3):131–133, 1995. 8, 2

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder Public-Key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *LNCS*, pages 147–166, 1996. 8, 2

[Mer89] Ralph C. Merkle. One way hash functions and des. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer, 1989. 6, 110

[MRH04] Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *Theory of Cryptography*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer Berlin / Heidelberg, 2004. i, 7, 8, 113

[MT07] Ueli M. Maurer and Stefano Tessaro. Domain extension of public random functions: Beyond the birthday barrier. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 187–204. Springer, 2007. 9, 141

[Now07] David Nowak. A framework for game-based security proofs. In *ICICS*, pages 319–333, 2007. 10

[NS78] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978. 8, 2

[Poi00] D. Pointcheval. Chosen-ciphertext security for any one-way cryptosystem. In *PKC'00*, pages 129–146, 2000. i, 6, 106, 107

[PP04] Duong Hieu Phan and David Pointcheval. On the security notions for public-key encryption schemes. In *SCN*, pages 33–46, 2004. 15

[Rab79] M. O. Rabin. Probabilistic algorithms in finite fields. Technical Report MIT/LCS/TR-213, 1979. 9, 2

[RDDM07] Arnab Roy, Anupam Datta, Ante Derek, and John C. Mitchell. Inductive proofs of computational secrecy. In *ESORICS*, pages 219–234, 2007. 10

[RDM07] Arnab Roy, Anupam Datta, and John C. Mitchell. Formal proofs of cryptographic security of diffie-hellman-based protocols. In *TGC*, pages 312–329, 2007. 10

[RS04] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *FSE*, pages 371–388, 2004. 6

[RSS11] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 487–506. Springer, 2011. 7, 9

[Sha48] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948. 7, 1

[Sha49] C.E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715, October 1949. 7, 1

[Sho02] Victor Shoup. Oaep reconsidered. *J. Cryptology*, 15(4):223–249, 2002. 10, 4

[Sho04] V. Shoup. Sequences of games: a tool for taming complexity in security proofs, 2004. URL: `http://eprint.iacr.org/2004/332`. 10, 4, 39

[Wu11] Hongjun Wu. The hash function jh. Submission to NIST (round 3), 2011. 12

[Zha08] Yu Zhang. The computational SLR: a logic for reasoning about computational indistinguishability. IACR ePrint Archive 2008/434, 2008. Also in Proc. of Typed Lambda Calculi and Applications 2009. 10

$$\frac{\dfrac{\mathbb{CDH} :_{\epsilon(\mathbf{1}_k,t+t')} \mathrm{R} = g^{\alpha.\beta}}{\mathbb{CE}l\mathbb{G}amal' :_{\epsilon} \mathrm{R} = g^{\mu}.(g^{\alpha.\beta})^{-1}}\, B-Sub \qquad \mathbb{CE}l\mathbb{G}amal \equiv_{\mathcal{R},\mathsf{true}} \mathbb{CE}l\mathbb{G}amal'}{\mathbb{CE}l\mathbb{G}amal :_{\epsilon} \mathrm{R} = g^{\mu}(\wedge \mathsf{G}_{true})}\, B-Bis\mathsf{G}$$

$$\begin{array}{l} I-Sub \\ Trans \end{array} \frac{\dfrac{\mathbb{R}eal\mathbb{DH} \sim_{\varepsilon(\mathbf{1}_k,t)} \mathbb{R}and\mathbb{DH}}{\mathbb{E}l\mathbb{G}amal = \mathbb{C}[\mathbb{R}eal\mathbb{DH}] \sim_{\varepsilon(k,t-k(\mathcal{E}).t')} \mathbb{C}[\mathbb{R}and\mathbb{DH}]} \qquad \dfrac{\text{right tree}}{\mathbb{C}[\mathbb{R}and\mathbb{DH}] \sim_0 \mathbb{RE}l\mathbb{G}amal}}{\mathbb{E}l\mathbb{G}amal \sim_{\varepsilon(k,t-k(\mathcal{E}).t')} \mathbb{RE}l\mathbb{G}amal}$$

