Symbolic Proofs for Lattice-Based Cryptography

Gilles Barthe  
IMDEA Software Institute  
Madrid, Spain  
gilles.barthe@imdea.org

Xiong Fan  
Cornell University  
Ithaca, NY, USA  
xfan@cs.cornell.edu

Joshua Gancher  
Cornell University  
Ithaca, NY, USA  
jrg358@cornell.edu

Benjamin Grégoire  
INRIA  
Sophia-Antipolis, France  
benjamin.gregoire@inria.fr

Charlie Jacomme  
LSV & CNRS & ENS Paris-Saclay  
Inria & Université Paris-Saclay  
charlie.jacomme@lsv.fr

Elaine Shi  
Cornell University  
Ithaca, NY, USA  
elaine@cs.cornell.edu

ABSTRACT
Symbolic methods have been used extensively for proving security of cryptographic protocols in the Dolev-Yao model, and more recently for proving security of cryptographic primitives and constructions in the computational model. However, existing methods for proving security of cryptographic constructions in the computational model often require significant expertise and interaction, or are fairly limited in scope and expressivity.

This paper introduces a symbolic approach for proving security of cryptographic constructions based on the Learning With Errors assumption (Regev, STOC 2005). Such constructions are instances of lattice-based cryptography and are extremely important due to their potential role in post-quantum cryptography. Following (Barthe, Grégoire and Schmidt, CCS 2015), our approach combines a computational logic and deducibility problems—a standard tool for representing the adversary’s knowledge, the Dolev-Yao model. The computational logic is used to capture (indistinguishability-based) security notions and drive the security proofs whereas deducibility problems are used as side-conditions to control that rules of the logic are applied correctly. We then use AutoLWE, an implementation of the logic, to deliver very short or even automatic proofs of several emblematic constructions, including CPA-PKE (Gentry et al., Eurocrypt 2010), Inner Product Encryption (Agrawal et al. Eurocrypt 2011), and Asiacrypt 2011, CCA-PKE (Micciancio et al., Eurocrypt 2012). The main technical novelty beyond AutoLWE is a set of (semi-)decision procedures for deducibility problems, using extensions of Gröbner basis computations for subalgebras in the (non-)commutative setting (instead of ideals in the commutative setting). Our procedures cover the theory of matrices, which is required for lattice-based assumption, as well as the theory of non-commutative rings, fields, and Diffie-Hellman exponentiation, in its standard, bilinear and multilinear forms. Additionally, AutoLWE supports oracle-relative assumptions, which are used specifically to apply (advanced forms of)

the Leftover Hash Lemma, an information-theoretical tool widely used in lattice-based proofs.

CCS CONCEPTS
• Security and privacy → Cryptography: Logic and verification;

KEYWORDS
Symbolic proofs; provable security; lattice-based cryptography

ACM Reference Format:

1 INTRODUCTION
Formal methods, and in particular formal verification, have long been used for building and checking mathematical claims of correctness or security for small but possibly very complex to moderately large and complex systems. In contrast to pen-and-paper counterparts, formally verified claims deliver higher assurance and independently verifiable proofs that can be replayed by third parties. Over the last 20 years, formal methods have been applied successfully to analyze the security of cryptographic protocols in the Dolev-Yao model [28], an idealized model in which cryptographic constructions are treated algebraically. 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 [5, 14, 59]. These tools have subsequently been used for analyzing numerous cryptographic protocols, including recently TLS 1.3. [26, 39]. Unfortunately, the Dolev-Yao model is focused on cryptographic protocols and cannot be used for reasoning about cryptographic primitives. A related approach is to use so-called refinement types (a.k.a. logical assertions) for reasoning about implementations written in a functional programming language [62]; this approach has also been used for analyzing TLS 1.3. [13, 27], but is also primarily limited to cryptographic protocols.

An alternative approach is to develop symbolic methods that reason directly in the computational model. This approach applies both to primitives and protocols, and instances of this approach have been instrumented in tools such as CertiCrypt [10], CryptHOL [42], CryptoVerif [15], EasyCrypt [9, 11], and FCF [53] (see also [7, 36])
for further approaches not supported by tools). However, these tools require significant user interaction and expertise, in particular when used for reasoning about cryptographic primitives.

A promising approach for analyzing cryptographic primitives in the computational model is to combine computational logics and 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) [6] and pairing-based cryptography [12]. Broadly speaking, computational logics formalize game-playing security proofs; each step of the proof corresponds to a hop, and symbolic side-conditions are used to ensure the validity of the hop. More specifically, computational logics, which can be seen as specializations of [7], are used to capture computational security goals and to drive security proofs whereas side-conditions use symbolic tools such as deducibility and static equivalence to guarantee that the rules of the logic are applied correctly. In particular, a key idea of this approach is to use deducibility for controlling the application of rules for performing reductions to hardness assumptions, and for performing optimistic sampling, a particularly common and useful transformation which simplifies probabilistic experiments by allowing to replace, under suitable circumstances, sub-computations by uniform samplings.

The use of deducibility in side conditions, as opposed to arbitrary mathematical conditions, is a necessary step for automating application of proof rules, and more generally for automating complete proofs. However, the interest of this approach is conditioned by the ability to check the validity of deducibility problems. 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 [22, 38, 43, 46, 49, 57, 58, 60]. 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 the 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. There are two possible approaches to address this challenge:

- Heuristics: rather than deciding deducibility, one considers weaker conditions that are easier for verification. As demonstrated with AutoG&P, such an approach may work reasonably well in practice. However, it is not fully satisfactory. First, the heuristics may be incomplete and fail to validate correct instances. Second, advanced proof rules that perform multiple steps at once, and proof search procedures, which explores the space of valid derivations, become unpredictable, even for expert users.
- (Semi-)decision procedures based on computational mathematics: in this approach, one provides reductions from deducibility problems to computational problems in the underlying mathematical setting. Then, one can reuse (semi-)decision procedures for the computational problems to verify deducibility problems.

This approach offers some important advantages. First, it eliminates a potential source of incompleteness, and in particular the possibility that a proof step fails. Second, it is more predictable. Predictability is very important when a high level of automation is sought. Indeed, automation is often achieved through advanced tactics. When they involve multiple heuristics, the outcome of advanced tactics cannot be anticipated, which is a major hurdle to the adoption of formal verification tools. Third, it formalizes connections between known mathematical problems, which may have been extensively studied, and verification problems that may arise for the first time. Lastly, it encourages reusing existing algorithms and implementations.

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.

Contributions

We propose 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. [52] for a recent survey of the field.

