EQUATIONAL REASONING FOR VERIFIED CRYPTOGRAPHIC SECURITY

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Joshua Ralph Gancher

December 2021

© 2021 Joshua Ralph Gancher
ALL RIGHTS RESERVED

EQUATIONAL REASONING FOR VERIFIED CRYPTOGRAPHIC SECURITY

Joshua Ralph Gancher, Ph.D.

Cornell University 2021

Modern software systems today have increasingly complex security requirements – such as supporting privacy-preserving computations, or resistance against quantum attackers – that are fulfilled by advanced forms of cryptography. At the same time, these advanced forms of cryptography often have subtle security proofs that require careful auditing. To ensure security, it is thus crucial to formally *verify* the security of the underlying cryptography, and to do so in a manner that is approachable to cryptographers.

This thesis explores the use of *equational reasoning* to conduct machine-checked security proofs. Equational reasoning is pervasive in cryptography, as it underlies the concepts of game-hopping hybrids and the simulation paradigm; thus, optimizing formal tools for equational reasoning delivers machine-checked proofs closer to their on-paper counterparts.

We first present AutoLWE, a prover for cryptographic primitives that supports reasoning about lattices. AutoLWE is built around *deducibility*, which (semi-) automatically applies hardness assumptions by partitioning the security game into an application of the hardness assumption with a context. Using AutoLWE, we deliver very short proofs of several representative constructions, including Public-Key Encryption, Identity-Based Encryption, and Inner Product Encryption.

We then present IPDL, a simple calculus and equational logic for distributed,

interactive cryptographic protocols in the computational model. The purpose of IPDL is to prove simulation results between real and idealized protocols in the style of Universal Composability (UC) [Can01]. IPDL does so by restricting its attention to *straight-line protocols*, a particularly simple but expressive subset of protocols. Using IPDL, we deliver short proofs of multiple case studies, including a semi-honest multiparty computation protocol over general circuits [GMW87], and an n-party coin toss protocol [Blu83].

BIOGRAPHICAL SKETCH

Joshua was raised in Portland, Oregon, wherein he graduated from Reed College in 2016. He finished in Ph.D. at Cornell University in 2021, and subsequently begins a postdoc at Carnegie Mellon University.

ACKNOWLEDGEMENTS

I first thank my co-advisors, Elaine Shi and Greg Morrisett, for their unending support, wisdom, and patience throughout my Ph.D. This dissertation is the product of the unique combination of their combined knowledge and perspectives.

I thank my other collaborators throughout my Ph.D.: Coşku Acay, Gilles Barthe, Xiong Fan, Benjamin Grégoire, Charlie Jacomme, Andrew Myers, Rolph Recto, and Kristina Sojakova. Academics is fruitless without collaboration, and I have been fortunate to work with such a bright community of researchers. I also thank my minor advisor, linguist Mats Rooth, for pushing me to think in new and different ways each time we speak.

Finally, I thank Melissa, for always being there for me; Stewart Little Coop, for the much-needed community in wintry Ithaca; and my parents, Steve and Sue, for making this journey possible.

TABLE OF CONTENTS

Biographical Sketch iii
Acknowledgements iv
Table of Contents v
List of Figures vii
List of Acronyms 1

1 Introduction **2**
1.1 Outline of Thesis 4

2 Symbolic Proofs for Lattice-Based Primitives **6**
2.1 Outline of Chapter 8
2.2 Example: Dual Regev Encryption 9
2.3 Logic 17
2.3.1 Games 19
2.3.2 Reasoning about expressions 23
2.3.3 Strongest postcondition 24
2.3.4 Judgment and proof rules 24
2.3.5 Soundness 28
2.3.6 Axioms Used 28
2.4 Deciding deducibility 33
2.4.1 Diffie-Hellman exponentiation 34
2.4.2 Fields and non-commutative rings 35
2.4.3 Matrices 37
2.5 Implementations and Case Studies 38
2.5.1 Implementation 39
2.5.2 Identity-Based Encryption 41
2.5.3 CCA1-PKE 44
2.5.4 Hierarchical Identity-Based Encryption 46
2.5.5 Inner Product Encryption 47
2.6 Related work 48
2.7 Conclusion 51
2.8 Proofs of section 2.4.1 51
2.8.1 Saturation into the target group 52
2.8.2 Reduction to polynomials 52
2.9 Proofs for section 2.4.3 56

3 Verifying Distributed Protocols using IPDL **62**
3.1 Outline of Chapter 67
3.2 IPDL by Example: Multi-use Secure Network 68
3.2.1 Terminology and Background 69
3.2.2 IPDL in a Nutshell 70
3.2.3 Multi-Use Secure Communication in IPDL 71

3.2.4 Simulator and Proof . . . 76
3.3 Core Logic . . . 81
3.3.1 Syntax . . . 81
3.3.2 Typing . . . 83
3.3.3 Equational Logic . . . 86
3.3.4 Semantics . . . 88
3.4 Parameterized Programs and Computational Soundness . . . 90
3.4.1 Soundness for PPT Adversaries . . . 90
3.4.2 Derived IPDL Constructs and Equations . . . 91
3.5 Encoding in Coq . . . 93
3.6 Case Studies . . . 96
3.6.1 Case Studies . . . 97
3.6.2 Proof Effort . . . 99
3.7 Additional Related Work . . . 100
3.8 Future Work . . . 101
3.9 Semantics . . . 102
3.9.1 Adversaries . . . 107
3.10 More Details on Case Studies . . . 114
3.10.1 OT from Trapdoor Permutations . . . 114
3.10.2 1-4 OT from 1-2 OT . . . 115
3.10.3 Two-Party GMW Protocol . . . 116
3.10.4 Coin Flip . . . 118

4 Future Work **127**

Bibliography **129**

LIST OF FIGURES

2.1 IND-CPA security of dual-Regev PKE. 13
2.2 Dual-Regev PKE: Game 2 14
2.3 Dual-Regev PKE: Game 3 15
2.4 Dual-Regev PKE: Game 4 15
2.5 AutoLWE proof for Dual Regev Encryption. 18
2.6 Syntax of expressions (selected) 19
2.7 Syntax of games 20
2.8 Selected proof rules 25
2.9 The LWE assumption, encoded in AutoLWE. 29
2.10 The LHL assumption combined with TrapGen, encoded in AutoLWE. 32
2.11 Example axiom capturing computational closeness of distributions. 33
2.12 Overview of case studies. All proofs took less than three seconds to complete. 39
2.13 Typing rules for matrix operators. 56

3.1 Definitions of authenticated and secure networks in IPDL. Both networks are parameterized by the number of queries in question, q, and the length of messages, m. 73
3.2 Authenticated-to-secure network protocol in IPDL. 75
3.3 Simulator for the authenticated-to-secure network example in IPDL. 76
3.4 Outline of proof for authenticated-to-secure network example. . . . 76
3.5 Syntax of IPDL Protocols. 81
3.6 Typing for Reactions. 83
3.7 The IPDL proof system for protocol equivalence. 84
3.8 Typing Rules for Protocols. 85
3.9 Case studies considered and lines of code. 100
3.10 Equivalence of Reactions in IPDL. 109
3.11 Derived rules for parameterized IPDL protocols. 124
3.12 Specification of OT functionality in IPDL. 125
3.13 Specification of ideal protocol for n-party coin flip in IPDL. 125
3.14 Real protocol for n-party coin flip in IPDL. 126

LIST OF ACRONYMS

CCA Chosen-Ciphertext Attack 44

CPA Chosen-Plaintext Attack 11

GMW Goldreich-Micali-Widgerson MPC Protocol 97

(H)IBE (Hierarchical) Identity-Based Encryption 41

IPE Inner-Product Encryption 47

LHL Leftover Hash Lemma 12

LWE Learning With Errors 11

MPC Multi-Party Computation 97

OT Oblivious Transfer 97

PKE Public-Key Encryption 9

SAGBI Subalgebra Analog to Gröbner Basis for Ideals 33

UC Universal Composability 70

CHAPTER 1

INTRODUCTION

Some of the most promising innovations in computing come from modern cryptography. Far from theoretical curiosity, there now exist real-world implementations of zero-knowledge proofs (ZKP) [GO94] for outsourced verifiable computation, multi-party computation (MPC) [EKR17] for privacy-preserving distributed computations, and lattice-based cryptosystems [Pei15] for quantum-resistance and homomorphic computations.

Cryptographic protocols are only useful when we can place trust in their security. This is made possible through the *provable security* paradigm: to show that a protocol is secure, the cryptographer proves on paper that subverting the protocol's security is at least as difficult as breaking a *hardness assumption* (e.g., computing a discrete logarithm). Since there are only a handful of hardness assumptions, it is possible for the academic community to systematically rule out all proposed attacks on them.

Unfortunately, the increasing complexity of cryptographic protocols makes peer-review an imperfect process. Cryptographic proofs often consist of multiple long, involved intermediate reductions which are frequently stated without proof, since proving them would involve an unreasonable amount of hard-to-read tedium. Indeed, the tradition of informal proofs in cryptography led to the famous declaration of a "crisis of rigor" by Bellare and Rogaway in 2008 [BR04], since informal proofs are "essentially unverifiable".

Incorrect proofs can and do evade human peer-review. A notable example is found in ZCash [zcaa], a blockchain for privacy-preserving cryptocurrency. Each

transaction in ZCash is supported by a zero-knowledge proof of well-formedness, based on a protocol which passed peer-review [BSCTV13]. Two years after its launch, an employee for ZCash found an error in the underlying protocol [zcab], allowing an attacker to counterfeit coins at will. Had a hacker discovered the attack, the protocol – along with the $1.3 billion dollars in the blockchain – would have been rendered valueless.

The solution is to move towards *verified security*, wherein on-paper proofs are replaced by fully formal artifacts which are checked by a mechanized theorem prover such as EasyCrypt [BGHZ11, BDG+13] or Coq. Verified proofs of security offer a number of advantages, including: raised trust in the proof's correctness; possibilities for automating technical tedium; executable prototypes that can be debugged and tested; and possibilities for formal security proofs for low-level machine code. Achieving verified security has been the subject of a long line of research [BBB+21], but is far from a settled matter.

Unfortunately, the strength of verified security is also its barrier to entry. Machine-checked proofs often require a great amount of detail and low-level reasoning which has no counterpart in an ordinary paper in cryptography. Because of this, cryptographers who wish to verify their security proofs must gain significant, additional training in formal methods in order to use the formal tools efficiently. While the community of cryptographers proficient in formal methods is growing, the academic community is still far away from machine-checked proofs being ubiquitous.

To decrease this barrier to entry, we must raise the level of abstraction for machine-checked proofs. The closer machine-checked proofs are to their on-paper counterparts, the more adoption of verified security we will see from the community.

In an ideal world, cryptographers would vew formal methods not as a hindrance towards easy proofs, but instead as a useful cognitive aid for guiding their proofs and modularizing their thought processes.

The goal of this thesis is to explore *equational reasoning* as a technique for bringing verified security closer to on-paper arguments. Equational reasoning is ubiquitous in cryptography: computational indistinguishibility is an observational equivalence between probability distributions; cryptographic reductions can be seen as equivalences between security games; and cryptographic protocols are proven secure by proving them equivalent to an idealization. Since equational reasoning is implicitly found in most cryptographic proofs, formal tools that optimize for equational reasoning are likely to be more approachable for cryptographic proofs.

1.1 Outline of Thesis

In this thesis, we construct new tools for using equational reasoning to mechanically prove cryptography secure. We focus on two domains: non-interactive lattice-based cryptosystems; and interactive protocols in the style of Universal Composability [Can01].

Symbolic Proofs for Lattice-based Cryptography Lattice-based cryptography [Pei15] is a relatively new paradigm for public-key cryptosystems based on problems involving noisy samples of integer lattices. Lattice-based hardness assumptions such as LWE [Pei15] are widely believed to be resistant to quantum attacks, unlike number-theoretic schemes based on RSA or Diffie-Hellman which are subverted by Shor's algorithm [Sho94].

In Chapter 2, we present AutoLWE, a prover for cryptographic primitives which can reason about lattices. The key technical novelty of this work is to extend *deducibility*, an algebraic technique to capture adversary knowledge, to the non-commutative setting of matrices. Deducibility allows the verifier to automatically (or semi-automatically) partition a security game into a hardness assumption and a reduction, relieving the cryptographer of most low-level reasoning steps about intermediate hybrids. Using the tool, we deliver very short proofs of several representative constructions, including CPA-PKE (Gentry et al., STOC 2008), (Hierarchical) Identity-Based Encryption (Agrawal et al. Eurocrypt 2010), Inner Product Encryption (Agrawal et al. Asiacrypt 2011), and CCA-PKE (Micciancio et al., Eurocrypt 2012).

A Simple Framework for Formally Verifying Cryptographic Protocols Most provers for cryptography, such as AutoLWE or EasyCrypt [BGHZ11, BDG+13], primarily support reasoning about security in the imperative, game-based paradigm. While highly successful for analyzing noninteractive primitives, they are less applicable for simulation-based proofs of general *interactive* protocols – and particularly, protocols involving more than two parties.

In Chapter 3, we present IPDL, a simple calculus and equational logic for reasoning about interactive cryptographic protocols. The main insight of IPDL is to focus our attention on *straight-line protocols*, which are protocols featuring control flow only on *data*, but not *communication*. By restricting our attention to straight-line protocols, we achieve a novel, *equational* proof system for reasoning about distributed cryptography. We demonstrate IPDL on a number of case studies, including the semi-honest GMW protocol over general circuits [GMW87], and an n-party coin toss protocol [Blu83].

CHAPTER 2
SYMBOLIC PROOFS FOR LATTICE-BASED PRIMITIVES

One of the major goals in computer-aided cryptography [BBB+21] is to use verification techniques to certify proofs of security for cryptographic protocols. Proof certificates for verified cryptosystems can be independently and automatically checked for correctness, dramatically raising trust in the cryptosystem's security.

A popular framework for verification of cryptographic protocols historically has been the Dolev-Yao model [DY83b], which idealizes the capabilities of the adversary through an algebraic abstraction of the protocol. By abstracting away from the probabilistic nature of cryptographic constructions, the Dolev-Yao model has served as a suitable and practical foundation for highly or fully automated tools [ABB+05, Bla01, SMCB12]. These tools have subsequently been used for analyzing numerous cryptographic protocols, including recently TLS 1.3. [CHH+17, KBB17].

Unfortunately, due to the algebraic abstraction, the Dolev-Yao model is focused on cryptographic protocols and cannot be used for reasoning about cryptographic primitives. Instead, tools such as CertiCrypt [BGB09], CryptHOL [Loc16] CryptoVerif [Bla06a], EasyCrypt [BGHZ11, BDG+13], and FCF [PM15a] develop probabilistic program logics to operate directly in the computational model. However, these tools require significant user interaction and expertise – doubly so when when used for reasoning about cryptographic primitives.

We follow an alternative approach, and *combine* logics for computational cryptography proofs with symbolic tools from the Dolev-Yao model. Prior work has

demonstrated that this approach works well for padding-based (combining one-way trapdoor permutations and random oracles) [BCG+13] and pairing-based cryptography [BGS15]. We combine the two techniques by designing a computational logic for game-playing security proofs which makes use of side conditions that are validated by symbolic tools. In particular, a key idea of this approach is to use *deducibility* – derivability of information from public data – to control applications of cryptographic hardness assumptions and for performing optimistic sampling, a particularly common and useful transformation in game-playing proofs.

Our approach, however, is limited to domain areas in which deducibility has efficient decision procedures. The problem of deciding deducibility has been studied extensively in the context of symbolic verification in the Dolev-Yao model, where deducibility formalizes the adversary knowledge [Low96, MS01, Pau00, KMM94, Sch96, RKTC03, CLS03, RT03]. This line of work has culminated in the design and implementations of decision procedures for classes of theories that either have some kind of normal form or satisfy a finite variant property. However, existing decidability results are primarily targeted towards algebraic theories that arise in the study of cryptographic protocols. In contrast, deducibility problems for cryptographic constructions require to reason about mathematical theories that may not have a natural notion of normal form or satisfy the finite variant property.

Thus, a main challenge for computational logics based on deducibility problems is to provide precise and automated methods for checking the latter. Prior work, such as AutoG&P, employ heuristics in order to check deducibility. While this approach may work reasonably well in practice, it is unsatisfactory: the heuristics may be incomplete or behave unpredictably for larger scale proofs. Instead, we use existing methods from computational mathematics to derive principled (semi-

)decision procedures for deducibility. By doing so, we obtain more complete and predictable algorithms. The idea using methods from computational mathematics to reason about deducibility is natural. However, we are not aware of prior work that exploits this connection in relation with the use of deducibility in a computational logic.

2.1 Outline of Chapter

This work develops symbolic methods for proving security of lattice-based cryptographic constructions. These constructions constitute a prime target for formal verification, due to their potential applications in post-quantum cryptography and their importance in the ongoing NIST effort to standardize post-quantum constructions; see e.g. [Pei16] for a recent survey of the field. We implement our logic in a tool called AutoLWE (`https://github.com/autolwe/autolwe`), and use the tool for proving (indistinguishability-based) security for several cryptographic constructions based on the Learning with Errors (LWE) assumption [Reg05a].

We first demonstrate our logic in Section 2.2, by showing how an example lattice-based cryptosystem is encoded and proven secure. We then detail our logic in Section 2.3. The logic follows the idea of combining computational proof rules with symbolic side-conditions, as in [BCG$^+$13, BGS15]. One important feature of our logic is that the proof rule for assumptions supports information-theoretic and computational assumptions that are stated using adversaries with oracle accesses. This extension is critical to capture (advanced cases of) the Leftover Hash Lemma [ILL89].

In Section 2.4, we show how we (partially) automate our logic with new algorithms for deducibility. Specifically, we present algorithms for deducibility in the

theory of Diffie-Hellman exponentiation – in its standard, bilinear, and multilinear versions – and in the theory of fields, non-commutative rings, and matrices. The central idea behind our algorithms for Diffie-Hellman exponentiation is to transform our deducibility problem into a standard problem from commutative algebra, which can be resolved using Gröbner bases. Our algorithms for non-commutative rings and matrices behave similarly, but are resolved through semi-decision procedures based on non-commutative variants of Gröbner bases known as Subalgebra Analog of Gröbner Basis on Ideals (SAGBI) [Nor98]. Additional details and proofs are given in Section 2.8 and Section 2.9.

In Section 2.5, we show how our logic behaves in practice on a number of case studies: in addition to our example from Section 2.2, we evaluate our tool on encryption schemes for chosen-ciphertext security, identity-based security, and an inner-product revealing scheme. We discuss related work in Section 2.6.

2.2 Example: Dual Regev Encryption

In this section, we describe an example public-key encryption scheme and show how it will be encoded in our formal system. We provide some mathematical background in Section 2.5.2. Recall that public-key encryption (PKE) is given by three probabilistic algorithms ($\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}$) for generating keys, encryption, and decryption, such that with overwhelming probability, decryption is the inverse of encryption for valid key pairs.

We consider the Dual Regev Encryption scheme [GPV08], an optimization of Regev's original encryption [Reg05b]. We focus on a simple version that encrypts single bits; however, standard techniques can be used to encrypt longer messages.

Definition 1 (Dual Regev Encryption). *Below, let $\lambda = n$ be the security parameter, $m = O(n \log q), q = O(m)$ and χ (or χ^n) be discrete Gaussian distribution over $\mathbb{Z}$ (or $\mathbb{Z}^n$).*

- *The key generation algorithm,* $\mathsf{KeyGen}(1^\lambda)$*, chooses a uniformly sampled random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a vector $\boldsymbol{r} \in \{-1, 1\}^m$ sampled uniformly, interpreted as a vector in $\mathbb{Z}_q^m$. The public key is* $\mathsf{pk} = (\mathbf{A}, \boldsymbol{u})$*, where $\boldsymbol{u} = \mathbf{A}\boldsymbol{r}$, and the secret key is* $\mathsf{sk} = \boldsymbol{r}$*.*
- *To encrypt a message $b \in \{0, 1\}$, the encryption algorithm* $\mathsf{Enc}(\mathsf{pk}, b)$ *chooses a random vector $\boldsymbol{s} \in \mathbb{Z}_q^n$, a vector $\boldsymbol{x}_0$ sampled from χ^n and an integer x_1 sampled from χ. The ciphertext consists of the vector $\boldsymbol{c}_0 = \boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{x}_0^\mathsf{T}$ and the integer $c_1 = \boldsymbol{s}^\mathsf{T}\boldsymbol{u} + x_1 + b\lceil q/2 \rceil$, where T denotes the transpose operation on matrices.*
- *The decryption algorithm checks whether the value $c_1 - \langle \boldsymbol{r}, \boldsymbol{c}_0 \rangle$ is closer to 0 or $b\lceil q/2 \rceil$ modulo p, and returns 0 in the first case, and 1 in the second.*

Decryption is correct with overwhelming probability, since we compute that $c_1 - \langle \boldsymbol{r}, \boldsymbol{c}_0 \rangle = x_1 + b\lceil q/2 \rceil - \langle \boldsymbol{r}, \boldsymbol{x}_0 \rangle$, so the norm of the term $x_1 - \langle \boldsymbol{r}, \boldsymbol{x}_0 \rangle$ will be much smaller than $b\lceil q/2 \rceil$.

Gentry, Peikert and Vaikuntanathan [GPV08] show that Dual Regev Encryption achieves chosen-plaintext indistinguishability under the *decisional LWE assumption*, defined below. Traditionally, chosen-plaintext indistinguishability is modeled by a probabilistic experiment, where an adversary proposes two messages m_0 and m_1, and is challenged with a ciphertext $c^\star$ corresponding to an encryption of message m_b, where b is sampled uniformly at random. The adversary is then requested to return a bit b'. The winning condition for the experiment is $b = b'$, which models that the adversary guesses the bit b correctly. Formally, one defines

the advantage of an adversary $\mathcal{A}$ against chosen-plaintext security (or *IND-CPA security*) as:

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{cpa}} = \left| \Pr_G[\, b = b' \,] - \frac{1}{2} \right|$$

where G is the probabilistic experiment that models chosen-plaintext security and $\frac{1}{2}$ represents the probability that a trivial adversary which flips a coin b' at random guesses the bit b correctly. We note that in our case, since the message space is $\{0, 1\}$, we can wlog set $m_0 = 0$ and $m_1 = 1$; thus, the adversary only needs to be queried once in this experiment.

The formal definition of G, instantiated to Dual Regev Encryption, is shown in Figure 2.1. We inline the key generation and encryption subroutines. In line 1, the public key $(\mathbf{A}, \boldsymbol{u})$ and its associated secret key $\boldsymbol{r}$ are randomly sampled. In lines 2 and 3, the message bit b is sampled uniformly, and the ciphertext (c_0, c_1) of this message is generated. Finally, in line 4, the adversary outputs a bit b', given as input the public key and the ciphertext.

Now, we outline the hardness assumptions and lemmas used in the proof of Dual Regev Encryption.

Learning with Errors

The *Learning With Errors* (LWE) assumption [Reg05b] is a computational assumption about the hardness of learning a linear function from noisy samples. We make use of the decisional variant, in which one distinguishes a polynomial number of "noisy" inner products with a secret vector from uniform.

Definition 2 (LWE)**.** *Let n, m, q, and χ be as in Definition 1. Given $\boldsymbol{s} \in \mathbb{Z}_q^n$, let $\mathsf{LWE}_{\boldsymbol{s},\chi}$ (dubbed* the LWE distribution*) be the probability distribution on $\mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^m$*

obtained by sampling $\boldsymbol{A} \in \mathbb{Z}_q^{n\times m}$ *at uniform, sampling* $\boldsymbol{e}$ *from* χ^n*, and returning the pair* $(\boldsymbol{A}, \boldsymbol{s}^\mathsf{T}\boldsymbol{A}+\boldsymbol{e})$*. The decision-*$\mathsf{LWE}_{q,n,m,\chi}$ *problem is to distinguish* $\mathsf{LWE}_{\boldsymbol{s},\chi}$ *from uniform, where* $\boldsymbol{s}$ *is uniformly sampled.*

We say the decision-$\mathsf{LWE}_{q,n,m,\chi}$ problem is infeasible if for all polynomial-time algorithms $\mathcal{A}$, the advantage $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{lwe}}(1^\lambda)$ is negligibly close to $1/2$ as a function of λ:

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{lwe}}(1^\lambda) = |\mathsf{Pr}[\mathcal{A} \text{ solves LWE}] - 1/2|$$

The works of [Reg05b, Pei09, BLP+13] show that the LWE assumption is as hard as (quantum or classical) solving GapSVP and SIVP under various settings of n, q, m and χ.

Leftover Hash Lemma

Let $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ be a collection of m samples of uniform vectors from $\mathbb{Z}_q^n$. The Leftover Hash Lemma (LHL) states that, given enough samples, the result of multiplying $\mathbf{A}$ with a random $\{-1,1\}$-valued matrix $\mathbf{R}$ is statistically close to uniform. Additionally, this result holds in the presence of an arbitrary linear leakage of the elements of $\mathbf{R}$. Specifically, the following leftover hash lemma is proved in [ABB10] (Lemma 13).

Lemma 1 (Leftover Hash Lemma)**.** *Let* q, n, m *be as in Definition 1. Let* k *be a polynomial of* n*. Then, the distributions* $\{(\mathbf{A}, \mathbf{AR}, \mathbf{R}^\mathsf{T}\mathbf{w})\}$ $\{(\mathbf{A}, \mathbf{B}, \mathbf{R}^\mathsf{T}\mathbf{w})\}$ *are negligibly close in* n*, where* $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ *in both distributions,* $\mathbf{R} \xleftarrow{\$} \{0,1\}^{m\times k}$*,* $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{n\times k}$*, and* $\mathbf{w} \in \mathbb{Z}_q^m$ *is any arbitrary vector.*

Given the above, security of Dual Regev Encryption is stated as follows:

Game $G^{\mathsf{pke}}_{\mathsf{org}}$:
$\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}, \boldsymbol{r} \xleftarrow{\$} \{-1,1\}^m$;
let $\boldsymbol{u} = \mathbf{A}\boldsymbol{r}$;
$b \xleftarrow{\$} \{0,1\}, \boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n, \boldsymbol{x}_0 \xleftarrow{\$} \mathcal{D}_{\mathbb{Z}^m}, x_1 \xleftarrow{\$} \mathcal{D}_{\mathbb{Z}}$;
let $\boldsymbol{c}_0 = \boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{x}_0, c_1 = \boldsymbol{s}^\mathsf{T}\boldsymbol{u} + x_1 + b\lceil q/2\rceil$;
$b' \leftarrow \mathcal{A}(\mathbf{A}, \boldsymbol{u}, \boldsymbol{c}_0, c_1)$;

Figure 2.1: IND-CPA security of dual-Regev PKE.

Proposition 1 ([GPV08]). *For any adversary $\mathcal{A}$ against chosen-plaintext security of Dual Regev Encryption, there exists an adversary $\mathcal{B}$ against* LWE*, such that:*

- $\mathbf{Adv}^{\mathsf{cpa}}_{\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{lwe}}_{\mathcal{B}} + \epsilon_{LHL}$*;*
- $t_{\mathcal{A}} \approx t_{\mathcal{B}}$*;*

where $\mathbf{Adv}^{\mathsf{lwe}}_{\mathcal{B}}$ *denotes the advantage of $\mathcal{B}$ against decisional LWE problem, ϵ_{LHL} is a function of the scheme parameters determined by the Leftover Hash Lemma, and $t_{\mathcal{A}}$ and $t_{\mathcal{B}}$ respectively denote the execution time of $\mathcal{A}$ and $\mathcal{B}$.*

Security proof We now outline the proof of Proposition 1.

The proof proceeds with a series of *game transformations*, beginning with the game in Figure 2.1. The goal is to transform the game into one in which the adversary's advantage is obviously zero. Each transformation is justified semantically either by semantic identities or by probabilistic assertions, such as the LWE assumption; in the latter case, the transformation incurs some error probability which must be recorded.

The first transformation performs an information-theoretic step based on the Leftover Hash Lemma. The Leftover Hash Lemma allows us to transform the joint

distribution $(\mathbf{A}, \mathbf{A}\boldsymbol{r})$ (where $\mathbf{A}$ and $\boldsymbol{r}$ are independently randomly sampled) into the distribution $(\mathbf{A}, \boldsymbol{u})$ (where $\boldsymbol{u}$ is a fresh, uniformly sampled variable). (This invocation does not use the linear leakage $\boldsymbol{w}$ from Lemma 1). In order to apply this lemma, we *factor* the security game from Figure 2.1 into one which makes use of $\boldsymbol{A}$ and $\boldsymbol{u}$, but not $\boldsymbol{r}$. That is, if G_0 is the original security game, then we have factored G into

$$G_0 = G'\{\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m};\ \boldsymbol{r} \xleftarrow{\$} \{-1,1\}^m;\ \mathsf{let}\ \boldsymbol{u} = \mathbf{A}\boldsymbol{r}\}_p,$$

where $G'\{\cdot\}_p$ is a game context with a hole at position p, such that G' does not make reference to $\boldsymbol{r}$ except in the definition of $\boldsymbol{u}$. By the Leftover Hash Lemma, we may now move to the game:

$$G_1 = G'\{\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m};\ \boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^{n}\}_p.$$

This transformation effectively removes $\boldsymbol{r}$ from the security game, thus removing any contribution of the secret key $\boldsymbol{r}$ to the information gained by the adversary $\mathcal{A}$. This transformation incurs the error probability ϵ_{LHL}. The resultant game is shown in Figure 2.2.