$$\begin{array}{l} I-Det \\ Trans \end{array} \frac{\dfrac{\mathbb{I}nterm \leq_{det,\gamma} \mathbb{C}[\mathbb{R}and\mathbb{DH}]}{\mathbb{C}[\mathbb{R}and\mathbb{DH}] \sim_0 \mathbb{I}nterm} \qquad \dfrac{\mathbb{I}nterm :_0 \mathsf{F}_{\neg\mathsf{true}} \qquad \mathbb{I}nterm \equiv_{\mathcal{R},\mathsf{true}} \mathbb{RE}l\mathbb{G}amal}{\mathbb{I}nterm \sim_0 \mathbb{RE}l\mathbb{G}amal}\,\text{I-Sub}}{\mathbb{C}[\mathbb{R}and\mathbb{DH}] \sim_0 \mathbb{RE}l\mathbb{G}amal}$$

Figure VII.1 – Proof Trees for ElGamal Encryption Confidentiality (Top) and ROR-plaintext Security (Bottom)

Generic preservation rules:

Below, c is $x \leftarrow \mathcal{U}$ or of the form $x := e'$ with e' being either $w||y$, $w \oplus y$, $f(y)$ or $\alpha \oplus H(y)$. Provided $x \notin V_1 \cup V_2$, and $z \neq x$:

(G1) $\{\mathsf{Indis}(z; V_1; V_2; \varepsilon)\}\ \mathsf{c}\ \{\mathsf{Indis}(z; V_1; V_2; \varepsilon')\}$

(G2) $\{\mathsf{WS}(z; V_1; V_2; \varepsilon)\}\ \mathsf{c}\ \{\mathsf{WS}(z; V_1; V_2; \varepsilon')\}$

where $\varepsilon'(k, \kappa, t) = \varepsilon(k, \kappa - \kappa_{\mathsf{c}}, t)$.

When $z \neq x$:

(G1') $\{\mathsf{Indis}(z; V_1; V_2; \varepsilon)\}\ \mathsf{c}\ \{\mathsf{Indis}(z; V_1, x; V_2; \varepsilon')\}$, if e' is constructible from $(V_1 \setminus \{z\}; V_2 \setminus \{z\})$,

(G2') $\{\mathsf{WS}(z; V_1; V_2; \varepsilon)\}\ \mathsf{c}\ \{\mathsf{WS}(z; V_1, x; V_2; \varepsilon')\}$, if e' is constructible from $(V_1; V_2)$,

where $\varepsilon'(k, \kappa, t) = \varepsilon(k + \kappa_{\mathsf{c}}, \kappa - \kappa_{\mathsf{c}}, t + \mathsf{T}_{\mathsf{c}}))$.

When $H' \neq H$:

(G3) $\{\mathsf{H}(H'; e[e'/x]; \varepsilon)\}\ \mathsf{c}\ \{\mathsf{H}(H'; e; \varepsilon')\}$, where $\varepsilon'(k, \kappa, t) = \varepsilon(k, \kappa - \kappa_{\mathsf{c}}, t)$.

(G3') $\{\mathsf{H}(H'; e; \varepsilon)\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{H}(H'; e; \varepsilon)\}$, if x does not appear in e.

Random assignment rules:

(R1) $\{\mathsf{true}\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{Indis}(x; \mathsf{Var} \cup \{\sigma\}, \emptyset; 0)\}$

(R2) $\{\mathsf{true}\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{H}(H; e; \frac{\kappa(H)}{2^l})\}$ if $x \in \mathsf{subvar}(e)$.

If $x \neq y$:

(R3) $\{\mathsf{Indis}(y; V_1; V_2; \varepsilon)\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{Indis}(y; V_1, x; V_2; \varepsilon)\}$

(R4) $\{\mathsf{WS}(y; V_1; V_2; \varepsilon)\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{WS}(y; V_1, x; V_2; \varepsilon)\}$

Hash functions rules:

When $x \neq y$:

(H1) $\{\mathsf{WS}(y; V_1; V_2; \varepsilon) \wedge \mathsf{H}(H; y; \varepsilon')\}\ x := \alpha \oplus H(y)\ \{\mathsf{Indis}(x; V_1, x; V_2; \varepsilon'')\}$, where $\varepsilon''(k, \kappa, t) = \varepsilon'(\kappa - \mathbf{1}^H) + k(H) * \varepsilon(k, \kappa - \mathbf{1}^H, t)$.