In this paper, we define a logic for proving computational security of lattice-based cryptographic constructions. The logic follows the idea of combining computational proof rules with symbolic side-conditions, as in [6, 12]. 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 [37]. The Leftover Hash Lemma is a powerful information-theoretical tool which allows to replace, under suitable conditions, a subcomputation by a sampling from a uniform distribution. The Leftover Hash Lemma is widely used in cryptographic proofs, in particular in the setting of lattice-based cryptography. 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 [54].

More specifically, our examples include: dual Regev PKE [33], MP-PKE [45], ABB-(H)IBE [1] and IPE [2]. All of our mechanized proofs are realistically efficient, running in at most three seconds (Fig. 12); efficiency in this setting is usually not an issue, since cryptographic constructions typically induce small instances of the deducibility problem. Recent progress on more advanced cryptographic constructions based on lattices, like attribute-based encryption [17] and predicate encryption [34], are closely related to both the structure of the schemes and strategy in the proofs in [1, 2, 33]. The MP-PKE [45] inspires development in some lattice-based constructions, like homomorphic encryption [3] and deniable attribute-based encryption [4].

The technical core of our contributions are a set of (semi-)decision procedures for checking deducibility in the theory of Diffie-Hellman exponentiation, in its standard, bilinear and multilinear versions,
and in the theories of fields, non-commutative rings, and matrices. In particular, we give decision procedures for checking deducibility in the theory of Diffie-Hellman exponentiation. This procedure has immediate applications to reasoning about security of cryptographic constructions based on bilinear and multilinear maps. The central idea behind our algorithm is to transform a deducibility problem into a problem from commutative algebra. The latter can be resolved through standard computations of Gröbner basis. Furthermore, we give a semi-decision procedure for checking deducibility in the theory of matrices. This has immediate applications to reasoning about security of lattice-based constructions. In this case, our algorithm extracts from a deducibility question a problem from non-commutative algebra. The problem can be resolved through semi-decision procedures based on non-commutative variants of Gröbner bases known as Subalgebra Analog of Gröbner Basis on Ideals (SAGBI) [48].

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 5.2. Recall that a public-key cryptosystem is given by three probabilistic algorithms (Setup, Enc, 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 [33], an optimization of Regev’s original encryption [55]. We focus on a simple version that encrypts single bits; however, standard techniques can be used to encrypt longer messages.

Definition 2.1 (Dual Regev Encryption). Below, let \( \lambda = n \) be the security parameter, \( m = O(n \log q) \), \( q = O(m) \) and \( \chi \) (or \( \chi^n \)) be discrete Gaussian distribution over \( \mathbb{Z} / (or \mathbb{Z}^n) \).

- The key generation algorithm, KeyGen(\( \lambda \)), chooses a uniformly sampled random matrix \( A \in \mathbb{Z}_{mq}^{n \times m} \) and a vector \( r \in \{-1, 1\}^m \) sampled uniformly, interpreted as a vector in \( \mathbb{Z}_q^m \). The public key is \( pk = (A, u) \), where \( u = Ar \), and the secret key is \( sk = r \).
- To encrypt a message \( b \in \{0, 1\} \), the encryption algorithm Enc(pk, b) chooses a random vector \( x \in \mathbb{Z}_q^n \), a vector \( x_0 \) sampled from \( \chi^n \) and an integer \( x_1 \) sampled from \( \chi \). The ciphertext consists of the vector \( c_0 = s^T A + x_0 \) and the integer \( c_1 = s^T u + x_1 + b[q/2] \), where \( s \) denotes the transpose operation on matrices.
- The decryption algorithm checks whether the value \( c_1 - (s, c_0) \) is closer to 0 or \( b[q/2] \) modulo \( p \), and returns 0 in the first case, and 1 in the second.

Decryption is correct with overwhelming probability, since we compute that \( c_1 - (s, c_0) = x_1 + b[q/2] - (s, x_0) \), so the norm of the term \( x_1 - (s, x_0) \) will be much smaller than \( b[q/2] \).

Gentry, Peikert and Vaikuntanathan [33] 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 \( c^* \) 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 as:

\[
\text{Adv}^{\text{cpe}}_{\mathcal{A}} = |\Pr_{c \leftarrow \mathcal{A}} [b = b'] - 1/2|
\]

where \( G \) is the probabilistic experiment that models chosen-plaintext security and \( 1/2 \) 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 1. We inline the key generation and encryption subroutines. In line 1, the public key \( (A, u) \) and its associated secret key \( r \) 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 [55] 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 2.2 (LWE). Let \( n, m, q, \) and \( \chi \) be as in Definition 2.1. Given \( s \in \mathbb{Z}_q^n \), let \( \text{LWE}_{s, \chi} \) (dubbed the LWE distribution) be the probability distribution on \( \mathbb{Z}_q^{n \times m} \times \mathbb{Z}_q^m \) obtained by sampling \( A \in \mathbb{Z}_q^{n \times m} \) at uniform, \( e \) from \( \chi^m \), and returning the pair \( (A, s^T A + e) \). The decision-LWE problem is to distinguish \( \text{LWE}_{s, \chi} \) from uniform, where \( s \) is uniformly sampled.

We say the decision-LWE problem is infeasible if for all polynomial-time algorithms \( \mathcal{A} \), the advantage \( \text{Adv}^{\text{lwe}}_{\mathcal{A}} (1^\lambda) \) is negligible close to 1/2 as a function of \( \lambda \):

\[
\text{Adv}^{\text{lwe}}_{\mathcal{A}} (1^\lambda) = |\Pr_{c \leftarrow \mathcal{A}} [A \text{solves LWE}] - 1/2|
\]

The works of [19, 50, 55] show that the LWE assumption is as hard as (quantum or classical) solving GapSVP and SIVP under various settings of \( n, q, m \) and \( \chi \).

Leftover Hash Lemma. Let \( A \in \mathbb{Z}_q^{n \times m} \) 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 \( A \) with a random \( \{-1, 1\} \) -valued matrix \( R \) is statistically close to uniform. Additionally, this result holds in the presence of an arbitrary linear leakage of the elements of \( R \). Specifically, the following leftover hash lemma is proved in [1] (Lemma 13).

Lemma 2.3 (Leftover Hash Lemma). Let \( q, n, m \) be as in Definition 2.1. Let \( k \) be a polynomial of \( n \). Then, the distributions \( (\langle A, R, R^T \rangle, w) \) \( \{(A, \mathbb{R}, R^T \mathbb{w})\} \) are negligibly close in \( n \), where \( A \xleftarrow{} \mathbb{Z}_q^{n \times m} \) in both distributions, \( R \xleftarrow{} (0, 1)^{m \times k} \), \( B \xleftarrow{} \mathbb{Z}_q^{n \times k} \), and \( \mathbb{w} \in \mathbb{Z}_q^{m} \) is any arbitrary vector.