Game G_2 :
$\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}, \boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^{n};$
$b \xleftarrow{\$} \{0,1\}, \boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^{n}, \boldsymbol{x}_0 \xleftarrow{\$} \mathcal{D}_{\mathbb{Z}^m}, x_1 \xleftarrow{\$} \mathcal{D}_{\mathbb{Z}};$
let $\boldsymbol{c}_0 = \boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{x}_0, c_1 = \boldsymbol{s}^\mathsf{T}\boldsymbol{u} + x_1 + b\lceil q/2\rceil;$
$b' \leftarrow \mathcal{A}(\mathbf{A}, \boldsymbol{u}, \boldsymbol{c}_0, c_1);$

Figure 2.2: Dual-Regev PKE: Game 2

The second transformation performs a reduction step based on the LWE assumption. Indeed, note that after the first transformation, the ciphertexts $(\boldsymbol{c}_0, c_1)$ contain an LWE distribution of dimension $n \times (m+1)$, with the message bit added

to c_1. By applying LWE, we then may safely transform $\boldsymbol{c}_0$ to be uniformly random, and c_1 to be uniformly random added to to the message bit. The resulting security game is shown in Figure 2.3.

$$\begin{array}{l}\textsf{Game } G_3: \\ \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}, \boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n; \\ b \xleftarrow{\$} \{0,1\}, \boldsymbol{r_0} \xleftarrow{\$} \mathbb{Z}_q^m, r_1 \xleftarrow{\$} \mathbb{Z}_q; \\ \text{let } \boldsymbol{c}_0 = \boldsymbol{r_0}, c_1 = r_1 + b\lceil q/2 \rceil; \\ b' \leftarrow \mathcal{A}(\mathbf{A}, \boldsymbol{u}, \boldsymbol{c}_0, c_1); \end{array}$$

Figure 2.3: Dual-Regev PKE: Game 3

The next transformation applies a semantics-preserving transformation known as *optimistic sampling*. To remove the message bit from the adversary input, note that the term c_1 is equal to the sum of r_1 and $b\lceil q/2 \rceil$, where r_1 is uniformly sampled and does not appear anywhere else in the game. Because of this, we know that c_1 itself is uniformly random. Thus, we can safely rewrite the body of c_1 to be equal to a fresh uniformly sampled r_1. The resulting game is shown in Figure 2.4.

$$\begin{array}{l}\textsf{Game } G_4: \\ \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}, \boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n; \\ b \xleftarrow{\$} \{0,1\}, \boldsymbol{r_0} \xleftarrow{\$} \mathbb{Z}_q^m, r_1 \xleftarrow{\$} \mathbb{Z}_q; \\ \text{let } \boldsymbol{c}_0 = \boldsymbol{r_0}, c_1 = r_1; \\ b' \leftarrow \mathcal{A}(\mathbf{A}, \boldsymbol{u}, \boldsymbol{c}_0, c_1); \end{array}$$

Figure 2.4: Dual-Regev PKE: Game 4

In this final game, there is no dependence between the challenge given to the adversary and the challenge b, so the probability that the adversary guesses b is upper bounded by $\frac{1}{2}$.

The most important point about the above proof is that while the cryptographic

theory underlying the Leftover Hash Lemma and Learning with Errors assumption is in nature analytic, the proof of security which uses them is only algebraic. That is, no complicated analytic arguments must be made in order to carry out the above proof; instead, each transformation is a straightforward syntactic transformation of the security game.

Our logic is designed to handle game transformations such as the ones in the above proof. Our implemented security proof for Dual Regev Encryption is shown in Figure 2.5. In lines 1-3, we apply the Leftover Hash Lemma. The `move` tactic is used to reorder samplings in the security game, as long as the two reorderings are semantically equivalent. The `assumption_decisional` tactic is used to apply hardness assumptions and information-theoretic lemmas. Note that all required factorings of games in this proof are performed automatically, handled by our use of the SAGBI method in Section 2.4.3. This is reflected by the "!" at the end of the tactic, which asks the proof system to automatically factor the game. (More complicated applications of `assumption_decisional` do require the user to provide some hints to the proof system about how to factor the game. These hints are minimal, however.) The arrow `->` after the tactic specifies that we wish to apply the transformation in the forward direction. (It is possible to apply the LHL and the LWE assumption in reverse, as well. This is used in later proofs.) Throughout, we use the `//` tactic to normalize the game. This tactic unfolds let bindings, and applies a syntactic normal form algorithm to all expressions in the game. The `mat_fold` and `mat_unfold` tactics are used to reason about uniformity of matrices of the form $\mathbb{Z}_q^{n\times(m+k)}$: the `mat_unfold` tactic will separate a uniform sampling of type $\mathbb{Z}_q^{n\times(m+k)}$ into two uniform samplings of types $\mathbb{Z}_q^{n\times m}$ and $\mathbb{Z}_q^{n\times k}$ respectively; the `mat_fold` does the corresponding inverse operation.

The `rnd` tactic is used to reason about transformations of uniform samplings: given two functions f, f^{-1} which must be mutual inverses, the `rnd` tactic allows one to "pull" a uniform sampling through f^{-1}. This is used in two ways in the proof: on lines 13 and 15, we use `rnd` to show that instead of sampling a matrix, we may instead sample its transpose. Whenever the original matrix is used, we now take the transpose of the new sampled matrix. Similarly, on line 19 we use `rnd` to perform an optimistic sampling operation, in which `B` is transformed in order to remove the additive factor `b?Mu(()):0_{1,1}`. Here, `Mu` is an uninterpreted function from the unit type to 1 by 1 matrices, modelling the message content $\lceil q/2 \rceil$, and `0_{1,1}` is the constant zero matrix of dimension 1 by 1. The notation `_?_:_` is the standard ternary if-then-else construct; thus, we can model the expression $b\lceil q/2 \rceil$ present in the Dual Regev scheme as the expression `b?Mu(()):0_{1,1}`.

Finally, the `indep!` tactic is used to reason about games such as the game in Figure 2.4, in which the adversary trivially has no advantage. Detail about the proof rules present in our logic is given in Section 2.3.4.

2.3 Logic

Our logic reasons about probabilistic expressions P, built from atomic expressions of the form $\Pr_G[\ \phi\ ]$, where G is a game, and ϕ is an event. Games are probabilistic programs with oracle and adversary calls, and ϕ is the winning condition of the game. The proof rules of the logic formalize common patterns of reasoning from the game-playing approach to security proofs. In their simpler form, proof steps will transform a proof goal $\Pr_G[\ \phi\ ] \leq p$ into a proof goal $\Pr_{G'}[\ \phi'\ ] \leq p'$, with $p = p' + c$, and G' a game derived from G; alternatively, they will directly discharge

```
(* apply LHL *)
move A 1.
assumption_decisional! LHL -> u; //.

(* fold A, u into single matrix Au *)
mat_fold 1 2 Au; //.

(* apply LWE assumption *)
move s 2.
assumption_decisional! LWE -> w; //.

(* unfold LWE distribution *)
rnd w (λ w. tr w) (λ w. tr w); //.
mat_unfold 2 wa wb; //.
rnd wb (λ B. tr B) (λ B. tr B); //.

(* perform optimistic sampling *)
move wb 4.
rnd wb (λ B. B - (b?Mu(()):0_{1,1}))
       (λ B. B + (b?Mu(()):0_{1,1})); //.
indep!.

qed.
```

Figure 2.5: AutoLWE proof for Dual Regev Encryption.

the proof goal $\Pr_G[\,\phi\,] \leq p$ (and give a concrete value for p) when the proof goal is of a simple and specific form, e.g. bounding the probability that an adversary guesses a uniformly distributed and secret value.

In order to be able to accommodate lattice-based constructions, the following novelties are necessary: the expression language includes vectors and matrices; new rules for probabilistic samplings and for oracle-relative assumptions (both in the information-theoretic and computational forms). These extensions do not pose any foundational challenge, but must be handled carefully to obtain the best trade-off between generality and automation.

Dimensions

d	$::=$	n	dimension variable
	$\mid$	$d_1 + d_2$	addition
	$\mid$	1	constant dimension 1

Types

t	$::=$	$\mathbb{B}$	boolean value
	$\mid$	$\mathbb{Z}_q$	prime field of order q
	$\mid$	$\mathbb{Z}_q^{d_1 \times d_2}$	integer matrix
	$\mid$	$\mathsf{list}_d\ t$	list
	$\mid$	$t \times \ldots \times t$	tuple

Expressions

M	$::=$	$\mathbf{0}$	null matrix
	$\mid$	$\mathbf{I}$	identity matrix
	$\mid$	$[M]$	constant list
	$\mid$	$M + M$	addition
	$\mid$	$M \times M$	multiplication
	$\mid$	$-M$	inverse
	$\mid$	$M \parallel M$	concatenation
	$\mid$	$\mathsf{sl}\ M$	left projection
	$\mid$	$\mathsf{sr}\ M$	right projection
	$\mid$	$M^\top$	transpose

Figure 2.6: Syntax of expressions (selected)

2.3.1 Games

Games consist of a security experiment in which an adversary with oracle access interacts with a challenger and of an assertion that determines the winning event.

Expressions The expression language operates over booleans, lists, matrices, and integers modulo q, and includes the usual algebraic operations for integer modulo q and standard operators for manipulating lists and matrices. The operations for matrices include addition, multiplication and transposition, together with *structural operations* that capture the functionalities of block matrices, and can be used for (de)composing matrices from smaller matrices. *concatenation*, *split*

Assertions (event expressions)

ϕ	::=	e	expression
	$\mid$	$\exists\, b_1, \ldots, b_k.\, e$	existential queries
	$\mid$	$\forall\, b_1, \ldots, b_k.\, e$	universal queries
where			
b	::=	$x \in Q_{\mathsf{o}}$	x ranges over queries

for all queries

Game commands

gc	::=	$\mathsf{let}\ x = e$	assignment
	$\mid$	$x \xleftarrow{\$} \mu$	sampling from distr.
	$\mid$	$\mathsf{assert}(\phi)$	assertion
	$\mid$	$y \leftarrow \mathcal{A}(x)\ \mathsf{with}\ \overrightarrow{\mathcal{O}}$	adversary call

Oracle commands

oc	::=	$\mathsf{let}\ x = e$	assignment
	$\mid$	$x \xleftarrow{\$} \mu$	sampling from distr.
	$\mid$	$\mathsf{guard}(b)$	guard

Oracle definitions

$\mathcal{O}$	::=	$\mathsf{o}(x) = \{\overrightarrow{oc};\ \mathsf{return}\ e\}$

Game definitions

G	::=	$\{\overrightarrow{gc};\ \mathsf{return}\ e\}; \overrightarrow{\mathcal{O}}$

where $\mathcal{A}$ and $\mathcal{O}$ range over adversary and oracle names respectively.

Figure 2.7: Syntax of games

left, and *split right*. The type of lists, list_d, denotes a list of length d. Lists are manipulated symbolically, so do not support arbitrary destructuring. Lists may be constructed through the *constant list* operation $[\cdot]$, which takes a type τ to the type $\mathsf{list}_d\ \tau$, for any d. All of the matrix operations are lifted pointwise to lists.

The syntax of expressions (restricted to expressions for matrices) is given in Figure 2.6. Selected typing rules for expressions are given in the Appendix, in Figure 2.13. Expressions are deterministic, and are interpreted as values over their intended types. Specifically, we first interpret dimensions as (positive) natural

numbers. This fixes the interpretation of types. Expressions are then interpreted in the intended way; for instance, transposition is interpreted as matrix transposition, etc.

Games Games are defined by a sequence of commands (random samplings, assignments, adversary calls) and by an assertion. The command defines the computational behavior of the experiment whereas the assertion defines the winning event. Each adversary call contains a list of oracles that are available to the adversary; oracles are also defined by a sequence of commands (random samplings, assignments, assert statements) and by a return expression. The grammars for oracle definitions and game definitions are given in Figure 2.7.

The operational behavior of oracles is defined compositionally from the operational behavior of commands:

- random sampling $x \xleftarrow{\$} \mu$: we sample a value from μ and store the result in the variable x;
- assignments: let $x = e$: we evaluate the expression e and store the result in the variable x;
- assertion $\mathsf{guard}(b)$: we evaluate b and return $\perp$ if the result is false. Guards are typically used in decryption oracles to reject invalid queries.

In addition, we assume that every oracle $\mathcal{O}$ comes with a value $\delta_{\mathcal{O}}$ that fixes the maximal number of times that it can be called by an adversary. To enforce this upper bound, the execution is instrumented with a counter $c_{\mathcal{O}}$ that is initially set to 0. Then, whenever the oracle is called, one checks $c_{\mathcal{O}} \geq \delta_{\mathsf{o}}$; if so, then $\perp$ is returned. Otherwise, the counter $c_{\mathcal{O}}$ is increased, and the oracle body is executed. In order

to interpret events, we further instrument the semantics of the game to record the sequence of interactions between the adversary and the oracle. Specifically, the semantics of oracles is instrumented with a query set variable $Q_{\mathcal{O}}$ that is initially set to $\emptyset$. Then, for every call the query parameters are stored in $Q_{\mathcal{O}}$. (Following [BDKL10] it would be more precise to hold a single list of queries, rather than a list of queries per oracle, but the latter suffices for our purposes.)

Informally, adversaries are probabilistic computations that must execute within a specific amount of resources and are otherwise arbitrary. One simple way to give a semantics to adversaries is through syntax, i.e. by mapping adversary names to commands, and then interpret these commands using the afore described semantics. However, our language of games is too restrictive; therefore, we map adversary names to commands in a more expressive language, and then resort to the semantics of this richer language. For convenience of meta-theoretic proofs, e.g. soundness, it is preferable to choose a language that admits a set-theoretical semantics. For instance, one can use the probabilistic programming language pWhile to model the behavior of the adversaries.

The semantics of games is defined compositionally from the operational behavior of commands, oracles, and adversaries:

- assertion $\mathsf{assert}(\phi)$: we evaluate ϕ and abort if the result is false.
- adversary call $y \leftarrow \mathcal{A}(e)$ with $\overrightarrow{\mathcal{O}}$: we evaluate e, call the adversary $\mathcal{A}$ with the result as input, and bind the output of the adversary to y. The adversary is provided with access to the oracles $\overrightarrow{\mathcal{O}}$.

Finally, the interpretation of $\Pr_G[\ \phi\ ]$ is to be the probability of ϕ in the sub-distribution obtained by executing G.

Throughout the paper, we assume that the games satisfy the following well-formedness conditions and (without loss of generality) hygiene conditions: (WF1) all variables must be used in scope; (WF2) commands must be well-typed; (Hyg1) adversary and oracle names are distinct; (Hyg2) bound variables are distinct.

2.3.2 Reasoning about expressions

Our indistinguishability logic makes use of two main relations between expressions: equality and deducibility. Equality is specified through a set of axioms $\mathcal{E}$, from which further equalities can be derived using standard rules of equational reasoning: reflexivity, symmetry, transitivity of equality, functionality of operators, and finally instantiation of axioms. We write $\Gamma \vdash_{\mathcal{E}} e = e'$ if e and e' are provably equal from the axioms $\mathcal{E}$ and the set of equalities Γ. Throughout the paper, we implicitly assume that the set of axioms includes standard identities on matrices.

Deducibility is defined using the notion of contexts. A context C is an expression that only contains a distinguished variable $\bullet$. We write $e \vdash^{C}_{\mathcal{E}} e'$, where e, e' are expressions and C is a context, if $\vdash_{\mathcal{E}} C[e] = e'$. We write $e \vdash_{\mathcal{E}} e'$ if there exists a context C such that $e \vdash^{C}_{\mathcal{E}} e'$. Similarly, we write $\Gamma \models e \vdash^{C}_{\mathcal{E}} e'$ if $\Gamma \vdash_{\mathcal{E}} C[e] = e'$ and $\Gamma \models e \vdash_{\mathcal{E}} e'$ if there exists a context C such that $\Gamma \models e \vdash_{\mathcal{E}} e'$. More generally, a (general) context C is an expression that only contains distinguished variables $\bullet_1, \ldots, \bullet_n$. We write $e_1, \ldots, e_n \vdash^{C}_{\mathcal{E}} e'$, where $e_1, \ldots, e_n, e'$ are expressions and C is a context, if $\vdash_E C[e_1, \ldots, e_n] = e'$. We write $e_1, \ldots, e_n \vdash_{\mathcal{E}} e'$ if there exists a context C such that $e_1, \ldots, e_n \vdash^{C}_{\mathcal{E}} e'$. Similarly, we write $\Gamma \models e_1, \ldots, e_n \vdash^{C}_{\mathcal{E}} e'$ if $\Gamma \models C[e_1, \ldots, e_n] =_{\mathcal{E}} e'$ and $\Gamma \models e_1, \ldots, e_n \vdash_{\mathcal{E}} e'$ if there exists a context C such that $\Gamma \models e_1, \ldots, e_n \vdash_{\mathcal{E}} e'$. Intuitively, a context is a recipe that shows how some expression may be computed given other expressions. If we consider matrices, we

may have $M + N, O, N \vdash M \times O$ with the context $C(\bullet_1, \bullet_2, \bullet_3) := (\bullet_1 - \bullet_3) \times \bullet 2$.

2.3.3 Strongest postcondition

A desirable property of any logic is that one can replace equals by equals. In particular, it should always be possible to replace an expression e by an expression e' that is provably equivalent to e. However, it is often desirable to use a stronger substitution property which allows to replace e by an expression e' that is provably equivalent to e relative to the context in which the replacement is to be performed. To achieve this goal, our proof system uses a *strongest postcondition* to gather all facts known at a position p in the main command. The computation of $sp_p(G)$ is done as usual, starting from the initial position of the program with the assertion true and adding at each step the assertion ϕ_c corresponding to the current command c, where:

$$
\begin{aligned}
\phi_{\mathsf{let}\ x = e} &= x = e \\
\phi_{\mathsf{guard}(b)} &= b \\
\phi_{\mathsf{assert}(e)} &= e \\
\phi_{\forall/\exists\, b_1, \ldots, b_k .\, e} &= \mathsf{true}
\end{aligned}
$$

2.3.4 Judgment and proof rules

Our computational logic manipulates judgments of the form $P \preceq P'$ where P and P' are probability expressions drawn from the following grammar:

$$P, P' ::= \epsilon \mid c \mid P + P' \mid P - P' \mid c \times P \mid |P| \mid \Pr_G[\ \phi\ ],$$

where ϵ ranges over variables, c ranges over constants, $|P|$ denotes absolute value, and $\Pr_G[\ \phi\ ]$ denotes the success probability of event ϕ in game G. Constants

$$[\textsc{False}]\ \frac{}{\Pr_G[\ \mathsf{false}\ ] \preceq 0} \qquad [\textsc{Case}]\ \frac{\Pr_G[\ \phi \wedge c\ ] \preceq \epsilon_1 \quad \Pr_G[\ \phi \wedge \neg c\ ] \preceq \epsilon_2}{\Pr_G[\ \phi\ ] \preceq \epsilon_1 + \epsilon_2}$$

$$[\textsc{Refl}]\ \frac{}{\Pr_G[\ \phi\ ] \preceq \Pr_G[\ \phi\ ]} \qquad [\textsc{Add}]\ \frac{P \preceq \epsilon_1 \qquad P' \preceq \epsilon_2}{P + P' \preceq \epsilon_1 + \epsilon_2}$$

$$[\textsc{Eq}]\ \frac{P \preceq \epsilon \qquad \vdash P' \leq P}{P' \preceq \epsilon}$$

$$[\textsc{Swap}]\ \frac{\Pr_{G\{c';\, c\}_p}[\ \phi\ ] \preceq \epsilon}{\Pr_{G\{c;\, c'\}_p}[\ \phi\ ] \preceq \epsilon} \qquad [\textsc{Insert}]\ \frac{\Pr_{G\{c;\, c'\}_p}[\ \phi\ ] \preceq \epsilon}{\Pr_{G\{c'\}_p}[\ \phi\ ] \preceq \epsilon}\ \boxed{\begin{array}{l} c \text{ sampling, let,} \\ \text{or } \mathsf{guard(true)} \end{array}}$$

$$[\textsc{Subst}]\ \frac{\Pr_{G\{e\}_p}[\ \phi\ ] \preceq \epsilon}{\Pr_{G\{e'\}_p}[\ \phi\ ] \preceq \epsilon}\ \boxed{sp_p(SE) \models e =_{\mathcal{E}} e'}$$

$$[\textsc{Abstract}]\ \frac{\left|\Pr_{G_1'}[\ \phi_1\ ] - \Pr_{G_2'}[\ \phi_2\ ]\right| \preceq \epsilon}{\left|\Pr_{G_1}[\ \phi_1\ ] - \Pr_{G_2}[\ \phi_2\ ]\right| \preceq \epsilon}\ \boxed{\begin{array}{l} G_1 \equiv G_1'[\mathcal{B}] \\ G_2 \equiv G_2'[\mathcal{B}] \end{array}}$$

$$[\textsc{Rand}]\ \frac{\Pr_{G\{s \xleftarrow{\$} t';\, \mathsf{let}\ r = C[s]\}_p}[\ \phi\ ] \preceq \epsilon}{\Pr_{G\{r \xleftarrow{\$} t\}_p}[\ \phi\ ] \preceq \epsilon}\ \boxed{sp_p(G) \models C'[C] =_{\mathcal{E}} \bullet}$$

$$[\text{RFold}]\ \frac{\Pr_{G\{x \xleftarrow{\$} \mathbb{Z}_q^{d_1 \times (d_2 + d_2')};\, \mathsf{let}\ x_1 = \mathsf{sl}\ x;\, \mathsf{let}\ x_2 = \mathsf{sr}\ x\}_p}[\ \phi\ ] \preceq \epsilon}{\Pr_{G\{x_1 \xleftarrow{\$} \mathbb{Z}_q^{d_1 \times d_2};\, x_2 \xleftarrow{\$} \mathbb{Z}_q^{d_1 \times d_2'}\}_p}[\ \phi\ ] \preceq \epsilon}$$

$$[\text{RUnfold}]\ \frac{\Pr_{G\{x_1 \xleftarrow{\$} \mathbb{Z}_q^{d_1 \times d_2};\, x_2 \xleftarrow{\$} \mathbb{Z}_q^{d_1 \times d_2'};\, \mathsf{let}\ x = x_1 \| x_2\}_p}[\ \phi\ ] \preceq \epsilon}{\Pr_{G\{x \xleftarrow{\$} \mathbb{Z}_q^{d_1 \times (d_2 + d_2')}\}_p}[\ \phi\ ] \preceq \epsilon}$$

$$[\textsc{Upto}]\ \frac{\Pr_{G\{\mathsf{guard}(c)\}_p}[\ \phi\ ] \preceq \epsilon_1 \qquad \Pr_{G\{\mathsf{guard}(c)\}_p}[\ \exists x \in Q_o.\ c(x) \neq c'(x)\ ] \preceq \epsilon_2}{\Pr_{G\{\mathsf{guard}(c')\}_p}[\ \phi\ ] \preceq \epsilon_1 + \epsilon_2}\ \boxed{p \text{ first position in } o}$$

$$[\textsc{Guess}]\ \frac{\Pr_{G;\, x \leftarrow \mathcal{A}()}[\ \phi\ ] \preceq \epsilon}{\Pr_G[\ \exists x \in Q_o.\ \phi\ ] \preceq \epsilon}$$

$$[\textsc{Find}]\ \frac{\Pr_{G;\, x \leftarrow \mathcal{A}(e)}[\ \phi_1 \wedge \phi_2\ ] \preceq \epsilon}{\Pr_G[\ (\exists x \in Q_o.\ \phi_1) \wedge \phi_2\ ] \preceq \epsilon}\ \boxed{\begin{array}{l} C \text{ efficient and} \\ sp_{|G|}(G) \models C[(e, x)] =_{\mathcal{E}} \phi_1 \end{array}}$$

Figure 2.8: Selected proof rules

include concrete values, e.g. 0 and $\frac{1}{2}$, as well as values whose interpretation will depend on the parameters of the scheme and the computational power of the adversary, e.g. its execution time or maximal number of oracle calls.

Proof rules are of the form

$$\frac{P_1 \preceq \epsilon_1 \quad \ldots \quad P_k \preceq \epsilon_k}{P \preceq \epsilon}$$

where P_is and P are probability expressions, ϵ_is are variables and finally ϵ is a probability expression built from variables and constants.

Figure 2.8 present selected rules of the logic. In many cases, rules consider judgments of the form $\Pr_G[\ \phi\ ] \preceq \epsilon$; similar rules exist for judgments of the form $|\Pr_G[\ \phi\ ] - \Pr_{G'}[\ \phi'\ ]| \preceq \epsilon$.

Rules [False] and [Case] formalize elementary axioms of probability theory. Rules [Refl] and [Add] formalize elementary facts about real numbers. Rule [Eq] can be used to replace a probability expression by another probability expression that is provably smaller within the theory of reals. For instance, derivations commonly use the identity $\epsilon_1 \leq |\epsilon_1 - \epsilon_2| + \epsilon_2$.

Rules [Swap], [Insert], [Subst] are used for rewriting games in a semantics-preserving way. Concretely, rule [Swap] swaps successive commands (at position p) that can be reordered (are dataflow independent in the programming language terminology). By chaining applications of the rule, one can achieve more general forms of code motion. Rule [Insert] inserts at position p command that does not carry any operational behaviour. Rule [Subst] substitutes at position p an expression e by another expression e' that is contextually equivalent at p, i.e. $sp_p(G) \models e =_{\mathcal{E}} e'$ holds.

The rule [Rand] performs a different transformation known as optimistic sampling. It replaces a uniform sampling from t by $s \xleftarrow{\$} t'$; $\mathsf{return}\ C[s]$. To ensure that this transformation is correct, the rule checks that C is provably bijective at the program point where the transformation arises, using a candidate inverse context C' provided by the user. Rules [RFold] and [RUnfold] are dual and are used to manipulate random samplings of matrices. The rule [RFold] is used to turn two uniform samplings of matrices into one uniform sampling of the concatenation; conversely, the rule [RUnfold] may be used to turn one uniform sampling of a concatenation into uniform samplings of its component parts. (We also have similar rules [LFold] and [LUnfold] in order to manipulate the vertical component of the dimension.) These rules are primarily used to apply axioms which are stated about matrices of compound dimension.

The rule [Abstract] is used for applying computational assumptions. The rule can be used to instantiate a valid judgment with a concrete adversary. The side-conditions ensure that the experiments G_1 and G_2 are syntactically equivalent to the experiment $G_1'[\mathcal{B} := B]$ and $G_2'[\mathcal{B} := B]$, where the notation $G'[\mathcal{B} := B]$ represents the game obtained by inlining the code of $\mathcal{B}$ in G'. Because of the requirement on syntactic equivalence, it is sometimes necessary to apply multiple program transformations before applying an assumption.

The rule [Upto] rule is used for replacing $\mathsf{guard}(c')$ at position p in an oracle with $\mathsf{guard}(c)$. According to the usual principle for reasoning up to failure events, the rule yields two proof obligations: bound the probability of the original event and the probability that the adversary performs a query where the results of c and c' differ.

The rules [Guess] and [Find] rules are used to deal with winning events in-

volving existential quantification.

The logic also contains a rule for hybrid arguments. The rule is similar to [BGS15].

2.3.5 Soundness

All proof rules of the logic are sound. To state soundness, we lift the interpretation of games to an interpretation of judgments and derivations. This is done by first defining a fixed interpretation of dimensions that is used for all the games of the derivation. Then, we define the interpretation of P inductively. We say that judgment $P \preceq P'$ is *valid* iff the inequality holds for every valid interpretation of P and P'. Finally, one can prove that $P \preceq P'$ is valid whenever $P \preceq P'$ is derivable in the logic.

2.3.6 Axioms Used

Here, we describe the axioms used to prove the schemes in Sections 2.2 and 2.5 secure. Each axiom is *decisional*, in that it is a claim about the closeness of two games. This is modeled by having both games end with a bit output b, so that each axiom is a claim of the form $|\Pr_{G_0}[\, b \,] - \Pr_{G_1}[\, b \,]| \preceq \epsilon$. This allows us to apply the [ABSTRACT] rule from Figure 2.8.

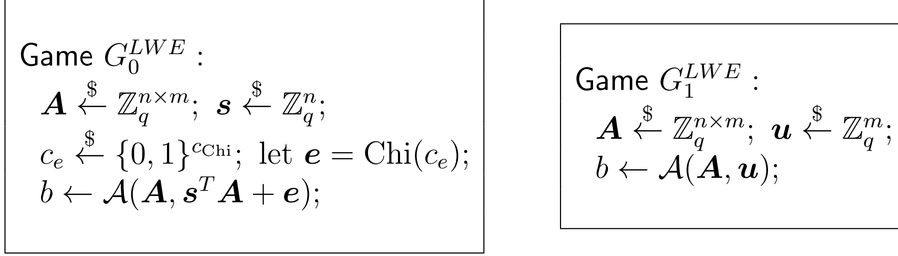

Figure 2.9: The LWE assumption, encoded in AutoLWE.

Learning with Errors

Recall from Section 2.2 that the LWE assumption states that the distribution $(\boldsymbol{A}, \boldsymbol{s}^{\mathsf{T}}\boldsymbol{A} + \boldsymbol{e})$ is indistinguishable from uniform, where $\boldsymbol{A}$ and $\boldsymbol{s}$ are uniformly sampled elements of $\mathbb{Z}_q^{n \times m}$ and $\mathbb{Z}_q^n$ respectively, and $\boldsymbol{e}$ is sampled from some given error distribution.

Our concrete encoding is given in Figure 2.9. Since our logic only deals with uniform samplings, in order to encode more complicated sampling algorithms such as the error distribution for LWE, we separate the sampling algorithm into a *coin sampling* stage and a *deterministic* stage. In the coin sampling stage, an element of $\{0,1\}^c$ is sampled, where c is the number of coins the sampling algorithm will use. (Since the sampling algorithm is polynomial time, c will be a polynomial of the security parameter.) In the deterministic stage, we call an uninterpreted function (here, Chi) which uses the sampled coins to produce the output of the distribution.

In various applications of the LWE assumption, the parameter settings of Figure 2.9 will alter slightly – for instance, in the Dual Regev scheme from Section 2.2, we do not use m on the nose, but rather $m + 1$. This difference is immaterial to the validity of the assumption.

Leftover Hash Lemma

The most subtle part of our proofs is often not applying the LWE assumption, but rather applying the Leftover Hash Lemma. This is because the LHL is an *information-theoretic* judgment rather than a computational one; information-theoretic judgments enjoy stronger composition properties than computational judgments.

Recall that the (basic) LHL states that the distribution $(\boldsymbol{A}, \boldsymbol{AR}, \boldsymbol{wR})$ is statistically close to the distribution $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{wR})$, where $\boldsymbol{A}$ is a uniformly random element of $\mathbb{Z}_q^{n\times m}$, $\boldsymbol{R}$ is a uniformly random element of $\{-1, 1\}^{m\times k}$ (interpreted as a matrix), and $\boldsymbol{w}$ is a fixed arbitrary vector in $\mathbb{Z}_q^m$. For the LHL to hold, however, we can actually relax the requirements on $\boldsymbol{A}$: instead of $\boldsymbol{A}$ being sampled uniformly, we only require that $\boldsymbol{A}$ is sampled from a distribution which is *statistically close* to uniform.

In the literature, it is often the case that the lemma being applied is not the LHL on the nose, but rather this weakened (but still valid) form in which $\boldsymbol{A}$ only need to be close to uniform. In many of our proofs, this occurs because $\boldsymbol{A}$ is not uniformly sampled, but rather sampled using an algorithm, TrapGen, which produces a vector $\boldsymbol{A}$ statistically close to uniform along with a *trapdoor* T_A, which is kepts secret from the adversary.