(H2) $\{\mathsf{WS}(y; V_1; V_2, y; \varepsilon) \wedge \mathsf{H}(H; y; \varepsilon')\}\ x := \alpha \oplus H(y)\ \{\mathsf{Indis}(x; V_1, x; V_2, y; \varepsilon'')\}$ if $y \notin V_1$, where $\varepsilon''(k, \kappa, t) = \varepsilon'(\kappa - \mathbf{1}^H) + \varepsilon(k, \kappa - \mathbf{1}^H, t + k(H) * \mathsf{T}_f)$

When $x \neq y$, and $x \in \mathsf{subvar}(e)$:

(H3) $\{\mathsf{H}(H; y; \varepsilon)\}\ x := \alpha \oplus H(y)\ \{\mathsf{H}(H'; e; \varepsilon')\}$
where $\varepsilon'(\kappa) = \varepsilon(\kappa - \mathbf{1}^H) + \frac{\kappa(H)}{2^{\ell(H)}}$.

Provided that $x \neq y, z$:

(H4) $\{\mathsf{WS}(y; V_1; V_2; \varepsilon_1) \wedge \mathsf{WS}(z; V_1; V_2; \varepsilon_2) \wedge \mathsf{H}(H; y; \varepsilon_3)\}\ x := \alpha \oplus H(y)$
$\{\mathsf{WS}(z; V_1, x; V_2; \varepsilon_4)\}$
where $\varepsilon_4(k, \kappa, t) = k(H) * \varepsilon_1(k, \kappa - \mathbf{1}^H, t) + \varepsilon_2(k, \kappa - \mathbf{1}^H, t) + \varepsilon_3(\kappa - \mathbf{1}^H)$.

Provided that $z \in \mathsf{subvar}(e) \wedge x \notin \mathsf{subvar}(e)$:

(H5) $\{\mathsf{H}(H; e; \varepsilon) \wedge \mathsf{WS}(z; y; \emptyset; \varepsilon')\}\ x := \alpha \oplus H(y)\ \{\mathsf{H}(H; e; \varepsilon'')\}$
where $\varepsilon''(\kappa) = \varepsilon(\kappa - \mathbf{1}_H) + \varepsilon'(0, \kappa - \mathbf{1}_H, 0)$.

If $x \neq y$:

(H6) $\{\mathsf{WS}(y; V_1; V_2, y; \varepsilon) \wedge \mathsf{H}(H; y; \varepsilon')\}\ x := \alpha \oplus H(y)\ \{\mathsf{WS}(y; V_1, x; V_2, y; \varepsilon'')\}$
where $\varepsilon''(k, \kappa, t) = \varepsilon(k, \kappa - \mathbf{1}_H, t + (k(H) + 1) * \mathsf{T}_f) + \varepsilon'(\kappa - \mathbf{1}_H) + \varepsilon(k, \kappa - \mathbf{1}_H, t)$.

When $x \neq y, z$:

(H7) $\{\mathsf{Indis}(z; V_1, z; V_2; \varepsilon_1) \wedge \mathsf{WS}(y; V_1, z; V_2; \varepsilon_2) \wedge \mathsf{H}(H; y; \varepsilon_3)\}\ x := \alpha \oplus H(y)\ \{\mathsf{Indis}(z; V_1, z, x; V_2; \varepsilon_4)\}$
where $\varepsilon_4(k, \kappa, t) = \varepsilon_1(k, \kappa - \mathbf{1}_H, t) + 2.k(H) * \varepsilon_2(k, \kappa - \mathbf{1}_H, t) + 2.\varepsilon_3(\kappa - \mathbf{1}_H)$.

One-way function rules:

If $x, y \notin V_1 \cup V_2$:

(P1) $\{\mathsf{Indis}(y; V_1; V_2, y; \varepsilon)\}\ x := f(y)\ \{\mathsf{Indis}(x; V_1, x; V_2; \varepsilon)\}$

If $y \notin V_1 \cup \{x\}$:

(P2) $\{\mathsf{Indis}(y; V_1; V_2, y; \varepsilon)\}\ x := f(y)\ \{\mathsf{WS}(y; V_1, x; V_2, y; \varepsilon'\})\}$
where $\varepsilon'(k, \kappa, t) = \varepsilon(k, \kappa, t + \mathsf{T}_f) + OW(t)$.