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

3
Proposition 2.4 ([33]). For any adversary $A$ against chosen-plaintext security of Dual Regev Encryption, there exists an adversary $B$ against LWE, such that:

- $\text{Adv}^B_A \leq \text{Adv}^{\text{LWE}}_B + \epsilon_{\text{LHL}}$;
- $t_A \approx t_B$;

where $\text{Adv}^{\text{LWE}}_B$ denotes the advantage of $B$ against decisional LWE problem, $\epsilon_{\text{LHL}}$ is a function of the scheme parameters determined by the Leftover Hash Lemma, and $t_A$ and $t_B$ respectively denote the execution time of $A$ and $B$.

Security proof. We now outline the proof of Proposition 2.4.

The proof proceeds with a series of game transformations, beginning with the game in Figure 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 $(A, Ar)$ (where $A$ and $r$ are independently randomly sampled) into the distribution $(A, u)$ (where $u$ is a fresh, uniformly sampled variable). (This invocation does not use the linear leakage $w$ from Lemma 2.3). In order to apply this lemma, we factor the security game from Figure 1 into one which makes use of $A$ and $u$, but not $r$. That is, if $G_0$ is the original security game, then we have factored $G$ into

$$G_0 = G'(A \xleftarrow{\$} \mathbb{Z}_q^{nxm}; r \xleftarrow{\$} [-1, 1]^m; \text{let } u = Ar)_p,$$

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

$$G_1 = G'(A \xleftarrow{\$} \mathbb{Z}_q^{nxm}; u \xleftarrow{\$} \mathbb{Z}_q^n)_p.$$

This transformation effectively removes $r$ from the security game, thus removing any contribution of the secret key $r$ to the information gained by the adversary $A$. This transformation incurs the error probability $\epsilon_{\text{LHL}}$. The resultant game is shown in Figure 2.

The second transformation performs a reduction step based on the LWE assumption. Indeed, note that after the first transformation, the ciphertexts $(c_0, c_1)$ contain an LWE distribution of dimension $n \times (m + 1)$, with the message bit added to $c_1$. By applying LWE, we may safely transform $c_0$ to be uniformly random, and $c_1$ to be uniformly random added to the message bit. The resulting security game is shown in Figure 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 $b[q/2]$, 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 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 cryptgraphic 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 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 4.3. This is reflected by the "!" at the end of the tactic, performed automatically, handled by our use of the SAGBI method in Section 4.3. This is used in two ways in the proof: on lines 13 and 15, we use \( \text{assumption decisional} ! \) LWE \( \rightarrow \) w; //. The move tactic allows one to "pull" a uniform sampling \( \text{move s 2.} \) into two uniform samplings of types \( Z^n \times Z^m \) and \( Z^n \times Z^m \) respectively; the \( \text{mat_fold} \) does the corresponding inverse operation.

The \( \text{assumption decisional} ! \) LWE \( \rightarrow \) w; //. unfold tactic is used to reason about transformations of uniform samplings: given two functions \( f, f^{-1} \) which must be mutual inverses, the \( \text{assumption decisional} ! \) LWE \( \rightarrow \) w; //. 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 \( \text{assumption decisional} ! \) LWE \( \rightarrow \) w; //. 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 \( \text{assumption decisional} ! \) LWE \( \rightarrow \) w; //. to perform an optimistic sampling transform, in which \( B \) is transformed in order to remove the additive factor \( b?\text{Mu}(()):0_{(1,1)} \). Here, \( \text{Mu} \) is an uninterpreted function from the unit type to 1 by 1 matrices, modelling the message content \( \{q/2\} \). 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 \( b[q/2] \) present in the Dual Regev scheme as the expression \( b?\text{Mu}(()):0_{(1,1)} \).

Finally, the \( \text{indep!} \) tactic is used to reason about games such as the game in Figure 4, in which the adversary trivially has no advantage. Detail about the proof rules present in our logic is given in Section 3.4.

### 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 \( \phi \) is an event. Games are probabilistic programs with oracle and adversary calls, and \( \phi \) 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 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

---

**Figure 5: AutoLWE proof for Dual Regev Encryption.**

<table>
<thead>
<tr>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d ) ::= ( n )</td>
</tr>
<tr>
<td>( d_1 \times d_2 )</td>
</tr>
<tr>
<td>( 1 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t ) ::= ( \mathbb{B} )</td>
</tr>
<tr>
<td>( Z_q )</td>
</tr>
<tr>
<td>( Z_q^{d_1 \times d_2} )</td>
</tr>
<tr>
<td>( \text{list}_{d} t )</td>
</tr>
<tr>
<td>( t \times \ldots \times t )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M ) ::= ( 0 )</td>
</tr>
<tr>
<td>( 1 )</td>
</tr>
<tr>
<td>([M])</td>
</tr>
<tr>
<td>( M + M )</td>
</tr>
<tr>
<td>( M \times M )</td>
</tr>
<tr>
<td>( -M )</td>
</tr>
<tr>
<td>( M | M )</td>
</tr>
<tr>
<td>( \text{sl} M )</td>
</tr>
<tr>
<td>( \text{sr} M )</td>
</tr>
<tr>
<td>( M^T )</td>
</tr>
</tbody>
</table>

**Figure 6: Syntax of expressions (selected)**
is given in Figure 6. Selected typing rules for expressions are given
pointwise to lists.

where

for all queries

Game commands
gc ::= \let\ x = e \assignment
| \x \delta. \mu \sampling\ from\ distr.
| \assert(\phi) \assertion
| \y \leftarrow A(x)\ with\ \vec{O} \adversary\ call

Oracle commands
oc ::= \let\ x = e \assignment
| \x \delta. \mu \sampling\ from\ distr.
| \guard(b) \guard

Oracle definitions
\O ::= \o(x) = [\vec{oc}; \return\ e]

Game definitions
\G ::= [\vec{gc}; \return\ e]; \vec{O}

where A and O range over adversary and oracle names respectively.

Figure 7: Syntax of games

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.

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 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 [], which takes a type τ to the type list_d τ, 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 6. Selected typing rules for expressions are given in the Appendix, in Figure 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 7. The operational behavior of oracles is defined compositionally from the operational behavior of commands:

- random sampling x \delta. \mu: we sample a value from \mu 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 \guard(b): we evaluate b and return \bot if the result is false. Guards are typically used in decryption oracles to reject invalid queries.