By combining the LHL with the TrapGen construction, we obtain the security games in Figure 2.10. Both games are displayed at once: the expressions which vary between the two games are annotated with which game they belong in. In order to model how $\boldsymbol{R}$ is sampled, we sample the component bits of $\boldsymbol{R}$ from $\{0, 1\}^{d_{LHL}}$, and apply a symbolic function, bitinj, which converts these component bits into a

matrix. Note in this security game that $\mathbf{w}$ comes from a symbolic adversary, $\mathcal{A}_1$. This models the universal quantification of $\mathbf{w}$ in the LHL. Additionally, note that $\mathcal{A}_2$ actually receives the trapdoor $T_{\mathbf{A}}$. This is counterintuitive, because adversaries in the cryptosystems do not have access to the trapdoor. However, remember that here we are constructing the adversary *for the LHL*; giving $\mathcal{A}_2$ the trapdoor reflects the assertion that the distribution $(\mathbf{A}, \mathbf{AR}, \mathbf{wR}, T_{\mathbf{A}})$ is statistically close to the distribution $(\mathbf{A}, \mathbf{B}, \mathbf{wR}, T_{\mathbf{A}})$, which follows from the information theoretic nature of the LHL.

While we use the assumption from Figure 2.10 in our proofs, we also use several small variations which are also valid. One such variation is in the proof of Dual Regev, where we do not use the TrapGen algorithm, but rather sample $\boldsymbol{A}$ uniformly (and do not give the adversary $T_{\boldsymbol{A}}$); additionally, we do not include this linear leakage $\boldsymbol{w}$. Another such variation is used in our CCA proof from Section 2.5. In this instance, we do not transform $\boldsymbol{AR}$ to $\boldsymbol{B}$, but rather to $\boldsymbol{AR} + \boldsymbol{B}$ (thus generalizing our [RAND] rule.) Additionally, we must state the LHL in the CCA proof to be relative to the decryption oracle, which makes use of $\boldsymbol{R}$. This relativized lemma is still valid, however, since the decryption oracle does not leak any information about $\boldsymbol{R}$. It will be interesting future work in order to unify these small variations of the LHL.

Distribution Equivalences

In addition to the two main axioms above, we also rely on several opaque probabilistic judgments about distributions from which the adversary may sample, but are written in terms of private variables which the adversary may not access. For instance, in an Identity-Based Encryption scheme, the adversary could have access

$$
\begin{array}{l}
\textsf{Game } G_\beta^{LHL}: \\
\quad c \xleftarrow{\$} \{0,1\}^{d_{TG}};\ \text{let } (\mathbf{A}, T_{\mathbf{A}}) = \text{TrapGen}(c); \\
\quad r \xleftarrow{\$} \{0,1\}^{d_{LHL}};\ \text{let } R = \text{bitinj}(r); \\
\quad \overline{\phantom{\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}}}^{\text{if } \beta=1} \\
\quad \mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{n\times m};\ \mathbf{w} \leftarrow \mathcal{A}_1(); \\
\quad b \leftarrow \mathcal{A}_2(\mathbf{A}, \overline{\mathbf{AR}}^{\text{if } \beta=0}\ \ \overline{\mathbf{B}}^{\text{if } \beta=1}, \mathbf{wR}, T_{\mathbf{A}}, \mathbf{w});
\end{array}
$$

Figure 2.10: The LHL assumption combined with TrapGen, encoded in AutoLWE.

to a KeyGen oracle, which must use the master secret key in order to operate. This is the case in Section 2.5.2. In the concrete proof, there is a step in which we change the implementation of the KeyGen oracle from one uninterpreted function to another. Transformations of this sort are encoded using oracle-relative assumptions, which are generalizations of axioms in AutoG&P which allow adversaries to query oracles.

For example, in Figure 2.11, we state closeness of the distributions $D_0(\boldsymbol{s_0}, \cdot)$ and $D_1(\boldsymbol{s_1}, \cdot)$, where both $\boldsymbol{s_0}$ and $\boldsymbol{s_1}$ are unknown to the adversary. (As before, each distribution is separated into a coin sampling stage and a deterministic stage.) Note that $\boldsymbol{s_0}$ and $\boldsymbol{s_1}$ need not be of the same type, since the adversary does not see them. Jumping ahead in (H)IBE part in the case study, D_0, D_1 correspond to the real/simulated key generation algorithms, where s_0 is the master secret key, and s_1 is the secret trapdoor information the simulator knows in order to answer secret key queries.

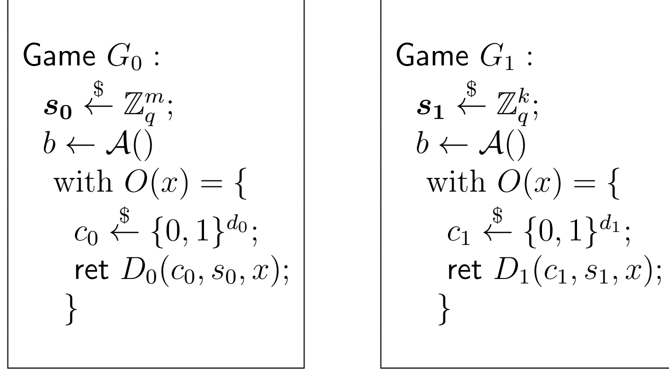

Figure 2.11: Example axiom capturing computational closeness of distributions.

2.4 Deciding deducibility

Several rules involve deducibility problems as side-conditions. For instance, in the [Abstract] rule from Fig 2.8, we may transform a bound involving G_1 and G_2 into a bound involving G_1' and G_2', if there exists a common subgame $\mathcal{B}$ which can be used to factor the former pair into the latter. Finding this subgame $\mathcal{B}$ will induce deducibility subproblems. In order to automate the application of the rules, it is thus necessary to provide algorithms for checking whether deducibility problems are valid. As previously argued, it is desirable whenever possible that these algorithms are based on decision procedures rather than heuristics.

In this section, we provide decision procedures for the theory of Diffie-Hellman exponentiation, both in its basic form and in its extension to bilinear groups, and for the theory of fields. The decision procedures for Diffie-Hellman exponentiation are based on techniques from Gröbner bases. In addition to being an important independent contribution on its own, the algorithms for Diffie-Hellman exponentiation also serve as a natural intermediate objective towards addressing the theory of matrices (although the problems are formally independent). For the latter, we require significantly more advanced algebraic tools. For the clarity of exposition,

we proceed incrementally. Concretely, we start by considering the case of fields and non-commutative rings. We respectively provide a decision procedure and a semi-decision procedure. Subsequently, we give a reduction from deducibility for matrices to deducibility for non-commutative rings. The reduction yields a semi-decision procedure for matrices. The algorithms for non-commutative rings and matrices are based on so-called SAGBI [RS90] (Subalgebra Analog to Gröbner Basis for Ideals) techniques, which as justified below provide a counterpart of Gröbner basis computations for subalgebras.

2.4.1 Diffie-Hellman exponentiation

Diffie-Hellman exponentiation is a standard theory that is used for analyzing key-exchange protocols based on group assumptions. It is also used, in its bilinear and multilinear version, in AutoG&P for proving security of pairing-based cryptography. In this setting, the adversary (also often called attacker in the symbolic setting) can multiply groups elements between them, i.e perform addition in the field, and can elevate a group element to some power he can deduce in the field. Previous work only provides partial solutions: for instance, Chevalier et al [CKRT03] only consider products in the exponents, whereas Dougherty and Guttman [DG14] only consider polynomials with maximum degree of 1 (linear expressions).

The standard form of deducibility problems that arises in this context is defined as follows: let Y be a set of names sampled in $\mathbb{Z}_q$, g some group generator, $\mathcal{E}$ the equational theory capturing field and groups operations, some set $X \subset Y$, $f_1, ...f_k, h \in \mathbb{K}[Y]$ be a set of polynomials over the names, and Γ be a coherent set

of axioms. The deducibility problem is then:

$$\Gamma \models X, g^{f_1}, ..., g^{f_k} \vdash_{\mathcal{E}} g^h$$

Proposition 2. *Deducibility for Diffie-Hellman exponentiation is decidable.*

The algorithm that supports the proof of the proposition proceeds by reducing an input deducibility problem to an equivalent membership problem of the saturation of some $\mathbb{Z}_q[X]$-module in $\mathbb{Z}_q[Y]$, and by using an extension for modules [Eis13] of Buchberger's algorithm [Buc76] to solve the membership problem.

The reduction to the membership problem proceeds as follows: first, we reduce deducibility to solving a system of polynomial equations. We then use the notion of saturation for submodules and prove that solving the system of polynomial equations corresponding to the deducibility problem is equivalent to checking whether the polynomial h is a member of the saturation of some submodule M. The latter problem can be checked using Gröbner basis computations.

2.4.2 Fields and non-commutative rings

Another problem of interest is when we consider deducibility inside the field rather than the group. The deducibility problem can then be defined as follows: let Y be a set of names sampled in $\mathbb{Z}_q$, $\mathcal{E}$ the equational theory capturing field operations, $f_1, ...f_k, h \in \mathbb{K}[Y]$ be a set of polynomials over the names, and Γ be a coherent set of axioms. The deducibility problem is then:

$$f_1, ..., f_k \vdash_{\mathcal{E}} h$$

We emphasize that this problem is in fact not an instance of the problem for Diffie-Hellman exponentiation. In the previous problem, if we look at field elements, the

adversary could compute any polynomial in $K[X]$ but he may now compute any polynomial in $K[f_1, ..., f_k]$, the subalgebra generated by the known polynomials.

Decidability is obtained thanks to [SS88], where they solve the subalgebra membership problem using methods based on classical Gröbner basis.

Proposition 3. *Deducibility for fields is decidable.*

If we wish to characterize the full adversary knowledge as done for Diffie-Hellman exponentiation using Gröbner basis, we would have to resort to so-called SAGBI [RS90] (Subalgebra Analog to Gröbner Basis for Ideals) techniques, which form the counterpart of Gröbner basis computations. However, some finitely generated subalgebras are known to have infinite SAGBI bases [RS90], thus it can only provide semi-decision for the membership problem.

For the case of non-commutative rings, we are not aware of any counterpart to [SS88], we resort to the non-commutative SAGBI [Nor98] theory.

Proposition 4. *Deducibility for non-commutative rings is semi-decidable.*

It is an open problem whether one can give a decision procedure for non-commutative rings. We note that the problem of module membership over a non-commutative algebra is undecidable [Mor94], as there is a reduction from the word problem over a finitely presented group. On the other hand, the problem is known to be decidable for some classes of subalgebras, notably in the the homogeneous case where all monomials are of the same degree.

2.4.3 Matrices

The case of matrices introduces a final difficulty: expressions may involve structural operations. To address the issue, we show that every deducibility problem in the theory of matrices is provably equivalent to a deducibility problem that does not involve structural operations, nor transposition—said otherwise, a deducibility problem in the theory of non-commutative rings.

Proposition 5. *Deducibility for matrices is semi-decidable.*

The algorithm that supports the proof of semi-decidability for matrices operates in two steps:

1. it reduces the deducibility problem for matrices to an equivalent deducibility problem for non-commutative rings;
2. it applies the semi-decision procedure for non-commutative rings.

The reduction to non-commutative rings is based on a generalization of the techniques introduced in [BDK+10] for the theory of bitstrings—note that the techniques were used for a slightly different purpose, i.e. deciding equivalence between probabilistic expressions, rather than for proving deducibility constraints.

The general idea for eliminating concatenation and splitting comes from two basic facts:

- $\mathcal{M} \vdash M \parallel N \Leftrightarrow \mathcal{M} \vdash M \wedge \mathcal{M} \vdash N$
- $\mathcal{M} \cup \{M \parallel N\} \vdash T \Leftrightarrow \mathcal{M} \cup \{M, N\} \vdash T$

For transposition, we observe that it commutes with the other operations, so in a proof of deducibility, we can push the transposition applications to the leaves. Everything that can be deduced from a set of matrices $\mathcal{M}$ and the transpose operation can also be deduced if instead of the transpose operation we simply provide the transposition of the matrices in $\mathcal{M}$.

2.5 Implementations and Case Studies

The implementation of our logic, called AutoLWE, is available at:

`https://github.com/autolwe/autolwe`

AutoLWE is implemented as a branch of AutoG&P and thus makes considerable use of its infrastructure.

Moreover, we have used AutoLWE to carry several case studies (see Figure 2.12): an Identity-Based Encryption scheme and an Hierarchical Identity-Based Encryption scheme by Agrawal, Boneh and Boyen [ABB10], a Chosen-Ciphertext Encryption scheme from Micciancio and Peikert [MP12], and an Inner Product Encryption scheme and proof from Agrawal, Freeman, and Vaikuntanathan [AFV11]. These examples are treated in Sections 2.5.2, 2.5.4, 2.5.3 and 2.5.5 respectively.

Globally, our tool performs well, on the following accounts: formal proofs remains close to the pen and paper proofs; verification time is fast (less than 3 seconds), and in particular the complexity of the (semi-)decision procedures is not an issue; formalization time is moderate (requiring at most several hours of programmer effort per proof). One of the main hurdles is the Leftover Hash Lemma, which

Case study			Proof
Reference	Scheme	Property	LoC
Gentry et al. '08 [GPV08]	dual-Regev PKE	IND-CPA	11
Micciancio et al. '12 [MP12]	MP-PKE	IND-CCA	98
Agrawal et al. '10 [ABB10]	ABB-IBE	IND-sID-CPA	56
Agrawal et al. '10 [ABB10]	ABB-HIBE	IND-sID-CPA	77
Agrawal et al. '11 [AFV11]	AFV-IPE	IND-wAH-CPA	106

Figure 2.12: Overview of case studies. All proofs took less than three seconds to complete.

must be applied in varying levels of sophistication. The Leftover Hash Lemma (and more generally all oracle-relative assumptions) increase the difficulty of guessing (chained) applications of assumptions, and consequently limits automation.

2.5.1 Implementation

Security games are written in a syntax closely resembling that shown in Figure 2.1. See Figure 2.5 for an example concrete proof in our system. Each line of the proof corresponds to a proof rule in our logic, as seen in Figure 2.8. All tactic applications are fully automated, except for the application of oracle-relative assumptions. The user must provide some hints to AutoLWE about how the security game needs to be factored in order to apply an oracle-relative assumption. The system in [BGS15] additionally supports a proof search tactic which automatically finds a series of tactics to apply to finish the goal; we do not have a version of that in our setting.

Oracle-relative Assumptions

AutoG&P allows one to add user defined axioms, both to express decisional assertions (two distributions are computationally close) and computational assertions

(a certain event has small chance of happening). In AutoG&P, these user-defined axioms are stated in terms of symbolic adversaries, which are related to the main security game by rules such as [Abstract] in Section 2.3.4. However, the symbolic adversaries present in axioms may not have oracles attached to them. While these restricted adversaries can be used to define the LWE assumption, they are not expressive enough to state the oracle-relative axioms we use throughout our proofs. In AutoLWE, we remove this restriction. An example axiom we now support which we did not before is that in Figure 2.11.

Recall that in order to apply a user defined axiom using [Abstract], we must factor the security game into one which is in terms of the axiom's game. This is done essentially by separating the security game into sections, where each section either reflects the setup code for the axiom, or an instantiation of one of the adversaries in the axiom. We still do this factoring in the case of oracle-relative axioms, but we must also factor oracles in the security game in terms of oracles in the axiom. Once this second step of factoring is done, oracles in the axiom can be compared syntactically to factored oracles in the security game.

Theory of Lists and Matrices

Note that in our case studies, we manipulate both matrices and *lists* of matrices (often simultaneously). Thus, both our normal form algorithm and our deducibility reduction from Section 2.4.3 must be lifted to apply to lists of matrices as well. This is what allows our system to reason about the more complicated HIBE scheme in a manner similar to the IBE scheme, which does not use lists.

In order to do this, we do not implement our main algorithms on expressions of matrices directly, but instead over a general *signature* of matrices, encoded as

a certain type of an ML module. We then instantiate this signature both with matrices and lists of matrices. By doing so, we receive an implementation for our new algorithms which operate uniformly across these two types of expressions.

Deduction algorithms

Many implementations of Gröbner basis computations can be found online, but all of them are only usable for polynomial ideals. In order to handle module and non-commutative subalgebra, we thus implemented generic versions of the Buchberger algorithm for $K[X]$-module and the SAGBI algorithm and plugged them into AutoLWE. The algorithms performed well: we could prove all the LWE examples, and the pairing-based examples very quickly, using the SAGBI methods. The efficiency of the computations contrasts with the complexity of the algorithms, which is high because the saturation squares up the number of inputs terms and the Gröbner Basis can be at worst a double exponential. However, we are dealing with relatively small instances of our problem that are extracted from concrete primitives.

2.5.2 Identity-Based Encryption

Mathematical background. Let Λ be a discrete subset of $\mathbb{Z}^m$. For any vector $\boldsymbol{c} \in \mathbb{R}^m$, and any positive parameter $\sigma \in \mathbb{R}$, let $\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x}) = \exp(-\pi||\boldsymbol{x} - \boldsymbol{c}||^2/\sigma^2)$ be the Gaussian function on $\mathbb{R}^m$ with center $\boldsymbol{c}$ and parameter σ. Next, we let $\rho_{\sigma,\boldsymbol{c}}(\Lambda) = \sum_{\boldsymbol{x}\in\Lambda} \rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x})$ be the discrete integral of $\rho_{\sigma,\boldsymbol{x}}$ over Λ, and let $\chi_{\Lambda,\sigma,\boldsymbol{c}}(\boldsymbol{y}) := \frac{\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{y})}{\rho_{\sigma,\boldsymbol{c}}(\Lambda)}$. Let S^m denote the set of vectors in $\mathbb{R}^m$ whose length is 1. The norm of a matrix $\mathbf{R} \in \mathbb{R}^{m\times m}$ is defined to be $\mathsf{sup}_{\boldsymbol{x}\in S^m}||\mathbf{R}\boldsymbol{x}||$. We say a square matrix is full

rank if all rows and columns are linearly independent.

Identity-based encryption is a generalization of public key encryption. In IBE, the secret key and ciphertext are associated with different identity strings, and decryption succeeds if and only if the two identity strings are equivalent. The security model, IND-sID-CPA, requires adversary to declare challenge identity upfront before seeing the public parameters, and allows adversary to ask for secret key for any identity except for the challenge identity, and CPA security holds for ciphertext associated with the challenge identity.

The IBE scheme our system supports is constructed by Agrawal et al. [ABB10]. The scheme operates as follows:

- Matrix $\mathbf{A}$ is generated by algorithm $\mathsf{TrapGen}$, which outputs a random $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and a small norm matrix $\mathbf{T} \in Z_q^{m\times m}$ such that $\mathbf{A} \cdot \mathbf{T_A} = 0$. Matrices $\mathbf{A}_1, \mathbf{B}$ are sampled randomly from $\mathbb{Z}_q^{n\times m}$, and $\boldsymbol{u}$ is sampled randomly from $\mathbb{Z}_q^n$. Set $\mathsf{pp} = (\mathbf{A}, \mathbf{A}_1, \mathbf{B}, \boldsymbol{u})$ and $\mathsf{msk} = \mathbf{T_A}$.
- To encrypt a message $\mu \in \{0, 1\}$ with identity $\mathsf{id} \in \mathbb{Z}_q^n$, one generates a uniform $\boldsymbol{s} \in \mathbb{Z}_q^n$, error vector $\boldsymbol{e}_0 \leftarrow \chi^m$ and error integer $e_1 \leftarrow \chi$ from discrete Gaussian, a random $\mathbf{R} \in \{0, 1\}^{m\times m}$, and computes ciphertext

  $\mathsf{ct} = \boldsymbol{s}^{\mathsf{T}} [\mathbf{A}||\mathbf{A}_1 + M(\mathsf{id})\mathbf{B}||\boldsymbol{u}] + (\boldsymbol{e}^{\mathsf{T}}||\boldsymbol{e}^{\mathsf{T}}\mathbf{R}||e') + (0||0||\lceil q/2\rceil\mu).$
- The secret key for identity $\mathsf{id} \in \mathbb{Z}_q^n$ is generated by procedure $\boldsymbol{r} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{A}_1 + M(\mathsf{id})\mathbf{B}, \mathbf{T_A}, \boldsymbol{u})$, where we have $\boldsymbol{r}$ is statistically close to χ^{2m}, and $[\mathbf{A}||\mathbf{A}_1 + M(\mathsf{id})\mathbf{B}]\, \boldsymbol{r} = \boldsymbol{u}$.

The idea of the proof is first to rewrite $\mathbf{A}_1$ as $\mathbf{AR} - M(\mathsf{id}^*)\mathbf{B}$, where id^* is the adversary's committed identity. If we do so, we then obtain that the challenge ciphertext is of the form

$$\mathbf{s}^{\mathsf{T}}\left[\mathbf{A}||\mathbf{AR}||\mathbf{u}\right] + (\mathbf{e}^{\mathsf{T}}||\mathbf{e}^{T}\mathbf{R}||e') + (0||0||\lceil q/2 \rceil \mu)$$

where $\mathbf{A}$ comes from TrapGen. We then apply a computational lemma about SampleLeft, in order to rewrite the KeyGen oracle to be in terms of another probabilistic algorithm, SampleRight. This is a statement about equivalence of distributions from which the adversary may sample, so must be handled using an oracle-relative assumption. This is done as described in Section 2.3.6. The computational lemma states that, for appropriately sampled matrices,

$$\text{SampleLeft}(\mathbf{A}, \mathbf{AR} + \mathbf{B}, T_{\boldsymbol{A}}, \boldsymbol{u}) \approx \text{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, T_{\boldsymbol{B}}, \boldsymbol{u}),$$

where $\mathbf{A}$ is sampled from TrapGen in the first and uniform in the second, and $\mathbf{B}$ is sampled uniformly in the first and from TrapGen in the second. By applying this transformation to our KeyGen oracle, we transform our matrix $\mathbf{A}$ from one sampled from TrapGen to uniform. Now that $\mathbf{A}$ is uniform, we finish the proof by noticing that our challenge ciphertext is equal to $\mathbf{b}||\mathbf{bR}||b + \lceil q/2 \rceil \mu$, where $(\mathbf{b}, b)$ forms an LWE distribution of dimension $n \times m + 1$. Thus we may randomize b to uniform, and apply the `rnd` tactic to erase μ from the ciphertext.

The main point of interest in this proof is the initial rewrite $\mathbf{A}_1 \to \mathbf{AR} - M(\mathsf{id}^*)\mathbf{B}$. Given that $\mathbf{A}_1$ is uniform, we may first apply optimistic sampling to rewrite $\mathbf{A}_1$ to $\mathbf{A}_2 - M(\mathsf{id}^*)\mathbf{B}$, where $\mathbf{A}_2$ is uniformly sampled. Thus, we now only need to perform the rewrite $\mathbf{A}_2 \to \mathbf{AR}$. This rewrite is not at all trivial, because $\mathbf{A}$ at this point in the proof comes from TrapGen. However, as noted in Section 2.3.6, it is sound to apply the LHL in this case, because TrapGen generates matrices which are close to uniform in distribution. Thus, we can use the LHL as encoded in Figure 2.10.

2.5.3 CCA1-PKE

The CCA1-PKE scheme we study is proposed by Micciancio and Peikert [MP12]. In comparison with the CPA-PKE scheme [GPV08] described in Section 2.2, the security model of CCA1-PKE is stronger: the adversary can query a decryption oracle for any ciphertext he desires before receiving the challenge ciphertext. The scheme operates as follows:

- Matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is sampled randomly and $\mathbf{R} \leftarrow \{-1, 1\}^{m \times m}$. Set $\mathsf{pk} = (\mathbf{A}, \mathbf{AR})$ and $\mathsf{sk} = \mathbf{R}$.
- Let $M : \mathbb{Z}_q^n \to \mathbb{Z}_q^{n \times m}$ be an embedding from $\mathbb{Z}_q^n$ to matrices, such that for distinct $\mathbf{u}$ and $\mathbf{v}$, $M(\mathbf{u}) - M(\mathbf{v})$ is full rank. To encrypt a message $\mu \in \{0, 1\}$, one generates a uniform $\mathbf{s} \in \mathbb{Z}_q^n$, a uniform $\mathbf{u} \in \mathbb{Z}_q^n$, a uniform matrix $R' \in \{-1, 1\}^{m \times m}$ and an error vector $\mathbf{e} \in \mathbb{Z}_q^m$ sampled from a discrete Gaussian, and computes the ciphertext

  $\mathbf{c}_0 = \mathbf{u}, \mathbf{c}_1 = \mathbf{s}^\mathsf{T} \mathbf{A_u} + (\mathbf{e}^\mathsf{T} || \mathbf{e}^\mathsf{T} * R') + (0 || \mathsf{Encode}(\mu)),$

  where $\mathbf{A_u} := [\mathbf{A} || -\mathbf{AR} + M(\mathbf{u})\mathbf{G}]$, $\mathbf{G}$ is a publicly known gadget matrix, and $\mathsf{Encode} : \{0, 1\} \to \mathbb{Z}_q^m$ sends μ to $\mu \lceil q/2 \rceil (1, \ldots, 1)$.
- To decrypt a ciphertext $(\mathbf{u} := \mathbf{c}_0, \mathbf{c}_1)$ with $\mathsf{sk} = \mathbf{R}$ and $\mathbf{u} \neq 0$, one computes $\mathbf{A_u}$ and calls a procedure $\mathsf{Invert}(\mathbf{A_u}, \mathbf{R}, \mathbf{c}_1)$, which will output $\mathbf{s}$ and $\mathbf{e}$ such that $\mathbf{c}_1 = \mathbf{s}^T \mathbf{A_u} + \mathbf{e}$, where $\mathbf{e}$ has small norm. By doing a particular rounding procedure using $\mathbf{c}_1, \mathbf{s}, \mathbf{e}$, and $\mathbf{R}$, the message bit μ can be derived.

The main subtlety of the proof is that the secret key $\mathbf{R}$ is used in the decryption oracle. Because of this, we must apply the Leftover Hash Lemma relative to this oracle, by using oracle-relative axioms. As we will see, not all uses of the LHL are

valid in this new setting; care must be taken to ensure that the axioms derived from the LHL are still cryptographically sound.

The high-level outline of the proof is as follows: first, we note that instead of using a fresh $\mathbf{R}'$ to encrypt, we can actually use the secret key $\mathbf{R}$. This is justified by the following corollary of the Leftover Hash Lemma: the distribution $(\mathbf{A}, \mathbf{AR}, \boldsymbol{e}, \boldsymbol{e}\mathbf{R}')$ is statistically close to the distribution $(\mathbf{A}, \mathbf{AR}, \boldsymbol{e}, \boldsymbol{e}\mathbf{R})$ where $\mathbf{A}, \mathbf{R}, \mathbf{R}'$, and $\boldsymbol{e}$ are sampled as in the scheme. This corollary additionally holds true relative to the decryption oracle, which makes use of $\mathbf{R}$.

Once we use $\mathbf{R}$ to encrypt instead of $\mathbf{R}'$, we again use the Leftover Hash Lemma to transform $\mathbf{AR}$ into $-\mathbf{AR} + M(\boldsymbol{u})G$, where $\boldsymbol{u}$ is generated from the challenge encryption. Again, this invocation of the Leftover Hash Lemma is stated relative to the decryption oracle. Crucially, note here that we do *not* transform $\mathbf{AR}$ directly into uniform, as we did before: the reason being is that this transformation would actually be *unsound*, because it would decouple the public key from $\mathbf{R}$ as it appears in the decryption oracle. Thus, we must do the transformation $\mathbf{AR} \to -\mathbf{AR} + M(\boldsymbol{u})G$ in one step, which is cryptographically sound relative to the decryption oracle. (Currently, we must write this specialized transformation as a unique variant of the Leftover Hash Lemma, as discussed in Section 2.3.6; future work will involve unifying these separate variants.)

At this point, we may apply the LWE assumption along with a more routine invocation of the LHL in order to erase the message content from the challenge ciphertext, which finishes the proof.

2.5.4 Hierarchical Identity-Based Encryption

Hierarchical IBE is an extension of IBE. In HIBE, the secret key for ID string id can delegate secret keys for ID strings id', where id is a prefix for id. Moreover, decryption succeeds if the ID string for the secret key is a prefix of (or equal to) the ID string for the ciphertext. The security model can be adapted according to the delegation functionality.

The HIBE construction our system supports is described in [ABB10]. The ID space for HIBE is $\mathsf{id}_i \in (\mathbb{Z}_q^n)^d$. The secret key for ID string $\mathsf{id} = (\mathsf{id}_1, \ldots, \mathsf{id}_\ell)$, where $\mathsf{id}_i \in \mathbb{Z}_q^n$, is a small-norm matrix $\mathbf{T}$, such that $\mathbf{F}_{\mathsf{id}}\mathbf{T} = 0$, and $\mathbf{F}_{\mathsf{id}} = [\mathbf{A}_0||\mathbf{A}_1 + M(\mathsf{id}_1)\mathbf{B}||\cdots||\mathbf{A}_\ell + M(\mathsf{id}_\ell)\mathbf{B}]$. We note that $\mathbf{T}$ can be computed as long as we know the secret key for id', where id' is a prefix of id. Ciphertext for ID string id can be generated similarly with respect to matrix $\mathbf{F}_{\mathsf{id}}$.

The security proof of HIBE is similar to the counterpart of IBE. The challenge ID string $\mathsf{id}^* = (\mathsf{id}_1^*, \ldots, \mathsf{id}_\ell^*)$ is embedded in pp as

$$\forall i \in [\ell], \mathbf{A}_i = \mathbf{A}\mathbf{R}_i - M(\mathsf{id}_i^*)\mathbf{B},\ \forall \ell < j \leq d, \mathbf{A}_j = \mathbf{A}\mathbf{R}_j$$

For admissible query $\mathsf{id} = (\mathsf{id}_1, \ldots, \mathsf{id}_k)$, where id is not a prefix of id^*, we have

$$\mathbf{B}_k = [(M(\mathsf{id}_1) - M(\mathsf{id}_1^*))\mathbf{B}||\cdots||(M(\mathsf{id}_k) - M(\mathsf{id}_k^*))\mathbf{B}] \neq 0$$

Then we can generate secret key for id using information $\mathbf{B}_k$ and $\mathbf{R}_k = (\mathbf{R}_1||\cdots||\mathbf{R}_k)$. In previous cases, we manipulate and apply rewriting rules to matrices. However, in order to reason about the security in a similar manner to pen-and-paper proof, we introduce the *list* notation, and adapt our implementation to operate uniformly across these two types of expressions.

2.5.5 Inner Product Encryption

The IPE scheme our scheme supports is described in [AFV11]. We briefly recall their construction as

- Matrix $\mathbf{A}$ is generated by algorithm $\mathsf{TrapGen}$. Matrices $\{matB_i\}_{i\in[d]}$ are sampled randomly from $\mathbb{Z}_q^{n\times m}$, and random vector $\boldsymbol{u}$ is from $\mathbb{Z}_q^n$. The public parameters $\mathsf{pp} = (\mathbf{A}, \{\mathbf{B}_i\}_{i\in[d]}, \boldsymbol{u})$, and $\mathsf{msk} = \mathbf{T}_{\mathbf{A}}$.
- Secret key $\mathsf{sk}_{\boldsymbol{v}} = \boldsymbol{r}$ for vector $\boldsymbol{v} \in \mathbb{Z}_q^d$ is computed by algorithm $\boldsymbol{r} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \sum_{i\in[d]} \mathbf{B}_i\mathbf{G}^{-1}(v_i\mathbf{G}), \mathbf{T}_{\mathbf{A}}, \boldsymbol{u})$, where for operation $\mathbf{G}^{-1}(\cdot) : \mathbb{Z}_q^{n\times m} \to \mathbb{Z}_q^{m\times m}$, for any $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, it holds that $\mathbf{G}\cdot\mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A}$ and $\mathbf{G}^{-1}(\mathbf{A})$ has small norm.
- To encrypt a message $\mu \in \{0,1\}$ for attribute $\boldsymbol{w}$, one generates a uniform $\boldsymbol{s} \in \mathbb{Z}_q^n$, error vector $\boldsymbol{e}_0 \leftarrow \chi^m$ and error integer $e_1 \leftarrow \chi$ from discrete Gaussian, random matrices $\{\mathbf{R}_i\}_{i\in[d]} \in \{0,1\}^{m\times m}$, and computes ciphertext $(\boldsymbol{c}_0, \{\boldsymbol{c}_i\}_{i\in[d]}, c)$ as

  $\boldsymbol{c}_0 = \boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{e}_0^\mathsf{T},\ \boldsymbol{c}_i = \boldsymbol{s}^\mathsf{T}(\mathbf{B}_i + w_i\mathbf{G}) + \boldsymbol{e}_0^\mathsf{T}\mathbf{R}, c = \boldsymbol{s}^\mathsf{T}\boldsymbol{u} + e + \lceil q/2 \rceil\mu$

The main challenge in the proof is to answer secret key queries for any vector $\boldsymbol{v}$ as long as $\langle \boldsymbol{v}, \boldsymbol{w}_0\rangle, \langle \boldsymbol{v}, \boldsymbol{w}_1\rangle$ are both not 0, where $(\boldsymbol{w}_0, \boldsymbol{w}_1)$ is declared by adversary upfront. The attribute $\boldsymbol{w}_b$ (b is a random bit) is first embedded in pp, i.e. $\mathbf{B}_i = \mathbf{A}\mathbf{R}_i - w_{bi}\mathbf{G}, \forall i \in [d]$, where $\mathbf{R}_i$ is a small matrix. By unfolding the matrix for query $\boldsymbol{v}$, we have

$$\left[\mathbf{A}|| \sum_{i\in[d]} \mathbf{B}_i\mathbf{G}^{-1}(v_i\mathbf{G})\right] = \left[\mathbf{A}||\mathbf{A}\sum_{i\in[d]} \mathbf{R}_i\mathbf{G}^{-1}(v_i\mathbf{G}) + \langle \boldsymbol{w}_b, \boldsymbol{v}\rangle\mathbf{G}\right]$$

If $\langle \boldsymbol{w}_b, \boldsymbol{v}\rangle \neq 0$, the algorithm $\mathsf{SampleRight}$ can be used to generate secret key for $\boldsymbol{v}$.

The sequence of hybrids generated in symbolic proof is a bit different from the pen-and-paper proof. In particular, instead of transforming from embedding of challenge attribute $\boldsymbol{w}_0$ directly to embedding of $\boldsymbol{w}_1$, we use the original scheme as a middle game, i.e. from embedding of $\boldsymbol{w}_0$ to original scheme, then to embedding of $\boldsymbol{w}_1$. The reason for using the original scheme again in the proof is that when using LHL to argue the indistinguishability between $(\mathbf{A}, \{\mathbf{B}_i = \mathbf{A}\mathbf{R}_i - w_{0i}\mathbf{G}\}_i)$ and $(\mathbf{A}, \{\mathbf{B}_i = \mathbf{A}\mathbf{R}_i - w_{1i}\mathbf{G}\}_i)$, the real public parameters $(\mathbf{A}, \{\mathbf{B}_i\}_i)$ actually serves as a middleman. Therefore, to ensure the consistency with respect to public parameters and secret key queries, the real scheme is used to make the transformation valid.

2.6 Related work

For space reasons, we primarily focus on related works whose main purpose is to automate security proofs in the computational model.

Corin and den Hartog [CdH06] show chosen plaintext security of ElGamal using a variant of a general purpose probabilistic Hoare logic. In a related spirit, Courant, Daubignard, Ene, Lafourcade and Lakhnech [CDE+08b] propose a variant of Hoare logic that is specialized for proving chosen plaintext security of padding-based encryption, i.e. public-key encryption schemes based on one-way trapdoor permutations (such as RSA) and random oracles. Later, Gagné, Lafourcade, Lakhnech and Safavi-Naini [GLLS09, GLL13] adapt these methods to symmetric encryption modes and message authentication codes.

Malozemoff, Katz and Green [MKG14] and Hoang, Katz and Malozemoff [HKM15] pursue an alternative approach for proving security of modes of operations and authenticated encryption schemes. Their approach relies on a sim-

ple but effective type system that tracks whether values are uniform and fresh, or adversarially controlled. By harnessing their type system into a synthesis framework, they are able to generate thousands of constructions with their security proofs, including constructions whose efficiency compete with state-of-the-art algorithms that were discovered using conventional methods. Using SMT-based methods, Tiwari, Gascón and Dutertre [TGD15] introduce an alternative approach to synthesize bitvector programs, padding-based encryption schemes and modes of operation.

Our work is most closely related to CIL [BDKL10], ZooCrypt [BCG+13] and AutoG&P [BGS15]. Computational Indistinguishability Logic (CIL) [BDKL10] is a formal logic for reasoning about security experiments with oracle and adversary calls. CIL is general, in that it does not prescribe a syntax for games, and side-conditions are mathematical statements. CIL does not make any provision for mechanization, although, as any mathematical development, CIL can be formalized in a proof assistant, see [CDL11]. ZooCrypt [BCG+13] is a platform for synthesizing padding-based encryption schemes; it has been used successfully to analyze more than a million schemes, leading to the discovery of new and interesting schemes. ZooCrypt harnesses two specialized computational logics for proving chosen-plaintext and chosen-ciphertext security, and effective procedures for finding attacks. The computational logics use deducibility to trigger proof steps that apply reduction to one-wayness assumptions, and to compute the probability of bad events using a notion of symbolic entropy. However, ZooCrypt is highly specialized.

AutoG&P [BGS15] introduce a computational logic and provide an implementation of their logic, called AutoG&P, for proving security of pairing-based crypto-

graphic constructions. Their logic uses deducibility for ensuring that proof rules are correctly enforced. Their implementation achieves a high level of automation, thanks to a heuristics for checking deducibility, and a proof search procedure, which decides which proof rule to apply and automatically selects applications of computational assumptions. We build heavily on this work; in particular, AutoLWE is implemented as an independent branch of AutoG&P. The main differences are:

- AutoLWE supports oracle-relative assumptions and general forms of the Left-over Hash Lemma, and (semi-)decision procedures for deducibility problems, for the theories of Diffie-Hellman exponentiation, fields, non-commutative rings and matrices. In contrast, AutoG&P only support more limited assumptions and implements heuristics for the theory of Diffie-Hellman exponentiation;
- AutoG&P supports automated generation of EasyCrypt proofs, which is not supported by AutoLWE. Rather than supporting generation of proofs a posteriori, a more flexible alternative would be to integrate the features of AutoG&P and AutoLWE in EasyCrypt.

Theodorakis and Mitchell [TM18] develop a category-theoretical framework for game-based security proofs, and leverage their framework for transferring such proofs from the group-based or pairing-based to the lattice-based setting. Their results give an elegant proof-theoretical perspective on the relationship between cryptographic proofs. However, they are not supported by an implementation. In contrast, we implement our computational logic. Furthermore, proofs in AutoLWE have a first-class status, in the form of proof scripts. An interesting direction for future work is to implement automated compilers that transform proofs from the group- and pairing-based settings to the lattice-based settings. Such proof compilers would offer a practical realization of [TM18] and could also implement

patches when they fail on a specific step.

2.7 Conclusion

We have introduced a symbolic framework for proving the security of cryptographic constructions based on the (decisional) Learning with Errors assumption. A natural step for future work is to broaden the scope of our methods to deal with other hardness assumptions used in lattice-based cryptography, including the Ring Learning with Errors assumption, the Short Integer Solution assumption. A further natural step would then be to analyze lattice-based key exchange protocols [Pei14, BCD+16]. To this end, it would be interesting to embed the techniques developed in this paper (and in [BGS15]) into the EasyCrypt proof assistant [BGHZ11, BDG+13], and to further improve automation of EasyCrypt for typical transformations used for proving security of protocols.

2.8 Proofs of section 2.4.1

In group theory, a multilinear map is a function which goes from a set of groups into a target group, and is linear with respect to all its arguments. They have been used in the past years to develop new schemes, such as Boneh-Boyen Identity Based Encryption [BB04] or Waters' Dual System Encryption [Wat09].

Given a multilinear map $\hat{e}$, $g_1, .., g_n, g_t$ a set of groups generators, let X be a set of public names sampled in $\mathbb{Z}_q$, Y be a set of private names sampled in $\mathbb{Z}_q$, $f_1, ...f_k, h \in \mathbb{K}[X, Y]$ be a set of polynomials over both public and secret names and Γ be a coherent set of axioms.

Our deducibility problem is to decide if $\Gamma \models X, g_{i_1}^{f_1}, ..., g_{i_k}^{f_k} \vdash_{\mathcal{E}} g_t^h$ Without loss of generality, we consider here the case of a bilinear map, to simplify the writing, but the proofs scale up to multilinear maps.

2.8.1 Saturation into the target group

First, we reduce our problem to the case of a single group. This result comes from the Proposition 1 of [KMT12]. Their constructive proof can trivially be used to obtain the following proposition:

Proposition 6. *For any sets X and Y, polynomials $f_1, ...f_n, h \in \mathbb{K}[X, Y]$ and groups elements $g_{i_1}^{f_1}, ..., g_{i_n}^{f_n}$, we denote*

$$\begin{aligned}(g_t^{e_i}) = \quad & \{\hat{e}(g_{i_j}, g_{i_k}) | 1 \leq j \leq k \leq n, g_{i_j} \in \mathbb{G}_1, g_{i_k} \in \mathbb{G}_2\} \\ & \cup \{\hat{e}(g_{i_j}, 1) | 1 \leq j \leq n, g_{i_j} \in \mathbb{G}_1, \} \\ & \cup \{\hat{e}(1, g_{i_j}) | 1 \leq j \leq n, g_{i_j} \in \mathbb{G}_2, \}\end{aligned}$$

Then $\Gamma \models X, g_{i_1}^{f_1}, ..., g_{i_n}^{f_n} \vdash_{\mathcal{E}} g_t^h \Leftrightarrow \Gamma \models X, g_t^{e_1}, ..., g_t^{e_N} \vdash_{\mathcal{E}-\hat{e}} g_t^h$.

We obtain a problem where we only have elements in the target group, we can therefore reduce the general problem to the single group case.

2.8.2 Reduction to polynomials

Lemma 2. *For any sets X and Y, polynomials $w_1, ...w_N, h \in \mathbb{K}[X, Y]$ we have $\Gamma \models X, g_t^{w_1}, ..., g_t^{w_N} \vdash_{\mathcal{E}} g_t^h$ if and only if:*

$$\exists (e_i, g_i) \in \mathbb{K}[X], (\forall i, \Gamma \models g_i \neq 0) \wedge \sum_i e_i \times \frac{f_i}{g_i} = h$$

Proof. If $\Gamma = \emptyset$, the adversary can construct elements of the form $(g_t^{w_i})^{e_i}$, where $e_i \in \mathbb{K}[X]$, i.e e_i is a polynomial constructed over variables fully known by the adversary, and then multiply this kind of term, yielding a sum in the exponent. If $\Gamma \neq \emptyset$, he may also divide by some $g_t^{g_i}$, with $g_i \in \mathbb{K}[X]$. We capture here the three capabilities of the adversary, which when looking in the exponent immediately translate into the formula on the right side. □

To handle this new problem, we notice that we can actually compute the set $\{g | \Gamma \models g \neq 0\}$. Indeed, for each axiom $f \neq 0$, we can extract a finite set of non zero irreducible polynomials by factorizing them (for example using Lenstra algorithm [Len85]). Any non annulling polynomial will be a product of all these irreducible polynomials. We can then obtain a finite set $Gs = (g_i)$ such that $G = \{g | \Gamma \models g \neq 0\} = \{\prod_{g \in Gs} g^{k_g} | \forall g, k_g \in \mathbb{N}\}$. With these notations, we can simplify proposition 1, because we know the form of the g_i. Moreover, as we do not want to deal with fractions, we multiply by the common denominator of all the $\frac{w_i}{g_i}$.

Lemma 3. *For any sets X and Y, polynomials $w_1, ...w_N, h \in \mathbb{K}[X, Y]$ we have $\Gamma \models X, g_t^{w_1}, ..., g_t^{w_N} \vdash_{\mathcal{E}} g_t^h$ if and only if:*

$$\exists (e_i) \in \mathbb{K}[X], (k_g) \in \mathbb{N}, \sum_i e_i \times w_i = h \prod_{g \in Gs} g^{k_g}$$

We do not prove this lemma, we will rather reformulate it using more refined mathematical structures and then prove it. Let us call $M = \{\sum_i e_i \times w_i | e_i \in \mathbb{K}[X]\}$ the free $\mathbb{K}[X]$-module generated by the (w_i). We recall that a S-module is a set stable by multiplication by S and addition, and that $\langle (w_i) \rangle_S$ is the S-module generated by (w_i). We also recall the definition of the saturation :

Definition 3. *Given a S-module T, $f \in S$ and $S \subset S'$, the saturation of T by f in S' is* $T :_{S'} (f)^\infty = \{g \in S' | \exists n \in \mathbb{N}, f^n g \in T\}$.

The previous lemma can be reformulated using saturation; if M is the module generated by $w_1, ..., w_N$:

Lemma 4. $\Gamma \models X, g_t^{w_1}, ..., g_t^{w_N} \vdash_{\mathcal{E}} g_t^h \Leftrightarrow h \in M :_{\mathbb{K}[X,Y]} (g_1...g_n)^\infty$

Proof. We recall that:

$$M :_{\mathbb{K}[X,Y]} (g_1...g_n)^\infty = \{x \in \mathbb{K}[X,Y] | \exists k \in \mathbb{N}, (g_1...g_n)^k \times x \in M\}$$

$\Rightarrow$ We have $\sum_i e_i \times w_i = h \prod_{g \in Gs} g^{k_g}$. With $K = max(k_g)$, we multiply both sides by $\prod_g g^{K-k_g}$ to get $h \prod_{g \in Gs} g^K = \sum_i \prod_g g^{K-k_g} e_i \times w_i \in M$. Which proves that $h \in M :_{\mathbb{K}[X,Y]} (g_1...g_n)^\infty$.

$\Leftarrow$ If $h \in M :_{\mathbb{K}[X,Y]} (g_1...g_n)^\infty$, we instantly have $(e_i) \in \mathbb{K}[X], k \in \mathbb{N}$ such that $h \prod_{g \in Gs} g^{k_g} = \sum_i e_i f_i$. □

We then simplify the saturation, by transforming it into the membership of the intersection of modules.

Lemma 5. *For any sets X and Y, $f_1, ...f_n, h \in \mathbb{K}[X,Y]$, $g \in \mathbb{K}[X]$,let $M = \{\sum_i e_i \times f_i | e_i \in \mathbb{K}[X]\}$. Then, with t a fresh variable* $M :_{\mathbb{K}[X,Y]} g^\infty = \langle (f_i) \cup ((gt - 1)Y^{\boldsymbol{j}})_{\boldsymbol{j} \in \{deg_Y(f_i)\}} \rangle_{\mathbb{K}[X,t]} \cap \mathbb{K}[X,Y]$.

Proof. $\subset$. Let there be $v \in M :_{\mathbb{K}[X,Y]} g^\infty$. Then, we have k such that $g^k \times v \in M$. The following equalities shows that v is in the right side set $v = g^k t^k v - (1 + gt + ... + g^{k-1}t^{k-1})(gt-1)v$. Indeed, $g^k t^k v \in M\mathbb{K}[X,t]$, so we have $(e_i) \in \mathbb{K}[X,t]$ such that $g^k t^k v = \sum_i e_i f_i$. Moreover, $g^k \times v \in M$ and $g \in \mathbb{K}[X]$ implies that

$deg_Y(v) \subset \{deg_Y(f_i)\}$. So we do have $(e'_i) \in \mathbb{K}[X,t]$ and $(\boldsymbol{j_i}) \subset \{deg_Y(f_i)\}$ such that

$$(1 + gt + ... + g^{k-1}t^{k-1})(gt-1)v = \sum e'_i(gt-1)Y^{\boldsymbol{j_i}}$$

Finally, $v \in \langle (f_i) \cup ((gt-1)Y^{\boldsymbol{j}})_{\boldsymbol{j} \in \{deg_Y(f_i)\}} \rangle_{\mathbb{K}[X,t]} \cap \mathbb{K}[X,Y]$.

$\supset$. Let there be $v \in \langle (f_i) \cup ((gt-1)Y^{\boldsymbol{j}})_{\boldsymbol{j} \in \{deg_Y(f_i)\}} \rangle_{\mathbb{K}[X,t]} \cap \mathbb{K}[X,Y]$. Then we have $(e_i), (e'_i) \in K[X,t]$ and $(\boldsymbol{j_i}) \subset \{deg_Y(f_i)\}$ such that :

$$v = \sum_i e_i f_i + \sum_i e'_i(gt-1)Y^{\boldsymbol{j_i}}$$

We have that $v \in \mathbb{K}[X,Y]$, so v is invariant by t. So, if we subsitute t with $\frac{1}{g}$, we have that $v = \sum_i e_i(X, \frac{1}{g})f_i$. Let us consider g^k the common denominator of all those fractions and call $e''_i = g^k e_i \in \mathbb{K}[X]$. We finally have $g^k \times v = \sum_i e''_i f_i \in M$, which means that $v \in M :_{\mathbb{K}[X,Y]} g^\infty$. □

The Buchberger algorithm allows us to compute a Gröbner basis of any free $\mathbb{K}[X]$-module [Eis13] and then decide the membership problem for a module. We thus solve our membership problem using this method.

Theorem 1. *For any sets X and Y, polynomials $f_1, ...f_n, h \in \mathbb{K}[X,Y]$, group elements $g_{i_1}, ..., g_{i_n}$ and a set of axioms Γ we can decide if $\Gamma \models X, g_{i_1}^{f_1}, ..., g_{i_n}^{f_n} \vdash_{\mathcal{E}} g_t^h$*

Proof. To decide if h is deducible, we first reduce to a membership problem with lemma 4 that can be solved using lemma 5 by computing the Gröbner basis of $\langle (f_i) \cup ((gt-1)Y^{\boldsymbol{j}})_{\boldsymbol{j} \in \{deg_Y(f_i)\}} \rangle_{\mathbb{K}[X,t]}$, keeping only the elements of the base that are independent of t and then checking if the reduced form of h is 0. □

As a side note, being able to decide the deducibility in this setting allows us to decide another classical formal method problem, the static equivalence. Indeed the

computation of the Gröbner basis allows us to find generators of the corresponding syzygies (Theorem 15.10 of [Eis13]), which actually captures all the possible distinguishers of a frame.

2.9 Proofs for section 2.4.3

$$[0]\ \frac{}{\Gamma \vdash 0 : {\mathbb{Z}_q}^{n,m}} \qquad [\textsc{id}]\ \frac{}{\Gamma \vdash \mathbf{I} : {\mathbb{Z}_q}^{n,n}} \qquad [\textsc{Tr}]\ \frac{\Gamma \vdash M : {\mathbb{Z}_q}^{m,n}}{\Gamma \vdash M^\top : {\mathbb{Z}_q}^{n,m}}$$

$$[\textsc{sL}]\ \frac{\Gamma \vdash M : {\mathbb{Z}_q}^{n,m+m'}}{\Gamma \vdash \mathsf{sl}\ M : {\mathbb{Z}_q}^{n,m}} \quad [\textsc{sR}]\ \frac{\Gamma \vdash M : {\mathbb{Z}_q}^{n,m+m'}}{\Gamma \vdash \mathsf{sr}\ M : {\mathbb{Z}_q}^{n,m'}} \qquad [\text{-}]\ \frac{\Gamma \vdash M : {\mathbb{Z}_q}^{n,m}}{\Gamma \vdash -M : {\mathbb{Z}_q}^{n,m}}$$

$$[\in]\ \frac{M \in \mathcal{M}}{\mathcal{M} \vdash M} \qquad [\times]\ \frac{\Gamma \vdash M : {\mathbb{Z}_q}^{n,\ell} \quad \Gamma \vdash M' : {\mathbb{Z}_q}^{\ell,n}}{\Gamma \vdash M \times M' : {\mathbb{Z}_q}^{n,m}}$$

$$[+]\ \frac{\Gamma \vdash M : {\mathbb{Z}_q}^{n,m} \quad \Gamma \vdash M' : {\mathbb{Z}_q}^{n,m}}{\Gamma \vdash M + M' : {\mathbb{Z}_q}^{n,m}} \qquad [||]\ \frac{\Gamma \vdash M : {\mathbb{Z}_q}^{n,m} \quad \Gamma \vdash M' : {\mathbb{Z}_q}^{n,m'}}{\Gamma \vdash M||M' : {\mathbb{Z}_q}^{n,m+m'}}$$

Figure 2.13: Typing rules for matrix operators.

We provide a more detailed proof of Proposition 5. To reason about matrices deducibility, written $\mathcal{M} \vdash M$ for a set of matrices $\mathcal{M}$ and a matrix M, we use the natural formal proof system $\mathcal{K}$ which matches the operations on expressions (see Figure 2.13), that we extend with the equality rule $[\textsc{Eq}]\ \frac{\mathcal{M} \vdash M_1 \quad \mathcal{M} \vdash M_1 = M_2}{\mathcal{M} \vdash M_2}$. For ease of writing, we denote $(\frac{A}{B}) := (A^\top||B^\top)^\top$.

Splits elimination

Proposition 7. *Given a set of matrices $\mathcal{M}$ and a matrix M, we can obtain $\mathcal{S}(\mathcal{M})$ a set of matrices without any concats, such that $\mathcal{M} \vdash M \Leftrightarrow \mathcal{S}(\mathcal{M}) \vdash H$.*

Proof. We notice that the concat operations commute with all the other operators: $(A||B)+(C||D) = (A+C||B+D)$, $(A||B)-(C||D) = (A-C||B-D)$, $A\times(B||C) = (AB||AC)$, $(A||B) \times \frac{C}{D} = AC + BD$, $(A||B)^\top = (\frac{A^\top}{B^\top})$. Given a set of matrices $\mathcal{M}$, we rewrite the matrices so that the concatenations operators are at the top. We can see the matrices as block matrices with submatrices without any concat, and then, we can create a set $\mathcal{S}(\mathcal{M})$ containing all the submatrices. This preserves deducibility thanks to the Eq rule for the rewriting, and to the split rules for the submatrices. □

Definition 4. *We call $\mathcal{N}$ the proof system based on $\mathcal{K}$ without splits, and write the deducibility with $\mathcal{M} \vdash_{\mathcal{N}} M$.*

Lemma 6. *If $\mathcal{M} \vdash (R||S)$ with a proof π (resp. $\mathcal{M} \vdash (\frac{R}{S})$) then $\mathcal{M} \vdash R$ and $\mathcal{M} \vdash S$ with smaller proofs (resp. $\mathcal{M} \vdash R$, $\mathcal{M} \vdash S$) .*

Proof. We prove it by induction on the size of the proof, and by disjunction on the last rule applied.

Base case: $|\pi| = 2$, then the proof is a concat on axioms and we can then obtain the sub matrix directly, with a proof of size one.

Induction case:

- The last rule is $[\text{Tr}]\,\dfrac{(\dfrac{R}{S})}{(R|S)}$.

  Then, we directly obtain by induction on $(\frac{R}{S})$ smaller proofs for R and S.

- The last rule is $[\times]\,\dfrac{M \quad (N^l||N^r)}{(MsN^l|MN^r)}$. Then, by induction on the proof of N, we obtain proofs of size smaller than $|\pi| - 1$ of N^l and N^r, and we just have to add a $[\times]$ to those proofs, yielding smaller proofs of MN^l and MN^r.

- If the last rule is [+], [-], [||], the proof can be done similarly to the two previous cases.

- The last rule is $[\text{sL}]\ \dfrac{((M|N)|L)}{(M|N)}$ Then, we have a proof of $((M|N)|L)$ of size $|\pi|-1$, so by induction we have a proof of $(M|N)$ smaller than $|\pi|-1$, and by adding a sL, we for instance obtain M with a proof smaller than $|\pi|$.

- [sR] is similar.

□

Lemma 7. *If $\mathcal{M}$ is a set of matrix without concatenations, and if $\mathcal{M} \vdash M$, then $\mathcal{M} \vdash_{\mathcal{N}} M$.*

Proof. We prove it by induction on the size of the proof, and by disjunction on the last rule applied.

Base case: $|\pi| = 1$, then the only problem might be if the rule used was a split, but as we have matrices without concatenations, this is not possible.

Induction case:

- If the last rule is [Tr], [+], [-], [||], we conclude by applying the induction hypothesis to the premise of the rule.

- The last rule is $[\text{sL}]\ \dfrac{(M|N)}{M}$. Then, we have a proof of $(M|N)$ of size $\pi - 1$, and with lemma 6 we have a smaller proof of M, on which we can then apply our induction hypothesis to obtain a proof of M without split.

- *splitR* is similar.

$\square$

Concatenations elimination

Definition 5. *We call $\mathcal{T}$ the proof system based on N without concatenations, and write the deducibility $\mathcal{M} \vdash_{\mathcal{T}} M$.*

Lemma 8. *If $\mathcal{M}$, M, N do not contain any concat, then:*

$$\mathcal{M} \vdash_{\mathcal{N}} (M|N) \Leftrightarrow \mathcal{M} \vdash_{\mathcal{T}} M \wedge \mathcal{M} \vdash_{\mathcal{T}} N$$

Proof. The left implication is trivial. For the right one, we once more do a proof by induction on the size of the proof.

Base case: $|\pi| = 1$, the last rule is a $[||]$, and we do have a proof of M and N.

Induction case:

- The last rule is $[\times]\ \dfrac{M \quad (N^l||N^r)}{(MN^l|MN^r)}$

  Then, by induction on the proof of N, we obtain proofs of size smaller than $|\pi| - 1$ of N^l and N^r without concats, and we just have to add a $[\times]$ to those proofs, yielding proofs of MN^l and MN^r without concats.

- If the last rule is [TR], [+], [-], [SL], [SR], we proceed as in the previous case

- The last rule is $[||]$. Then the induction rule instantly yields the expected proofs. Then, we have a proof of $(M|N)$ of size $\pi - 1$, and with lemma 6 we have a smaller proof of M, on which we can then apply our induction hypothesis to obtain a proof of M without split.

$\square$

Lemma 9. $\mathcal{M} \vdash_{\mathcal{N}} M \Leftrightarrow \forall G \sqsubseteq M, \mathcal{M} \vdash_{\mathcal{T}} G$ *Where* $G \sqsubseteq H$ *denotes the fact that* G *is a sub matrix of* M *without any concatenation.*

Proof. The left implication is trivial, we prove the right one. As was done in Lemma 7, we can see M has a bloc matrix, i.e with all the concat at the top.

We are given a proof of $\mathcal{M} \vdash_{\mathcal{N}} M$, which must contain all its concatenations at the bottom of the proof tree. If we look at all the highest concat rule in the proof such that no concat is made before, we have some proof of $\mathcal{M} \vdash_{\mathcal{N}} (M_i|M_j)$, and thanks to lemma 8, we have $\mathcal{M} \vdash_{\mathcal{T}} M_i \wedge (A_i) \vdash_{\mathcal{T}} M_j$. Applying this to all the highest concat rules in the proof yields the result. □

Removal of the transposition

Definition 6. *We call* $\mathcal{V}$ *the proof system based on* $\mathcal{N}$ *without concat, and write the deducibility* $\mathcal{M} \vdash_{\mathcal{V}} M$.

The transposition commutes with the other operations, given a matrix M we define the normal form $N(M)$ where the transposition is pushed to the variables. We extend the notation for normal form to sets of matrices.

Lemma 10. $\mathcal{M} \vdash_{\mathcal{T}} M \Leftrightarrow \mathcal{M} \cup (N(\mathcal{M}^t)) \vdash_{\mathcal{V}} N(M)$

Proof. $\Leftarrow$ This is trivial, as the normal form can be deduced using the rule [Eq]. $\Rightarrow$ Given the proof of M, we can commute the trans rule with all the others, and obtain a proof tree where all the transposition are just after a [$\in$] rule. Then, any [$\in$] followed by [TRANS] can be replaced by a [$\in$] and a [EQ] when given the input $\mathcal{M} \cup (N(\mathcal{M}^t))$ instead of $\mathcal{M}$. We can thus construct a valid proof of $\mathcal{M} \cup (N(\mathcal{M}^t)) \vdash_{\mathcal{V}} N(M)$ □

Conclusion The proof of proposition 5 is a direct consequence of Lemmas 7, 7, 9 and 10.

CHAPTER 3

VERIFYING DISTRIBUTED PROTOCOLS USING IPDL

In Chapter 2, we presented AutoLWE, a tool for formally verifying noninteractive primitives, whose security is defined in terms of security games (i.e., imperative probabilistic programs). While expressive, tools such as AutoLWE or EasyCrypt are mostly relegated to proofs in the *noninteractive* and *two-party* settings, and not easily usable for general distributed cryptographic tasks, such as MPC.

In the cryptographic literature, the *de facto* framework for reasoning about distributed cryptography is Universal Composability (UC) [Can01]. UC adopts a so-called simulation-paradigm, where we want to prove the protocol in question satisfies *approximate observational equivalence* (i.e., computational indistiniguishability) to a particular idealization which relies on trusted functionalities rather than cryptographic mechanisms. Any cryptographic protocol proven secure in UC is known to be concurrently and modularly composable.

To enable scalable formal verification for complex cryptography, our goal is to provide an easy-to-use system for encoding and mechanically checking proofs for multi-party protocols. For the system to be usable by cryptographers, it is important that proofs in the system *approximately match how cryptographers write proofs on paper.*

State of affairs. While a line of work focused on formal verification of cryptography, most earlier works fall short of verifying general, multi-party cryptographic protocols. Earlier works either focused on verification of non-interactive primitives and do not provide a native protocol-calculus for encoding interactive multi-party protocols (e.g., Easycrypt [BGHZ11] and FCF [PM15b]), or fo-

cused on restricted classes of protocols using specific cryptographic primitives such as encryption and message authentication, symbolically modeling encryption and authentication as certain ideal abstractions to facilitate formal verification [BSCS18, MSCB13, DY83a, Cre08]. For the latter line of works (often referred to as the *symbolic* approach), it is imperative that the "symbolic" ideal abstractions of cryptography exactly match what actual cryptography can provide, and this turns out to be subtle and non-trivial. It was observed that in some works, the symbolic abstractions are a mismatch of what actual cryptography can provide, and consequently, even a formally verified protocol can be broken when instantiated with actual cryptography [AR00, Mic19]. CryptoVerif is a related symbolic-style prover that reasons directly in the computational model [Bla06b], but is focused on automation over expressivity.

Formally verifying distributed cryptographic protocols is an exciting nascent area that has recently attracted increasing attention from the cryptography as well as the formal methods communities. A couple recent systems, EasyUC [CSV19] and CryptHOL [LSBM19] made initial attempts at this goal. These systems adopt bisimulation (i.e., relational invariant) to reason about observational equivalence between protocols (i.e., the observable traces induced by protocols are identically distributed). Bisimulation-style proofs mix reasoning about low-level distributional equivalence with higher-level cryptographic reasoning, making it somewhat cumbersome and unnatural for cryptographers to use these systems — improving the usability of such systems was phrased as a major open question in earlier works [CSV19]. A few recent works [EP19, HKO+18, SV17, ABB+17] made one-off efforts to mechanize the proofs of a single, specific MPC protocol in Easycrypt; however, these works do not aim to provide a general logic for encoding cryptographic protocols in general, and their approach does not na-

tively reason about concurrent composition of protocols. Finally, an elegant line of work [BCL12, BCL14, BCEO19] beginning with Bana and Comon [BCL12] combines both symbolic-style and equational reasoning with unconditional computational soundness. While their framework natively supports automation, it has not been mechanized and it remains unclear how easy it is to encode larger-scale developments such as secure function evaluation or *multi*-party protocols. We give a more detailed comparison with Bana-Comon in Section 3.7.

Our Contributions In this paper, we propose IPDL (short for Interactive Probabilistic Dependency Logic), a language and proof system for reasoning about multi-party cryptographic protocols. IPDL is designed with the following desiderata in mind:

- *Ease of use.* As mentioned, we would like the experience of using IPDL to resemble how cryptographers write proofs on paper. One novelty of IPDL is that its logic directly captures approximate observational equivalence reasoning, which is at the core of common simulation-style proofs for cryptographic protocols Unlike previous works [CSV19, LSBM19] which reason about entire actors (i.e. parties or functionalities), IPDL adopts a *channel-centric* logic, which decomposes the behavior of the protocol into the behaviors along each communication channel. This is the insight that enables us to have a simple equational logic.
- *Support for a broad class of protocols.* IPDL's logic supports straightline protocols with *statically bounded loops* (i.e., loops with a-priori known bounds) — we stress, however, that adversaries are treated as arbitrary probabilistic polynomial-time machines in IPDL. The statically bounded loops can be used to *parametrize* the number of parties, the size of the circuit (e.g., in a

secure function evaluation example), the number of invocations in a reactive functionality, and so on. This allows us to capture a broad class of protocols, including most protocols studied in the cryptography literature (see our case studies for examples).

- *Computational soundness.* Since IPDL's logic is straightline supporting statically bounded loops, it allows the logic to symbolically track the runtime and error of the reduction. In this way, the core logic can reason about the security loss in IPDL proofs.
- *Compositional guarantees.* IPDL's approximate observational equivalence notion (also referred to as approximate equivalence for simplicity) follows the elegant Universal Composability (UC) paradigm [Can01]. In a typical simulation-style proof, we want to reason that some real-world protocol's security is as strong as some ideal-world specification. To do this, we can encode a simulator in IPDL that interacts with the real-world adversary and the ideal specification, and we prove in IPDL that the real-world and ideal-world executions are approximately equivalent in terms of the view from an external environment. In this way, IPDL supports the reasoning of concurrent composition of cryptographic protocols (either with itself or with other protocols) [Can01].

While IPDL follows the UC paradigm to provide concurrent composition, it is not our goal to capture the full extent of the expressiveness of UC. For example, currently we assume a static corruption model where the set of corrupt parties are determined a-prori. As mentioned, we also impose certain restrictions on the protocols (i.e., straightline with statically bounded loops) to allow explicit tracking of the runtime of the programs, and ensure computational soundness. This seems to be the right sweet-spot between ease-of-use and comprehensiveness: we chose

the simplifications carefully such that IPDL can nonetheless capture a broad class of protocols studied in the cryptography literature; and these simplifications allow us to encode proofs in IPDL that are concise and resemble on-paper proofs.

Mechanization using Coq. We have implemented IPDL in Coq, and open sourced it at `https://github.com/ipdl/ipdl`. The main strength of our implementation is that we support *parameterized* protocols, or families of protocols definable by a function in Coq. By doing so, we are able to support protocols indexed over arbitrary Coq types – this includes protocols defined for q adversary queries, n parties, or inductive protocols such as MPC for general circuits.

Rich case studies. We have implemented several case studies which demonstrate how IPDL can help scale up formal verification to more complex protocols than before. The case studies and proofs are included in our open source too. Currently, we provide the following case studies:

1. a multi-use, secure communication network out of an authenticated one using a CPA-secure encryption scheme;
2. a maliciously secure n-party coin toss with abort protocol assuming idealized commitments;
3. several semi-honest oblivious transfer (OT) constructions, including OT from trapdoor permutations [GMW87], 1-out-of-4 OT from 1-out-of-2 [14o, NP99], and a preprocessing protocol for OT [pre, Bea95]; and
4. a semi-honest 2-party GMW protocol [GMW87], defined over a general family of circuits.

Proof effort. The secure network example can be encoded in 195 lines of code, including description and proofs. In comparison, the recent EasyUC work [CSV19] required 12203 lines of code to realize a *single-use* secure network [1]. Among these above case studies, the most sophisticated is GMW, which uses the OT as a building block. Encoding the description and proof of GMW+OT in IPDL is accomplished with less than 3201 lines of code. As a rough point of comparison, the related work by [ABB+17] took 11069 lines of code to encode the garbled circuit + OT [Yao82] protocol (and moreover, their work is a one-off effort focused on mechanizing a single protocol rather than providing a general logic and framework). For some other one-off efforts at mechanizing MPC proofs [EP19, SV17, ABB+17], we were not able to find open source code online, so we cannot provide a direct comparison in terms of lines of code.

3.1 Outline of Chapter

In Section 3.2, we first demonstrate IPDL by way of a simple example – proving that an authenticated network can realize a secure network by using an encryption scheme. While the example is simple, the security guarantee is strong – *any other protocol* which assumes a secure network can use this proof to safely use an authenticated one instead (given a trusted key setup). This equational logic is proved sound using a novel semantics developed in Section 3.9.

[1]Among the 12203 lines for EasyUC's secure network example, roughly speaking, 260 lines describe the real-world protocol, 182 lines describe the ideal functionality, 190 lines describe the simulator, and the remaining lines are the proof. We did not count their key exchange since our implementation assumed a trusted key setup. As the EasyUC paper [CSV19] itself acknowledges, part of the complexity arises from the fact that EasyCrypt is procedure-based and does not natively support a protocol calculus; consequently, there was a significant amount of tedious work in writing "boilerplate" code that route messages in between parties and functionalities.

We then detail the formal account of IPDL in Section 3.3. IPDL is unique in that it exposes an *equational logic* which allows one to reason directly over the code of protocols. This is in contrast to related tools such as EasyUC [CSV19], which makes use of Easycrypt's relational Hoare logic, requiring the use of heavyweight invariants to reason equationally.

In Section 3.4, we discuss derived rules and soundness for reasoning about *parameterized protocols* in IPDL. While cryptographic protocols generally have parameters (e.g., the number of adversary queries, the size of a circuit, the number of parties), the core logic of IPDL is purely about concrete programs. We bridge this gap by verified, derived rules for reasoning about parameterized protocols.

In Section 3.5, we discuss the encoding of IPDL into Coq. In Section 3.6 and Section 3.10, we detail our case studies (described in the previous section). Finally, we discuss additional related work and future directions in Sections 3.7 and 3.8.

3.2 IPDL by Example: Multi-use Secure Network

In this section, we introduce IPDL and its equational style of reasoning through an example protocol for constructing a simple secure communication network out of an authenticated one using a CPA-secure encryption scheme. Before we introduce the protocol, we will first introduce the basic syntax and semantics of IPDL, along with some background information protocol security.

3.2.1 Terminology and Background

We define some basic terminology and review some background on UC [Can01]. A *protocol* is any (distributed) message-passing system, which may give outputs and react to inputs. For protocols to have a meaningful security definition, they may exhibit probabilistic behavior but *not* (possibilistic) nondeterminism. Protocols communicate over *channels*, which for us are unidirectional (i.e., either input or output). The channels of a given protocol can be split into *interfaces*, which are subsets of the channels of a given protocol.

We express the security properties of a protocol through two interfaces: the *external* interface, which is used for the high-level I/O behavior of the protocol; and the *attacker* interface, which is used to define the threat model. Given two protocols P and Q with the same external interface but (possibly) differing attacker interfaces, we say that P *realizes* Q when there exists a *simulator* S which converts the attacker interface of Q to that of P, such that P is observationally equivalent to Q composed with S. This type of simulator is also called a converter in earlier work, such as Constructive Cryptography [Mau11].

We often call the attacker for P the *adversary*, and call the attacker for Q the simulator. Intuitively, when P realizes Q, any adversary's capability of influencing the external interface in P is *upper bounded* by the simulator, since any attack on P can be turned into an equivalent attack on Q.

We typically think of P as the implementation (or the *real* protocol) and Q as the specification (or the *ideal* protocol). While P is comprised of parties who interact with each other using cryptographic mechanisms and may be arbitrarily corrupted by the adversary, Q is comprised of idealized parties who instead interact

with a trusted third party (the *functionality*) which performs the protocol's logic in a centralized and trustworthy way. The simulator's capabilities in the ideal protocol are constrained by the functionality and ideal parties to be simple and easy to understand.

Relation to UC. Here, we compare our terminology to UC [Can01]. In UC, a real world protocol is defined to be the composition of all honest party's code and any hybrid functionalities, while an ideal protocol is usually taken to specifically mean the ideal functionality and all ideal parties. Our notion of protocol is more generic, and refers to any message-passing system definable in IPDL. In particular, any individual party's code in IPDL is considered a protocol, as is any arbitrary composition of protocols.

3.2.2 IPDL in a Nutshell

IPDL is a calculus and equational theory for reasoning about probabilistic protocols. The main judgement of IPDL is that $P \overset{\delta}{=} Q$, where P and Q are interactive protocols, and δ is a *computational error*, which upper-bounds the advantage of a computational attacker from distinguishing P from Q. IPDL supports UC-style reasoning: given protocols P and Q, one can ask whether there exists an appropriately typed simulator S such that $P \overset{\delta}{=} Q||S$, for a reasonable choice of δ.

In contrast to the semantics of UC [Can01] which utilizes systems of Interactive Turing Machines that must be activated sequentially, IPDL allows for messages to happen in any order consistent with the protocol. This is possible since all messages in IPDL are assumed to be scheduled by the attacker.

The key insight of IPDL is to adopt a *channel-centric* rather than agent-centric viewpoint. All channels in IPDL are write-once, and have a unique behavior associated to them by a *reaction*, which is a program that may sample probabilistic values and read the value of other channels (once those channels have fired). To enable a simple equational theory, all dependencies between channels are required to be statically determined. Thus, control flow may only happen at the level of *data*, but not on the level of the protocol. We stress, however, that the equational logic of IPDL is proven sound relative to a general cryptographic adversary who may use more complex control flows.

3.2.3 Multi-Use Secure Communication in IPDL

As a first example, we show how to construct a *secure* network from an *authenticated* network[2]. This is accomplished through message encryption, such that an eavesdropper cannot learn any information about the messages sent between Alice, the sender, and Bob, the recipient. Our secure network abstraction is multi-use, i.e., it is parametrized by a parameter q that denotes the number of messages exchanged between Alice and Bob. In this example, we assume that both Alice and Bob are honest and the adversary is a passive eavesdropper (although later on, in our case studies, we will have cases where the parties can be semi-honest or maliciously corrupt).

We conduct the example using syntax from our Coq mechanization which encodes our IPDL core logic. In our Coq implementation, the counterpart of our main judgement in IPDL, $P \overset{\delta}{=} Q$, is written as `P ~= Q`. Our current Coq mecha-

[2]The cryptographic literature uses the terms "secure channel" and "authenticated channel" [Can01], but we avoid overloading the term "channel" so "channel" always means the low-level, write-once IPDL channels.

nization has not yet implemented the tracking of error bounds which is part of the IPDL core logic described in Section 3.3; nonetheless we prove on paper that our logic is computationally sound (see Section 3.3).

More details for our Coq embedding is given in Section 3.5.

Definitions of authenticated and secure network. Figure 3.1 shows the IPDL encoding of the definitions of an authenticated network and a secure network, respectively. A network `Net` is parametrized with 1) q, which denotes the maximum number of messages exchanged, 2) m, which denotes the length of each message, and 3) `leakage`, which denotes a leakage function, and a parameter `l` which denotes the length of the leakage. In the definition of `Net`, `I` denotes a vector of q input channels between Alice and the ideal functionality `Net`, `O` denotes a vector q output channels between Bob and the ideal functionality `Net`; and `leak` and `ok` each denotes a vector of q channels between the ideal functionality and the adversary.

In Lines 8-9, for each $j < q$, `Net` reads from the jth channel in `I` to obtain a message `x`, applies the leakage function to `x`, and returns the result to the adversary through the jth channel in `leak`. To do so, we first we perform a parallel composition in IPDL over all $j < q$ (written `\||_(j < q)`). We then assign the jth channel of `leak` (written `leak ## j`) to the *reaction* through the syntax `::=`.

Similarly, in Lines 10-12, for each $j < q$, `Net` reads from the jth channel in `ok`, the jth channel in `I` to obtain a message, and assigns the message to the jth channel in `O`. Note here that we do not use the value along the jth `ok` channel, but only its *timing*: thus, the jth channel of `O` only fires once the jth channel of `ok` does. In other words, the receiver Bob receives the message only when the adversary okays it.

```
1  Definition Net (q m : nat) {l : nat} (leakage : m -> l)
2     (* External channels *)
3     (I O : q.-tuple (chan m))
4     (* Attacker channels *)
5     (leak : q.-tuple (chan l)) (ok : q.-tuple (chan TUnit))
6          :=
7    [||
8       \||_(j < q) (leak ## j) ::= (x <-- Read (I ## j);;
9                                   Ret (leakage x));
10      \||_(j < q) (O ## j) ::= (_ <-- Read (ok ## j);;
11                                  x <-- Read (I ## j);;
12                                  Ret x)
13   ].
14
15 Definition Auth q m := Net q m id.
16 Definition Sec q m := Net q m (fun _ => [tuple]).
```

Figure 3.1: Definitions of authenticated and secure networks in IPDL. Both networks are parameterized by the number of queries in question, q, and the length of messages, m.

Given our definition of `Net`, we can now define both the authenticated and secure networks. On Line 15, we define an authenticated network `Auth q m` to be equal to `Net` instantiated with the leakage function `id` (i.e., the identity function). This means that the adversary can read the contents of all messages in an authenticated network. Similarly, on Line 16 we define a secure network `Sec q m` to to be equal to `Net` with the constant leakage function `fun _ => [tuple]`, which returns the empty bitstring. Thus in `Sec`, the adversary receives no information about message contents. However, the adversary still learns the timing information of each message; and moreover, since the messages length m is a public parameter, the adversary is assumed to know m, too.

Protocol realizing a secure network. Figure 3.2 shows the IPDL encoding of a protocol that realizes a secure network from an authenticated one, through the use of encryption. The construction takes the same parameters as `Sec`, i.e., q and m, for the number of messages and length of each message, respectively.

Additionally, it takes the following parameters for c and k, for the ciphertext length and key length, and `genK`, `enc`, and `dec`, for generating keys, encryption, and decryption.

We first define the key generation functionality (in Line 12), which samples a key from the distribution `genK`, and assigns it to the channel `K` (which is taken as a parameter of the functionality). Then, we define the code for Alice in Line 14, who is parameterized by a vector of channels `I` for message inputs, a channel `K` for the encryption key, and a vector of channels `send` for communicating with the authenticated network. For each $j < q$, Alice: reads a message from the jth channel of `I`; a key from the channel `K`; generates a ciphertext by encrypting the message under the key; and assigns the ciphertext to the jth `send` channel. We define Bob similarly on Line 21: for each $j < q$, we read the jth ciphertext, read the key, and output the corresponding decryption to the jth channel of `O`.

We have the real protocol in total on Line 27. It is parameterized similarly to the `Sec` functionality, except it has leakage channels of length c instead of zero. We compose protocols together in IPDL through first *generating* local communication channels, and composing the subprotocols together using these local channels. On Line 30, we generate the channel `K` for key generation (using the Coq syntax `k <- new k`). Then, on Lines 31 and 32, we generate two fresh *vectors* of channels, `send` and `recv` for the underlying authenticated network. This is done with similar syntax `send <- newvec q @ c`: here, `q` is the length of the vector, and `c` is the length of messages on each channel. Finally, on Lines 34-37, we compose together the key generation functionality, Alice, Bob, and the underlying authenticated network. Thus, the authenticated network will deliver ciphertexts from `send` to `recv`, but only after leaking the ciphertexts along `leak` and receiving the `ok` message.

```
Section AuthToSec.
(* Same as in Net *)
Context (m q : nat).
(* Ciphertext and Key length *)
Context (c k : nat).

(* Algorithms for encryption *)
Context (genK : Dist k).
Context (enc : m -> k -> Dist c).
Context (dec : c -> k -> m).

Definition FKey (K : chan k) := (K ::= Samp genK).

Definition alice (I : q.-tuple (chan m)) (K : chan k)
                 (send : q.-tuple (chan c)) :=
  \||_(j < q) (send ## j) ::=
                 (msg <-- Read (I ## j) ;;
                  key <-- Read K ;;
                  ctxt <-- Samp (enc msg key) ;;
                  Ret ctxt)).

Definition bob (recv : q.-tuple (chan c)) (K : chan k)
               (O : q.-tuple (chan m)) :=
  \||_(j < q) (O ## j) ::=
                 (ctxt <-- Read (recv ## j) ;;
                  key <-- Read K ;;
                  Ret (dec ctxt key)).

Definition Real (I O : q.-tuple (chan m))
           (leak : q.-tuple (chan c))
           (ok : q.-tuple (chan TUnit)) :=
  K <- new k ;;
  send <- newvec q @ c ;;
  recv <- newvec q @ c ;;
  [||
      FKey K;
      alice I k send;
      bob recv K O;
      Auth q c send recv leak ok
  ].
```

Figure 3.2: Authenticated-to-secure network protocol in IPDL.

```
1  Definition Sim
2     (leakI : q.-tuple (chan 0)) (okI : q.-tuple (chan TUnit))
3     (leakR : q.-tuple (chan c)) (okR : q.-tuple (chan TUnit))
4  :=
5  K <- new key ;;
6  [||
7    K ::= (Samp genK) ;
8    \||_(j < q) (leakR ## j) ::=
9            (_ <-- Read (tnth leakI j);;
10            key <-- Read K;;
11            e <-- Samp (enc [tuple of nseq _ false] key);;
12            Ret e);
13   \||_(j < q) (okI ## j) ::= (x <-- Read (okR ## j) ;;
14                               Ret x)
15 ]
```

Figure 3.3: Simulator for the authenticated-to-secure network example in IPDL.

$$\begin{aligned} \text{Real World} &\equiv \text{Simplified RW (by decryption correctness)} \\ &\approx \text{Simplified RW + Message-Free Ciphertext (by CPA-security)} \\ &\equiv \text{Simplified Ideal World} \\ &\equiv \text{Ideal World} \end{aligned}$$

Figure 3.4: Outline of proof for authenticated-to-secure network example.

3.2.4 Simulator and Proof

To show that our real protocol is secure, we must show that the attacker's capabilities in the real world are *upper bounded* by those in the ideal world, wherein Alice and Bob rely on the functionality `Sec` to communicate. Recall that in the real world, the attacker learns any ciphertexts Alice sends through the `leak` channels, but in the ideal world, the attacker learns only the timing of each message. In both the real and ideal worlds, the attacker may use the `ok` channels to schedule the delivery of each message.

Definition of the simulator. The simulator is shown in Figure 3.3. We describe the simulator informally. It is parameterized by four vectors of channels: two, `leakI` and `okI`, communicate with the ideal world; and two, `leakR` and `okR` communicate with the real world. The simulator must do two things: it must receive timing information for the jth message through `leakI`, and produce a real-looking ciphertext along `leakR`; and it must receive scheduling information along `okR`, and produce scheduling information along `okI`.

To produce the real-looking ciphertexts for `leakR`, the simulator first generates a key channel `K`, similar to the real world, and samples `K` using `genK` (Line 6). Then, when it receives timing information for the jth message through `leakI` (Line 8), encrypts the all-zeroes message using `K` (Line 10), and outputs the ciphertext along the jth channel of `leakR`. (The all-zeroes message is encoded in Coq by `[tuple of nseq _ false]`.) This simulation is successful, since ciphertexts reveal no information about the message if the encryption scheme is CPA secure. Finally, the simulator may simply forward all scheduling decisions for from `okR` to `okI` (Line 12).

Proof of security. Once the simulator is constructed, we compose the ideal world with the simulator using locally generated channels and ask whether the result is approximately equivalent to the real world. This equivalence judgement proves that the adversary's view (`leakR` and `okR`), as well as Alice and Bob's view (`I` and `O`) cannot be distinguished between the two protocols. In the following theorem, `~=` is the Coq notation for the approximate equivalence judgement in IPDL (written on paper as $\stackrel{\delta}{=}$).

```
Theorem AutSec_Security I O leakR okR :
  real I O leakR okR ~=
   (leakI <- newvec q @ 0 ;;
    okI <- newvec q @ TUnit ;;
    [||
        ideal I O leakI okI;
        Sim leakI okI leakR okR
    ].
```

We now outline the IPDL proof required to prove the above theorem. The proof is outlined in Figure 3.4, and contains a number of steps:

1. *Simplifying the ideal world with simulator:* We first apply a number of equational rewrites to the ideal world. In effect, these equational rewrites will inline the behavior of the simulator into the ideal functionality. Recall from Figure 3.1 in Line 10 that the jth channel of `O` reads from the jth channel of `ok`, which for our ideal functionality is named `okI`. However, the simulator from Figure 3.3 in Line 13 forwards the value along the jth channel of `okR` into `okI`. In this instance, we can *fold* the definition of `okI` into `O`, which rewrites `O` so that it reads from `okR` directly. Since the internal channel `okI` is now now longer used, we may eliminate it from the protocol. IPDL is specifically designed to perform these kinds of rewrites, and do so in a succinct manner.

   After doing the same inlining step for the internal channel `leakI`, we receive the following protocol:

```
K <- new k ;;
[||
  K ::= Samp genK;
  \||_(j < q) (leakR ## j) ::=
                  (key <-- Read K ;;
                   _ <-- Read (I ## j) ;;
                   c <-- enc [tuple of nseq _ key] key ;;
                   Ret c);
  \||_(j < q) (O ## j) ::=
                  (_ <-- Read (okR ## j) ;;
                   msg <-- Read (I ## j) ;;
                   Ret msg)
]
```

2. *Simplifying the real world:* After simplifying the ideal world, we perform similar inlinings in the real world. Specifically, we inline the definition of `send` (coming from Alice) into the authenticated network, and inline the definition of `recv` (coming from the authenticated network) into Bob. Once we do so, we get that the value of Bob's output is equal to the decryption of Alice's generated ciphertext. To simplify Bob's output we apply an *axiom* which models the correctness of decryption for the encryption scheme. The axiom allows us to perform an equational rewrite to each of Bob's output channels in `O`, and transform it to the reaction that simply reads the message from `I`.

3. *Applying CPA security in the real world:* When we apply the rewrites in Step 2, we receive the following protocol:

```
K <- new k ;;
[||
  K ::= Samp genK;
  \||_(j < q) (leakR ## j) ::=
                   (key <-- Read K ;;
                    msg <-- Read (I ## j) ;;
                    c <-- enc msg key ;;
                    Ret c);
  \||_(j < q) (O ## j) ::=
                   (_ <-- Read (okR ## j) ;;
                    msg <-- Read (I ## j) ;;
                    Ret msg)
]
```

Note that this protocol is almost the same as the simplified ideal world from Step 1: The behavior along the `O` channels are exactly the same, but the behavior of the `leakR` channels here encrypts the real message, while the protocol in Step 1 encrypts the filler message.

To prove these two protocols equivalent, we first apply a *congruence* rule, which allows us to factor out the common behaviors for `O`, and focus only on the equivalence of the `leakR` channels between the real and ideal worlds. At this point, we can directly apply our equational axiom for CPA security, which states that no adversary can tell the difference between encryptions of arbitrarily chosen messages from encryptions of filler messages (given that they key is secret). This axiom applies directly to our two worlds, which finishes the proof.

$$
\begin{array}{llcl}
\text{Variables} & x & & \\
\text{Channels} & c & & \\
\text{Channel Sets} & I, O, C & ::= & \{c_1, \ldots, c_n\} \\
\text{Data Types} & \tau & ::= & \mathsf{unit} \mid \mathbb{B} \mid \tau_1 \times \tau_2 \\
 & & \mid & \mathsf{bits}(n) \qquad (\text{with } n \in \mathbb{N}) \\
 & & \mid & \ldots \\
\text{Variable Contexts} & \Gamma & ::= & x_1 : \tau_1, \ldots, x_n : \tau_n \\
\text{Channel Contexts} & \Delta & ::= & c_1 : \tau_1, \ldots, c_n : \tau_n \\
\text{Expressions} & e & ::= & x \mid \mathsf{tt} \mid \mathsf{true} \mid \mathsf{false} \\
 & & \mid & \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \\
 & & \mid & (e_1, e_2) \mid \mathsf{fst}(e) \mid \mathsf{snd}(e) \\
 & & \mid & \mathsf{f}(e_1, \ldots, e_n) \\
\text{Distributions} & D & ::= & 1_e \mid x : \tau \leftarrow D_1;\ D_2 \\
 & & \mid & \mathsf{Unif}(\tau) \mid \mathsf{D}(e_1, \ldots, e_n) \\
\text{Reactions} & R & ::= & \mathsf{Ret}\ e \\
 & & \mid & \mathsf{Samp}\ D \\
 & & \mid & \mathsf{Read}\ c \\
 & & \mid & x : \tau \leftarrow R_1;\ R_2 \\
\text{Protocols} & P, Q & ::= & c := R \mid P_1 \mid\mid P_2 \\
 & & \mid & \nu c : \tau.\ P \mid 0
\end{array}
$$

Figure 3.5: Syntax of IPDL Protocols.

3.3 Core Logic

In this section, we describe the core calculus of IPDL in detail, along with its semantics. Sections 3.3.1, 3.3.2, and 3.3.3 describe the syntax, typing rules, and equational logic of IPDL. In Sections 3.9 and 3.9.1 we describe the semantics of IPDL protocols and their interaction with adversaries.

3.3.1 Syntax

The syntax of IPDL is shown in Figure 3.5. All data types τ in IPDL are assumed to have a bitstring length $|\tau|$, along with an interpretation $[\![\tau]\!] : \{0,1\}^{|\tau|}$. For our examples, we assume a unit type, booleans, products, and bitstrings of a given

length $n \in \mathbb{N}$, along with their standard bitstring interpretations.

Protocols in IPDL are composed of *expressions*, *distributions*, *reactions*, and *protocols*. Expressions are built out of collection of function symbols $\mathsf{f}(e_1, \ldots, e_n)$, with an assumed typing rule and interpretation mapping bitstrings to bitstrings (of the appropriate lengths, depending on the type of f). For clarity, we show the standard connectives for unit, bool, products, and bitstrings (not shown).

Distributions represent probabilistically determined messages. Along with distribution symbols $\mathsf{D}(e_1, \ldots, e_n)$ which, similarly to function symbols, have a type and an interpretation, we assume the unit distribution 1_e, monadic bind, and the uniform distribution $\mathsf{Unif}(\tau)$ for any IPDL type τ. Distributions are assumed to always have unit mass.

Reactions can be seen as effectful programs which may sample from probability distributions and read from channels. Reactions themselves also carry a monadic structure. reaction with no variables is called *closed*; a reaction with no reads is necessarily equal to a sampling. Note that reactions may *not* contain any control flows themselves; thus, all effects which a reaction may perform are statically determined. Reactions intuitively have a semantics mapping valuations on channels to either distribution on return values or an error (if the required input channels do not have values set yet.)

Finally, a protocol is an interacting network of reactions. A protocol can either be defined by assigning a closed reaction to a channel, a parallel composition, the spawning of a new, fresh channel, or the zero protocol 0.

$$\frac{\Gamma \vdash e : \tau}{\Delta;\ \Gamma \vdash \mathsf{Ret}\ e : \emptyset \to \tau}\ \textsc{ret} \qquad \frac{\Gamma \vdash D : \tau}{\Delta;\ \Gamma \vdash \mathsf{Samp}\ D : \emptyset \to \tau}\ \textsc{Samp}$$

$$\frac{\Delta \vdash c : \tau}{\Delta;\ \Gamma \vdash \mathsf{Read}\, c : \{c\} \to \tau}\ \textsc{Read}$$

$$\frac{\Delta;\ \Gamma \vdash R_1 : I_1 \to \tau_1 \qquad \Delta;\ \Gamma, x : \tau_1 \vdash R_2 : I_2 \to \tau_2}{\Delta;\ \Gamma \vdash x : \tau_1 \leftarrow R_1;\ R_2 : I_1 \cup I_2 \to \tau_2}\ \textsc{Bind}$$

Figure 3.6: Typing for Reactions.

3.3.2 Typing

Typing $\Gamma \vdash e : \tau$ for expressions and $\Gamma \vdash D : \tau$ for distributions is standard. The typing $\Delta;\ \Gamma \vdash R : I \to \tau$ for reactions is shown in Figure 3.6; it says that R is a reaction reading from channels in I and returning a distribution of type τ, if successful. The channel context Δ declares the channels available for sending and receiving messages (we note that Δ stays unchanged throughout the typing judgement), and the variable context Γ is used for constructing messages.

Typing for programs has the form $\Delta \vdash P : I \to O$, where I and O are finite sets of channel names denoting inputs and outputs, respectively. The typing rules for IPDL are given in Figure 3.8. Rule RCT states that the inputs and outputs to a reaction $c \coloneqq R$ are given by set I of the channels R reads from except c, and the single channel c, respectively. The most subtle rule is PAR, which states that $P \mid\mid Q$ is well-typed if P and Q have disjoint outputs; and if so, then the inputs of $P \mid\mid Q$ are inputs of either P or Q (or both) that do not appear as outputs in the other program, and the outputs of $P \mid\mid Q$ are the outputs of P or Q. This rule bears a close resemblance to typed approaches to module linking; *e.g.*, as in [GM99]. We note that $\Delta \vdash I \to O$ implies $I \cap O = \emptyset$.

$$\frac{}{\Delta \vdash P = P}\ [\textsc{Refl}] \qquad \frac{\Delta \vdash P_1 \stackrel{\delta}{=} P_2}{\Delta \vdash P_2 \stackrel{\delta}{=} P_1}\ [\textsc{Sym}] \qquad \frac{\Delta \vdash P_1 \stackrel{\delta_1}{=} P_2 \qquad \Delta \vdash P_2 \stackrel{\delta_2}{=} P_3 \qquad \delta_3 = \delta_1 + \delta_2}{\Delta \vdash P_1 \stackrel{\delta_3}{=} P_3}\ [\textsc{Trans}]$$

$$\frac{\Delta_1, x : \tau_1, y : \tau_2, \Delta_2 \vdash P_1 \stackrel{\delta}{=} P_2}{\Delta_1, y : \tau_2, x : \tau_1, \Delta_2 \vdash P_1 \stackrel{\delta}{=} P_2}\ [\textsc{Exchange}] \qquad \frac{\Delta \vdash P_1 \stackrel{\delta}{=} P_2 \qquad x \notin \Delta}{\Delta, x : \tau \vdash P_1 \stackrel{\delta}{=} P_2}\ [\textsc{Weakening}]$$

$$\frac{\Delta;\ \cdot \vdash R_1 = R_2}{\Delta \vdash (c \coloneqq R_1) = (c \coloneqq R_2)}\ [\textsc{ReactCong}] \qquad \frac{\Delta \vdash P \stackrel{\delta}{=} Q : I \to O \ \ \mathsf{axiom}}{\Delta \vdash P \stackrel{\delta}{=} Q}\ [\textsc{Axiom}]$$

$$\frac{\Delta \vdash P_1 \stackrel{\delta}{=} P_2 \qquad Q\ b\text{-bounded} \qquad \delta'(k) = \delta(c_{\mathsf{comp}} * |\Delta| * \mathsf{max}(k, b))}{\Delta \vdash P_1 \mid\mid Q \stackrel{\delta'}{=} P_2 \mid\mid Q}\ [\textsc{CompCong}]$$

$$\frac{\Delta, c : \tau \vdash P \stackrel{\delta}{=} Q}{\Delta \vdash \nu c : \tau.\ P \stackrel{\delta}{=} \nu c : \tau.\ Q}\ [\textsc{NewCong}] \qquad \frac{}{\Delta \vdash P_1 \mid\mid P_2 = P_2 \mid\mid P_1}\ [\textsc{CompSym}]$$

$$\frac{}{\Delta \vdash (P_1 \mid\mid P_2) \mid\mid P_3 = P_1 \mid\mid (P_2 \mid\mid P_3)}\ [\textsc{CompAssoc}] \qquad \frac{\Delta \vdash P : C \to \emptyset \qquad C \subseteq I \cup O}{\Delta \vdash P \mid\mid Q = Q}\ [\textsc{AbsorbComp}]$$

$$\frac{c \notin P}{\Delta \vdash P \mid\mid \nu c : \tau.\ Q = \nu c : \tau.\ P \mid\mid Q}\ [\textsc{CompNew}]$$

$$\frac{}{\Delta \vdash \nu c_1 : \tau_1.\ \nu c_2 : \tau_2.\ P = \nu c_2 : \tau_1.\ \nu c_1 : \tau_2.\ P}\ [\textsc{NewExchange}]$$

$$\frac{x \notin R_2}{\begin{array}{c}\Delta \vdash \Big(c_1 \coloneqq \big(x : \tau_0 \leftarrow \mathsf{Read}\ c_0;;\ R_1\big) \mid\mid \big(c_2 \coloneqq \big(y : \tau_2 \leftarrow \mathsf{Read}\ c_1;;\ R_2\big)\big)\Big) \\ = \Big(c_1 \coloneqq \big(x : \tau_1 \leftarrow \mathsf{Read}\ c_0;; R_1\big) \mid\mid \big(c_2 \coloneqq \big(x : \tau_0 \leftarrow \mathsf{Read}\ c_0;;\ y : \tau_1 \leftarrow \mathsf{Read}\ c_1;;\ R_2\big)\big)\Big)\end{array}}\ [\textsc{ResourceTrans}]$$

$$\frac{c_1 \notin \Delta \qquad c_1 \notin R_1 \qquad c_1 \notin R_2}{\Delta \vdash \Big(\nu c_1 : \tau_1.\ c_1 \coloneqq R_1 \mid\mid c_2 \coloneqq \big(x : \tau_1 \leftarrow \mathsf{Read}\ c_1;;\ R_2\big)\Big) = \Big(c_2 \coloneqq \big(x : \tau_1 \leftarrow R_1;;\ R_2\big)\Big)}\ [\textsc{UnFold}]$$

$$\frac{\Delta \vdash \big(x : \tau_1 \leftarrow R_1;;\ y : \tau_1 \leftarrow R_1;;\ \mathsf{Ret}(x, y)\big) = \big(x : \tau_1 \leftarrow R_1;;\ \mathsf{Ret}(x, x)\big)}{\Delta \vdash \Big(c_1 \coloneqq R_1 \mid\mid c_2 \coloneqq \big(x : \tau_1 \leftarrow \mathsf{Read}\ c_1;;\ R_2\big)\Big) = \Big(c_1 \coloneqq R_1 \mid\mid c_2 \coloneqq \big(x : \tau_1 \leftarrow R_1;;\ R_2\big)\Big)}\ [\textsc{Substitution}]$$

$$\frac{x \notin R_2}{\Delta \vdash \Big(c_1 \coloneqq R_1 \mid\mid c_2 \coloneqq \big(x : \tau_1 \leftarrow \mathsf{Read}\ c_1;;\ R_2\big)\Big) = \Big(c_1 \coloneqq R_1 \mid\mid c_2 \coloneqq \big(x : \tau_1 \leftarrow R_1;;\ R_2\big)\Big)}\ [\textsc{UnusedResource}]$$

Figure 3.7: The IPDL proof system for protocol equivalence.

$$\frac{}{\Delta \vdash 0 : \emptyset \to \emptyset}\ \textsc{zero} \qquad \frac{\Delta;\ \cdot \vdash R : I \to \tau \qquad \Delta \vdash c : \tau}{\Delta \vdash (c \coloneqq R) : I \setminus \{c\} \to \{c\}}\ \textsc{rct}$$

$$\frac{\Delta, c : \tau \vdash P : I \to O \cup \{c\} \qquad c \notin I \qquad c \notin O}{\Delta \vdash \nu c : \tau.\ P : I \to O}\ \textsc{hide}$$

$$\frac{\begin{array}{c}\Delta \vdash P : I_1 \to O_1 \qquad \Delta \vdash Q : I_2 \to O_2 \\ O_1 \cap O_2 = \emptyset \qquad I = (I_1 \cup I_2) \setminus (O_1 \cup O_2) \qquad O = O_1 \cup O_2\end{array}}{\Delta \vdash P \mid\mid Q : I \to O}\ \textsc{par}$$

Figure 3.8: Typing Rules for Protocols.

Typing $\Gamma \vdash e : \tau$ for expressions and $\Gamma \vdash D : \tau$ for distributions is standard. The typing $\Delta;\ \Gamma \vdash R : I \to \tau$ for reactions is shown in Figure 3.6; it says that R is a reaction reading from channels in I and returning a distribution of type τ, if successful. The channel context Δ declares the channels available for sending and receiving messages (we note that Δ stays unchanged throughout the typing judgement), and the variable context Γ is used for constructing messages.

Typing for protocols has the form $\Delta \vdash P : I \to O$, where I and O are finite sets of channel names denoting inputs and outputs, respectively. The typing rules for IPDL are given in Figure 3.8. Rule RCT states that the inputs and outputs to a reaction $c \coloneqq R$ are given by set I of the channels R reads from except c, and the single channel c, respectively. The most subtle rule is PAR, which states that $P \mid\mid Q$ is well-typed if P and Q have disjoint outputs; and if so, then the inputs of $P \mid\mid Q$ are inputs of either P or Q (or both) that do not appear as outputs in the other program, and the outputs of $P \mid\mid Q$ are the outputs of P or Q. This rule bears a close resemblance to typed approaches to module linking; *e.g.*, as in [GM99].

3.3.3 Equational Logic

The main feature of IPDL is that we are enabled to reason *equationally* about protocols using rewrite rules. To obtain computational soundness, our equational logic tracks the adversary's run time and computational error incurred during the proof.

At the level of expressions, we assume a user-defined equational theory supporting judgements of the form $\Gamma \vdash e_1 = e_2 : \tau$ for well-typed e_1 and e_2. We assume a similar judgement $\Gamma \vdash D_1 = D_2 : \tau$ for distributions. We assume that equality (both for expressions and distributions) is well-behaved with respect to substitution. For distributions, we additionally assume the equational theory for commutative monads as well as the weakening rule:

$$\frac{\Gamma \vdash D_1 : \tau \qquad x \notin D_2}{\Gamma \vdash x : \tau \leftarrow D_1;\, D_2 = D_2}$$

We now describe the equational theory for reactions, similarly written $\Gamma \vdash R_1 = R_2$. Most rules are standard, and encode the equational theory of commutative monads. We highlight the most interesting rules here, and leave the rest for the appendix in Figure 3.10. We first have two rules for relating the monadic structure of reactions and distributions:

$$\frac{}{\begin{array}{l}\Delta;\ \Gamma \vdash \mathsf{Samp}\ \big(x : \tau_1 \leftarrow D_1;\, D_2\big) \\ = \big(x : \tau_1 \leftarrow \mathsf{Samp}\ D_1;\, \mathsf{Samp}\ D_2\big)\end{array}}\ [\textsc{SampBind}]$$

$$\frac{}{\Delta;\ \Gamma \vdash \mathsf{Samp}\ 1_e = \mathsf{Ret}\ e}\ [\textsc{SampRet}]$$

Next, we have the *contraction* rule, stating that reading from the same channel

twice is equivalent to reading it once:

$$\frac{}{\Delta;\ \Gamma \vdash \big(x : \tau_1 \leftarrow \mathsf{Read}\ c;\ y : \tau_1 \leftarrow \mathsf{Read}\ c;\ R\big) = \big(x : \tau_1 \leftarrow \mathsf{Read}\,c;\ R[y/x]\big)}\ [\textsc{Contr}]$$

For protocols, we have the judgement $\Delta \vdash P \overset{\delta}{=} Q$, which states (informally) that any computational adversary (called the "environment" in UC [Can01]) with running time at most k cannot distinguish interaction with P from Q with advantage greater than $\delta(k)$. Here, $\delta : \mathbb{N} \to \mathbb{R}$ is an *error*, which maps adversary running times to an upper bound on distinguishing advantage. Since greater computation power allows the adversary to gain distinguishing advantage, we assume throughout that δ is an increasing function. We allow user defined axioms for (approximate) program equivalences, which are used to define assumptions on the security of a cryptosystem or hardness assumption. The equational theory of programs is given in Figure 3.7. Our judgement is directly inspired from the work on Task-PIOA [CCK+07]. We will write $\Delta \vdash P = Q$ for the special case of exact equality when δ is the constant zero function.

We now discuss a selection of the rules from Figure 3.7. The most important rule is [CompCong], which states that if P_1 is approximately equivalent to P_2, then for any Q (of the appropriate type), $P_1||Q$ is approximately equivalent to $P_2||Q$. This is the rule that enables modular reasoning in IPDL. To reason about the error incurred by using this rule, we define the notion of *b-boundedness*: an IPDL program Q is b-bounded if, intuitively, its behavior can be simulated with a probabilistic algorithm with at most b time steps (defined formally in 3.9). Given this notion, the rule [CompCong] changes the attacker's running time to $O(|\Delta| * \mathsf{max}(k, b))$; this is because the attacker for P_1 (and P_2) must simulate the behavior of Q, which increases its running time.

Similarly, [HideCong] states that $\stackrel{\delta}{=}$ forms a congruence under the spawning of a new channel. Rule [HideComp] states that our hiding operator commutes with parallel composition, under the assumption that no extra channels are affected. We note that this rule is closely related to the concept of *scope extrusion* in the π-calculus (i.e., as in [MPW92]).

Rule [Resource Trans] states that if c_0 is an input to the reaction defining c_1, and c_1 is an input to the reaction defining c_2, then we may freely add c_0 to the inputs of c_2.

The last three rules specify under what conditions we may replace a read from a channel c by the reaction R defining it. The first scenario in which this is sound is when the reaction R is non-probabilistic – this is rule [Subst]. The second case when we may replace a read from c by the reaction R is when the value read from c is in fact never used – this is rule [UnusedResource]. Lastly, we may perform this replacement if the channel c is read from in precisely one place – this is rule [UnFold]. Reading from right to left, this rule also serves to relate the monadic bind at the level of reactions to the parallel composition of programs.

3.3.4 Semantics

We now informally describe our semantic model for IPDL, as well as our proof of soundness. Technical details can be found in Section 3.9. We interpret each IPDL program P as an *I/O automaton*, which is a probabilistic transition system that can deliver outputs and react to inputs.

IPDL equivalence judgements are proven sound relative to a semantic adver-

sary, who is formulated as a *dual* automaton (along with some extra data). The adversary is responsible for interacting with the protocol, choosing the order in which outputs occur, and eventually outputting a decision bit after some k number of rounds. We define k-bounded adversaries to be those which run for k rounds, and each round may only take k time steps in its internal transition functions. [3] Given a k-bounded adversary A and IPDL program P, we write $A(P)$ to mean the distribution on booleans defined by letting A and P interact for k rounds, and observing the decision bit of A. We stress that our automata model, and thus our adversarial model, is *not* limited to the syntax of IPDL, but instead can describe arbitrary behaviors, including conditional branching and other forms of control flow.

Soundness. Our soundness theorem states that whenever $\Delta \vdash P \stackrel{\delta}{=} Q$, any k-bounded adversary has an advantage at most $\delta(k)$ in distinguishing P from Q. Note here that δ is derived from a proof in our logic, and will consist of the sum of a number of errors incurred by applying IPDL axioms.

Theorem 2. *Suppose $\Delta \vdash P : I \to O$ and $\Delta \vdash Q : I \to O$ are two IPDL programs such that $\Delta \vdash P \stackrel{\delta}{=} Q$. Then for all k-bounded adversaries A, $|\Pr[A(P) = 1] - \Pr[A(Q) = 1]| \leq \delta(k)$.*

The proof of Theorem 2 is given in Section 3.9. We now give some detail about the proof. For the rules with error zero, we employ *bisimulation* arguments, to directly show the two protocols have the same behaviors. For the [CompCong] rule, we must transform an arbitrary adversary A for the composition $P_1||Q$ to an adversary $A||Q$ for the protocol P_1. The bound $c_{\mathsf{comp}} * |\Delta| * \mathsf{max}(k, b)$ comes

[3]Without loss of generality we take k be the upper bound on the adversary's running time per round and the number of rounds.

directly from the proof.

3.4 Parameterized Programs and Computational Soundness

In this section, we consider *parameterized protocols*: families of IPDL protocols $\{P_j\}$, ranging over some index set j. Parametrization in IPDL can be used to encode the number of parties (e.g., our n-party coin flip with abort example), number of reactive sessions (e.g., our secure network example), as well as for ranging over more complicated index sets (e.g., for expressing arbitrary circuits in our GMW example). In Section 3.4.1, we describe how Theorem 2 applies to PPT adversaries and computational indistinguishability. In Section 3.4.2, we describe some derived equational rules for reasoning about parameterized programs.

3.4.1 Soundness for PPT Adversaries

While our core logic in Section 3.3 does not reason about parameterization, we show here that we can use the logic to reason about protocols which depend on a security parameter. In this section, we consider parameterized IPDL protocols of the form $\{P^\lambda\}$, parameterized by a security parameter $\lambda \in \mathbb{N}$. Similarly, we consider families of channel contexts $\{\Delta^\lambda\}$, and families of errors $\{\delta^\lambda\}$.

We lift computational indistinguishability to parameterized IPDL protocols in a straightforward manner. First, note that the family of errors $\{\delta^\lambda\}$ can be seen as a two-place function: the first argument is the security parameter, while the second is the adversary's running time. Correspondingly, we say that the family $\{\delta^\lambda\}$ is *negligible* if for all polynomials p, $\delta^\lambda(p(\lambda))$ is a negligible function of λ. We

define *PPT adversaries* to be families of adversaries $\{A^\lambda\}$ such that there exists a polynomial p where A^λ is $p(\lambda)$-bounded. Then, we have the following corollary immediate from Theorem 2:

Corollary 1. *Suppose that* $\Delta^\lambda \vdash P^\lambda \overset{\delta^\lambda}{=} Q^\lambda$, *and the family* $\{\delta^\lambda\}$ *is negligible. Then, for any PPT adversary* $\{A^\lambda\}$, *the quantity*

$$|\Pr[A^\lambda(P^\lambda) = 1] - \Pr[A^\lambda(Q^\lambda) = 1]|$$

is a negligible function of λ.

The (parameterized) error parameter $\{\delta^\lambda\}$ may grow in IPDL for two reasons: either by applying an axiom, or by applying the [CompCong] rule, which grows the adversary's runtime by the runtime of the common context. As long as the proof has polynomially many rewrites, the error family for each axiom is negligible, and the runtime of each context for the [CompCong] rule is polynomial, we are guaranteed that $\{\delta^\lambda\}$ is a negligible family of errors.

3.4.2 Derived IPDL Constructs and Equations

We now turn to reasoning principles in IPDL for parameterized programs. To build parameterized programs systematically, we introduce two pieces of syntactic sugar on top of the core IPDL syntax. Let $n \in \mathbb{N}$ be a variable in the ambient meta-logic. First, *vectorized channel generation*, $\nu\ \vec{v}^n : \tau.\ P$, generates a fresh vector of channels $\{v_i\}_{i \in \{1 \dots n\}}$ for use in protocol P. Second is the notation $||_{j \in J}\, P_j$ for composing a family of protocols P_j together, for all j in some finite index set J. Both pieces of syntactic sugar are reflected in our Coq formalization, as seen e.g. in Section 3.2. While each P_j must be an IPDL program, we emphasize that the

mapping $j \mapsto P_j$ and the index set J are all defined in the ambient logic and may make use of arbitrary set theoretic reasoning. This reflects our Coq formalization, which uses the `bigop` and `fintype` libraries from ssreflect [GMT16] to manage parameterized composition and index sets of bounded natural numbers.

Derived IPDL rules. We additionally introduce a number of derived rules for IPDL for reasoning about parameterized programs. We describe the most important rules here, and leave the rest for Figure 3.11 in the Appendix.

One of our most widely used rules is [EqBig], which states that parameterized composition is a congruence, which states that in order to prove that $||_{j \in J} \; P_j$ is equivalent to $||_{j \in J} \; Q_j$, it suffices to show that P_i is equivalent to Q_i for each $i \in J$.

Next, we have a number of rules involving manipulating the index sets for parameterized composition, directly inspired from the `bigop` library. Most importantly, we have that we can arbitrarily split up compositions: any composition $||_{j \in J} \; P_j$ can be split into the composition of $||_{j \in J \cap K} P_j$ and $||_{j \in J \cap \widetilde{K}} P_j$, where $\widetilde{K}$ is the complement of K. We additionally have that composition is compatible with parameterized composition: that is, $||_{j \in J} P_j$ composed with $||_{j \in J} Q_j$ is equivalent to $||_{j \in J}(P_j \mid\mid Q_j)$.

Finally, we describe our most powerful rule, [Hybrid]:

$$\frac{\begin{array}{c}\forall k < n, \Gamma \vdash (\bigg|\bigg|_{j<k} P_j)||R = (\bigg|\bigg|_{j<k} Q_j)||R \\ \Rightarrow \Gamma \vdash (\bigg|\bigg|_{j<k} P_j)||P_k||R = (\bigg|\bigg|_{j<k} P_j)||Q_k||R\end{array}}{\Gamma \vdash (\bigg|\bigg|_{j<n} P_j)||R = (\bigg|\bigg|_{j<n} Q_j)||R} \;[\textsc{Hybrid}]$$

This rule states that to transform one composition of a protocol family $\{P_j\}$

into another one $\{Q_j\}$ (say, for the index set $\{0 \dots n\}$) in the presence of a common context R, we may instead prove that for any $k < n$, if we have the composition of $\{P_j\}_{j \leq k}$ along with R, we may rewrite the last P_k to Q_k.

3.5 Encoding in Coq

In this section, we describe our encoding of IPDL in Coq.

Basic syntax. First, we describe how we embed types, expressions, and distributions. Our encoding is *shallow*, meaning that expressions and functions in IPDL are represented using their native Coq analogues. IPDL types are given by an inductive Coq type `type := TBool | TUnit | TBits (n : nat) | TPair (t1 t2 : type)`. As is standard, IPDL types in Coq come equipped with a function `interpType : type -> Type`, which maps each IPDL type into its interpretation as a Coq type. This mapping is standard; we use the `tuple` library of ssreflect [GMT16] to model bitstrings. We model distributions syntactically, as finite boolean decision trees.

We now turn to channels, reactions, and IPDL protocols:

```
1  Axiom chan : type -> Type.
2
3  Definition Chan := {t : type & chan t}.
4
5  Inductive rxn : type -> Type :=
6  | Samp {t : type} : Dist t -> rxn t
7  | Ret {t : type} : t -> rxn t
8  | Read {t : type} (c : chan t) : rxn t
```

```
9  | Bind {t1 t2 : type} : rxn t1
10                         -> (t1 -> rxn t2)
11                         -> rxn t2.
12
13 Inductive WfRxn : list Chan -> rxn t -> Prop := ...
14
15 Inductive ipdl : Type :=
16 | prot0 : ipdl
17 | Out {t} (c : chan t) : rxn t -> ipdl
18 | Par : ipdl -> ipdl -> ipdl
19 | New t : (chan t -> ipdl) -> ipdl.
```

To model channel binding in Coq, we opt for the *weak HOAS* approach [CS13], which models channels through a `type`-indexed abstract Coq type, given by an axiom. Since channels have type tags, we use a dependent sum to speak about the collection of all channels, `Chan`. Reactions are encoded monadically, as in Section 3.3. For ease of use, we adopt the usual monadic syntax `x <-- r ;; k` to represent monadic binds. For convenience, we do *not* enforce that reactions are well-typed through the Coq type system, but instead embed the typing judgement in the proposition `WfRxn`: if `WfRxn G r` holds, then `r` performs exactly the reads as specified through the sequence of channels `G`. This encoding is faithful to the syntax in Section 3.3, which does not allow pattern matching or branching at the level of reactions: since `WfRxn` enforces that all `Read` effects must be identical in all branches, the reaction is equivalent to one without reaction-level branching.

Finally, in Line 17, we encode IPDL programs through the datatype `ipdl`. In the datatype we use syntax `Out` and `Par`, but these are also captured by the Coq notations `::=` and `||` respectively. Since we use weak HOAS, we are enabled to encode channel binding in `New` through an ordinary Coq function. We allow use of the more

natural syntax `x <- new t ;; P`. As is standard [CS13], this encoding requires us to additionally encode the predicate `chansOf : ipdl -> Chan -> Prop` to model the free variables of IPDL programs, since we cannot soundly assume decidable equality of channels. We provide tactics for (mostly) automatically discharging goals involving `chansOf` and related constructions.

Typing judgements. We similarly encode the typing judgement of IPDL programs through an inductive datatype. It follows the same rules as in Section 3.3, except for the ν operator for local channel generation. Since we cannot directly assume a specific channel is globally fresh in Coq (e.g., as in nominal calculi [AGM+04]), we parameterize the typing judgement by a finite collection of channels `X` and assume that the new channel `c` is only fresh *against* the channels in `X`.

Equational theory. We encode equivalences of reactions and IPDL programs through the inductive datatypes `EqRxn` and `EqProt`, respectively. Our libraries for IPDL make heavy use of Coq's support for setoid rewriting to enable easy proofs. Their definitions closely follow the rules in Section 3.3, except for the following differences: 1) we do not reason about a separate monadic bind operator for distributions and reactions, but only the one for reactions; 2) we give ourselves the liberty to include a few derived rules for managing channel dependencies and reasoning about probability distributions; 3) our Coq implementation currently does not reason about computational error (i.e., the δ parameter). We plan on introducing reasoning about computational error and protocol run time to a future iteration of our implementation.

Encoding of parameterized protocols. One of the major strenghts of our Coq encoding is that we are able to write arbitrary Coq programs to generate IPDL protocols, effectively using Coq as a meta-programming environment for IPDL. Following Section 3.4, we use the `bigop` library from ssreflect [GMT16]: we model parameterized composition $||_{j \in J} P_j$ using the syntax `\||_(j < n | p j) f j` where `f` is a function of type `'I_n -> ipdl`, and `'I_n` is the type of natural numbers less than `n`. While we do have support for more general index sets – as in the bigop library, we support using sequences for index sets, as well as general finite types – we only use bounded natural numbers for our proof developments. We model parameterized channel generation $\nu\ \overrightarrow{v}^n : \tau.\ P$ through the notation `x <- newvec n @ t ;; P`, which is defined by induction on `n`. Here, the type of `x` is `n.-tuple (chan t)`, or the type of lists of length exactly `n`. This type is borrowed from the `tuple` library of `ssreflect`. All the derived rules in Section 3.4 are implemented as lemmas in Coq, proven from the basic equational rules of IPDL.

3.6 Case Studies

In this section, we briefly describe all case studies we have mechanized in IPDL. We defer more detailed description to Appendix 3.10, including Coq sources for selected protocols.

3.6.1 Case Studies

Multi-use secure network. Our first case study is a multi-use secure network, and we refer the reader to the earlier Section 3.2 for more details. [4]

Semi-honest OT constructions. In (1-out-of-2) OT, there is a *sender* who has a pair of messages m_0 and m_1, and a *receiver* who has an index bit i. The ideal functionality for OT receives these three protocol inputs, and returns to the receiver m_i. The sender receives no protocol output. All OT protocols we consider are in the *semi-honest* setting, in which the adversary observes corrupted parties' private data, but cannot harm integrity. We encode semi-honest security in IPDL by *annotating* each corrupted party with explicit leakage channels for the adversary, and extending their protocol code appropriately.

We verify three OT protocols: OT from trapdoor permutations, the OT construction by Goldreich et al. [GMW87], a simple preprocessing scheme for OT [pre, Bea95], and construction of 1-out-of-4 OT from 1-out-of-2 OT [14o, NP99]. All OT constructions are roughly of the same complexity, and emphasize different parts of the system; in particular, the proofs for OT often require complex *rerandomization* steps, in which we transform uniform randomness to eliminate channel dependencies. More details about all OT protocols is given in Section 3.10.1.

Semi-honest, two-party GMW protocol. Our second major case study for IPDL is the GMW protocol [GMW87], a semi-honest secure multiparty computation protocol over bits based on secret sharing. First, we model boolean circuits

[4] While our example reasons only about a fixed size of message, it is straightforward to adapt our example to the variable length case by considering a type of messages *up to* a given length.

in Coq as follows:

```
1  Inductive Op (A B k : nat) :=
2    | InA : 'I_A -> Op A B k
3    | InB : 'I_B -> Op A B k
4    | And : 'I_k -> 'I_k -> Op A B k
5    | Xor : 'I_k -> 'I_k -> Op A B k
6    | Not : 'I_k -> Op A B k.
7
8  Definition Circ A B n := forall (k : 'I_n), Op A B k.
9
10 Definition CircOutputs n o := o.-tuple ('I_n).
```

Above, we first introduce the type `Op A B k` of *operations* which may make use of all of Alice's inputs (numbered $0 \ldots A-1$), Bob's inputs (numbered $0 \ldots B-1$), and all wire IDs from 0 to $k-1$. We then define a circuit to be a mapping from all wire IDs $j < n$ to an operation which may make use of all wires up to $j-1$. This definition of boolean circuits is equivalent to a more ordinary, inductively defined variant, but is nicer to work with in proofs. Our circuits support multiple outputs, which are modeled through a finite mapping from wire IDs to output IDs, which we define using a `o.-tuple`, or fixed-size list of length `o`. (We assume the same outputs for each party.)

We describe how we encode the ideal/real protocol of GMW in Appendix 3.10.3.

Coin flip with abort. This protocol allows n mutually distrusting parties to collaboratively generate fair randomness [Blu83]. To do so, each party locally generates a bitstring uniformly from $\{0, 1\}^k$ and sends a cryptographic commitment of the bitstring to all other parties. We assume a broadcast channel for the commit-

ments to prevent equivocation. Once all other commitments have been collected, each party opens their respective bit, and all parties output the collective XOR of all opened bitstrings. We model the commitment and broadcast channels using a standard UC commitment functionality, which prevents equivocation by construction. Our proof is secure in the malicious model. Modeling details about the protocol are given in depth in Section 3.10.4.

3.6.2 Proof Effort

In Figure 3.9, we outline the lines of code needed for each case studies considered. Our simplest example is our secure network example from Section 3.2, which consists of a number of simple rewriting steps along with the application of two IPDL axioms. Our OT examples, while simple to define, take a modest effort to prove, with the largest proof being the 1-4 OT at 749 lines of code. While the number of lines is moderate, the complexity of the proof script is low: most of the lines consist of repetitive tactic invocations as well as intermediate rewriting steps being explicitly defined as hybrids. It is likely that proofs like these can be further automated with additional engineering effort. Our most complex examples are the GMW protocol and the n-Coin Flip, both of which have proofs of less than 2000 lines of code. Out of the 1995 lines of code for the n-Coin Flip, 345 of them were definitions of intermediate hybrids while the rest were either proof scripts or auxiliary lemmas.

We compare our proof effort with related mechanization efforts in Section **??**.

Case study	LoC (Definitions)	LoC (Proof)
Secure Network	73	122
Trapdoor OT	75	568
Preprocessing OT	40	249
1-4 OT	88	749
n-Coin Flip	100	1995
GMW	324	1397

Figure 3.9: Case studies considered and lines of code.

3.7 Additional Related Work

More detailed comparison with Bana-Comon. A promising direction ([BCL12], [BCL14], [BCEO19]) for protocol verification is initiated by Bana and Comon, where the attacker is not limited by interacting with idealized cryptography, but instead constrained by a number of logical axioms which state what the attacker is not able to do. While this framework has made advances compared to symbolic systems, there is to date no publicly available mechanization of their framework. While some IPDL proofs can likely be automated using these techniques, we anticipate that our more complicated parameterized proofs (e.g., inducting over circuits, handling n parties) would require significant engineering effort similar to ours to mechanize using their framework. Indeed, the strength of our parameterized approach is derived from the usage of a general-purpose theorem prover for defining parameterized protocols; this has no counterpart yet in the Bana-Comon framework.

Frameworks for cryptographic protocols. In the cryptography literature, Universal Composability [Can01] and Constructive Cryptography [Mau11] are the two dominant definitional frameworks for simulation-based security. Several automata-based frameworks also exist, such as [BPW07] and [CCK$^+$18], which,

while similar in spirit, aim for a more formal treatment. Additionally, some works use process calculi to model computational cryptographic protocols, such as [MRST01]. A recent effort to formalize the semantics of UC is ILC [LHM19]. While a useful step towards giving formal reasoning support for UC, it does not yet provide support for verification. Additionally, a number of works formalize standalone (non-UC) proofs of interactive protocol security using special-purpose embeddings of protocols into Easycrypt. For example, [HKO+18] gives an on-paper reduction of the security of Maurer's MPC protocol [Mau06] to a certain trace property which is directly verified in Easycrypt

An interesting alternative framework is given in Micciancio and Tessaro [MT13] (hereafter M&T), where they use *complete partial orders* to represent cryptographic protocols as the least fixed point of a recursive set of equations. There is some amount of conceptual overlap between M&T and IPDL: their monotonicity requirement (that further inputs can only create more outputs) is similar to our encoding of protocols, which cannot make use of non-determinism through observing scheduling decisions. However, the framework is not mechanized, and cannot reason about computational error.

3.8 Future Work

The first direction of future work for IPDL is to increase its expressivity while still retaining the equational flavor of its logic. For example, support for adaptive corruption and more flexible control flows would be interesting.

An exciting future direction is to integrate IPDL with an underlying battle-hardened cryptographic proof system (such as EasyCrypt [BGHZ11]) which may

enable more expressiveness, thus achieving ease-of-use and generality simultaneously. Other exciting future directions include to provide a greater degree of proof automation, compiling IPDL programs to executable code (e.g., in C) and proving the correctness of the compilation. We anticipate that IPDL could also be seen as an equational interface for more expressive tools such as EasyUC [CSV19] or CryptHOL [LSBM19]. Additionally, it would be interesting to combine IPDL with ILC [LHM19], n programming language for UC semantics.

3.9 Semantics

In this section, we give semantics to well-typed IPDL programs. Every type can be straightforwardly interpreted as the set of bitstrings of a certain length; if c is a channel declared in a typing context Δ, we will write $|c|$ for the length of bitstrings assigned to c and by abuse of notation we will use natural numbers n to stand for the set of bitstrings of length n. Analogously to types, we interpret variable typing contexts Γ as natural numbers, again corresponding to a set of bitstrings of the specified length. We interpret channel typing contexts Δ as mappings of channel names c to natural numbers $|c|$, specifying that a given channel will carry bitstrings of the given length.

We first describe our semantic model of I/O automata. Let Δ be a channel context, as above. Then, an I/O automaton consists of the following data, for I and O disjoint sets of channels which form Δ:

- a finite set St of *states*
- a start state $\mathsf{s}_\star : \mathsf{St}$

- a *valuation function* $\mathsf{St} \times (o : O) \to 1 + |o|$
- an *input transition function* $\mathsf{St} \times (\Sigma_{i:I}|i|) \to \mathcal{D}(\mathsf{St})$
- an *output transition function* $\mathsf{St} \times (o : O) \to \mathcal{D}(\mathsf{St})$

The valuation function tells us the value of the output o, if any, in a given state. The input transition function takes a state s and an assignment $i \coloneqq v$, where v is a value of the correct type, and returns a distribution on states. The output transition function takes a state s and an output o, and returns a distribution on states.

We write $s|_o$ for the value of the output o in state s. Given a state s, we write $\langle i \coloneqq v \rangle\, s$ and $\langle o \rangle\, s$ for the distribution resulting from performing the specified input or output. Using the monadic bind, we can generalize this to distributions σ as $\langle i \coloneqq v \rangle\, \sigma$ and $\langle o \rangle\, \sigma$.

There are several canonical ways to produce new protocols from old ones. For our purposes, the following three are important:

- Given a protocol P in the typing context $\Delta, c \mapsto |c|$ with an output c, we can restrict P in the obvious way to obtain a new protocol $\nu c : |c|.\ P$ in the reduced typing context Δ. The new protocol has the same states as P but its valuation and output transition functions are now restricted to channels from Δ.
- Given a protocol P with an output o, we define a new protocol $P|_o$ as follows: we have the same states as in P but both before and after performing any input assignment or output computation, we perform o.

- Given two protocols P and Q in the same typing context with inputs I_1 and I_2 and outputs O_1 and O_2 such that $O_1 \cap O_2 = \emptyset$, we can define a new protocol $P \mid\mid Q$ as follows:
  - the states are pairs (s, t), where s and t are states of P and Q, respectively
  - the start state is $(\mathsf{s}_\star, \mathsf{t}_\star)$, where $\mathsf{s}_\star$ and $\mathsf{t}_\star$ are the start states of P and Q, respectively
  - the valuation is defined as
    * $(s, t)|_o \coloneqq s|_o$ if $o \in O_1$
    * $(s, t)|_o \coloneqq t|_o$ if $o \in O_2$
  - to perform an input assignment $i \coloneqq v$ in (s, t), we perform $i \coloneqq v$ in s and/or t as applicable:
    * If $i \in I_1$ and $i \notin I_2$, the resulting distribution is $\langle i \coloneqq v \rangle\, s \times 1_t$.
    * If $i \notin I_1$ and $i \in I_2$, the resulting distribution is $1_s \times \langle i \coloneqq v \rangle\, t$.
    * If $i \in I_1$ and $i \in I_2$, the resulting distribution is $\langle i \coloneqq v \rangle\, s \times \langle i \coloneqq v \rangle\, t$.
  - to compute an output o in (s, t), we compute o in s or t, accordingly as to whether o is an output of P or Q. If applicable, we forward the result to the other protocol:
    * If $o \in O_1$ and $o \notin I_2$, the resulting distribution is $\langle o \rangle\, s \times 1_t$.
    * If $o \in O_2$ and $o \notin I_1$, the resulting distribution is $1_s \times \langle o \rangle\, t$.
    * If $o \in O_1$ and $o \in I_2$, we draw a new state r from $\langle o \rangle\, s$. If $r|_o = u$ for some $u \in |o|$, the resulting distribution is $1_r \times \langle o \coloneqq u \rangle\, t$, otherwise $1_r \times 1_t$.
    * If $o \in O_2$ and $o \in I_1$, we draw a new state r from $\langle o \rangle\, t$. If $r|_o = u$ for

some $u \in |o|$, the resulting distribution is $\langle o \coloneqq u \rangle\, s \times 1_r$, otherwise $1_s \times 1_r$.

For our soundness result, we also need to introduce the concept of a *channel embedding*. Given two contexts Δ and Δ', a channel embedding $\theta : \Delta \to \Delta'$ is an injective function from the indices in Δ to Δ' which preverse channel typing.

We are now ready to describe the interpretation of an IPDL program $\Delta \vdash P : I \to O$. We will proceed in two steps: in the first step we define a *one-step* interpretation $[[-]]_1$ using the above constructs, and in the second step we define the final interpretation $[[-]]$ in terms of the one-step interpretation. When asked to perform an output o, the one-step interpretation attempts to first compute all the hidden channels that o may directly or indirectly depend on; however, it does not yet attempt to compute any output channels, even those that o directly depends on. This is the job for the final interpretation.

Formally, we define $[[P]]_1$ by induction on the structure of P as follows:

- $[[0]]_1$ has a unique state and no output actions
- $[[o \coloneqq R : \tau]]_1$ has mappings of channel names c to bitstrings of length $1 + |c|$ as states, where c is either an input to the reaction R or the output o. The start state maps every channel name to $\bot$. Performing an input assignment $i \coloneqq v$ in a state s amounts to setting the value of i in s to v, if not already defined. To compute an output o in a state s, we check if c is already defined in s; if so, we do nothing. Otherwise we execute the reaction R in s (yielding $\bot$ if any of the required input channels are not defined in s).
- $[[P \mid\mid Q]]_1 \coloneqq [[P]]_1 \mid\mid [[Q]]_1$
- $[[\nu c : \tau.\ P]]_1 \coloneqq \nu c : [[\tau]].\ [[P]]_1|_c$

It is now possible to prove that for any output o, we have $[[P]]_1|_o|_o = [[P]]_1|_o$ and for any two outputs o_1, o_2 we have $[[P]]_1|_{o_1}|_{o_2} = [[P]]_1|_{o_2}|_{o_1}$. If $\{o_0, \ldots, o_n\}$ are the outputs of P, we define the final interpretation of P to be $[[P]] \coloneqq [[P]]_1|_{o_0} \ldots |_{o_n}$. Thus, if an output o_2 depends on an output o_1, in the final interpretation the computation of o_2 will take into account the result of the computation of o_2, if any.

Another important property of our semantics of IPDL is that the $|_o$ operator is compatible with composition, in the following sense:

Lemma 11. *For any IPDL programs P and Q with disjoint outputs, and any output o of P,* $([[P]]_1||[[Q]]_1)|_o = ([[P]]_1|_o||[[Q]]_1)|_o$.

The above lemma can be verified manually by enumerating the cases in which o may fire in each state of the composition, and whether o is an input of Q. A similar result holds for the symmetric case where we add $|_o$ to Q, instead of P.

By applying the above lemma many times, we have that $[[P||Q]] = ([[P]]||[[Q]])|_{o_1,\ldots,o_k}$, where $o_1, \ldots, o_k$ is an arbitrary enumeration of the outputs of P and Q.

Boundedness for IPDL programs In order to reason in a computationally sound manner, we need to estimate the running times of IPDL protocols. We say that an IPDL protocol Q is b-bounded when the size of the state of $[[Q]]$ (in bits) is bounded by b, and for each transition function of the final interpretation $[[Q]]$, there exists a probabilistic Turing machine that runs for at most b time steps which computes this function.

3.9.1 Adversaries

An *environment* or *adversary* for a protocol P with inputs I and outputs O is specified by:

- a dual adversary protocol A with states St, inputs $I' \subseteq O$, and outputs $I \subseteq O'$
- a *stepping function* $\mathsf{St} \to \mathcal{D}(\mathsf{St})$
- a *decision function* $\mathsf{St} \to \mathbb{B}$
- an *accept function* $(O \cup O') \to \mathsf{St} \to \mathbb{B}$
- a *schedule* $\{0, \ldots, k-1\} \to (O \cup O')$

In particular, the adversary does not have access to the states of the protocol. At each step, the schedule decides on performing one of the outputs (of either the protocol or the adversary). In each case, the adversary probabilistically steps to a new state as given by the stepping function. The adversary has the ability to refuse the execution of any scheduled channel.

We now describe how the adversary interacts with a semantic protocol P. Given a state s of the adversary, a state t of the protocol, and an output $o : O \cup O'$ to be performed, we probabilistically determine a new adversary state and a new protocol state as follows:

- We call the stepping function in state s and draw a new adversary state r from the resulting distribution.
- If the accept function for o at s is false, the resulting distribution is $1_r \times 1_t$.

- Otherwise we ask the composed protocol $A \mid\mid P$ to perform o in the state (r, t), to obtain the resulting distribution $\langle o \rangle (r, t)$.

We can lift this single execution step to act on distributions of pairs (s, t) of adversary and protocol states. We inductively perform this lifted execution step on each scheduled channel to obtain a final distribution on pairs of adversary and protocol states. At this point we call the decision function to turn the resulting distribution on adversary states to a distribution on booleans. This distribution, denoted $A(P)$, will be the result of the interaction between the adversary and the protocol.

We call an adversary *k-bounded* if:

- the states have length at most k
- the schedule has length at most k
- for each i, the corresponding input transition function is k-bounded
- for each o, the corresponding output transition function is k-bounded
- for each i or o, the corresponding accept function is k-bounded
- the stepping function is k-bounded
- the decision function is k-bounded

We define a *bisimulation* between two comparable protocols P and Q as a binary relation $\sim$ on distributions on the states of P and Q respectively, satisfying the following conditions:

- Initial: the unit distributions on the respective initial states of P and Q are related by $\sim$

$$\frac{}{\Delta;\ \Gamma \vdash R = R}\ [\textsc{Refl}] \qquad \frac{\Delta;\ \Gamma \vdash R_1 = R_2}{\Delta;\ \Gamma \vdash R_2 = R_1}\ [\textsc{Sym}]$$

$$\frac{\Delta;\ \Gamma \vdash R_1 = R_2 \qquad \Delta;\ \Gamma \vdash R_2 = R_3}{\Delta;\ \Gamma \vdash R_1 = R_3}\ [\textsc{Trans}]$$

$$\frac{\Gamma \vdash e_1 = e_2}{\Delta;\ \Gamma \vdash \mathsf{Ret} e_1 = \mathsf{Ret} e_2}\ [\textsc{RetCong}]$$

$$\frac{\Gamma \vdash D_1 = D_2}{\Delta;\ \Gamma \vdash \mathsf{sample}\ D_1 = \mathsf{sample}\ D_2}\ [\textsc{SampleCong}]$$

$$\frac{\Delta;\ \Gamma \vdash R_1 = R_3 \qquad \Delta;\ \Gamma, x : \tau_1 \vdash R_2 = R_4}{\Delta;\ \Gamma \vdash \big(x : \tau_1 \leftarrow R_1;; R_2\big) = \big(x : \tau_1 \leftarrow R_3;;\ R_4\big)}\ [\textsc{BindCong}]$$

$$\frac{}{\Delta;\ \Gamma \vdash \mathsf{sample}\ 1_e = \mathsf{Ret} e}\ [\textsc{SampleRet}]$$

$$\frac{}{\Delta;\ \Gamma \vdash \mathsf{sample}\ \big(x : \tau_1 \leftarrow D_1;\ D_2\big) = \big(x : \tau_1 \leftarrow \mathsf{sample}\ D_1;;\ \mathsf{sample}\ D_2\big)}\ [\textsc{SampleBind}]$$

$$\frac{}{\Delta;\ \Gamma \vdash \big(y : \tau_2 \leftarrow (x : \tau_1 \leftarrow R_1;;\ R_2);;\ R_3\big) = \big(x : \tau_1 \leftarrow R_1;;\ y : \tau_2 \leftarrow R_2;;\ R_3\big)}\ [\textsc{BindBind}]$$

$$\frac{}{\Delta;\ \Gamma \vdash \big(x : \tau_1 \leftarrow \mathsf{Ret} e;;\ R\big) = [e/x]R}\ [\textsc{RetBind}]$$

$$\frac{}{\Delta;\ \Gamma \vdash \big(x : \tau \leftarrow R;;\ \mathsf{Ret} x\big) = R}\ [\textsc{BindRet}]$$

$$\frac{x \notin R_2 \qquad y \notin R_1}{\Delta;\ \Gamma \vdash \big(x : \tau_1 \leftarrow R_1;;\ y : \tau_2 \leftarrow R_2;;\ R_3\big) = \big(y : \tau_2 \leftarrow R_2;;\ x : \tau_1 \leftarrow R_1;;\ R_3\big)}\ [\textsc{Exchange}]$$

$$\frac{}{\Delta;\ \Gamma \vdash \big(x : \tau_1 \leftarrow \mathsf{read}\ c;;\ y : \tau_1 \leftarrow \mathsf{read}\ c;;\ R\big) = \big(x : \tau_1 \leftarrow \mathsf{read}\, c;;\ [y/x]R\big)}\ [\textsc{Contr}]$$

Figure 3.10: Equivalence of Reactions in IPDL.

- Inputs: if $\mu \sim \eta$, then for any input assignment $i \coloneqq v$ there exist (convex) coefficients $c_1, \ldots, c_n$ and distributions $\mu_1, \ldots, \mu_n$, $\eta_1, \ldots, \eta_n$ such that $\mu_k \sim \eta_k$ for each $k = 1, \ldots, n$ and

$$\langle i \coloneqq v \rangle \mu = \Sigma_{k:=1\ldots n} c_k \mu_k = \Sigma_{k:=1\ldots n} c_k \eta_k = \langle i \coloneqq v \rangle \eta$$

- Outputs: if $\mu \sim \eta$, then for any output o there exist (convex) coefficients $c_1, \ldots, c_n$ and distributions $\mu_1, \ldots, \mu_n$, $\eta_1, \ldots, \eta_n$ such that $\mu_k \sim \eta_k$ for each $k = 1, \ldots, n$ and

$$\langle o \rangle \mu = \Sigma_{k:=1\ldots n} c_k \mu_k = \Sigma_{k:=1\ldots n} c_k \eta_k = \langle o \rangle \eta$$

Any bisimulation between P and Q is also a bisimulation between $P|_o$ and $Q|_o$, and likewise between $\nu c : \tau.\ P$ and $\nu c : \tau.\ Q$. Of special interest are bisimulations where $\mu \sim \eta$ implies $\mu = 1_x$ and $\eta = 1_y$ for some states x and y (denoted $x \sim y$) such that $x|_o = y|_o$ for any output o. It is easy to see that the existence of such a bisimulation between protocols P and Q implies indistinguishablility of P and Q by any adversary of any bound.

Validity and Proof of Soundness We say the judgement $\Delta \vdash P \overset{\delta}{=} Q : I \to O$ is *valid* if for any channel embedding $\theta : \Delta' \to \Delta$ between channel contexts, and any k-bounded adversary,

$$|\Pr[A(\theta\ [[P]]) = 1] - \Pr[A(\theta\ [[Q]])]| \leq \delta(k).$$

Note here that the bound we prove is invariant up to channel embedding. This immediately implies Theorem 2, by applying the identity embedding.

We now sketch the proof of soundness for the equational rules in our logic:

- The [Refl], [Sym], and [Trans] rules are clear.
- The [Exchange] and [Weakening] rules follow at once from the invariance under protocol embeddings.
- The [ReactCong] rule is also clear and [axiom] holds by assumption.
- To prove [CompSym] and [CompAssoc], we define bisimulations by $(s, t) \sim (t, s)$ and $((s, t), u) \sim (s, (t, u))$ respectively.
- To prove [AbsorbComp], we define a bisimulation by $(_, t) \sim t$.
- The rules [CompNew] and [NewExchange] are clear since both sides are interpreted as identical protocols.
- The rules [ResourceTrans], [Subst], and [UnusedResource] follow from the fact that we can choose our final interpretation of both sides to be $[[\cdot]]_1|_{c_1}|_{c_2}$, *i.e.*, prior to any query we attempt to fire c_1 before c_2.
- In the rule [UnFold], we can similarly choose our final interpretation of the body inside the program-level bind on the left-hand side to be $[[\cdot]]_1|_{c_1}|_{c_2}$. This again attempts to fire c_1 before c_2, and this amounts precisely to performing the reaction R_1 inside the reaction-level bind on the right-hand side.
- The rule [NewCong] follows from the fact that any adversary for the protocols on the bottom is also an adversary for the protocols on top.

It remains to prove the rule [CompCong]. We first give the following two constructions on adversaries:

Composition Given an adversary A for a semantic composition of protocols P and Q (not necessarily coming from IPDL), we can compose A with Q to form an

adversary for P whose interaction with P yields precisely the same final distribution on booleans as the interaction of the original adversary A with $P \mid\mid Q$. Let $A : I' \to O'$ and $Q : I_2 \to O_2$. Let $\mathsf{d}, \mathsf{a}, \mathsf{s}$ be the decision, accept, and stepping functions of the adversary. The protocol for the new adversary is $A \mid\mid Q$; the schedule is the same as the one for A; the decision function maps a state $(s, _)$ to $\mathsf{d}(s)$; the accept function for a channel $c \in O_2 \cup O'$ maps a state $(s, _)$ to $\mathsf{a}_c(s)$; the accept function for a channel $c \in (I_2 \cup O_2)\ (I' \cup O')$ maps any state to true; the step function maps a state (s, t) to $\mathsf{s}_c(s) \times 1_t$.

Restriction Given an adversary A for a protocol $P|_o$ (not necessarily coming from IPDL), we can turn A into an adversary for P whose interaction with P yields precisely the same final distribution on booleans as the interaction of the original adversary A with $P|_o$. Let S be the set of states of A. The new schedule is obtained by scheduling o before and after every channel in the schedule for A. The set of states for the new adversary is $\mathsf{S} + \mathsf{S} + \mathsf{S}$. We now define the rest of the structure:

- The states in the left branch encode the original states of A. All inputs and outputs leave a left-branch state unchanged (and will never be called on a left-branch state). Their decision function is the original decision function for A. They accept all channels for scheduling, even though the structure of the new schedule guarantees that only o is ever scheduled in a left-branch state. The step function for any channel turns a left-branch state into the corresponding middle-branch state.
- The states in the middle branch encode the states of A after performing o on the left. All inputs and outputs leave a middle-branch state unchanged (and will never be called on a middle-branch state). Their decision function

maps every state to false (and will never be called on a middle-branch state). Their accept function is the original accept function for A. The step function for any channel is the original step function for A with the proviso that it furthermore turns a middle-branch state into a right-branch state.

- The states in the right branch encode the states of A before performing o on the right. The input and output transition functions are the original ones for A. Their decision function maps every state to false (and will never be called on a right-branch state). They accept all channels for scheduling, even though only o will ever be scheduled in a right-branch state. The step function for any channel turns a right-branch state into the corresponding left-branch state.

Now, we may prove the rule sound. Given a k-bounded adversary A for the protocols $[[P_1||Q]]$ and $[[P_2||Q]]$, we will turn it into an appropriate adversary for $[[P_1]]$ and $[[P_2]]$. First, by Lemma 11, we see that $[[P_1||Q]] = ([[P_1]] \mid\mid [[Q]])|_{o_1,\ldots,o_\ell}$ (and similarly for P_2), where ℓ is the number of outputs of P_1 and Q. We then apply the second construction for restriction above ℓ times to receive an equivalent adversary for $[[P_1]] \mid\mid [[Q]]$. Finally, we apply the first construction for composition to receive an adversary A' for $[[P_1]]$. By construction, the behavior of A' interacting with $[[P_1]]$ produces the same final output distribution on booleans as the behavior of A interacting with $[[P_1||Q]]$, and similarly for P_2.

It now remains to estimate the bound for A', as a function of k, the bound for A. Suppose Q is b-bounded. Then, the first construction has a bound of $O(|\Delta| * \mathsf{max}(k, b))$, by estimating the runtime of each transition in the protocol $A \mid\mid [[Q]]$. The second construction has a bound of $O(k)$, since the schedule for the adversary grows by a constant amount, and each transition of the semantic

protocol has a runtime of at most $O(k)$. Since we run the second construction at most $|\Delta|$ times (bounding the number of outputs), we have that the adversary A' is bounded by $O(|\Delta| * \mathsf{max}(k, b))$.

3.10 More Details on Case Studies

3.10.1 OT from Trapdoor Permutations

The ideal functionality for (1-2) OT is given in Figure 3.12. It is given by a single reaction which simply selects a boolean from the receiver, a pair of messages from the sender, and outputs the appropriate component of the pair. Our definition of ideal OT is parameterized by the type of messages, `L`. (Recall that all IPDL types are in bijection with bitstrings of an appropriate length.) For this simple definition, we eschew the use of ideal parties; instead, if the receiver is corrupted, we simply spawn another copy of the OT functionality with the same inputs, but an output for the adversary. The adversary learns nothing if the sender is corrupted.

The trapdoor OT protocol depends on the security of a *hardcore predicate*, which consists of a family of trapdoor permutations f along with a predicate B such that it is difficult to distinguish the value $B(x)$ from uniform, given only f and $f(x)$ for a uniformly chosen x in the domain of f. While the type system of IPDL does not include general functions (since they take exponential space to describe), we can still model trapdoor permutations by representing f with the following data: an *evaluation key*, a *trapdoor key*, an distribution for generating trapdoor keys, a *derivation function* from trapdoor keys to evaluation keys, and *evaluation functions*, both forwards using the evaluation key, and backwards using

the trapdoor key. Only the evaluation and trapdoor keys need to represented as IPDL values: the generation algorithm, derivation function, and evaluation functions can instead be represented as distributions and function symbols in IPDL, respectively. Given this data, we can easily model the hard-core predicate's security as an approximate equivalence between IPDL programs.

In the trapdoor OT protocol, the sender (Alice) sends a randomly chosen trapdoor permutation f to the receiver (Bob), but keeps the inverse of f secret. In return, Bob sends a pair of values, appropriately constructed using uniform randomness and f. Finally, Alice sends her pair of messages back to Bob, appropriately blinded by Bob's message. Assuming Bob constructed his message correctly, and that B is a hard-core predicate for f, this is a secure construction.

In this protocol, and as is common to all of our OT constructions, the adversary learns nothing in the case when Alice is corrupted; thus, we only focus on the case when Bob is corrupted. In this case, the simulator is able to read Bob's index bit and the output of the OT, and must reconstruct Bob's view of the protocol for the adversary. The most subtle part of the proof is that for Bob's view to be simulatable, we cannot reason only about the uniformity of a single bit $B(x)$, but instead of a *pair* of bits $B(x)$ and $B(y)$ (given only f, $f(x)$, and $f(y)$). We thus prove as a lemma that this generalized notion of security for the hard-core predicate follows from the usual one.

3.10.2 1-4 OT from 1-2 OT

While the above two OT protocols only operate over pairs of messages, the GMW protocol in Section 3.6.1 instead requires an OT protocol which operates over

four messages, instead of two. This case study mechanizes a construction for 1-4 OT from three instances of 1-2 OTs. In the protocol, the sender blinds their four messages by a combination of *six* random strings, and sends these blinded messages to the receiver. These random strings are additionally given as input to the underlying OTs as messages. The receiver uses their two index bits as index bits for the underlying OTs. The randomness is carefully chosen so that the appropriate randomness only cancels out for the intended message, and all other messages appear uniformly random.

The IPDL proof of the above construction requires a subtle analysis which uses *rerandomization*, or mapping uniform randomness through a bijection. Specifically, we show the following two protocols are (exactly) equivalent in IPDL: the first takes as input a boolean on a channel i, and returns uniformly random values an channels c and d; instead, the second uniformly samples two values x and y, and sets c to be the value $\mathsf{if}\ i\ \mathsf{then}\ x\ \mathsf{else}\ y$, and similarly sets d to be the value $\mathsf{if}\ i\ \mathsf{then}\ y\ \mathsf{else}\ x$. Once the above lemma is established, the proof follows from a number of straightforward channel inlinings.

3.10.3 Two-Party GMW Protocol

Given the definitions in Section 3.6.1, the ideal protocol for GMW is straightforward. The ideal parties for Alice and Bob forward their inputs to the functionality, and eventually receive outputs from the functionality. We focus on the case when Alice is corrupt, so she also forwards her inputs and outputs to the simulator. We additionally assume that the simulator learns the *timing* of Bob's inputs (but not their content); this is important for a technical reason, which we will explain below. Given inputs from Alice and Bob along channel vectors $\overrightarrow{u}^A$ and $\overrightarrow{v}^B$,

the functionality generates a fresh set of vectors $\overrightarrow{w}^n$ for the circuit wires, runs $\mathsf{evalCirc}(c, \overrightarrow{u}, \overrightarrow{v}, \overrightarrow{w})$, and delivers the circuit outputs to the ideal parties accordingly.

Notably, this definition of the functionality – and thus, also our GMW formalization – encodes *reactive* MPC, in which Alice and Bob can give inputs to the protocol depending on prior outputs. This is possible since our encoding has the feature that the only causal relationships between wires are those imposed by dataflow; thus, if an output wire w_k does not depend on Alice's jth output, Alice is enabled to give the jth input to the protocol after she receives the value for w_k.

The implementation of the GMW protocol is also straightforward, and follows the standard construction: Alice and Bob secret share their inputs, collaboratively compute the circuit over their secret shares, and open their shares for the output wires. To compute the nonlinear AND operation, Alice must use a 1-4 OT protocol to obliviously send Bob a single bit which encodes the XOR of the cross-terms of the two secret shared variables. As described in 3.10.1, we model semi-honest corruption by instrumenting the corrupted party (here, Alice) to leak to the adversary any inputs she receives from Bob, and any randomness she generates during the protocol. Thus, the adversary receives five types of messages from Alice: Alice's randomness generated during the OTs, Alice's protocol input, Alice's secret share for Bob of her input, Alice's share of Bob's input, and Bob's opening of the output wires.

The simulator follows a standard construction in which it evaluates a "blinded" copy of the real protocol in its head, having access to only Alice's private data, but not Bob's. The central step in the proof of security is the construction of an *invariant* between the real world and ideal world with simulator, such that Bob's

share of wire w in the real world is equal to the XOR of the true value of wire w in the ideal world with Alice's simulated share, coming from the simulator. By constructing this invariant, we use the [HYBRID] rule to easily reason about the GMW protocol without needing to perform an explicit induction on the circuit.

3.10.4 Coin Flip

Security for the coin flip protocol is defined as an ideal functionality that: generates a uniform bitstring; leaks it to the simulator; and once the simulator returns with an `ok` message, broadcasts the bitstring to all ideal parties. All non-corrupted ideal parties then output the same randomness from the functionality.

This functionality is intended to model three main properties: *fairness* (if one honest party receives output, they all do); *agreement* (all honest parties receive the same output); and *uniformity* of the agreed-upon output. However, we do *not* prove privacy of the output bit, or guaranteed delivery.

Unlike the other case studies, we prove this example secure in the *malicious* setting, where the adversary is able to take over the behavior of all corrupted parties. In order to do so in a structured way, we do not allow the adversary to directly control internal protocol channels, but instead give it access to distinguished adversarial channels as proxies. We then, for each corrupted party, write a *shim* which simply forwards messages between the internal protocol channels and those for the adversary (and vice versa).

Our protocol is defined over an arbitrary number of parties and an arbitrary corruption scenario, modeled as a function `honest : 'I_n -> bool`. However, for simplicity our proof assumes that there are at least two parties such that the first

one is corrupted and the last is honest. This is without loss of generality, since the protocol is clearly symmetric, and the security goal is degenerate if all parties are corrupt and immediate if no parties are corrupt.

Since IPDL is channel-centric rather than process-centric, modeling and reasoning about a protocol with n parties and a fixed number of messages is no harder than reasoning about a protocol with a fixed number of parties, and n messages (such as the GMW protocol). Indeed, one of the first simplification steps we take in the proof is to isolate the behaviors among all channels. For a simple example, suppose that we have a protocol where n parties each first send a message x, and then a second message y. Instead of reasoning about the protocol $||_j P_j$, where P_j is the code of the jth party, we instead use the [BigPar-Par] rule from Section 3.4 to rewrite the protocol as $(||_j\, x.j ::= r_j) \mid\mid (||_j\, y.j ::= r'.j)$. While a simple observation, this form of rewrite enables a much smoother verification than without.

Encoding of the Ideal Protocol in IPDL The functionality and corresponding ideal protocol is given in IPDL in Figure 3.13. The functionality is parameterized over three channels: `leak` and `ok`, which are thought of as meant for the simulator, and `send`, which will be used to broadcast a value to all ideal parties. First, on Line 6, it generates a fresh channel `b` carrying a boolean for internal use. It then spawns off three subcomputations: first, on Line 8 we set `b` to be a uniformly random boolean; second, on Line 9, we leak `b` to the simulator, by copying its value to the `leak` channel; finally, on Line 10, we wait for the `ok` message from the simulator, then copy the value of `b` to the `send` channel.

On Lines 14 – 19, we have the ith ideal party, `CoinIParty`. The

ideal party is parameterized by the total number of parties `n`, a predicate `honest : 'I_n -> bool` where `'I_n` is the type of natural numbers less than `n` (from ssreflect [GMT16]), the index of the current party, `i : 'I_n`, and two channels, `send` and `out`. If the ith party is honest, then we simply copy the value from `send` to `out` (Line 17); otherwise, we do nothing (Line 19), given by the empty protocol `prot0`.

Finally, on Lines 21-29, we define the ideal protocol, which is composed of the functionality and all n ideal parties. In addition to the `ok` and `leak` channels for the simulator, the protocol is also parameterized by a *n-length vector* of output channels `out : n.-tuple (chan K)`. The protocol generates the internal `send` channel, and first invokes the functionality on Line 27. It then on Line 28 spawns, for each $i < n$, a copy of the ith ideal party, taking input along the `send` channel, and producing output on the ith output channel (written here as `out ## i`.) We make heavy use of the `bigop` library from `ssreflect` to handle n-ary compositions over an index set, as in Line 28. Also, note that while the `send` channel is defined once inside of the functionality, it is able to be read by all n parties; thus, all channels in IPDL naturally support broadcast.

Encoding of the Real Protocol In the real protocol, each party broadcasts a commitment of their randomly chosen bit, receives everyone else's commitments, and then broadcasts an opening of their commitment. We model the commitment by operating in a *hybrid* setting, wherein each party has access to an ideal functionality for performing commitments. This functionality is given below:

```
1 Definition FComm (K : nat)
2       (* inputs from party *)
3       (commit : chan K) (open : chan TUnit)
```

```
4       (* outputs to broadcast *)
5       (committed : chan TUnit) (opened : chan K) :=
6   [||
7      committed ::= (_ <-- Read commit ;; Ret tt);
8      opened ::= (x <-- Read commit ;;
9                  _ <-- Read open ;; Ret x)
10  ].
```

The commitment functionality is parameterized by input channels `commit` and `open`, which are to be used by the party the functionality is meant for, and output channels `committed` and `opened`, which will be broadcast to all. On Line 7, the channel `committed` is set to wait for `commit` before firing. On Line 8, the channel `opened` is set to the value of `commit`, but only after the channel `opened` has fired.

We now turn to the actual protocol, which is given in Figure 3.14. Similar to the ideal protocol, we model malicious corruption by splitting the party's code into two parts: one for if the party is honest, and one if the party is corrupted.

We first describe the honest party, given on Lines 25-36. We note that the party is parameterized by all channels appearing at the top of the Section, on Lines 15-21. These include the inputs from all broadcast commitments and openings, and the outputs from the party itself, both for its own commitment as well as its protocol output. First on Lines 25-26 we generate two fresh *vectors* of channels, `committed_sum` and `opened_sum`, which will be used for aggregation of multiple values. The first parameter to `newvec`, `n`, is the length of the channel vector, while the second parameter is the type of the channels. The party first commits to a uniformly chosen input, as given on Line 28. On Lines 29-32, the party then computes a *fold* over the signals coming from the channels in `committed`: since each channel in this vector carries a unit value, we are merely accumulating *timing dependen-*

cies into the channels in `committed_sum`. On Line 33, the party then opens their commitment, based on the timing of the last channel in `committed_sum`. In effect, this causes the party to wait for all commitments to happen before the party opens theirs. Line 34 similarly folds the channels in `opened` together into `opened_sum`, so that the last channel in `opened_sum` carries the collective XOR of all opened commitments. The party outputs this value on Line 35.

To encode the corrupted party on Lines 39-46, for convenience we define a *shim* for the corrupted party, which acts to separate the adversary's channels from the internal protocol channels. The adversary's channels, defined on Lines 4-9, are divided into inputs and outputs. The inputs from the adversary are `advCommit` and `advOpen`, which allow the adversary to control the ith party's `commit` and `open` messages (if the ith party is corrupt.) This is reflected in Lines 40 and 41 in the corrupted party, which copy the ith channel of `advCommit` to the corrupted party's `commit` channel, and similarly for `open`. The outputs to the adversary are `advCommitted` and `advOpened`, which are both *tuples of tuples* of channels. On Line 42, the ith component of `advCommitted` is set equal (pointwise) to the ith party's view of the `committed` tuple of input message. Similarly, on Line 44 the ith component of `advOpened` is set to the ith party's view of `opened`. Finally, to define the party we again case split on whether party i is honest, and choose the corresponding implementation.

Finally, we now define the real protocol in total in Lines 55-68. We first generate all internal channel vectors for the commitment functionalities, and then spawn all n commitment functionalities and n parties.

Simulator and Proof Sketch To show that `CoinReal` realizes `CoinIdeal`, we must show the existence of a simulator `CoinSim` which transforms the adversarial channels of `CoinReal` into those of `CoinIdeal`.

Since the security condition is degenerate in the case when all parties are corrupted or all are honest, we focus without loss of generality on the case where the first party is corrupted, and the last party is honest. Intuitively, the simulator runs a copy of the real world protocol "in its head", but modified in the following way: the last party, instead of generating a commitment uniformly, generates its commitment by reading the commitments of all other parties (honest or not), and XORing all other commitments together, along with the value along the channel `leak` from the ideal world. This ensures that the bit that all the parties inside the simulated real world all agree to the same value as is chosen by the functionality. In turn, when all simulated parties open their commitments, the simulator then outputs `ok` to the functionality. Since all commitments by honest players appear uniform, and the simulator only submits `ok` after all corrupted players open their commitments, it follows that the adversary's view in the real and ideal worlds are identical, and all honest party's behavior is identical as well.

$$\frac{}{\Gamma \vdash \nu\ \overrightarrow{v}^{n+1} : \tau.\ P = \nu\ \overrightarrow{u}^{n} : \tau.\ \nu x : \tau.\ P[\overrightarrow{u}\ x/\overrightarrow{v}]}\ [\textsc{NuVec-R}]$$

$$\frac{}{\Gamma \vdash \nu\ \overrightarrow{v}^{n+1} : \tau.\ P = \nu x : \tau.\ \nu\ \overrightarrow{u}^{n} : \tau.\ P[x\ \overrightarrow{u}/\overrightarrow{v}]}\ [\textsc{NuVec-L}]$$

$$\frac{}{\Gamma \vdash \nu\ \overrightarrow{v}^{0} : \tau.\ P = P}\ [\textsc{NuVec-0}]$$

$$\frac{}{\Gamma \vdash \nu\ \overrightarrow{v}^{n} : \tau.\ \nu\ \overrightarrow{w}^{m} : \sigma.\ P = \Gamma \vdash \nu\ \overrightarrow{w}^{m} : \tau.\ \nu\ \overrightarrow{v}^{n} : \sigma.\ P}\ [\textsc{NuVec-Comm}]$$

$$\frac{\Gamma, \overrightarrow{v}^{n} : \tau \vdash P = Q}{\Gamma \vdash \nu\ \overrightarrow{v}^{n} : \tau.\ P = \overrightarrow{v}^{n} : \tau.\ Q}\ [\textsc{Eq-NuVec}] \qquad \frac{\forall j \in J, \Gamma \vdash P_j = Q_j}{\Gamma \vdash \Big|\Big|_{j \in J} P_j = \Big|\Big|_{j \in J} Q_j}\ [\textsc{EqBig}]$$

$$\frac{}{\Gamma \vdash \Big|\Big|_{j \in J} P_j = \Big|\Big|_{j \in J \cap K} P_j \ ||\ \Big|\Big|_{j \in J \cap \widetilde{K}} P_j}\ [\textsc{BigPar-Decomp}]$$

$$\frac{}{\Gamma \vdash \Big|\Big|_{j \in \{k\}} P_j \ = \ P_k}\ [\textsc{BigPar-1}] \qquad \frac{}{\Gamma \vdash \Big|\Big|_{j \in \emptyset} P_j \ = \ 0}\ [\textsc{BigPar-0}]$$

$$\frac{}{\Gamma \vdash (\Big|\Big|_{j \in J} P_j) \ ||\ (\Big|\Big|_{j \in J} Q_j) = \Big|\Big|_{j \in J} (P_j \ ||\ Q_j)}\ [\textsc{BigPar-Par}]$$

$$\frac{\forall ij, i \neq j \Rightarrow v.i \notin P_j}{\Gamma \vdash \nu\ \overrightarrow{v}^{n} : \tau.\ \Big|\Big|_{j<n} P_j = \Big|\Big|_{j<n} (\nu\ v : \tau.\ P_j[v/v.j])}\ [\textsc{BigPar-Nu}]$$

$$\frac{\forall k < n, \Gamma \vdash (\Big|\Big|_{j<k} P_j)||R = (\Big|\Big|_{j<k} Q_j)||R \Rightarrow \Gamma \vdash (\Big|\Big|_{j<k} P_j)||P_k||R = (\Big|\Big|_{j<k} P_j)||Q_k||R}{\Gamma \vdash (\Big|\Big|_{j<n} P_j)||R = (\Big|\Big|_{j<n} Q_j)||R}\ [\textsc{Hybrid}]$$

Figure 3.11: Derived rules for parameterized IPDL protocols.

```
1  Definition OTIdeal (L : type) (i : chan TBool)
2  (m : chan (L ** L)) (o : chan L) :=
3   o ::= (
4      x_i <-- Read i ;;
5      x_m <-- Read m ;;
6      Ret (if x_i then x_m.`2 else x_m.`1)).
```

Figure 3.12: Specification of OT functionality in IPDL.

```
1  Definition CoinFunc (K : nat)
2     (* Channels for simulator *)
3     (leak : chan K) (ok : chan TUnit)
4     (* Broadcast channel for ideal party *)
5     (send : chan K) :=
6      b <- new K ;;
7      [||
8          b ::= (Unif K);
9          leak ::= (x <-- Read b ;; Ret x);
10         send ::= (_ <-- Read ok ;;
11                   x <-- Read b ;; Ret x)
12     ].
13
14 Definition CoinIParty (K : nat) {n : nat}
15    (honest : 'I_n -> bool)
16    (i : 'I_n)
17    (send out : chan K) :=
18        if honest i then
19            (out ::= (x <-- Read send ;; Ret x))
20                    else
21            prot0.
22
23 Definition CoinIdeal (K : nat) {n : nat}
24           (honest : 'I_n -> bool)
25           (out : n.-tuple (chan K))
26           (ok : chan TUnit)
27           (leak : chan K) :=
28     send <- new K ;;
29     [||
30         CoinFunc leak ok send;
31         \||_(i < n) CoinIParty honest i send (out ## i)
32     ].
```

Figure 3.13: Specification of ideal protocol for n-party coin flip in IPDL.

```
Context
    (K : nat) {n} (honest : 'I_n -> bool)
    (* inputs from adversary *)
    (advCommit : n.-tuple (chan K))
    (advOpen : n.-tuple (chan TUnit))
    (* outputs to adversary *)
    (advCommitted : n.-tuple (n.-tuple (chan K)))
    (advOpened : n.-tuple (n.-tuple (chan K)))
    (* output channels of protocol *)
    (out : n.-tuple (chan K)).

Section CoinRealParty.
  Context {n} (i : 'I_n)
          (* inputs to party *)
          (committed : n.-tuple (chan TUnit))
          (opened : n.-tuple (chan K))
          (* outputs from party *)
          (commit : chan K)
          (open : chan TUnit)
          (partyOut : chan K).

  Definition CoinRealParty_honest
    :=
      committed_sum <- newvec n @ TUnit ;;
      opened_sum <- newvec n @ K ;;
      [||
        commit ::= (Unif K);
        cfold committed
            (fun _ _ => tt)
            (fun _ => tt)
            committed_sum;
        open ::=
           (_ <-- Read (committed_sum ## ord_max);;
            Ret tt);
        cfold opened xort id opened_sum;
        partyOut ::= (Read (opened_sum ## ord_max))
      ].

  Definition CoinRealParty_corr
    [||
        commit ::= (Read (advCommit ## i));
        open ::= ((advOpen ## i));
        \||_(j < n) (advCommitted ## i ## j) ::=
                        (Read (committed ## j))
        \||_(j < n) (advOpened ## i ## j) ::=
                        (Read (opened ## j))
    ].

  Definition CoinParty
    if honest i then CoinRealParty_honest
                else CoinRealParty_corr.
End CoinRealParty.

Definition CoinReal :=
    commit <- newvec n @ K ;;
    committed <- newvec n @ TUnit ;;
    open <- newvec n @ TUnit ;;
    opened <- newvec n @ K ;;
    [||
        \||_(i < n)
            FComm (commit ## i)
                  (committed ## i)
                  (open ## i)
                  (opened ## i);
        \||_(i < n) CoinParty
                     i committed opened
                     (commit ## i) (open ## i) (out ## i)
    ].
```

Figure 3.14: Real protocol for n-party coin flip in IPDL.

CHAPTER 4
FUTURE WORK

In this thesis, we presented two systems for equationally reasoning about cryptography: AutoLWE, which makes use of deducibility to (semi-)automatically transform security games for lattice-based cryptosystems; and IPDL, an equational calculus for distributed cryptographic protocols. While both systems are steps in the right direction, further research is needed to achieve our vision of ubiquitous verified proofs for cryptography. Below, we outline future directions for our line of work.

Automatic Program Partitioning Proof steps involving hardness assumptions in both AutoLWE and IPDL operate by essentially *partitioning* the security experiment into two parts: the hardness assumption and its external environment. Partitioning is handled through deducibility in AutoLWE, while it is currently manual in IPDL. It would be very profitible to extend IPDL to support automatic proof partitioning, as this would deliver much higher-level proofs.

An interesting direction would be to use automatic paritioning not only for automating proofs, but for *guiding* new proofs. The cryptographer could design a protocol without a proof of security, and the formal tool could automatically search through its library of hardness assumptions to find one that applies to the protocol through a partitioning. Indeed, formal tools become more widely adopted, the practice of proving security *in tandem* with a formal tool will likely become more common.

Automatic Equivalence Checking More generally, it may be possible for a formal tool to automatically and generically decide whether two security exper-

iments are (exactly) equivalent. Automatic verification of program equivalences would drastically reduce the proof burden for the user, as only a description of the hybrid steps would be required for a proof. Model checking techniques as seen in symbolic cryptographic tools such as Tamarin [MSCB13] or more general symbolic execution engines such as Klee [CDE+08a] are likely applicable.

GUIs and Semi-Formal Tools While verified proofs guarantee the highest degree of formality, a useful and complementary approach is to develop *lightweight* techniques which allow cryptographers to write on-paper arguments, but still benefit from some mechanized support for structuring proofs. These lightweight tools would be analogous to static analyzers for systems code (e.g., Staticcheck [sta]), which aim to eliminate most common bugs but not perform full verification.

A recent work in this direction is state-separating proofs (SSP) [BDLF+18]. The SSP paradigm is an on-paper method for structuring cryptographic proofs as a collection of *packages*, similar to a module in Easycrypt [BGHZ11]. The SSP approach aids formal reasoning, as it provides an expressive on-paper framework that makes it easy to modularize proof efforts.

Interestingly, a *proof-viewer* has been developed for SSP [Pun21], which allows for interactive folding/unfolding of on-paper proof steps and package code in a graphical user interface to aid proof development and comprehension. A similar structured editor for IPDL would be very useful for supporting on-paper, partially verified proof developments.

BIBLIOGRAPHY

[14o] Secure multi-party computation (gmw protocol + malicious model). `https://people.eecs.berkeley.edu/~sanjamg/classes/cs276-fall14/scribe/lec16.pdf`. Accessed: 2020-11-30.

[ABB+05] Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 553–572, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[ABB+17] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, and Vitor Pereira. A fast and verified software stack for secure function evaluation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1989–2006, 2017.

[AFV11] Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 21–40, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.

[AGM+04] Samson Abramsky, Dan R Ghica, Andrzej S Murawski, C-HL Ong, and Ian David Bede Stark. Nominal games and full abstraction for the nu-calculus. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004.*, pages 150–159. IEEE, 2004.

[AR00] Martın Abadi and Phillip Rogaway. Reconciling two views of cryptography. In *Proceedings of the IFIP International Conference on Theoretical Computer Science*, pages 3–22. Springer, 2000.

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 223–238, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[BBB+21] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. Sok: Computer-aided cryptography. In *To Appear in Proceedings of the IEEE Symposium on Security and Privacy*, May 2021.

[BCD+16] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1006–1018. ACM, 2016.

[BCEO19] Gergei Bana, Rohit Chadha, Ajay Kumar Eeralla, and Mitsuhiro Okada. Verification methods for the computationally complete symbolic attacker based on indistinguishability. *ACM Transactions on Computational Logic (TOCL)*, 21(1):2, 2019.

[BCG+13] Gilles Barthe, Juan Manuel Crespo, Benjamin Grégoire, César Kunz, Yassine Lakhnech, Benedikt Schmidt, and Santiago Zanella Béguelin. Fully automated analysis of padding-based encryption in the computational model. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 1247–1260. ACM, 2013.

[BCL12] Gergei Bana and Hubert Comon-Lundh. Towards unconditional soundness: Computationally complete symbolic attacker. In *International Conference on Principles of Security and Trust*, pages 189–208. Springer, 2012.

[BCL14] Gergei Bana and Hubert Comon-Lundh. A computationally complete

symbolic attacker for equivalence properties. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 609–620, 2014.

[BDG+13] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. In Alessandro Aldini, Javier Lopez, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, volume 8604 of *Lecture Notes in Computer Science*, pages 146–166. Springer, 2013.

[BDK+10] Gilles Barthe, Marion Daubignard, Bruce M. Kapron, Yassine Lakhnech, and Vincent Laporte. On the equality of probabilistic terms. In Edmund M. Clarke and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, volume 6355 of *Lecture Notes in Computer Science*, pages 46–63. Springer, 2010.

[BDKL10] Gilles Barthe, Marion Daubignard, Bruce M. Kapron, and Yassine Lakhnech. Computational indistinguishability logic. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010: 17th Conference on Computer and Communications Security*, pages 375–386, Chicago, Illinois, USA, October 4–8, 2010. ACM Press.

[BDLF+18] Chris Brzuska, Antoine Delignat-Lavaud, Cedric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. State separation for code-based game-playing proofs. Cryptology ePrint Archive, Report 2018/306, 2018. `https://eprint.iacr.org/2018/306`.

[Bea95] Donald Beaver. Precomputing oblivious transfer. In *Annual International Cryptology Conference*, pages 97–109. Springer, 1995.

[BGB09] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In Zhong Shao and Benjamin C. Pierce, editors, *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, pages 90–101. ACM, 2009.

[BGHZ11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working

cryptographer. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.

[BGS15] Gilles Barthe, Benjamin Grégoire, and Benedikt Schmidt. Automated proofs of pairing-based cryptography. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1156–1168. ACM, 2015.

[Bla01] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*, pages 82–96. IEEE Computer Society, 2001.

[Bla06a] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *27th IEEE Symposium on Security and Privacy, S&P 2006*, pages 140–154. IEEE Computer Society, 2006.

[Bla06b] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *2006 IEEE Symposium on Security and Privacy*, pages 140–154, Berkeley, CA, USA, May 21–24, 2006. IEEE Computer Society Press.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 575–584, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[Blu83] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*, 15(1):23–27, 1983.

[BPW07] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (rsim) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.

[BR04] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331, 2004. `https://eprint.iacr.org/2004/331`.

[BSCS18] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. Proverif 2.00: Automatic cryptographic protocol verifier, user manual and tutorial, 2018.

[BSCTV13] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. Cryptology ePrint Archive, Report 2013/879, 2013. `https://eprint.iacr.org/2013/879`.

[Buc76] B. Buchberger. A theoretical basis for the reduction of polynomials to canonical forms. *SIGSAM Bull.*, 10(3):19–29, August 1976.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

[CCK+07] Ran Canetti, Ling Cheung, Dilsun Kaynar, Nancy Lynch, and Olivier Pereira. Compositional security for task-pioas. In *20th IEEE Computer Security Foundations Symposium (CSF'07)*, pages 125–139. IEEE, 2007.

[CCK+18] Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Task-structured probabilistic i/o automata. *Journal of Computer and System Sciences*, 94:63–97, 2018.

[CDE+08a] Cristian Cadar, Daniel Dunbar, Dawson R Engler, et al. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI*, volume 8, pages 209–224, 2008.

[CDE+08b] Judicaël Courant, Marion Daubignard, Cristian Ene, Pascal Lafourcade, and Yassine Lakhnech. Towards automated proofs for asymmetric encryption schemes in the random oracle model. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 371–380. ACM, 2008.

[CdH06] Ricardo Corin and Jerry den Hartog. A probabilistic hoare-style logic for game-based cryptographic proofs. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata,*

Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II, volume 4052 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2006.

[CDL11] Pierre Corbineau, Mathilde Duclos, and Yassine Lakhnech. Certified security proofs of cryptographic protocols in the computational model: An application to intrusion resilience. In Jean-Pierre Jouannaud and Zhong Shao, editors, *Certified Programs and Proofs - First International Conference, CPP 2011, Kenting, Taiwan, December 7-9, 2011. Proceedings*, volume 7086 of *Lecture Notes in Computer Science*, pages 378–393. Springer, 2011.

[CHH+17] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1773–1788. ACM, 2017.

[CKRT03] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. Deciding the security of protocols with diffie-hellman exponentiation and products in exponents. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FSTTCS 2003: Foundations of Software Technology and Theoretical Computer Science: 23rd Conference, Mumbai, India, December 15-17, 2003. Proceedings*, pages 124–135, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[CLS03] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *18th Annual IEEE Symposium of Logic in Computer Science, 2003. Proceedings.*, pages 271–280, June 2003.

[Cre08] Cas JF Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *International Conference on Computer Aided Verification*, pages 414–418. Springer, 2008.

[CS13] Alberto Ciaffaglione and Ivan Scagnetto. A weak hoas approach to the poplmark challenge. *arXiv preprint arXiv:1303.7332*, 2013.

[CSV19] Ran Canetti, Alley Stoughton, and Mayank Varia. Easyuc: Using easycrypt to mechanize proofs of universally composable security. In

32nd IEEE Computer Security Foundations Symposium, 2019. `https://eprint.iacr.org/2019/582`.

[DG14] Daniel J. Dougherty and Joshua D. Guttman. Decidability for lightweight diffie-hellman protocols. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 217–231, 2014.

[DY83a] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[DY83b] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Trans. Information Theory*, 29(2):198–207, 1983.

[Eis13] David Eisenbud. *Commutative Algebra: with a view toward algebraic geometry*, volume 150. Springer Science & Business Media, 2013.

[EKR17] David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2(2-3), 2017.

[EP19] Karim Eldefrawy and Vitor Pereira. A high-assurance evaluator for machine-checked secure multiparty computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 851–868, 2019.

[GLL13] Martin Gagné, Pascal Lafourcade, and Yassine Lakhnech. Automated security proofs for almost-universal hash for MAC verification. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, volume 8134 of *Lecture Notes in Computer Science*, pages 291–308. Springer, 2013.

[GLLS09] Martin Gagné, Pascal Lafourcade, Yassine Lakhnech, and Reihaneh Safavi-Naini. Automated security proof for symmetric encryption modes. In Anupam Datta, editor, *Advances in Computer Science - ASIAN 2009. Information Security and Privacy, 13th Asian Computing Science Conference, Seoul, Korea, December 14-16, 2009. Proceedings*, volume 5913 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2009.

[GM99] Neal Glew and Greg Morrisett. Type-safe linking and modular assembly language. In *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 250–261, 1999.

[GMT16] Georges Gonthier, Assia Mahboubi, and Enrico Tassi. *A small scale reflection extension for the Coq system*. PhD thesis, Inria Saclay Ile de France, 2016.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, 1994.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206, Victoria, BC, Canada, May 17–20, 2008. ACM Press.

[HKM15] Viet Tung Hoang, Jonathan Katz, and Alex J. Malozemoff. Automated analysis and synthesis of authenticated encryption schemes. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 84–95. ACM, 2015.

[HKO+18] Helene Haagh, Aleksandr Karbyshev, Sabine Oechsner, Bas Spitters, and Pierre-Yves Strub. Computer-aided proofs for multiparty computation with active security. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 119–131. IEEE, 2018.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 12–24. ACM, 1989.

[KBB17] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *2017 IEEE*

European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017, pages 435–450. IEEE, 2017.

[KMM94] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, Jun 1994.

[KMT12] Steve Kremer, Antoine Mercier, and Ralf Treinen. Reducing Equational Theories for the Decision of Static Equivalence. *Journal of Automated Reasoning*, 48(2):197–217, 2012.

[Len85] A.K. Lenstra. Factoring multivariate polynomials over finite fields. *Journal of Computer and System Sciences*, 30(2):235 – 248, 1985.

[LHM19] Kevin Liao, Matthew A Hammer, and Andrew Miller. Ilc: a calculus for composable, computational cryptography. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 640–654, 2019.

[Loc16] Andreas Lochbihler. Probabilistic functions and cryptographic oracles in higher order logic. In Peter Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 503–531. Springer, 2016.

[Low96] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 147–166, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[LSBM19] Andreas Lochbihler, S. Reza Sefidgar, David Basin, and Ueli Maurer. Formalizing constructive cryptography using crypthol. In *32nd IEEE Computer Security Foundations Symposium*, 2019. `http://www.andreas-lochbihler.de/pub/lochbihler2019csf.pdf`.

[Mau06] Ueli Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154(2):370–381, 2006.

[Mau11] Ueli Maurer. Constructive cryptography–a new paradigm for security definitions and proofs. In *Joint Workshop on Theory of Security and Applications*, pages 33–56. Springer, 2011.

[Mic19] Daniele Micciancio. Symbolic encryption with pseudorandom keys. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 64–93. Springer, 2019.

[MKG14] Alex J. Malozemoff, Jonathan Katz, and Matthew D. Green. Automated analysis and synthesis of block-cipher modes of operation. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 140–152. IEEE Computer Society, 2014.

[Mor94] Teo Mora. An introduction to commutative and noncommutative gröbner bases. *Theoretical Computer Science*, 134(1):131 – 173, 1994.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i. *Information and computation*, 100(1):1–40, 1992.

[MRST01] John Mitchell, Ajith Ramanathan, Andre Scedrov, and Vanessa Teague. A probabilistic polynomial-time calculus for analysis of cryptographic protocols:(preliminary report). *Electronic Notes in Theoretical Computer Science*, 45:280–310, 2001.

[MS01] Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001: 8th Conference on Computer and Communications Security*, pages 166–175, Philadelphia, PA, USA, November 5–8, 2001. ACM Press.

[MSCB13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The tamarin prover for the symbolic analysis of security protocols. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, pages 696–701, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[MT13] Daniele Micciancio and Stefano Tessaro. An equational approach to secure multi-party computation. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 355–372. ACM, 2013.

[Nor98] Patrik Nordbeck. Canonical subalgebraic bases in non-commutative polynomial rings. In *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation*, ISSAC '98, pages 140–146, New York, NY, USA, 1998. ACM.

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 245–254, 1999.

[Pau00] Lawrence Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6, 12 2000.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 333–342, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

[Pei14] Chris Peikert. Lattice cryptography for the internet. In Michele Mosca, editor, *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, October 1-3, 2014. Proceedings*, volume 8772 of *Lecture Notes in Computer Science*, pages 197–219. Springer, 2014.

[Pei15] Chris Peikert. A decade of lattice cryptography. Cryptology ePrint Archive, Report 2015/939, 2015. `https://eprint.iacr.org/2015/939`.

[Pei16] Chris Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4):283–424, 2016.

[PM15a] Adam Petcher and Greg Morrisett. The foundational cryptography framework. In Riccardo Focardi and Andrew C. Myers, editors, *Principles of Security and Trust - 4th International Conference, POST*, volume 9036 of *Lecture Notes in Computer Science*, pages 53–72. Springer, 2015.

[PM15b] Adam Petcher and Greg Morrisett. The foundational cryptography framework. In *International Conference on Principles of Security and Trust*, pages 53–72. Springer, 2015.

[pre] Introduction to secure computation, lecture 5. `http://www.cs.umd.edu/~jkatz/gradcrypto2/f13/lecture5.pdf`. Accessed: 2020-11-30.

[Pun21] Kirthivaasan Puniamurthy. A proof viewer for State-separating proofs: Yao's Garbling Scheme. Master's thesis, Aalto University. School of Science, 2021.

[Reg05a] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.

[Reg05b] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[RKTC03] M. Rusinowitch, R. Küsters, M. Turuani, and Y. Chevalier. An np decision procedure for protocol insecurity with xor. In *Logic in Computer Science, Symposium on(LICS)*, volume 00, page 261, 06 2003.

[RS90] Lorenzo Robbiano and Moss Sweedler. Subalgebra bases. In Winfried Bruns and Aron Simis, editors, *Commutative Algebra*, pages 61–87, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

[RT03] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with a finite number of sessions and composed keys is np-complete. *Theoretical Computer Science*, 299(1):451 – 475, 2003.

[Sch96] Steve Schneider. Security properties and csp. In *Proceedings of the 1996 IEEE Conference on Security and Privacy*, SP'96, pages 174–187, Washington, DC, USA, 1996. IEEE Computer Society.

[Sho94] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[SMCB12] Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In Stephen Chong, editor, *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, pages 78–94. IEEE Computer Society, 2012.

[SS88] David Shannon and Moss Sweedler. Using gröbner bases to determine

algebra membership, split surjective algebra homomorphisms determine birational equivalence. *J. Symb. Comput.*, 6(2-3):267–273, December 1988.

[sta] Staticcheck. `https://staticcheck.io`. Accessed: 2021-05/20.

[SV17] Alley Stoughton and Mayank Varia. Mechanizing the proof of adaptive, information-theoretic security of cryptographic protocols in the random oracle model. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 83–99. IEEE, 2017.

[TGD15] Ashish Tiwari, Adrià Gascón, and Bruno Dutertre. Program synthesis using dual interpretation. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 482–497. Springer, 2015.

[TM18] Eftychios Theodorakis and John C. Mitchell. Semantic security invariance under variant computational assumptions. *IACR Cryptology ePrint Archive*, 2018:51, 2018.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, pages 619–636, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Yao82] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[zcaa] Zcash. `https://z.cash`.

[zcab] Zcash counterfeiting vulnerability successfully remediated. `https://electriccoin.co/blog/zcash-counterfeiting-vulnerability-successfully-remediated/`.