If $z \neq x, y$:

(P3) $\{\mathsf{Indis}(z; V_1, z; V_2, y; \varepsilon)\}\ x := f(y)\ \{\mathsf{Indis}(z; V_1, z, x; V_2, y; \varepsilon)\}$.

If $z \neq x, y$

(P4) $\{\mathsf{WS}(z; V_1; V_2; \varepsilon) \wedge \mathsf{Indis}(y; V_1; V_2, y, z; \varepsilon')\}\ x := f(y)\ \{\mathsf{WS}(z; V_1, x; V_2, y; \varepsilon'')\}$ where $\varepsilon''(k, \kappa, t) = \varepsilon(k, \kappa, t + \mathsf{T}_f) + \varepsilon'(k, \kappa, t + \mathsf{T}_f)$.

Exclusive or rules:

If $y \notin V_1 \cup V_2$, $y \neq x, z$:

(X1) $\{\mathsf{Indis}(y; V_1, y, z; V_2; \varepsilon)\}\ x := y \oplus z\ \{\mathsf{Indis}(x; V_1, x, z; V_2; \varepsilon)\}$

(X2) $\{\mathsf{Indis}(w; V_1, y, z; V_2; \varepsilon)\}\ x := y \oplus z\ \{\mathsf{Indis}(w; V_1, x, y, z; V_2; \varepsilon)\}$, if $w \neq x, y, z$

(X3) $\{\mathsf{WS}(w; V_1, y, z; V_2; \varepsilon)\}\ x := y \oplus z\ \{\mathsf{WS}(w; V_1, x, y, z; V_2; \varepsilon)\}$, if $w \neq x$

Concatenation rules:

If $x \notin V_1 \cup V_2$:

(C1) $\{\mathsf{WS}(y; V_1; V_2; \varepsilon)\}\ x := y||z\ \{\mathsf{WS}(x; V_1; V_2; \varepsilon)\}$

A dual rule applies for z.

If $y, z \notin V_1 \cup V_2 \cup \{x\}$:

(C2) $\{\mathsf{Indis}(y; V_1, y, z; V_2; \varepsilon) \wedge \mathsf{Indis}(z; V_1, y, z; V_2; \varepsilon')\}\ x := y||z\ \{\mathsf{Indis}(x; V_1, x; V_2; \varepsilon + \varepsilon')\}$,

(C3) $\{\mathsf{Indis}(w; V_1, y, z; V_2; \varepsilon)\}\ x := y||z\ \{\mathsf{Indis}(w; V_1, x, y, z; V_2; \varepsilon)\}$, if $w \neq x, y, z$,

(C4) $\{\mathsf{WS}(w; V_1, y, z; V_2; \varepsilon)\}\ x := y||z\ \{\mathsf{WS}(w; V_1, y, z, x; V_2; \varepsilon)\}$, if $w \neq x$,

Consequence and sequential composition rules:

(Csq) if $\varphi_0 \Rightarrow \varphi_1$, $\{\varphi_1\}\ \mathsf{c}\ \{\varphi_2\}$ and $\varphi_2 \Rightarrow \varphi_3$ then $\{\varphi_0\}\ \mathsf{c}\ \{\varphi_3\}$

(Seq) if $\{\varphi_0\}\ \mathsf{c}_1\ \{\varphi_1\}$ and $\{\varphi_1\}\ \mathsf{c}_2\ \{\varphi_2\}$, then $\{\varphi_0\}\ \mathsf{c}_1; \mathsf{c}_2\ \{\varphi_2\}$

(Conj) if $\{\varphi_0\}\ \mathsf{c}\ \{\varphi_1\}$ and $\{\varphi_0\}\ \mathsf{c}\ \{\varphi_1\}$, then $\{\varphi_0\}\ \mathsf{c}\ \{\varphi_1 \wedge \varphi_2\}$

Rules for Indis_f: c is $x \leftarrow \mathcal{U}$ or of the form $x := e'$ with e' being either $w||y$, $w \oplus y$, $f(y)$ or $\alpha \oplus H(y)$.

$(P1)^f$ $\{\mathsf{Indis}(y; V_1; V_2, y; \varepsilon)\}\ x := f(y)\ \{\mathsf{Indis}_f(x; V_1, x; V_2; \varepsilon)\}$ if $x, y \notin V_1 \cup V_2$.

$(G1)^f$ $\{\mathsf{Indis}_f(z; V_1; V_2; \varepsilon)\}\ \mathsf{c}\ \{\mathsf{Indis}_f(z; V_1; V_2; \varepsilon')\}$, when $z \neq x$, $x \notin V_1 \cup V_2$, and where $\varepsilon'(k, \kappa, t) = \varepsilon(k, \kappa - \kappa_{\mathsf{c}}, t)$.

$(G1')^f$ $\{\mathsf{Indis}_f(z; V_1; V_2; \varepsilon)\}\ \mathsf{c}\ \{\mathsf{Indis}_f(z; V_1, x; V_2; \varepsilon')\}$, when $z \neq x$, if e' is constructible from $(V_1 \setminus \{z\}; V_2 \setminus \{z\})$ and where $\varepsilon'(k, \kappa, t) = \varepsilon(k + \kappa_{\mathsf{c}}, \kappa - \kappa_{\mathsf{c}}, t + \mathsf{T}_{\mathsf{c}}))$.

$(R3)^f$ $\{\mathsf{Indis}_f(y; V_1; V_2; \varepsilon)\}\ x \leftarrow \mathcal{U}(l)\ \{\mathsf{Indis}_f(y; V_1, x; V_2; \varepsilon)\}$, if $x \neq y$.

$(H7)^f$ $\{\mathsf{Indis}_f(z; V_1, z; V_2; \varepsilon_1) \wedge \mathsf{WS}(y; V_1, z; V_2; \varepsilon_2) \wedge \mathsf{H}(H; y; \varepsilon_3)\}\ x := \alpha \oplus H(y)\ \{\mathsf{Indis}_f(z; V_1, z, x; V_2; \varepsilon_4)\}$ if $x \neq y, z$ and where $\varepsilon_4(k, \kappa, t) = \varepsilon_1(k, \kappa - \mathbf{1}_H, t) + 2.k(H) * \varepsilon_2(k, \kappa - \mathbf{1}_H, t) + 2.\varepsilon_3(\kappa - \mathbf{1}_H)$.

$(P3)^f$ $\{\mathsf{Indis}_f(z; V_1, z; V_2, y; \varepsilon)\}\ x := f(y)\ \{\mathsf{Indis}_f(z; V_1, z, x; V_2, y; \varepsilon)\}$ if $z \neq x, y$.

$(X2)^f$ $\{\mathsf{Indis}_f(w; V_1, y, z; V_2; \varepsilon)\}\ x := y \oplus z\ \{\mathsf{Indis}_f(w; V_1, x, y, z; V_2; \varepsilon)\}$, if $w \neq x, y, z$.

$(C3)^f$ $\{\mathsf{Indis}_f(w; V_1, y, z; V_2; \varepsilon)\}\ x := y||z\ \{\mathsf{Indis}_f(w; V_1, x, y, z; V_2; \varepsilon)\}$, if $w \neq x, y, z$.

Injective partially trapdoor one-way functions:
If $x, y \notin (V_1 \cup V_2 \cup \{z\})$:

$(IPO1)^f$ $\{\mathsf{Indis}(x; V_1, x, y; V_2; \varepsilon) \wedge \mathsf{Indis}(y; V_1, x, y; V_2; \varepsilon')\}\ z := f(x||y)$
$\{\mathsf{Indis}_f(z; V_1, z; V_2; \varepsilon + \varepsilon')\}$.

$(IPO2)$ $\{\mathsf{Indis}(x; V_1, x, y; V_2; \varepsilon) \wedge \mathsf{Indis}(y; V_1, x, y; V_2; \varepsilon')\}\ z := f(x||y)$
$\{\mathsf{WS}(z; V_1, z; V_2; \varepsilon'')\}$, where $\varepsilon''(k, \kappa, t) = POW(t) + \varepsilon(k, \kappa, t + \mathsf{T}_f) + \varepsilon'(k, \kappa, t + \mathsf{T}_f)$.