In addition, we assume that every oracle O comes with a value δ_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_O that is initially set to 0. Then, whenever the oracle is called, one checks c_O ≥ δ_O; if so, then \bot is returned. Otherwise, the counter c_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_O that is initially set to \emptyset. Then, for every call the query parameters are stored in Q_O. (Following [7] 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-theoretical 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 \assert(\phi): we evaluate \phi and abort if the result is false.
- adversary call \y \leftarrow A(e)\ with\ \vec{O}: we evaluate e, call the adversary A with the result as input, and bind the output of the
adversary to \( y \). The adversary is provided with access to the oracles \( \mathcal{O} \).

Finally, the interpretation of \( \Pr_G[\phi] \) is to be the probability of \( \phi \) 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.

### 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 \( \Gamma \vdash e = e' \) if \( e \) and \( e' \) are provably equal from the axioms \( \mathcal{E} \) and the set of equalities \( \Gamma \). 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 \( e \vdash_C e' \), where \( e, e' \) are expressions and \( C \) is a context, if \( e \vdash e' \). Similarily, we write \( \Gamma \vdash e \vdash_C e' \) if \( \Gamma \vdash e = e' \) and \( \Gamma \vdash e \vdash e' \) if there exists a context \( C \) such that \( \Gamma \vdash e = e'. \) More generally, a (general) context \( C \) is an expression that only contains distinguished variables \( \bullet_1, \ldots, \bullet_n \). We write \( e_1, \ldots, e_n \vdash_C e', \) where \( e_1, \ldots, e_n, e' \) are expressions and \( C \) is a context, if \( e \vdash_C e' \). We write \( e_1, \ldots, e_n \vdash e' \) if there exists a context \( C \) such that \( e_1, \ldots, e_n \vdash_C e' \). Similarly, we write \( \Gamma \vdash e_1, \ldots, e_n \vdash e' \) if \( \Gamma \vdash C[e_1, \ldots, e_n] = e' \) and \( \Gamma \vdash e_1, \ldots, e_n \vdash e' \) if there exists a context \( C \) such that \( \Gamma \vdash e_1, \ldots, e_n \vdash_C e' \). Intuitively, a context is a recipe that shows how some expression may be computed given other expressions. If we consider matrices, we may have \( M \times N \) matrices, we may have \( M \times N \times O \times N = O \times M \times O \) with the context \( C[\bullet_1, \bullet_2, \bullet_3] := (\bullet_1 - \bullet_3) \times 2 \).

### 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 \( \phi_k \) and adding at each step the assertion \( \phi_k \) corresponding to the current command \( c \), where:

\[
\begin{align*}
\phi_{let\ x = e} & = x = e \\
\phi_{guard(b)} & = b \\
\phi_{assert(e)} & = e \\
\phi_{if/\exists h_1, \ldots, h_k \cdot e} & = true
\end{align*}
\]

### 3.4 Judgment and proof rules

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

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

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

\[
P_1 \leq e_1 \ldots P_k \leq e_k \quad \frac{\ \ \ (P \leq e)}{}
\]

where \( P_i \)s and \( P \) are probability expressions, \( e_i \)s are variables and finally \( e \) is a probability expression built from variables and constants.

Figure 8 presents selected rules of the logic. In many cases, rules consider judgments of the form \( \Pr_G[\phi] \leq e \); similar rules exist for judgments of the form \( |\Pr_G[\phi] - \Pr_G[\phi']| \leq e \).

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 \( e_1 \leq |e_1 - e_2| + e_2 \).

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 computational behaviour. Rule [SUBST] substitutes at position \( p \) an expression \( e' \) by another expression \( e' \) that is contextually equivalent at \( p \), i.e. \( sp_p(G) \vdash e \equiv e' \).

The rule [RAND] performs a different transformation known as optimistic sampling. It replaces a uniform sampling from \( s \) by \( s \Rightarrow t' \), return \( C[x] \). 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′[\{B := B\}] and G′′[\{B := B\}], where the notation G′[\{B := B\}] represents the game obtained by inlining the code of 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 guard(c′) at position p in an oracle with 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 involving existential quantification.

The logic also contains a rule for hybrid arguments. The rule is similar to [12] and omitted For lack of space.

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 ⪯ P′ is valid if the inequality holds for every valid interpretation of P and P′. Finally, one can prove that P ⪯ P′ is valid whenever P ⪯ P′ is derivable in the logic.

3.6 Axioms Used

Here, we describe the axioms used to prove the schemes in Sections 2 and 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[\{b \} - \Pr_G[\{b\}]] ≤ \epsilon. This allows us to apply the [Abstract] rule from Figure 8.

3.6.1 Learning with Errors. Recall from Section 2 that the LWE assumption states that the distribution (A, sTA + e) is indistinguishable from uniform, where A and s are uniformly sampled elements of \mathbb{Z}_q^{n\times m} and \mathbb{Z}_q^n respectively, and e is sampled from some given error distribution.

Our concrete encoding is given in Figure 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. This allows us to apply the [Abstract] rule from Figure 8.
rather \( m + 1 \). This difference is immaterial to the validity of the assumption.

3.6.2 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 \((A, AR, wR)\) is statistically close to the distribution \((A, B, wR)\), where \(A\) is a uniformly random element of \(\mathbb{Z}_q^{nxm}\). \(R\) is a uniformly random element of \([-1, 1]^{mxk}\) (interpreted as a matrix), and \(w\) is a fixed arbitrary vector in \(\mathbb{Z}_q^m\). For the LHL to hold, however, we can actually relax the requirements on \(A\): instead of \(A\) being sampled uniformly, we only require that \(A\) 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 \(A\) only need to be close to uniform. In many of our proofs, this occurs because \(A\) is not uniformly sampled, but rather sampled using an algorithm, TrapGen, which produces a vector \(A\) statistically close to uniform along with a trapdoor \(T_A\), which is kept secret from the adversary.

By combining the LHL with the TrapGen construction, we obtain the security games in Figure 10. Both games are displayed at once: the expressions which vary between the two games are annotated as \(w\). This models the universal quantification of \(w\) in the LHL. Additionally, note that \(A_2\) actually receives the trapdoor \(T_A\). 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 \(A_2\) the trapdoor reflects the assertion that the distribution \((A, AR, wR, T_A)\) is statistically close to the distribution \((A, B, wR, T_A)\), which follows from the information theoretic nature of the LHL.

While we use the assumption from Figure 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 \(A\) uniformly (and do not give the adversary \(T_A\)); additionally, we do not include this linear leakage \(w\). Another such variation is used in our CCA proof from Section 5. In this instance, we do not transform \(AR\) to \(B\), but rather to \(AR + B\) (thus generalizing our \([\text{RAND}]\) rule.) Additionally, we must state the LHL in the CCA proof to be relative to the decryption oracle, which makes use of \(R\). This relativized lemma is still valid, however, since the decryption oracle does not leak any information about \(R\). It will be interesting future work in order to unify these small variations of the LHL.

3.6.3 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 to a KeyGen oracle, which must use the master secret key in order to operate. This is the case in Section 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 11, we state closeness of the distributions \(D_0(s_0, \cdot)\) and \(D_1(s_1, \cdot)\), where both \(s_0\) and \(s_1\) are unknown to the adversary. (As before, each distribution is separated into a coin sampling stage and a deterministic stage.) Note that \(s_0\) and \(s_1\) 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.

4 DECIDING DEDUCIBILITY

Several rules involve deducibility problems as side-conditions. For instance, in the \([\text{ABSTRACT}]\) rule from Fig 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.
We give a reduction from deducibility for matrices to deducibility we proceed incrementally. Concretely, we start by considering the of polynomials over the names, and $\Gamma$ and groups operations, some set $Y$ context is defined as follows: let polynomials with maximum degree of 1 (linear expressions). exponents, whereas Dougherty and Guttman [29] only consider deduce in the field. Previous work only provides partial solutions: adversary (also often called attacker in the symbolic setting) can proving security of pairing-based cryptography. In this setting, the AutoG&P is also used, in its bilinear and multilinear version, in Diffie-Hellman exponentiation is a standard theory that is used for one 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 $\mathbb{Z}_q$, $E$ the equational theory capturing field operations, $f_1, \ldots, f_k, h \in \mathbb{K}[Y]$ be a set of polynomials over the names, and $\Gamma$ be a coherent set of axioms. The deducibility problem is then:

$$f_1, \ldots, f_k \vdash_{E} h$$

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 $K[\{f_1, \ldots, f_k\}]$, the subalgebra generated by the known polynomials.

Decidability is obtained thanks to [61], where they solve the subalgebra membership problem using methods based on classical Gröbner basis.

**Proposition 4.1.** 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 [30] of Buchberger’s algorithm [20] 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.

### 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 $\mathbb{Z}_q$, $E$ the equational theory capturing field operations, $f_1, \ldots, f_k, h \in \mathbb{K}[Y]$ be a set of polynomials over the names, and $\Gamma$ be a coherent set of axioms. The deducibility problem is then:

$$f_1, \ldots, f_k \vdash_{E} h$$

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 $K[\{f_1, \ldots, f_k\}]$, the subalgebra generated by the known polynomials.

Decidability is obtained thanks to [61], where they solve the subalgebra membership problem using methods based on classical Gröbner basis.

**Proposition 4.2.** 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 [56] (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 [56], 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 [61], we resort to the non-commutative SAGBI [48] theory.

**Proposition 4.3.** 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 [47], 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 homogeneous case where all monomials are of the same degree.
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 4.4.** 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 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 [8] 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:

- \( M \parallel N \Rightarrow M \parallel M \parallel M \parallel N \)
- \( M \parallel \{M \parallel N\} \parallel T \Rightarrow M \parallel \{M, N\} \parallel T \)

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 \( M \) and the transpose operation can also be deduced if instead of the transpose operation we simply provide the transposition of the matrices in \( M \).

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 12): an Identity-Based Encryption scheme and an Hierarchical Identity-Based Encryption scheme by Agrawal, Boneh and Boyen [1], a Chosen-Ciphertext Encryption scheme from Micciancio and Peikert [45], and an Inner Product Encryption scheme and proof from Agrawal, Freeman, and Vaikuntanathan [2]. These examples are treated in Sections 5.2, 5.4, 5.3 and 5.5 respectively.

Globally, our tool performs well, on the following accounts: formal proofs remain 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 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.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Scheme</th>
<th>Property</th>
<th>Proof</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gentry et al. ’08 [33]</td>
<td>dual-Regev PKE</td>
<td>IND-CPA</td>
<td>11</td>
</tr>
<tr>
<td>Agrawal et al. ’10 [1]</td>
<td>ABB-HIBE</td>
<td>IND-sID-CPA</td>
<td>77</td>
</tr>
</tbody>
</table>

Figure 12: Overview of case studies. All proofs took less than three seconds to complete.

5.1 Implementation

Security games are written in a syntax closely resembling that shown in Figure 1. See Figure 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 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 [12] 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.

5.1.1 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 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 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.

5.1.2 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 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.

5.1.3 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 \( \mathbb{K}[X] \)-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.

5.2 Identity-Based Encryption

Mathematical background. Let \( \Lambda \) be a discrete subset of \( \mathbb{Z}^m \). For any vector \( v \in \mathbb{R}^m \), and any positive parameter \( \sigma \in \mathbb{R} \), let \( \rho_{v, \sigma}(x) = \exp(-\pi |x - c|^2/\sigma^2) \) be the Gaussian function on \( \mathbb{R}^m \) with center \( c \) and parameter \( \sigma \). Next, we let \( \rho_{\sigma, \Lambda}(\lambda) = \sum_{x \in \Lambda} \rho_{v, \sigma}(x) \) be the discrete integral of \( \rho_{\sigma, \Lambda} \) over \( \Lambda \), and let \( \lambda_{\sigma, \Lambda}(y) := \rho_{\sigma, \Lambda}(y) \). Let \( S^m \) denote the set of vectors in \( \mathbb{R}^m \) whose length is \( 1 \). The norm of a matrix \( R \in \mathbb{R}^{m \times m} \) is defined to be \( \sup_{x \in \mathbb{S}^m} \| R x \| \). 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. InIBE, 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-\( \text{ID} \)-\( \text{CPA} \), 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. [1]. The scheme operates as follows:

- **Matrix A** is generated by algorithm TrapGen, which outputs a random \( A \in \mathbb{Z}^{n \times m} \) and a small norm matrix \( T \in \mathbb{Z}^{m \times m} \) such that \( A \cdot T_A = 0 \). Matrices \( A_1, B \) are sampled randomly from \( \mathbb{Z}^{n \times m} \), and \( u \) is sampled randomly from \( \mathbb{Z}_q^m \). Set \( \text{pp} = (A_1, A, B, u) \) and \( \text{msk} = T_A \).

- **Encrypt** a message \( m \in \{0, 1\} \) with identity \( id \in \mathbb{Z}_q^m \), one generates a uniform \( s \in \mathbb{Z}_q^m \), error vector \( e_0 \rightarrow \chi_m \) and error integer \( e_1 \rightarrow \chi \) from discrete Gaussian, a random \( R \in \{0, 1\}^{m \times m} \), and computes ciphertext
  
  \[ c_t = s^T [A][s] + M(id)[B][u] + (e_0^T) || e^T R || e_1 + (0) || (q/2) \mu \]

- **The secret key for identity** \( id \in \mathbb{Z}_q^m \) is generated by procedure \( r \rightarrow \text{SampleLeft}(A, A_1 + M(id)[B], T_A, u) \), where we have \( r \) is statistically close to \( \chi_m^2 \), and \( [A][A_1 + M(id)[B]] r = u \).

The idea of the proof is first to rewrite \( A_1 \) as \( AR - M(id^*)B \), where \( id^* \) is the adversary’s committed identity. If we do so, we then obtain that the challenge ciphertext is of the form

\[ s^T [A][AR][u] + (e_0^T) || e^T R || e_1 + (0) || (q/2) \mu \]

where \( A \) 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 3.6.3. The computational lemma states that, for appropriately sampled matrices, \( \text{SampleLeft}(A, AR + B, T_A, u) \approx \text{SampleRight}(A, B, R, T_B, u) \), where \( A \) is sampled from TrapGen in the first and uniform in the second, and \( B \) is sampled uniformly in the first and from TrapGen in the second. By applying this transformation to our KeyGen oracle, we transform our matrix \( A \) from one sampled from TrapGen to uniform. Now that \( A \) is uniform, we finish the proof by noticing that our challenge ciphertext is equal to \( b||BR||b + [q/2]\mu \), where \( (b, b) \) forms an LWE distribution of dimension \( n \times m \). Thus we may randomize \( b \) to uniform, and apply the rnd tactic to erase \( \mu \) from the ciphertext.

The main point of interest in this proof is the initial rewrite \( A_1 \to AR - M(id^*)B \). Given that \( A_1 \) is uniform, we may first apply optimistic sampling to rewrite \( A_1 \to A_2 - M(id^*)B \), where \( A_2 \) is uniformly sampled. Thus, we now only need to perform the rewrite \( A_2 \to AR \). This rewrite is not at all trivial, because \( A \) at this point in the proof comes from TrapGen. However, as noted in Section 3.6.2, 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 10.

5.3 CCA1-PKE

The CCA1-PKE scheme we study is proposed by Micciancio and Peikert [45]. In comparison with the CPA-PKE scheme [33] described in Section 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 A** \( \in \mathbb{Z}_q^{n \times m} \) is sampled randomly and \( R \leftarrow \{0, 1\}^{m \times m} \). Set \( \text{pk} = (A, AR) \) and \( \text{sk} = R \).

- **Let M** \( : \mathbb{Z}_q \rightarrow \mathbb{Z}_q^{n \times m} \) be an embedding from \( \mathbb{Z}_q \) to matrices, such that for distinct \( u \) and \( v \), \( M(u) - M(v) \) is full rank. To encrypt a message \( m \in \{0, 1\} \), one generates a uniform \( s \in \mathbb{Z}_q^m \), a uniform \( u \in \mathbb{Z}_q^m \), a uniform matrix \( R' \in \{0, 1\}^{m \times m} \) and an error vector \( e \in \mathbb{Z}_q^m \) sampled from a discrete Gaussian, and computes the ciphertext
  
  \[ c_0 = u, c_1 = s^T A u + (e_0^T) || e^T R' || e_1 + (0) || (q/2) \mu \]

  \[ c_2 = M(u) [A] - AR + M(u) [G], G \text{ is a publicly known gadget matrix, and } \text{Encode} : \{0, 1\} \rightarrow \mathbb{Z}_q^m \text{ sends } m \mu \text{ to } p[q/2](1, \ldots, 1) \]

  \[ \text{To decrypt a ciphertext } (u = c_0, c_1) \text{ with } \text{sk} = R \text{ and } u \neq 0, \text{ one computes } A_0 \text{ and calls a procedure } \text{Invert}(A_0, R, c_1) \text{, which will output } s \text{ and } e \text{ such that } c_1 = s^T A u + e \text{, where } e \text{ has small norm} \]

  \[ \text{By doing a particular rounding procedure using } c_1, s, e, \text{ and } R, \text{ the message bit } m \text{ can be derived} \]

The main subtlety of the proof is that the secret key \( R \) 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 $R'$ to encrypt, we can actually use the secret key $R$. This is justified by the following corollary of the Leftover Hash Lemma: the distribution $(A, AR, e, eR')$ is statistically close to the distribution $(A, AR, e, eR)$ where $A, R, R'$, and $e$ are sampled as in the scheme. This corollary additionally holds true relative to the decryption oracle, which makes use of $R$.

Once we use $R$ to encrypt instead of $R'$, we again use the Leftover Hash Lemma to transform $AR$ into $-AR + M(u)G$, where $u$ 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 $AR$ 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 $R$ as it appears in the decryption oracle. Thus, we must do the transformation $AR \rightarrow -AR + M(u)G$ 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 3.6.2; 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.

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 we support is described in [1]. The ID space for HIBE is $id \in \mathbb{Z}_q$. The secret key for ID string $id = (id_1, \ldots, id_e)$, where $id_1, \ldots, id_e$ is a small-norm matrix $T$, such that $F_q T = 0$, and $F_d = A_0 || A_1 + M(id_2)B || \cdots || A_e + M(id_e)B$. We note that $T$ 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 $F_d$.

The security proof of HIBE is similar to the counterpart of IBE. The challenge ID string id' = $(id_1', \ldots, id_e')$ is embedded in pp as

$$\forall i \in [\ell], A_i = AR_i - M(id_i)B, \quad \forall \ell < j \leq d, A_j = AR_j$$

For admissible query $id = (id_1, \ldots, id_e)$, where id is not a prefix of id', we have

$$B_k = [(M(id_1) - M(id_1'))B] || \cdots || [(M(id_e) - M(id_e'))B] \neq 0$$

Then we can generate secret key for id using information of $B_k$ and $R_k = (R_1 || \cdots || R_k)$. In previous cases, we handle 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.

5.5 Inner Product Encryption

The IPE scheme our scheme supports is described in [2]. We briefly recall their construction as:

- Matrix $A$ is generated by algorithm TrapGen. Matrices $\{matB_i\}_{i\in[d]}$ are sampled randomly from $\mathbb{Z}_q^{n\times m}$, and random vector $u$ is from $\mathbb{Z}_q^n$.
- The public parameters $pp = (A, (B_i)_{i\in[d]}, u)$, and $msk = T_A$.
- Secret key $sk_x = r$ for vector $v \in \mathbb{Z}_q^n$ is computed by algorithm $r \leftarrow SampleLeft(A, \sum_{i\in[d]} B_i G_1^{-1}(v_i G), T_A, u)$, where for operation $G_1^{-1} : \mathbb{Z}_q^{m \times m} \rightarrow \mathbb{Z}_q^{m \times m}$, for any $A \in \mathbb{Z}_q^{m \times m}$, it holds that $G \cdot G_1^{-1}(A) = A$ and $G_1^{-1}(A)$ has small norm.
- To encrypt a message $\mu \in \{0, 1\}$ for attribute $w$, one generates a uniform $s \in \mathbb{Z}_q$, error vector $e_0 \leftarrow \mathcal{C}_1$ and error integer $e_1 \leftarrow \mathcal{C}_2$ from discrete Gaussian, random matrices $[R_i]_{i\in[d]} \in \{0, 1\}^{m \times m}$, and computes ciphertext $(c_0, (c_i)_{i\in[d]}, c)$ as

$$c_0 = s^T A + e_0^T, \quad c_i = s^T (B_i + w_i G) + e_1^T R_i, \quad c = s^T u + e + [q/2]$$

The main challenge in the proof is to answer secret key queries for any vector $v$ as long as $(\nu, w_0, \nu, w_1)$ are both not 0, where $(w_0, w_1)$ is declared by adversary upfront. The attribute $w_k (b)$ is a random bit is first embedded in pp, i.e. $B_i = AR_i - w_i G, \forall i \in [d]$, where $R_i$ is a small matrix. By unfolding the matrix for query $v$, we have

$$A||A \sum_{i\in[d]} B_i G_1^{-1}(v_i G) = A||A \sum_{i\in[d]} R_i G_1^{-1}(v_i G) + \langle w_k, v \rangle$$

If $(w_k, v) \neq 0$, the algorithm SampleRight can be used to generate secret key for $v$.

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 $w_0$ directly to embedding of $w_1$, we use the original scheme as a middle game, i.e. from embedding of $w_0$ to original scheme, then to embedding of $w_1$. The reason for using the original scheme again in the proof is that when using LHL to argue the indistinguishability between $(A, (B_i = AR_i - w_i G), v)$ and $(A, (B_i = AR_i - w_i G)\| c)$, the real public key $(A, (B_i))$ 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.

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 [24] 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 Lakhnachev [25] 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, Lakhnachev and Safavi-Naini [31, 32] adapt these methods to symmetric encryption modes and message authentication codes.

Malozemoff, Katz and Green [44] and Hoang, Katz and Malozemoff [35] pursue an alternative approach for proving security of modes of operations and authenticated encryption schemes. Their approach relies on a simple 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, including constructions whose efficiency competes with state-of-the-art algorithms that were discovered using conventional methods. Using SMT-based methods, Tiwari, Gascón and Dutertre [64] introduce an alternative approach to synthesize bitvector programs, padding-based encryption schemes and modes of operation.

Our work is most closely related to CIL [7], ZooCrypt [6] and AutoG&P [12]. Computational Indistinguishability Logic (CIL) [7] 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 [23]. ZooCrypt [6] 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 [12] introduce a computational logic and provide an implementation of their logic, called AutoG&P, for proving security of pairing-based cryptographic 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 Leftover 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 [63] 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 [63] and could also implement patches when they fail on a specific step.

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 [18, 51]. To this end, it would be interesting to embed the techniques developed in this paper (and in [12]) into the EasyCrypt proof assistant [9, 11], and to further improve automation of EasyCrypt for typical transformations used for proving security of protocols.

Acknowledgements. This work is partially supported by ONR Grant N000141512750 (Barthe), by IBM under Agreement 4915013672 (Fan), and by NSF Grant 1704788 (Ganchar).

REFERENCES

A PROOFS OF SECTION 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 [16] or Waters’ Dual System Encryption [65].

Given a multilinear map \( e, g_1, \ldots, g_n, g \), 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 \( \mathbb{Z}_q \), \( f_1, \ldots, f_k, h \in \mathbb{K}[X, Y] \) be a set of polynomials over both public and secret names and \( \Gamma \) be a coherent set of axioms.

Our deducibility problem is to decide if \( \Gamma \models X, g_{f_1}^1, \ldots, g_{f_k}^k + e; g^h \). 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.

A.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 [40]. Their constructive proof can trivially be used to obtain the following proposition:

**Proposition A.1.** For any sets \( X, Y \), and polynomials \( f_1, \ldots, f_n, h \in \mathbb{K}[X, Y] \) and groups elements \( g_{f_1}^1, \ldots, g_{f_n}^n \), we denote \( (g^h_f) = \langle [e(g_1, g_j)] \rangle_{1 \leq j \leq n, g_j \in G_1, g_k \in G_2} \rangle \cup \langle [e(g_1, g_j)] \rangle_{1 \leq j \leq n, g_j \in G_1, g_k \in G_2} \rangle \cup \langle [e(1, g_j)] \rangle_{1 \leq j \leq n, g_j \in G_2} \rangle \)

Then \( \Gamma \models X, g_{f_1}^1, \ldots, g_{f_n}^n + e; g^h \) \( \iff \Gamma \models X, g_{f_1}^1, \ldots, g_{f_n}^n + e; g^h \).

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.

A.2 Reduction to polynomials

**Lemma A.2.** For any sets \( X, Y \), and polynomials \( w_1, \ldots, w_N, h \in \mathbb{K}[X, Y] \), we have \( \Gamma \models X, g_{w_1}^1, \ldots, g_{w_N}^N + e; g^h \) if and only if:

\[ \exists (e_1, g_1) \in \mathbb{K}[X], (vi, \Gamma \models g_i \neq 0) \land \sum_{i \leq N} e_i \times g_i = h \]

**Proof.** If \( \Gamma = 0 \), the adversary can construct elements of the form \( (g_{w_i}^i)^{g_i^h} \), where \( e_i \in \mathbb{K}[X] \), i.e. \( g_i \) 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 \( \Gamma \neq 0 \), he may also divide by some \( g_i^j \), 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 \( \{g \mid \Gamma \models g \neq 0\} \). 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 [41]). Any non annuling polynomial will be a product of all these irreducible polynomials. We can then obtain a finite set \( G = \{ g_i \} \) such that \( G = \{g \mid \Gamma \models g \neq 0\} \). 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 \( w_i / g_i \).
Theorem A.7. For any sets $X$ and $Y$, polynomials $f_1, \ldots, f_n, h \in \mathbb{K}[X, Y]$, group elements $g_1, \ldots, g_n$, and a set of axioms $\Gamma$ we can decide if $\Gamma \vdash X, g_1^{f_1}, \ldots, g_n^{f_n} \forall \varepsilon g_i^h$.

Proof. To decide if $h$ is deducible, we first reduce to a membership problem with Lemma A.5 that can be solved using Lemma A.6 by computing the Gröbner basis of $(f_i) \cup ((gt-1)Y^j) \in \text{deg}(f_i)) \in \mathbb{K}[X, Y]$, 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 [30]), which actually captures all the possible distinguishers of a frame.

B PROOFS FOR SECTION 4.3

We provide a more detailed proof of Proposition 4.4. To reason about matrices deducibility, written $M \vdash M$ for a set of matrices $M$ and a matrix $M$, we use the natural formal proof system $\mathcal{K}$ which matches the operations on expressions (see Figure 13), that we extend with the equality rule $\begin{array}{c|c}
\text{Eq} & \text{M} \vdash M_1 \quad \text{M} \vdash M_2 \\
\hline
\end{array}$. For ease of writing, we denote $\left(\frac{M}{M}\right) := (A^\top \mid B^\top)^\top$.

Splits elimination.

Proposition B.1. Given a set of matrices $M$ and a matrix $M$, we can obtain $S(M)$ a set of matrices without any concats, such that $M \vdash M \equiv S(M) \vdash H$.

Proof. We notice that the concat operations commute with all the other operators: $(A||B) + (C||D) = (A + C||B + D)$, $(A||B) - (C||D) = (A - C||B - D)$, $A \times (B||C) = (A||B) \times (B||C)$, $(A||B) \div (B||C) = AC + BD$, $(A||B)^\top = (B||A)^\top$. Given a set of matrices $M$, 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 $S(M)$ containing all the submatrices. This preserves deducibility thanks to the Eq rule for the rewriting, and to the split rules for the submatrices.

Definition B.2. We call $N$ the proof system based on $\mathcal{K}$ without splits, and write the deducibility with $M \vdash_N M$.

Lemma B.3. If $M \vdash (R|S)$ with a proof $\pi$ (resp. $M \vdash (R | S)$) then $M \vdash R$ and $M \vdash S$ with smaller proofs (resp. $M \vdash R$, $M \vdash S$).

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: $\begin{array}{c|c}
\text{Induction case:} & \text{The last rule is} \\
\hline
\text{Induction case:} & \text{The last rule is} \\
\end{array}$

Then, we directly obtain by induction on $(R | S)$ smaller proofs for $R$ and $S$. 

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 A.4. Given a $S$-module $T$, $f \in S$ and $S \subseteq S'$, the saturation of $T$ by $f$ in $S'$ is $T_{S'}^f = \{g \in S' | \exists n \in \mathbb{N}, f^n g \in T \}$.

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

Lemma A.5. $\Gamma \vdash X, g_1^{w_1}, \ldots, g_N^{w_N} \forall \varepsilon g_i^h$ if $h \in M \mathbb{K}[X, Y]$.

Proof. We recall that:

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

$\Rightarrow$ We have $\sum_i e_i \times w_i = h \prod_{g \in G(S)} g^{k_y}$. With $K = \max(k_y)$, we multiply both sides by $\prod_y g^{k_y} - k_x$ to get $h \prod_{g \in G(S)} g^k = \sum_i \prod_y g^{k_y} e_i \times w_i \in M$. Which proves that $h \in M_{\mathbb{K}[X, Y]}(g_1, \ldots, g_n)$.

$\Rightarrow$ If $h \in M_{\mathbb{K}[X, Y]}(g_1, \ldots, g_n)$, we instantly have $(e_i) \in \mathbb{K}[X], k \in \mathbb{N}$ such that $\sum_i e_i \times f_i = g^h$.

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

Lemma A.6. For any sets $X$ and $Y$, $f_1, \ldots, f_n, h \in \mathbb{K}[X, Y]$, $g \in \mathbb{K}[X]$, $\exists M = \{\sum_i e_i \times f_i | e_i \in \mathbb{X}[X]\}$. Then, with a fresh variable $M_{\mathbb{K}[X, Y]}(g^\omega) = \{(f_i) \cup ((gt-1)Y^j) \in \text{deg}(f_i)) \in \mathbb{K}[X, Y] \}$.

Proof. C. Let there be $v \in M_{\mathbb{K}[X, Y]}(g^\omega)$. Then, we have $k$ such that $g^k \times u \in M$. The following equalities shows that $v$ is in the right side set $v = g^k \times u = (1 + gt + \ldots + g^{k-1}1k-1)(gt-1)c$.

Indeed, $g^k \times u \in \mathbb{K}[X, t]$ so we have $(e_i) \in \mathbb{K}[X, t]$ such that $g^k \times v = \sum_i e_i f_i$. Moreover, $g^k \times u \in M$ and $g \in \mathbb{K}[X]$ implies that $\text{deg}(v) \subseteq \text{deg}(f_i)$.

So we do have $(e_i) \in \mathbb{K}[X, t]$ and $(j) \subseteq \text{deg}(f_i)$ such that $$(1 + g^k \times u) = \sum_i e_i^j (gt-1)Y^j$$ Finally, $u \in \{(f_i) \cup ((gt-1)Y^j) \in \text{deg}(f_i)) \in \mathbb{K}[X, Y] \}$.

$\Rightarrow$ Let there be $u \in \{(f_i) \cup ((gt-1)Y^j) \in \text{deg}(f_i)) \in \mathbb{K}[X, Y] \}$. Then we have $(e_i, (j) \subseteq \text{deg}(f_i))$ such that $v = \sum i e_i f_i$.

We have that $v \in \mathbb{K}[X, Y]$, so $v$ is invariant by $t$. So, if we substitute $t$ with $\frac{1}{2}$, we have that $v = \sum_i e_i(\mathbb{X}, \frac{1}{2} f_i)$. 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 $v \in M_{\mathbb{K}[X, Y]}(g^\omega)$.

The Buchberger algorithm allows us to compute a Gröbner basis of any free $\mathbb{K}[X]$-module [30] and then decide the membership problem for a module. We thus solve our membership problem using this method.
\[ \frac{[\times]}{\Gamma \vdash M : N^{l} || N^{r}} \]

- The last rule is \( (M) N \). 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 \([\times]\), \([-]\), \([\|]\)], the proof can be done similarly to the two previous cases.

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

- \( [sR] \) is similar.

\[ \Box \]

**Lemma B.4.** If \( M \) is a set of matrix without concatenations, and if \( M \vdash M \), then \( M \vdash_{N} M \).

**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 \( [\times]\), \([\|]\), \([-]\), \([\|]\), we conclude by applying the induction hypothesis to the premise of the rule.
  - The last rule is \( [\times] \). Then, we have a proof of \( M \) of size \(|\pi| = 1\), and with lemma B.3 we have a smaller proof of \( M \), on which we can then apply our induction hypothesis to obtain a proof of \( M \) without split.

- \( [sR] \) is similar.

\[ \Box \]

**Concatenations elimination.**

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

**Lemma B.6.** If \( M, M, N \) do not contain any concat, then:

\[ M \vdash_{N} (M) N \Leftrightarrow M \vdash_{T} M \land M \vdash_{T} N \]

**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: