Symbolic Methods in Computational Cryptography Proofs
Gilles Barthe, Benjamin Grégoire, Charlie Jacomme, Steve Kremer, Pierre-Yves Strub

To cite this version:

HAL Id: hal-02404701
https://hal.archives-ouvertes.fr/hal-02404701
Submitted on 11 Dec 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Symbolic methods in computational cryptography proofs

Gilles Barthe*, Benjamin Grégoire†, Charlie Jacomme‡, Steve Kremer§, Pierre-Yves Strub¶

*MPI for Security and Privacy & IMDEA Software Institute, †Inria Sophia-Antipolis, ¶École Polytechnique
‡LSV, CNRS & ENS Paris-Saclay & Inria & Université Paris-Saclay
§LORIA, Inria Nancy-Grand Est & CNRS & Université de Lorraine

Abstract—Code-based game-playing is a popular methodology for proving security of cryptographic constructions and side-channel countermeasures. This methodology relies on treating cryptographic proofs as an instance of relational program verification (between probabilistic programs), and decomposing the latter into a series of elementary relational program verification steps. In this paper, we develop principled methods for proving such elementary steps for probabilistic programs that operate over finite fields and related algebraic structures. We focus on three essential properties: program equivalence, information flow, and uniformity. We give characterizations of these properties based on deducibility and other notions from symbolic cryptography. We use (sometimes improve) tools from symbolic cryptography to obtain decision procedures or sound proof methods for program equivalence, information flow, and uniformity. Finally, we evaluate our approach using examples drawn from provable security and from side-channel analysis—for the latter, we focus on the masking countermeasure against differential power analysis. A partial implementation of our approach is integrated in EASYCRYPT, a proof assistant for provable security, and in MASKVERIF, a fully automated prover for masked implementations.

1. Introduction

It is notoriously difficult to design cryptographic constructions with strong security. A common measure to deal with this difficulty is to require that designs of cryptographic constructions are validated by mathematical proofs showing that the constructions are secure against arbitrary computationally bounded adversaries [1]. Many of these proofs take the form of reductionist arguments. In such arguments, both the security goals (indistinguishability of encryption, unforgeability of signature…) and the computational assumptions (hardness of decisional Diffie-Hellman…) are modelled as probabilistic experiments where a challenger interacts with an adversary; such experiments come with a winning condition, which captures the situation where an adversary has broken the security property. In the simple case, only one assumption is involved. The reductionist argument is then given by a method for transforming an adversary $A$ against the cryptographic construction under consideration into an adversary $B$ against the computational assumption, and a proof that $p_A \leq f(p_B)$, where $p_B$ denotes the probability of $B$ winning the experiment (against the construction), $p_A$ denotes the probability of $A$ winning the experiment (against the assumption), and $f$ is a function such that $f(x)$ is “small” whenever $x$ is “small”. This rigorous approach is a pillar of modern cryptography, and arguably one of the keys to its success. However, reductionist proofs are becoming increasingly complex, as a consequence of new application scenarios (requiring more complex constructions) and theoretical advances in the field (yielding stronger but more complex constructions).

The game-playing technique [2] is a popular methodology for proving security of cryptographic constructions. This technique decomposes reductionist arguments into elementary steps that can be justified individually with relative ease. In the simple case above, involving a single computational assumption, the technique involves defining a sequence of probabilistic experiments (which are called games in this setting), such that the first experiment captures the security of the construction, and the last experiment captures the security assumption. In addition, the technique requires proving for all steps that $p_{A_i} \leq f_i(p_{A_{i+1}})$, where $A_i$ is an adversary and $p_{A_i}$ is his winning probability in the $i$-th experiment. One then concludes by applying transitivity of inequality. The game-playing technique is helpful to tame the complexity of reductionist arguments. However, it remains difficult to build and verify game-playing proofs of complex constructions.

The code-based game-playing technique [3] is a common variant of the game-playing technique where experiments are modelled as probabilistic programs. This approach has been instrumental in the mechanization of reductionist arguments using tools based on program verification [4], [5], [6], [7]. These tools have been used for verifying many representative examples of cryptographic constructions. However, these tools remain difficult to use by cryptographers, because automation is limited and expertise in program verification is required.

Specializations of the code-based game-playing approach have also been developed and implemented for padding-based [8], pairing-based [9] and lattice-based [10] constructions. Proofs are fully automated for several constructions. However, the most advanced constructions cannot be proved secure in these tools. Furthermore, automation is
This paper focuses on developing principled, automated, proof methods for elementary steps in code-based game-playing proofs. We focus on three particularly common classes of steps:

1) program equivalence: such steps, known as bridging steps in the cryptography literature, require proving that two programs produce the same distribution. These steps are used for proving that the probability of an adversary winning the game is the same in both games (assuming that the winning events are the same).

2) independence: such steps require proving that some values in a given experiment are independent of some secrets. This is modeled as a probabilistic information flow problem, requiring that the joint distribution of these values (viewed as a function of the secrets and other inputs) is a constant function in the secrets. Such steps are used to justify code transformations, and also to conclude that the probability of some events is “small”;

3) uniformity: such steps require proving that some values in a given experiment are uniformly distributed. These steps are very important for justifying lazy sampling, a useful technique which allows to resample values from uniform distributions immediately before they are used.

Contributions. The overall goal is to leverage, in a modular way, existing techniques from symbolic cryptography in order to solve these problems. Symbolic cryptography completely abstracts away probabilities and provides syntactic rather than semantic reasoning techniques.

We first define syntax and semantics for probabilistic programs built over arbitrary function symbols. When operating over finite fields, this allows us to capture programs with conditionals, but without support for loops. We are able to prove that the three problems introduced above, as well as some of their variants, are inter-reducible. This yields several ways to tackle a same problem, some of them more suited for mathematical reasoning and other more suited for symbolic reasoning.

We provide a sound and complete semantic characterization of the three classes of problems. The characterization is based on exhibiting a bijection between random coins (of the two programs in the case of program equivalence or two runs of the same program in the case of probabilistic information flow), and performing equality checks. Intuitively, two programs are equivalent if for any result of the first program there exists a valuation of random variables for the second program such that the results coincide—the bijection ensures that the set of valuations used in the second program follows the required probability distribution.

Based on this characterization we give a sound and complete syntactic characterization of the three problems. The characterization is based on the notion of primal algebra used previously for proving decidability of unification in the theory of finite fields [12]. The syntactic characterization replaces the existence of a bijection by the existence of a term satisfying specific syntactic properties.

We then leverage (and sometimes extend) methods from symbolic cryptography, including deducibility, deduction constraint and static equivalence, to check the syntactic characterization of our properties.

Our abstract framework allows us to derive sound and complete algorithms, as well as heuristics that may be only sound or only complete. Given the high complexity, or lack of decision procedures, such heuristics are of particular interest in practice. Previously mentioned tools for proof mechanization do use some heuristics, but they often lack theoretical foundations, leading to a misunderstanding regarding the precision and limitations. Our results clarify these questions for the heuristics we propose.

In particular, in the case of finite fields of a fixed size we obtain sound and complete algorithms. Even though we show that this problem has high computational complexity, our algorithms appear to be more efficient in practice than the straightforward ones. While the case of finite fields of a fixed size is already useful in some cases, cryptographic proofs (i) are often performed for an abstract size of the finite field, for which we do not have a decision result, and (ii) may require complex combinations of symbol functions. We first define syntax and semantics for probabilistic programs built over arbitrary function symbols. When operating over finite fields, this allows us to capture programs with conditionals, but without support for loops. We are able to prove that the three problems introduced above, as well as some of their variants, are inter-reducible. This yields several ways to tackle a same problem, some of them more suited for mathematical reasoning and other more suited for symbolic reasoning.

We provide a sound and complete semantic characterization of the three classes of problems. The characterization is based on exhibiting a bijection between random coins (of the two programs in the case of program equivalence or two runs of the same program in the case of probabilistic information flow), and performing equality checks. Intuitively, two programs are equivalent if for any result of the first program there exists a valuation of random variables for the second program such that the results coincide—the bijection ensures that the set of valuations used in the second program follows the required probability distribution.

Based on this characterization we give a sound and complete syntactic characterization of the three problems. The characterization is based on the notion of primal algebra used previously for proving decidability of unification in the theory of finite fields [12]. The syntactic characterization replaces the existence of a bijection by the existence of a term satisfying specific syntactic properties.

We then leverage (and sometimes extend) methods from symbolic cryptography, including deducibility, deduction constraint and static equivalence, to check the syntactic characterization of our properties.

Our abstract framework allows us to derive sound and complete algorithms, as well as heuristics that may be only sound or only complete. Given the high complexity, or lack of decision procedures, such heuristics are of particular interest in practice. Previously mentioned tools for proof mechanization do use some heuristics, but they often lack theoretical foundations, leading to a misunderstanding regarding the precision and limitations. Our results clarify these questions for the heuristics we propose.

In particular, in the case of finite fields of a fixed size we obtain sound and complete algorithms. Even though we show that this problem has high computational complexity, our algorithms appear to be more efficient in practice than the straightforward ones. While the case of finite fields of a fixed size is already useful in some cases, cryptographic proofs (i) are often performed for an abstract size of the finite field, for which we do not have a decision result, and (ii) may require complex combinations of symbol functions. We then leverage (and sometimes extend) methods from symbolic cryptography, including deducibility, deduction constraint and static equivalence, to check the syntactic characterization of our properties.

Our abstract framework allows us to derive sound and complete algorithms, as well as heuristics that may be only sound or only complete. Given the high complexity, or lack of decision procedures, such heuristics are of particular interest in practice. Previously mentioned tools for proof mechanization do use some heuristics, but they often lack theoretical foundations, leading to a misunderstanding regarding the precision and limitations. Our results clarify these questions for the heuristics we propose.

In particular, in the case of finite fields of a fixed size we obtain sound and complete algorithms. Even though we show that this problem has high computational complexity, our algorithms appear to be more efficient in practice than the straightforward ones. While the case of finite fields of a fixed size is already useful in some cases, cryptographic proofs (i) are often performed for an abstract size of the finite field, for which we do not have a decision result, and (ii) may require complex combinations of symbol functions. We then leverage (and sometimes extend) methods from symbolic cryptography, including deducibility, deduction constraint and static equivalence, to check the syntactic characterization of our properties.

Our abstract framework allows us to derive sound and complete algorithms, as well as heuristics that may be only sound or only complete. Given the high complexity, or lack of decision procedures, such heuristics are of particular interest in practice. Previously mentioned tools for proof mechanization do use some heuristics, but they often lack theoretical foundations, leading to a misunderstanding regarding the precision and limitations. Our results clarify these questions for the heuristics we propose.

In particular, in the case of finite fields of a fixed size we obtain sound and complete algorithms. Even though we show that this problem has high computational complexity, our algorithms appear to be more efficient in practice than the straightforward ones. While the case of finite fields of a fixed size is already useful in some cases, cryptographic proofs (i) are often performed for an abstract size of the finite field, for which we do not have a decision result, and (ii) may require complex combinations of symbol functions. We then leverage (and sometimes extend) methods from symbolic cryptography, including deducibility, deduction constraint and static equivalence, to check the syntactic characterization of our properties.

We demonstrate the usefulness of our approach in practice through the implementation of a library that we interfaced with two existing tools: EASYCRYPT [2] and MASKVERIF [11]. We do not implement all heuristics or decision procedures discussed in this paper, but the ones implemented are sufficient to improve the existing tools. The source codes of the library and modified tools are available online: [13], [14], [15]. We consider examples in the area of masking, which provide challenging examples of probabilistic information flow. In particular, unlike the original MASKVERIF tool, our extension allows for insightful feedback as it may provide attack witnesses when proofs fail. Furthermore, the integration of our approach into the EASYCRYPT proof assistant improves automation.

2. Setting

We start by providing background on terms built over function symbols and their interpretation over a given domain. Such terms capture the operations performed by a program, which are then interpreted over the domain.
\( t ::= x \) \( \) variables
\( f(t_1, \ldots , t_n) \) \( \) operation of arity \( n \)
\( p ::= x := t_1 \) \( \) deterministic assignment
\( r_1, \ldots , r_n \xleftarrow{\mathcal{A}} \) \( \) probabilistic assignments
\( p_1; p_2 \) \( \) sequential composition
\( \text{return} \ (t_1, \ldots , t_n) \) \( \) return of arity \( n \)

Figure 1. Syntax of terms and programs. \( f \) ranges over operations. \( x \) ranges over variables.

\( x, x_1, x_2, y_1, y_2, z_1, z_2 \xleftarrow{\mathbb{F}_q}; \)
\( sk := (g^{x}, x_1, x_2, y_1, y_2, z_1, z_2); \)
\( pk := (g^{x}, g^{x_1 + x_2}, g^{y_1 + x_2}, g^{z_1 + x_2}); \)
\( \text{return} \ sk \)

Figure 2. Key generation in Cramer-Shoup encryption (abstracted and simplified)

### 2.1. Background

A signature \( \Sigma \) is an indexed set \( \{ \mathcal{F}_n \}_{n \in \mathbb{N}} \) of function symbols with their arity. Given a set \( X \) of variables, the set \( \mathcal{T}_\Sigma(X) \) of terms is defined inductively as usual (see Figure 1). \( \Sigma \)-algebra \( \mathcal{A} \) for the signature \( \Sigma = \{ \mathcal{F}_n \}_{n \in \mathbb{N}} \) is given by a set \( \mathcal{A} \) and the interpretation of \( \Sigma \), which consists of a total function \( f^\mathcal{A} : A^n \mapsto A \) for each \( f \in \mathcal{F}_n \).

We denote by \( A^X \) the set of valuations \( \rho : X \mapsto A \) and for \( I \subset X \) we denote \( \rho_I \) the restriction of \( \rho \) to \( I \). \( \text{bij}^X \) denotes the set of bijections \( f : A^X \mapsto A^X \). Every valuation \( \rho \in A^X \) is extended inductively to terms and tuples of terms as follows:

\[
[x]_\rho = \rho x
\]

\[
[f(t_1, \ldots , t_n)]_\rho = f^\mathcal{A}([(t_1, \ldots , t_n)]_\rho)
\]

\[
[(t_1, \ldots , t_n)]_\rho = (([t_1]_\rho, \ldots , [t_n]_\rho)
\]

2.2. Syntax and semantics of probabilistic programs

We consider a core probabilistic language over a signature \( \Sigma \). Expressions of the language are built from variables using function symbols from \( \Sigma \). Commands are built from probabilistic and deterministic assignments, and from return commands. We assume that each program contains a single return command as its last instruction. Figure 1 gives the syntax of programs. We provide in Figure 2 a program example which corresponds to a simplified version of the key generation of the Cramer-Shoup encryption scheme. The program is performing random sampling over a finite field, and is using exponentiation in a group with fixed generator \( g \).

The semantics for such programs is standard and omitted. To preserve the generality of our results with respect to the signatures, we only consider linear programs. Note however that as it will be shown later, as soon as the signature captures at least finite fields, we can encode conditionals. Loops are currently out of scope, but this restriction still allows us to consider many interesting examples.

**Lemma 1.** For every program \( p \), there exist sets \( R \) and \( X \) and terms \( t_1, \ldots , t_k \in \mathcal{T}_\Sigma(R \cup X) \) such that \( p \) is equivalent to

\( r_1, \ldots , r_m \xleftarrow{\mathcal{A}} \) \( \text{return} \ \langle t_1, \ldots , t_k \rangle \)

with \( R = \{ r_1, \ldots , r_m \} \).

From now on, we identify commands as tuples of terms, and let \( C_k(X, R) \) denote the set of tuples \( \langle t_1, \ldots , t_k \rangle \) of terms such that \( t_i \in \mathcal{T}_\Sigma(R \cup X) \) for \( i = 1, \ldots , k \). Intuitively, \( X \) corresponds to the input of the program, which we call input variables, and \( R \) corresponds to the random samplings of the program, the random variables.

**Example 2.** The program in Figure 2 can be written as the tuple \( p \in C_k(\emptyset, \{ x, y, z, x_1, x_2, y_1, y_2, z_1, z_2 \}) \) which does not take any input, where

\[
p = (g^{x}, x_1, x_2, y_1, y_2, z_1, z_2)
\]

We may also consider simpler programs corresponding to booleans expressions.

**Example 3.** We provide Let \( \Sigma = \{ 0, 1, +, \times \} \) where 0, 1 are of arity 0, i.e., constants and +, \times are binary symbols, and \( \mathcal{A} = \{ 0, 1 \} \). With the interpretation \( +^\mathcal{A} = \oplus \) (the xor operator) and \( \times^\mathcal{A} = \land \) (the and operator) \( \mathcal{A} \) defines Boolean expressions.

We often write \( uv \) for \( u \times v \). We define the programs \( p_1, p_2, p_3 \in \mathcal{C}(\{ x \}, \{ u, v, w \}) \):

- \( p_1 = x + v \),
- \( p_2 = xv \),
- \( p_3 = uv + uv + uv \).

Let \( p = \langle t_1, \ldots , t_k \rangle \) be a command in \( C_k(X, R) \). For every \( \rho \in A^X \), we let \( [p]_\rho \in \text{Distr}(A^k) \) be the distribution \( [\langle t_1, \ldots , t_k \rangle]_{\rho_{|I}} \), where \( I \) is sampled uniformly over \( A^R \).

The next lemma gives a counting semantics of commands. The proof follows immediately from the observation that sampling is uniform.

**Lemma 4.** Let \( p \in C_k(X, R), \rho \in A^X \) and \( \bar{v} \in A^k \). Then

\[
[p]_\rho(\bar{v}) = \frac{1}{|A^k|} \cdot |S[p]_\rho,\bar{v}|
\]

where \( [p]_\rho(\bar{v}) \) is the probability of \( \bar{v} \) in the distribution \( [p]_\rho \) and \( S[p]_\rho,\bar{v} \) is defined as

\[
\{ \rho' \in A^{X \cup R} | \rho'_{|X} = \rho \land [p]_{\rho'} = \bar{v} \}
\]

**Example 5.** Let \( \rho = \{ x \mapsto 0 \} \). Continuing Example 3 we have that

\[
[p_1]_\rho(0) = [p_1]_\rho(1) = \frac{1}{2}
\]

which corresponds to the fact that \( 0 + v \) follows the uniform distribution when \( v \) is uniformly sampled. However, \( [p_2]_\rho(0) = 1 \) and \( [p_2]_\rho(1) = 0 \), as \( 0 \times v \) always evaluates to 0.
be a highlighted.

classical notion, probabilistic non-interference, which can be defined in our framework as follows.

We first introduce the notion of \( I \)-equivalence where \( I \) is a subset of inputs: we require equivalent programs to have the same distribution on all inputs that coincide on \( I \).

**Definition 10 (Independence).** Let \( I \subseteq X \) and \( p, q \in C_k(X, R) \). \( p \) and \( q \) are \( I \)-equivalent, denoted \( p \simeq_I q \), if
\[
\forall \rho^1, \rho^2 \in A^X. \rho^1 |_I = \rho^2 |_I \Rightarrow [p]_{\rho^1} = [q]_{\rho^2}
\]

When \( I = X \) we simply say that \( p \) and \( q \) are equivalent and write \( p \simeq q \).

**Example 7.** Continuing Example 5 we see that \( p_1 \simeq p_3 \). Indeed, both programs correspond to the uniform distribution.

**Remark 8.** Note that pointwise equivalence of a tuple does not imply the equivalence of the tuple. We have that \( u \simeq v \) and \( u \simeq w \) but \( (u, v) \not\simeq (v, v) \).

Another useful property is uniformity. This is easily encoded as a program equivalence.

**Definition 9 (Uniformity).** \( p \in C_k(X, R) \) is uniform if and only if \( p \simeq x_1, \ldots, x_k \) where \( \{x_1, \ldots, x_k\} \subseteq R \).

Finally we introduce the notion of independence. Intuitively, a program is independent of a subset of its inputs \( X_s \), if the output does not vary when changing the value of \( X_s \).

**Definition 10 (Independence).** \( p \in C_k(X_s \uplus X_{pub}, R) \) is independent of \( X_s \), denoted \( p \perp X_s \), if \( p \simeq x_{pub}, p \).

Independence typically captures the notion of non-interference: when the input variables are partitioned into secret variables \( X_s \) and public variables \( X_{pub} \), non-interference expresses that the program does not leak anything about the values of \( X_s \).

**Example 11.** Continuing Example 5 we have that \( p_2 \perp \{x\} \) as \( \llbracket x + v \rrbracket_{\{x + 1\}} = \llbracket x + v \rrbracket_{\{x \to 0\}} \).

The definition of independence is strongly linked to a classical notion, probabilistic non-interference, which can be defined in our framework as follows.

\[ \pi_O(p) \simeq_I \pi_O(p) \]

where \( \pi_O \) denotes the projection of \( p = p_1, \ldots, p_k \) keeping only indices in \( O \).

**2.4. Links between problems**

In this section we show relationships between equivalence, uniformity and independence. A summary of reductions between these problems is depicted in Figure 3. Conditions on the algebra for these reductions are highlighted. When the algebra is interpreted as a finite field, all decision problems are equivalent, and reduce to the uniformity of a program with only random variables. Proofs for this Section can be found in Section A.

We start by giving a Lemma which states that some program \( p \) is independent from a set of secret \( X_s \) if and only if \( p \) and a copy of \( p \) with a distinct set of secrets are equivalent.

**Lemma 13.** Let \( p \in C_k(X_s \uplus X_{pub}, R) \). We have that
\[
p \perp X_s \Leftrightarrow p \simeq p[X'_s/X_s]
\]

where \( X_s \cap X'_s = \emptyset \) and \( |X_s| = |X'_s| \).

Using this Lemma we can directly relate the decision problems for independence and uniformity.

**Proposition 14.** Independence and uniformity are polynomial time reducible to equivalence.

From the previous Lemma, we also get that if a program is uniform, it is independent of all its inputs.

**Corollary 15.** Let \( p \in C_k(X, R) \) with \( \{r_1, \ldots, r_k\} \subseteq R \). We have that
\[
p \simeq r_1, \ldots, r_k \Rightarrow p \perp X
\]

We now provide a Lemma which states that a program \( p \) is uniform if and only if it can be used to hide the value of some secret \( s \).

**Lemma 16.** Let \( A \) be a \( \Sigma \)-algebra that contains a binary symbol \( + \) such that \( + \) is right invertible. Let \( p = (p_1, \ldots, p_k) \in C_k(X, R) \) with \( \{r_1, \ldots, r_k\} \subseteq R \) and \( x_1, \ldots, x_k \) fresh input variables. We have that
\[
p \simeq r_1, \ldots, r_k \Leftrightarrow (p_1 + x_1, \ldots, p_k + x_k) \perp \{x_1, \ldots, x_k\}
\]

The reduction from uniformity to independence directly follows.

**Proposition 17.** If \( A \) contains a binary symbol \( + \) such that \( + \) is right invertible then uniformity is polynomial time reducible to independence.

We now focus on how to reduce program equivalence to uniformity. Our first result applies to finite fields \( \mathbb{F}_q \) where \( q \) is a prime power \( p^n \). We model \( \mathbb{F}_q \) by a \( \Sigma \)-algebra where
\[ p \perp X_s \iff p \simeq p[\chi_s / X_s]. \]

**Lemma 13**

\[ p \perp X_s \iff p \simeq p[\chi_s / X_s]. \]

**Lemma 16**

if \( + \) is right-invertible then \( p \simeq r \iff p + x_r \perp x_r \)

**Lemma 18**

over \( \mathbb{F}_q \), \( p \simeq p' \iff \left( \text{if } r = 0 \text{ then } (p + x)^q \text{ else if } r = 1 \text{ then } (1_k - (p' + x)^q) \right) \simeq r \)

**Lemma 20**

\[ p \simeq q \iff (p[rx / x], x) \simeq (q[rx / x], x). \]

It follows that: \( t \simeq r \iff \forall \rho, [t]_\rho(0) = \frac{1}{2} \iff \forall \rho, [p]_\rho(0) = \frac{1}{2} \)

**Proposition 19.**

In a finite field, program equivalence is polynomial time reducible to uniformity for programs.
Finally, we show that program equivalence can be encoded into equivalence of two programs that do not use any input variables, by replacing the input variables of the program by random variables that are revealed at the end of the program.

**Lemma 20.** Let \( p, q \in C_k(X, R) \) and \( r_X \) a sequence of random variables such that \( |X| = |r_X| \).
\[
p \simeq q \iff (p[r_X|X], r_X) \simeq (q[r_X|X], r_X)
\]

Hence we have the following proposition.

**Proposition 21.** \( \simeq \) is polynomial time reducible to equivalence without input variables.

3. Characterizations

To leverage existing methods from symbolic cryptography, we need to be able to reason in term of syntax rather than semantic on our different problems, as symbolic cryptographic abstract probabilities away. In this Section, we first provide a general characterization of equivalence using the semantic of equivalence.

Then, we identify precisely what are the properties required from our algebras to be able to reason only on the syntax, yielding the notion of effective algebra. Using those effective algebras, we are able to translate the semantic characterization into a purely syntactic characterization. Proofs can be found in [Section B].

3.1. Semantic characterization

We start by giving a semantic characterization of equivalence, which will later on allow us to obtain a syntactic characterization. Intuitively, two programs are equivalent if they evaluate the same number of times to the same values. More formally, they are equivalent if, for each possible value of the programs, the number of evaluations that result in the given value coincides. This means that the set of valuations for which the two equivalent programs output a given value have the same cardinality. We may then exhibit a bijection between the random variables of the two programs so that their results coincide on every valuation.

**Proposition 22.** Let \( p, q \in C_k(X, R) \). For every \( \rho_1, \rho_2 \in \mathcal{A}^X \), the following are equivalent:
1. \( [p]_{\rho_1} = [q]_{\rho_2} \)
2. \( \forall \sigma \in \mathcal{A}^R, [S[p]_{\rho_1, \sigma}] = [S[q]_{\rho_2, \sigma}] \)
3. \( \exists f \in \text{bij}^R, \forall \tau \in \mathcal{A}^R, [p]_{\rho_1 \cup \tau} = [q]_{\rho_2 \cup f(\tau)} \)

The following corollary will be used extensively throughout the paper.

**Corollary 23.** Let \( p, q \in C_k(X, R) \) and \( I \subseteq X \). We have that
\[
p \simeq_I q \iff \forall \rho^1, \rho^2 \in \mathcal{A}^X, \exists f \in \text{bij}^R, \forall \tau \in \mathcal{A}^R, \rho^1 |_I \equiv \rho^2 |_I \Rightarrow [p]_{\rho^1 \cup \tau} = [q]_{\rho^2 \cup f(\tau)}
\]

**Example 24.** Using Corollary 23, we can prove that \( x + v \simeq v \) using the bijection \( f : v \mapsto x + v \) (we may see \( f \) as a function over the terms, rather than the valuations) which satisfies \( x + v = f(v) \).

Consider now the equivalence \( p_3 \simeq u \) where \( p_3 = uv + vw + wu \). We define the function
\[
f(r, s, t) := \begin{cases} (1, 0, 0) & \text{if } (r, s, t) = (0, 1, 1) \\ (0, 1, 1) & \text{if } (r, s, t) = (1, 0, 0) \\ (r, s, t) & \text{otherwise} \end{cases}
\]
where a valuation is denoted by a tuple of values. Obviously, \( f \) is a bijection and, extending the truth table given in\[Example 5\], we see that \( f \) verifies \( \forall \tau \in \mathcal{A}^\{u,v,w\}, \left[ uv + vw + wu \right]_\tau = [u]_{f(\tau)} \)

where \( f_u \) denotes the projection of \( f(u, v, w) \) on the first component.

Relying on Corollary 23, we can now introduce a characterization of uniformity based on the notion of \( R \)-bijection. Intuitively, a program is \( R \)-bijective if for any fixed value of its inputs, the output produced by the program can be seen as a bijection over its random variables. For \( p \in C_k(X, R) \), it is possible if and only if \( k = |R| \).

**Definition 25.** \( p \in C_k(X, R) \) is \( R \)-bijective if and only if
\[
\forall \rho \in \mathcal{A}^X, \tau \mapsto [p]_{\rho, \tau} \in \text{bij}^R
\]

**Example 26.** We have that \( x + u \in \{u\}\)-bijective as for any \( \rho \in \mathcal{A}^\{x\}, \tau \mapsto (u \mapsto [x + u]_{\rho, \tau}) \) is a bijection, with itself as inverse.

**Corollary 27.** If \( p \in C_k(X, R) \) with \( R = \{r_1, \ldots, r_k\} \), then
\[
p \simeq r_1, \ldots, r_k \iff p \text{ is } R\text{-bijective}
\]

Note that for uniformity to imply \( R \)-bijectivity the condition that \( |R| = k \) is necessary.

**Example 28.** As seen before, \( x + u \simeq u \) (Example 24), and \( x + u \in \{u\}\)-bijective (Example 26). Note that \( |R| = k \) is important here, as for instance \( uv + vw + wu \) is not \( \{u, v, w\}\)-bijective, \( |uv + vw + wu| = 1 \neq |\{u, v, w\}| \), and \( uv + vw + wu \simeq u \).

3.2. Symbolic abstraction

In this Section we introduce a framework to completely axiomatize \( \Sigma \)-algebras: in some cases the term algebra equipped with an equational theory \( E \) gives a fully abstract representation of the \( \Sigma \)-algebra.
3.2.1. Primal Algebra. A central notion of this section is primality introduced in [12]:

**Definition 29.** $A$ is said to be primal if and only if

$$\forall n, \forall f : A^n \rightarrow A. \exists t \in T(\Sigma_n(x_1, \ldots, x_n)). f = t^A$$

Primality expresses that for any function in the algebra, there exists a term whose interpretation is equal to the function.

**Term algebra for $F_q$.** We denote by $F_q$ a finite field, where $q = p^n$ and $p$ is prime, and by $P$ an irreducible polynomial over $F_p[\alpha]$ of degree $n$. The $F_q$-algebra is defined by the signature

$$\Sigma_{F_q} = \{0, 1, \alpha, +, *\}$$

and their usual mathematical interpretation with $F_q$ seen as $F_p[\alpha] / (P(\alpha))$. Nipkow [12] has shown that the $F_q$-algebra is a primal algebra:

**Proposition 30 ([12]).** $F_q$ is a primal algebra.

The main idea underlying the proof relies on the encoding of conditionals of the form if $x=i$ then $t_1$ else $t_2$. We already presented such an encoding for $F_q$ in Section 2.4. This allows for a basic encoding of any function as

if $x = 0$ then $f(0)$ else ... if $x = i$ then $f(i)$ else ...

As we are working on finite sets, this encoding completely captures a function. In the case of Booleans ($q = 2$), we basically write down the truth table of the function inside a term.

**Term algebra for $(F_q)^m$.** It is interesting to note that $(F_q)^m$ can be made primal. We write tuples directly as sequences of elements, for example we denote $(0, 0, 0)$ with 000, or 03. The $(F_q)^m$-algebra is then defined by the signature

$$\Sigma_{(F_q)^m} = \{0_m, (0k10m-1-k)_{0 \leq k \leq m-1}, (0k\alpha0m-1-k)_{0 \leq k \leq m-1}, \alpha, +, *\}$$

and their mathematical interpretations, where we extend multiplication and addition to tuples component by component. This is similar to the classical notation for xor on bitstrings.

Defined this way, we still have primality for those algebras:

**Proposition 31.** $(F_q)^m$ is a primal algebra.

3.2.2. Equational theories. To fully abstract our algebras, we need to be able to capture equalities between terms. We achieve this using equational theories. An equational theory $E$ is a set of equalities $\{t_i = u_i\}$, where $t_i, u_i \in T(\Sigma, V)$ for some set of variables $V$. $E$ induces a relation $=E$ on terms defined as the smallest equivalence relation that contains equalities in $E$ and that is closed under substitutions of variables by terms, and application of function symbols.

**Definition 32.** An equational theory $E$ is said to be sound ($\iff$) and faithful with respect to $A$ if and only if

$$\forall t_1, t_2 \in T(\Sigma), t_1^A = t_2^A \iff t_1 = t_2$$

When $E$ is both sound and faithful we may use $= $ for $=E$.

The **equational theory** $E_{F_q}$. We consider the equational theory $E_{F_q}$ parameterized by $n$ and $P$ such that $q = p^n$, and $P$ is an irreducible polynomial $P \in F_p[\alpha]$ of degree $n$. Denoting by $P(\alpha)$ the term corresponding to this polynomial, we define $E_{F_q}$ as follows.

$$x + 0 = x \quad x * 0 = 0$$
$$x + y = y + x \quad x * y = y * x$$
$$x + y + z = (x + y) + z \quad x * (y * z) = (x * y) * z$$

**Proposition 33.** $E_{F_q}$ is sound and faithful with respect to $F_q$.

3.2.3. Effective algebra. We can finally define the new general class of algebras we will be able to reason about.

**Definition 34.** $(A, \Sigma, E)$ is called an effective algebra if $A$ is a finite primal term algebra over $\Sigma$, and $E$ is sound and faithful with respect to $A$.

We consider the examples where $A$ is a finite field, denoted $(F_q)^n$, where $q$ is an explicit value, and $n$ is a parameter. We may for instance study programs manipulating bitstrings of length $n$ using $F_q^n$.

Some interesting effective algebra are, for an explicit $q$ and a parameter $n$, $(Z_{p^n}, \Sigma_q, E_{Z_p})$ and $(F_q, \Sigma_{F_q}, E_{F_q})$.

We remark that effective algebras provide in the following work equivalences between probabilistic programs and symbolic methods. Yet, if the equational theory is sound but not faithful, or if the algebra is not primal, we lose completeness of our reductions, but we still keep sound proof techniques.

3.3. Symbolic characterization

We provide an extension of Proposition 22 based on effective algebras. This is the abstraction which allows us to reason only at the syntax level.

**Lemma 35.** Let $(A, \Sigma, E)$ be an effective algebra and $p, q \in C_k(X, R)$

$$p \simeq q \iff \exists T \in T(\Sigma, X \cup R)[R], T \text{ R-bijective and } p =_E q[T / R]$$

Moreover, if $E$ is sound, but not faithful, then the implication from right to left $(\Rightarrow)$ still holds.

**Example 36.** Consider again $uv + vw + uw \simeq u$ from Example 5. A valid witness of this equivalence is $t(u, v, w) = (uv + uw + vw, u + v, u + w)$.
To show that \( t = (t_1, t_2, t_3) \) is indeed a valid witness, we show that \( t \) is \( R \)-bijective, which we do by exhibiting the inverse. We have \( t_2t_3 = uw + uw + uv + u = t_1 + u \). Thus, we have \( t_2t_3 + t_1 = u \). Then, \( t_2 + t_2t_3 + t_1 = v \) and \( t_3 + t_2t_3 + t_1 = w \). Finally, if we set \( g(x_1, x_2, x_3) := (x_2x_3 + x_1 + x_2 + x_2x_3 + x_1, x_3 + x_2x_3 + x_1) \), we find that \( g(t(u, v, w)) = (u, v, w) \), i.e \( t \) is a bijection.

4. Symbolic methods for probabilistic programs

Using the previous syntactic characterization of equivalence, we can now leverage several techniques coming from the theory or symbolic cryptography, such as deduction, deduction constraints and static equivalence. Proofs for this Section can be found in Section C.

4.1. Using deduction to check uniformity

One of the most fundamental notions in symbolic cryptography is deduction: it captures what terms can be computed by an adversary from a given set of terms. It is formally defined as follows.

Definition 37. Let \( \Sigma \) be a signature equipped with \( E, V \) a set of variables, and \( t_1, \ldots, t_k, s \in T(\Sigma, V) \). We say that \( s \) is deductible from \( t_1, \ldots, t_k \), denoted \( t_1, \ldots, t_k \vdash_E s \) if and only if:

\[ \exists R \in T(\Sigma, \{x_1, \ldots, x_k\}). R \sigma = E \]

where \( x_1, \ldots, x_n \) are variables disjoint from \( V \) and \( \sigma = \{(x_1 \mapsto t_1, \ldots, x_k \mapsto t_k)\} \).

Intuitively, the term \( u \) models the computation of the adversary, and the variables \( x_i \) are used as handles to refer to the corresponding terms \( t_i \).

Example 38. Consider \( \Sigma_{\mathbb{G}} = \{0, +\} \) and the equational theory \( E_{\mathbb{G}_q} \) defined as the subset of \( E_{\mathbb{G}_q} \) with \( q = 2 \) defined over \( \Sigma_{\mathbb{G}} \). We have that:

\[ u + v + w, v + w \vdash_{E_{\mathbb{G}_q}} u \]

witnessed by the term \( R = x_1 + x_2 \). However,

\[ u + v + w, v + w \not\vdash_{E_{\mathbb{G}_q}} u + w \]

We now show that, on effective algebras, deduction can be used to decide uniformity. When the equational theory is sound (which is generally straightforward), but not necessarily complete, deduction can still be used as a proof technique, as in that case our encoding still implies uniformity.

The intuition behind the result is given through Corollary 27 which links uniformity and bijectivity. As a bijective function is a function which gives its outputs allows to recompute its inputs, it can be checked if a function is bijective using deduction.

Proposition 39. Let \( (A, \Sigma, E) \) be an effective algebra and \( p \in C_k(X, R) \) with \( R = \{r_1, \ldots, r_k\} \). We have that

\[ p \simeq r_1, \ldots, r_k \Leftrightarrow \forall r_i \in R, p, X \vdash_E r_i \]

Moreover, if \( E \) is sound, but not faithful, the implication from right to left \((-\Rightarrow\) still holds.

4.2. Deduction constraints and unification for program equivalence

In this section we show how deduction constraint as used in symbolic cryptography [10] and (equational) unification can be used to verify program equivalence. Deduction constraints generalize deduction from ground terms to terms that contain variables that have to be instantiated by the attacker.

Definition 40. Let \( \Sigma \) be a signature equipped with \( E, V \) a set of variables. A deduction constraint is an expression \( \exists \sigma \in C \). We have that:

\[ T_1 \vdash_E u_1 \land \ldots \land T_n \vdash_E u_n \]

where \( T_1, \ldots, T_n \) are finite sets of terms, \( u_1, \ldots, u_n \) are terms.

A substitution \( \sigma \) with \( \text{dom}(\sigma) = V \) is a solution over variables \( V \) of a deduction constraint system if and only if

\[ \forall i. T_i \sigma \vdash u_i \sigma \]

A deduction constraint system may satisfy additional properties:

- monotonicity: \( \emptyset \subseteq T_1 \subseteq \ldots \subseteq T_n \)
- origination: \( \forall i. \text{vars}(T_i) \subseteq \text{vars}(u_1, \ldots, u_{i-1}) \)
- one-turn: \( \forall i, j. T_i \equiv T_j \land \text{vars}(u_i) = \emptyset \)

Monotonicity and origination are classical notions that are naturally satisfied in the context of security protocols and exploited in decision procedures: monotonicity ensures that the attacker knowledge (the \( T_i \)'s) only grows, and origination ensures that any variable appearing in the attacker knowledge has been instantiated in a previous constraint.

The one-turn property is novel: it requires that the attacker knowledge is invariant and that all variables actually appear in the attacker knowledge. We show in Section D that extending the signature with a homomorphic function symbol allows to transform a one-turn constraint system into a constraint system that satisfies origination and monotonicity while preserving solutions.

Unification [17] is the problem that, given two terms, asks to find a substitution which makes the terms equal in the equational theory. For any terms \( u, v \), we denote by \( \text{unifier}(\Sigma, \mathcal{V})(u, v) \) the set of most general unifiers of \( u \) and \( v \) over \( \Sigma \), equational theory \( E \) and variables \( \mathcal{V} \). A set of unifiers is a most general set of unifiers if for any unifier \( \sigma \), there exists a most general unifier \( \mu \), such that \( \sigma \) is an
instance of \( \mu \), i.e., there exists a substitution \( \theta \) such that 
\[ \sigma = E \, \mu \theta. \]

We now reduce program equivalence to unification and solving of one-turn deducibility constraint systems. For any set of variables \( V \), we denote by \( V^0 \) a set of corresponding function symbols of arity 0, one for each element of \( V \).

**Lemma 41.** Let \( \langle A, \Sigma, E \rangle \) be an effective algebra and \( p, q \in C_k(X, R) \). Let \( R' \) be a set of variables such that \( |R'| = |R| \) and \( R' \cap R = \emptyset \). We have that
\[ p \sim q \iff \exists \, r \in R' \exists \, \sigma \in U_{\Sigma \cup X \cup p} \, E, R \, (p, q). \]
Moreover, if \( E \) is only sound, but not faithful, the implication from right to left (\( \Rightarrow \)) still holds.

Note that in some cases, the most general unifier may not contain any fresh variables. Then the constraint solving problem is simply a deduction problem. For instance, by restricting equivalence to uniformity, the unifier becomes trivial and we obtain the following corollary:

**Corollary 42.** Let \( \langle A, \Sigma, E \rangle \) be an effective algebra and \( p_1, \ldots, p_m \in C_1(X, R) \) where \( R = \{ r_1, \ldots, r_n \} \) and \( m < n \). Let \( R' \) be a disjoint set of variables such that \( |R'| = n \). We have that
\[ p_1, \ldots, p_m \sim r_1, \ldots, r_m \iff \exists \, p_{m+1}, \ldots, p_n \in C_1(X, R). \]
Moreover, if \( E \) is sound, but not faithful, then the implication from right to left (\( \Rightarrow \)) still holds.

### 4.3. Static equivalence and non equivalence

The notion of static equivalence was introduced in [18] and its decidability has been studied in [19]. Static equivalence expresses the inability of an adversary to distinguish two sequences of messages.

**Definition 43.** Let \( \Sigma \) be a signature equipped with \( E, V \) a set of variables, and \( t_1^1, \ldots, t_k^1, t_1^2, \ldots, t_k^2 \in T(\Sigma, V) \). Two sequences \( t_1^1, \ldots, t_k^1 \) and \( t_1^2, \ldots, t_k^2 \) are statically equivalent in \( E \), written \( t_1^1, \ldots, t_k^1 \sim_E t_1^2, \ldots, t_k^2 \) iff
\[ \forall u_1, u_2 \in T(\Sigma, (x_1, \ldots, x_k)). \quad u_1 \sigma_1 \uparrow \iff u_2 \sigma_2 \uparrow \]
where \( x_1, \ldots, x_n \) are variables disjoint from \( V \) and \( \sigma^i = \{ x_1 \mapsto t_1^i, \ldots, x_k \mapsto t_k^i \} \).

Intuitively, two sequences of terms are statically equivalent if the set of relations between terms are the same on both sequences.

**Example 44.** Consider again the signature \( \Sigma_{\oplus} \) and the equational theory \( E_{\oplus} \) introduced in [Example 38]. Then we have that
\[ u \oplus v, v \oplus w, u \oplus w \not\sim_{E_{\oplus}} u, v, w \]
as the relation \( x_1 + x_2 = x_3 \) holds on the left hand side but not on the right hand side. However,
\[ u \oplus v, v \oplus w, w' \sim_{E_{\oplus}} u, v, w \]

This notion has similarities to program equivalence.

We show that indeed program equivalence implies static equivalence, and hence static non-equivalence may be used to show that two programs are not equivalent.

**Proposition 45.** Let \( \langle A \rangle \) be a finite primal algebra over \( \Sigma \) with a sound equational theory \( E \), and \( p, q \in C_k(X, R) \). We have that
\[ p \not\sim_{E} q \Rightarrow p \not\sim q \]

**Example 46.** The converse does not hold. Consider the Boolean algebra \( \mathbb{F}_2 \) and let \( u \) and \( v \) be random variables. We have that \( uv \sim_{E_{\oplus}} u \), but \( uv \not\sim u \), as \( uv \) and \( v \) do not follow the same distribution.

Static non-equivalence can then be used to contradict non-interference.

**Corollary 47.** Let \( \langle A \rangle \) be a finite primal algebra over \( \Sigma \) with a sound equational theory \( E \), \( p, q \in C_k(X_{pub} \uplus X_s, R) \), and \( X_s' \) such that \( |X_s| = |X_s'| \).
\[ p, X_s \not\sim_{E} p[X'_s/X_s], X_s \Rightarrow p \not\sim X_s \]

### 5. Deriving heuristics

We can derive from our framework different ways to solve a specific problem, providing principled algorithms that are sound and/or complete. In this section, we illustrate how such algorithms might be obtained either for general algebras or more precisely for Boolean algebras. We focus mostly on the case of Boolean algebras, which is of particular interest for cryptography. Nevertheless, procedures for more general settings can be obtained, such as the bilinear setting.

Regarding Boolean algebras, a first interesting subcase is soundness and/or completeness. In the linear case, where only the xor is used. It was previously explored and shown useful in [20], and Using our work we are able to derive more general results, going beyond linearity and uniformity.

In the general case of Boolean algebras (xor and conjunction), we develop several heuristics. As deciding equivalence for a given finite field \( \mathbb{F}_q \) is at least coNP-hard (most likely strictly above NP and coNP, unless the polynomial hierarchy collapses; we refer the reader to Section D for our results on computational complexity) developing such heuristics is particularly important.

Our techniques rely on well established symbolic methods, and novel extensions of these techniques (detailed in Section D). A summary of related, existing results is given in Section F.

#### 5.1. Soundness and completeness

A first important consideration is that when given an algebra and an instance of a problem, we might be unable
to solve it using a sound and faithful equational theory, i.e., a theory that matches exactly the algebra. However, our previous results allow us to either add equations and maintain completeness, or remove equations maintaining soundness.

Example 48. Let \( r \) be a random variable and \( x \) an input variable.
- \( u + x \) is uniformly distributed for any \( \mathbb{F}_p \), as we can prove that it is uniformly distributed in \( \mathbb{F}_2 \) ring theory, which is a sound theory for any \( \mathbb{F}_p \);
- \( u \times x \) is never uniform for any \( \mathbb{F}_p \), as we can prove that it is not uniform in \( E_{\mathbb{F}_2} \), which is a faithful theory for \( \mathbb{F}_p \);
- \( uv + uw + uw \) is uniform over \( \mathbb{F}_2 \) but not \( \mathbb{F}_4 \), while \( E_{\mathbb{F}_2} \) is faithful for \( \mathbb{F}_4 \);
- \( u \times u \) is not uniform over a ring, but is over \( \mathbb{F}_2 \), while a ring theory is sound for \( \mathbb{F}_2 \).

5.2. Boolean algebras: the linear case

Consider programs only built on Booleans and the \text{xor} operator, i.e., without conjunction, which corresponds to an Associative Commutative Unit Nilpotent (ACUN) theory. Uniformity and independence can be decided by program equivalence (see Figure 3). Program equivalence can be decided using unification and solving one-turn deduction constraint systems thanks to Lemma 41. Unification is solvable in polynomial time for ACUN theories [21]. Solving one-turn deduction constraints for the ACUN theory can be reduced to solving classical reduction constraints for the ACUNh theory where \( h \) is a private symbol (Lemma 56 of Section D).

This problem is decidable in polynomial time [22]. Note that [22] does not claim to handle private symbols, but they compute a basis of all possible solutions, and there exists a solution with \( h \) being private if only if we can find one in the basis that does not use \( h \). The general decision procedures consist of two steps, first a unification, and then the resolution of a linear system over a polynomial ring using Gaussian elimination.

The authors of [20] solved uniformity in the linear case using Gaussian elimination. Our decision procedures are essentially the same for uniformity. However we extend the procedures to independence and program equivalence by adding the unification step, and also add support for the non linear case.

5.3. Boolean algebras: the general case

Consider programs over a general Boolean algebra, i.e., including \text{xor} and conjunction operators. This setting is unlikely to admit an efficient procedure for solving program equivalence. We therefore use our framework to derive several heuristics.

Example 49. Going back to Example 5, let \( p = uv + uv + uv \). We cannot directly show that \( p \equiv u \). However, if we extend \( p \) into a program of size 3, we may decide that
\[
(uv + uv + w, u + w, u + v) \equiv (u, v, w)
\]
using deduction in finite fields (cf Section D).

Example 50. Let \( u, v, w \) be three random variables. We can show using deduction in a commutative ring [23] that:
\[
(u, v + uv, w - uv - v) \equiv (u, v, w)
\]
Indeed, \( u, v + uv, w - uv - v \equiv u, v, w \) by computing:
\[
r, s, t \mapsto r, s - rt - rs, s + t
\]

We now consider several heuristics that may be used in this context.

5.3.1. Unification. Lemma 41 may yield a deduction constraint system with a trivial solution \( R' \rightarrow R \) that can be checked with deduction. This provides an efficient heuristic for program equivalence.

Example 51. Let \( u \) be a random variable, \( x, s, s' \) three input variables, \( p = x(u + s) \) and \( q = x(u + s') \). We have that \( x(u + s) \equiv x(u + s') \), by Lemma 41 with the unifier \( u' \mapsto u + s' + s \) on \( x(u + s) \equiv x(u' + s') \) which is trivially bijective when seen as a function on \( u \).

5.3.2. Derandomization. Given a program, all of its randomness may not be needed to satisfy a given property (this is a trivial consequence of Proposition 22). We may for instance prove that a program is uniform even when we replace some of its random variables by input variables. This reduces the domain of the bijection needed to prove uniformity, and may simplify the proof. The same derandomization technique can be applied to equivalence or independence.

Example 52. Let \( u, v \) be random and \( x, s, s' \) be input variables. We may solve \( v(u + s) \perp s \) by replacing \( v \) with the input variable \( x, s \) and prove \( x(u + s) \perp s \), which follows directly from Example 51 and Lemma 13.

5.3.3. Solving uniformity through independence. Proposition 17 may be used to prove uniformity through independence. This may be useful as it may simplify the deducibility constraints obtained by Lemma 41.

Example 53. Consider again \( p = uv + uv + uv \). Applying Lemma 41 directly we obtain the deducibility constraint
\[
(uv + uv + vu, v') \vdash t \quad u, v, w
\]
which admits no obvious solution. Let \( s, s' \) be two input variables. Applying Proposition 17 uniformity of \( p \) is equivalent to \( uv + uv + vu + s \perp s \), which in turn holds if and only if
\[
(uv + uv + vu + s \equiv u'v' + v'u' + v'u' + s')
\]
By setting \( s \) to \( s + s' \), this is directly equivalent to
\[
(uv + uv + vu + s \equiv u'v' + v'u' + v'u')
\]
On this equivalence, Lemma 41 provides a unifier \( u', v', w' \mapsto u + s, v + s, w + s \), which is a faithful theory for a ring containing \( u, v, w \) with \( u, v, w \mapsto u + s, v + s, w + s \), which is never uniform for any \( \mathbb{F}_2 \), but is over \( \mathbb{F}_4 \), while \( E_{\mathbb{F}_2} \) is faithful for \( \mathbb{F}_4 \).
5.3.4. Brute forcing the witnesses space. We know that we can reduce each of the problems we study to testing uniformity of fully random programs, using the encoding given in Figure 3. Let \( p \in C_n(X, R) \) where \( R = \{r_1, \ldots, r_m\} \). Using Proposition 39, uniformity is equivalent to the existence of a program \( \alpha \in C_{n-m}(X, R) \) such that \( P, \alpha \) is uniform, which can be verified using deducibility by Proposition 39. Such an \( \alpha \) exists if and only if \( p \) is uniform.

This yields a sound and complete procedure. However, the procedure has an exponential running time, as we need to search over all possible polynomials. This technique is nevertheless of interest when combined with approximations of the algebra, because the witness found might be valid only using a subset of the equations over the algebra.

5.4. Extension to more complex algebras

In [24] the question of deducibility or static equivalence for the union of disjoint theories is reduced to the deducibility or the static equivalence in each theory. We extend their result for deducibility to typed equational theory in Section D. By Proposition 39 we reduce uniformity to deducibility. Thus, we may use our results on algebras that are the union of disjoint algebras.

We may for instance consider programs over both Boolean linear expressions and group exponentiation with linear maps. Indeed, deducibility for Boolean linear expressions is decidable in polynomial time [25], and deducibility in the bilinear setting was studied in [23] (which we extend to the case where the finite field is of explicit size in Section D).

We may also consider a combination of any theory extended with free function symbol. The free function symbols might for instance represent arbitrary code, e.g., attacker actions inside cryptographic games.

5.5. Interference witnesses

Corollary 47 allows to find witnesses of non interference. The idea is that given \( p = (p_1, \ldots, p_k) \in C_{k}(X, R) \), every relation of the form \( C_1[p_1, \ldots, p_n] = C_2[X'] \) where \( X' \subset X \) and \( C_1, C_2 \) are contexts is a witness that \( p \notin \{X'\} \).

The techniques for deciding static equivalence developed in [23] provide an algorithm that computes the set of relations satisfied between multiple polynomials. Based on this work, we develop an algorithm that, given a program, returns a set of variables that do not verify non interference, and witnesses to verify the leakage. This is done by computing the set of all relations over the program and the secrets, keeping those that are actual witnesses, and simply outputting the set of all secrets leaked according to the witnesses.

5.6. Sampling from multiple distributions

We considered that programs were only performing random sampling over a single distribution, the uniform distribution over some algebra. In practice, we often sample random variables over multiple distributions or domains. Our results, and notably Proposition 22 can be extended to this case, for instance by considering programs built over \( C_k(X, R_1, \ldots, R_n) \) and exhibiting bijections over each \( R_i \) to prove program equivalence.

6. Applications

We provide in this section applications of our techniques, describing how we developed a library based on some of our results and integrated it into two cryptographic tools, EASYCRYPT [7], [26], and MASKVERIF [11]. The implementations are available online [13], [14], [15].

6.1. Implementation of a library

We implemented parts of our framework as an OCaml library. Given that our main focus is automation of cryptographic proofs, the library provides procedures to handle programs over finite fields. We implemented Gröbner basis techniques developed for deciding deducibility in rings [23]. This allows for rings of both characteristic 0 and 2 to:

- compute the inverse of a function over multiple variables,
- compute the set of relations between elements.

With those building blocks and based on the practical considerations of the previous section, we thus provide algorithms that can either be sound or complete (Section 5.1) to:

- decide uniformity through deducibility (Proposition 39);
- heuristically prove uniformity through derandomization (Section 5.3.2);
- provide a witness of interference through static non-equivalence (Section 5.5);
- provide a witness of non-interference through uniformity (Corollary 15).

In particular, our building blocks for rings of characteristic 0 and 2, yield a sound heuristic to show uniformity through derandomization and deducibility for any finite field \( \mathbb{F}_2^n \). We may also use static non-equivalence to provide a witness of interference which is valid for any finite field \( \mathbb{F}_2^n \).

6.2. Integration in MASKVERIF

MASKVERIF is a tool developed to formally verify masking schemes, i.e., counter measure against differential power analysis attacks.

MASKVERIF allows to check security properties like \( n \)-probing security, non-interference (NI) and strong non-interference (SNI). Those notions are strongly related to Definition 12 of probabilistic non-interference. All properties require that for any \( n \)-tuple of sub-expressions \( p \) used inside the program the following base property is satisfied: there exists a subset of the input \( I \) such that \( p \preceq_I p \). Furthermore, there is a cardinality constraint on \( I \) but this is
6.3. Integration in EASYCRYPT

6.3.1. The \texttt{rnd} rule. In cryptographic games, one may replace an expression by another expression as long as both expressions range over the same distribution. E.g., if an expression is uniformly distributed, one may replace it by a variable that is sampled at random.

In EASYCRYPT, this is done via the proof tactic \texttt{rnd}. It allows to replace an expression of the form $f(x)$ by $x$ if $f$ is invertible - i.e., if one can give an effective expression for $f^{-1}$. For instance, we may replace $x \oplus y$ by $x$ because $\oplus$ is an involution, i.e., $(x \oplus y) \oplus y = x$.

Listing 1 presents actual examples of the \texttt{rnd} tactic, where the bijection inverse is specified by hand.

Using our library, it has been possible to enhance the \texttt{rnd} tactic by making optional the bijection inverse expressions — in that case, EASYCRYPT tries to automatically compute the inverse. Our library is powerful enough to remove all the explicit inverse expressions in every occurrence of the \texttt{rnd} tactic in the EASYCRYPT standard libraries and examples.

6.3.2. Simultaneous \texttt{rnd} rules. Based on the fact that we can actually compute inverse for tuples, we developed an EASYCRYPT tactic which allows to reduce the distance between cryptographic pen and paper proofs and EASYCRYPT proofs. It appears that in this field of applications, the hypothesis of Proposition 39 about the number of outputs of the program equal to the number of its random variables is not a restriction.

As an example, let us consider once again the Cramer-Shoup encryption scheme of Figure 2. We consider a goal - in that case, EASYCRYPT tries to automatically compute the inverse. Our library is powerful enough to remove all the explicit inverse expressions in every occurrence of the \texttt{rnd} tactic in the EASYCRYPT standard libraries and examples.

\begin{verbatim}
> examples/elgamal.ec
rnd (fun x z ⇒ x - log (if b then \texttt{ml} else \texttt{m0})(2))
(fun x z ⇒ log (if b then \texttt{ml} else \texttt{m0})(2)).
> examples/cramer-shoup/cramer_shoup.ec
rnd (fun x2 ⇒ \{x2 + G2.v \* G1.y2\} + G1.u \* (G1.x + G2.v \* G1.y))\{2\}
(fun x ⇒ \{x - G1.u \* (G1.x + G2.v \* G1.y)\} / (G1.w \* (G1.u' - G1.u)) - G2.v \* G1.y2){2}.
> examples/incomplete/oaep/OAEP.ec
rnd (fun x ⇒ x + \texttt{pad} (if \texttt{b} then \texttt{n0} else \texttt{m1})(2))
(fun x ⇒ x - \texttt{pad} (if \texttt{b} then \texttt{n0} else \texttt{m1})(2)).
\end{verbatim}

Listing 1. Examples of the EASYCRYPT \texttt{rnd} tactic

We propose here a new method that limits incompleteness and, importantly, provides a witness of interference.

We use two functionalities of our library. A witness of interference can be obtained when checking static non-equivalence and a witness of non-interference can be obtained by showing uniformity which allows to prove that a given tuple verifies the base property. Those two procedures are sound but not complete, yielding results for all finite fields $\mathbb{F}_2^n$.

Combining both procedures, we obtain an efficient algorithm. We first compute a set of secrets that are leaked from the tuple. If this set of secrets is already larger than the expected size of $I$, we have a proof that the tuple depends on too many inputs. Otherwise, we try to prove that the remainder of the tuple, the part for which we did not find any obvious secret leakage, is actually independent of the remaining secrets by proving that it is uniform. This last step is still incomplete.

The modification of \texttt{MaskVerif} is minor: we first try the original heuristic and if it fails we call the new heuristic based on our library. This change does not affect efficiency when the original heuristic succeeds but allows to prove new examples, such as proving that the masked multiplication proposed in [27] is NI and SNI. The key point for this example is to prove independence of tuples of the form:

\[
x_1y_1 + (x_1(y_0 + r) + (x_1 + 1)r)
\]

where $r_0$ and $r_1$ are the random variables.

A major advantage is that we can now provide witnesses of interference, ensuring that a proof failure is not a false negative. For instance, when analysing a masked implementation of an AND gate, among the 3,784 tuples to check, there is a tuple $(t_1, t_2, t_3)$ of the form

\[
((a_1b_1 + r_1) + a_2b_0 + a_0b_1 + r_0, r_0, r_1)
\]

A simple witness of interference that we obtain is then the equation $t_1 + t_2 + t_3 = a_1b_1 + a_2b_0 + a_0b_1$.

Our procedure does output a witness demonstrating that the masked implementation of the Verilog implementation of the AES Sbox as designed in [28] is not NI at order 1.

We remark that our algorithm may also be applied to the verification of multi party computation protocols, as shown in Section G.
is a bijection - for example, the inverse of $X_1 \mapsto X - w \cdot X^2$ being $r \mapsto r + w \cdot X^2$.

Currently, performing this part of the proof in EASYCRYPT requires a mixture of code motion, code inlining and single application of the $\text{rnd}$ rules, as shown in Listing 2.

Listing 2. EASYCRYPT proof script

Using our techniques, we were able to replace it with a single line tactic, using the new tactic $\text{rndmatch}$.

\begin{verbatim}
\text{swap1} \{1\} \{1\} \{0\}; \text{wp}; \text{swap} \{-1\}; \text{swap} \{-1\}.
\text{rnd} \{\text{fun} \ z \Rightarrow z + \text{G1} \cdot \text{G2} \cdot \text{G3} \cdot \text{G4}\}
\{\text{fun} \ z \Rightarrow z - \text{G1} \cdot \text{G2} \cdot \text{G3} \cdot \text{G4}\}.
\text{rnd}; \text{wp}; \text{swap} \{-1\}.
\text{rnd} \{\text{fun} \ z \Rightarrow z + \text{G1} \cdot \text{x2} \cdot \text{G3} \cdot \text{G4}\}
\{\text{fun} \ z \Rightarrow z - \text{G1} \cdot \text{x2} \cdot \text{G3} \cdot \text{G4}\}.
\text{rnd}; \text{wp}; \text{rnd}; \text{wp}.
\end{verbatim}

The underlying idea is that a user should only specify the variables on the left which he wishes to map to expressions on the right. The tactic handles the necessary code motions and inlinings into the game until it produces a tuple at the end. The tactic automatically solves the equivalence using the enhanced $\text{rnd}$ tactic:

\begin{verbatim}
\text{rnd} \{\text{fun} \ \{v1, v2, v3, v4, v5, v6, v7, v8, v9\} \Rightarrow
\{v1, v3, v4 \cdot v5 + v5, v5, v6 - v3 \cdot v7, v7, v8 - v2 \cdot v9, v9\},
\end{verbatim}

7. Related work

Our work explores the relationship between probabilistic and symbolic approaches to cryptography. The probabilistic approach is focused on computational or information-theoretic notions of security, which are modelled using probabilistic experiments. The symbolic approach uses methods from universal algebra, automated reasoning and logic to model and reason about security. Both models have been used extensively in the literature, and there is active research to develop formal methods and tools for proving security in these models.

The connection between these two approaches was first established by Abadi and Rogaway [29], who prove computational soundness of symbolic security proofs for symmetric key encryption: under specific assumptions, protocols that are secure in the symbolic model are also secure in the computational model. Their seminal work triggered a long series of results for other cryptographic constructions [30]. The difficulty in computational soundness results stems from the fact that the soundness of a security proof requires that every possible behaviour of a computational adversary is captured by a symbolic adversary. In our work, we exploit soundness of symbolic attacks: every symbolic attack (e.g., an attacker deduction) corresponds to a computational attack. This form of soundness is generally obtained by construction, as every symbolic term induces a probabilistic algorithm. This form of connection originates from the work of Barthe et al [8] on automatically verifying and synthesizing RSA-based public-key encryption. This connection was further extended in [9] and [10] to deal with pairing-based and lattice-based cryptography.

Our work is also closely related to approaches to reason about equivalence and simulatability of probabilistic programs. Barthe et al [31] show decidability of equality for probabilistic programs (without conditionals or oracle calls) over fixed-length bitstrings. Jutla and Roy [32] show decidability of simulatability for programs (with conditionals but no oracle calls) over finite-length bitstrings.

Applications of symbolic methods to masking were considered by Barthe et al in [11], [33], who develop special-
ized logics to prove different notions of (threshold) non-interference.

8. Conclusion

We have developed new symbolic methods for information-theoretic and computational security proofs of cryptographic constructions. Our methods leverage the natural correspondence between commonly used techniques from the code-based game-playing technique and symbolic tools, including deducibility and unification. The benefits of our approach are two-fold. From a theoretical point of view, it provides new insights on the relationship between computational and symbolic approaches to cryptography. From a practical perspective, it helps improving automation and robustness of computer-aided proofs (in contrast with heuristics, which are often limited and brittle).

An important direction for future work is to extend our approach to discover cryptographic reductions automatically. Such an extension could form the basis of a principled embedding into EASYCRYPT of the automated logics from [9] and [10]. More speculatively, it would be interesting to leverage connections between equivalence of programs (for the class considered in this paper) and permutation polynomials [34]. Indeed, it follows from our results that program equivalence over finite fields reduces in polynomial time to deciding if a set of multivariate polynomials is an orthogonal system. Thus, it would be interesting to develop efficient procedures for checking whether a system of polynomials is orthogonal. We provide a more detailed presentation of those perspectives in [Section C].

Finally, it would be interesting to obtain a better understanding of the decidability and computational complexity of equivalence of programs over finite fields $\mathbb{F}_2^\alpha$ for all $\alpha$. While we show that the use of procedures for program equivalence over a commutative ring of characteristic two is a sound abstraction, completeness is currently open. Also note that while we can decide uniformity in some cases on such rings, program equivalence can only be verified heuristically. To the best of our knowledge, there currently exists no unification procedure for commutative rings of characteristic 2: there seems to be a gap in the literature, between unification over commutative rings which is undecidable (due to Hilbert tenth problem) and the procedure for primal algebras and finite fields [12].

Acknowledgements. We wish to thank the anonymous reviewers for their useful comments. We are grateful for the support by the ERC under the EU’s Horizon 2020 research and innovation program (grant agreements No 645865-SPOOC), the French National Research Agency (ANR) under the project Tecap (ANR-17-CE39-0004-01), and ONR Grant N000141512750.

References

The second equivalence is proven by choosing \( \rho \), respectively \( \rho_1, \rho_2 \), such that \( \rho_1(X_s) = \rho(X_s), \rho_2(X_s) = \rho(X'_s) \) and \( \rho_1(X_{pub}) = \rho_2(X_{pub}) = \rho(X_{pub}) \).

**Corollary 15.** Let \( p \in C_k(X, R) \) with \( \{ r_1, \ldots, r_k \} \subseteq R \). We have that

\[
p \succeq r_1, \ldots, r_k \Rightarrow p \perp X
\]

**Proof.** Let \( p \in C_k(X, R) \) and \( \{ r_1, \ldots, r_k \} \subseteq R \) such that \( p \succeq r_1, \ldots, r_k \), and \( X' \) a set of input variables such that \( |X| = |X'| \) and \( X \cap X' = \emptyset \). We have that \( p \succeq r_1, \ldots, r_k \), thus \( p[X'/X,] \succeq r_1, \ldots, r_k \). By transitivity, we have that \( p \succeq p[X'/X,] \), and we conclude with Lemma 13 instantiated with \( X_{pub} = \emptyset \) and \( X_s = X \).

**Lemma 16.** Let \( A \) be a \( \Sigma \)-algebra that contains a binary symbol \( + \) such that \(+A\) is right invertible. Let \( p = (p_1, \ldots, p_k) \in C_k(X, R) \) with \( \{ r_1, \ldots, r_k \} \subseteq R \) and \( x_1, \ldots, x_k \) fresh input variables. We have that

\[
p \succeq r_1, \ldots, r_k \Leftrightarrow (p_1 + x_1, \ldots, p_k + x_k) \perp \{ x_1, \ldots, x_k \}
\]

**Proof.** Let \( p \in C_1(X, R) \) with \( r \in R \) and \( x \) a fresh deterministic variable. We have that \( p \succeq r \Leftrightarrow p + x \perp x \):

\[
p + x \perp x
\]

\[
\Leftrightarrow \forall \rho \in A^{(x)\cup X}. \ [p + x]_{\rho} = [p + x]_{\rho_1}
\]

\[
\Leftrightarrow \forall \rho \in A^{(x')\cup X}. \ [p + x + x']_{\rho} = [p]_{\rho} \ (\text{as } x \text{ is right invertible})
\]

\[
\Leftrightarrow \forall \rho \in A^{(z)\cup X}, \forall c \in A, \ [\{ \tau \in A^R | z \in A^R | \rho_{\tau} = c \}]
\]

\[
\Leftrightarrow \forall \rho \in A^{(z)\cup X}, \forall c \in A,
\]

\[
\ |\{ \tau \in A^R | \rho_{\tau} = c' \} | = |\{ \tau \in A^R | \rho_{\tau} = c \}|
\]

\[
\ (\text{as } x \text{ is right invertible})
\]

\[
p \succeq r
\]

We may easily generalize for any tuple of values. The reduction is indeed polynomial as the size (number of symbols) of \( p + x \) is the size of \( p \) plus two.

**Lemma 18.** Let \( p, p' \in C_k(X, R) \) be programs over \( F_q \). Let \( r = (r_1, \ldots, r_k) \) be a tuple of random variables and \( x = (x_1, \ldots, x_k) \) a tuple of input variables, not appearing in \( p \) nor \( p' \). We have that

\[
p \succeq p' \Leftrightarrow T(p + x, p' + x) \succeq r
\]

where

\[
T(p, p') := \begin{cases} 
  p^{q-1} & \text{if } r = 0_k \\
  1 - p^{q-1} & \text{else if } r = 1_k \\
  1 - k - p^{q-1} & \text{else } r
\end{cases}
\]

**Proof.** Fix a valuation \( \rho \) of \( X \). We denote by \([s]_{\rho}(i)\) the probability that the program \( s \) is equal to \( i \). For any program \( s \in C_k(X, R) \), we have that \([s^{q-1}]_{\rho}(0_k) = [s]_{\rho}(0_k)\), \([s^{q-1}]_{\rho}(0_k) = 1 - [s]_{\rho}(0_k)\), and \([1 - s^{q-1}]_{\rho}(1_k) = [s]_{\rho}(0_k)\). Then, we have that:

\[
\begin{align*}
[T(p, p')]_{\rho}(0_k) &= [r]_{\rho}(0_k)[p]_{\rho}(0_k) + [r]_{\rho}(1_k)(1 - [p']_{\rho}(0_k)) + 0 \\
&= \frac{[p]_{\rho}(0_k)}{q^r} + 1 - \frac{|p']_{\rho}(0_k)}{q^r}
\end{align*}
\]

\[
[T(p, p')]_{\rho}(1_k) = [r]_{\rho}(0_k)(1 - [p']_{\rho}(0_k)) + [r]_{\rho}(1_k)[p']_{\rho}(0_k) + 0 = \frac{1 - |p']_{\rho}(0_k)}{q^r} + \frac{|p]_{\rho}(0_k)}{q^r}
\]

\[
\forall i \in F_q \setminus \{ 0_k, 1_k \}, \ [T(p, p')]_{\rho}(i) = [r]_{\rho}(0_k) + [r]_{\rho}(0_k) + [r]_{\rho}(1_k) \times 1
\]

We thus have that \( T(p, p') \succeq r \Leftrightarrow [p]_{\rho}(0_k) = [p']_{\rho}(0_k) \), i.e., this encoding allows us to test if the distribution of \( p \) and \( p' \) coincide in one point. To test all possible points, we add a fresh input variables \( x = (x_1, \ldots, x_k) \), which allows us to test all points.

\[
T(p + x, p' + x) \succeq R' \Leftrightarrow [p + x]_{\rho}(0_k) = [p' + x]_{\rho}(0_k) \Leftrightarrow \forall i \in F_q \setminus \{ 0_k, 1_k \}, \ [p]_{\rho}(i) = [p']_{\rho}(i)
\]

**Lemma 20.** Let \( p, q \in C_k(X, R) \) and \( r_X \) a sequence of random variables such that \( |X| = |r_X| \).

\[
p \succeq q \Leftrightarrow (p[r_X/X,]) \succeq (q[r_X/X,])
\]

**Proof.**

\[
p \succeq q
\]

\[
\Leftrightarrow \forall \rho \in A^X. \exists f_{\rho} \in \text{bij}_{\emptyset}. \forall \tau \in A^R.
\]

\[
[p]_{\rho}[\tau] = [q]_{f_{\rho}([\tau])} \ (\text{by Cor. 23})
\]

\[
\Leftrightarrow \exists f \in \text{bij}_{X \cup R}. \forall (\rho \cup \tau) \in A^{X \cup R}
\]

\[
\rho = \rho \cup \tau \ \ L(\rho \cup \tau) = [q]_{f([\rho \cup \tau])} \ (\text{by defining } f([\rho \cup \tau]) = \rho \cup f(\tau))
\]

\[
\Leftrightarrow \exists f \in \text{bij}_{X \cup R}. \forall (\rho \cup \tau) \in A^{X \cup R}
\]

\[
[X]_{\rho \cup \tau} = [X]_{f([\rho \cup \tau])} \land [\rho \cup \tau] = [q]_{f([\rho \cup \tau])}
\]

\[
\Leftrightarrow \exists f \in \text{bij}_{X \cup R} \forall \tau \in A^{R \cup X}
\]

\[
[p[r_X/X,] \tau] = [q[r_X/X,] \tau]
\]

**Appendix B. Proofs of Section 3**

**Proposition 22.** Let \( p, q \in C_k(X, R) \). For every \( \rho_1, \rho_2 \in A^X \), the following are equivalent:

1. \([p]_{\rho_1} = [q]_{\rho_2}\)
2. \(\forall \rho \in A^X. \left[ S[p], \rho_1, \rho_2 \right] = \left[ S[q], \rho_1, \rho_2 \right]\)
3. \(\exists f \in \text{bij}^R. \forall \tau \in A^R. \left[ p \right]_{\rho \cup f(\tau)} = \left[ q \right]_{\rho \cup f(\tau)}\)


Proof. The equivalence between 1 and 2 is directly obtained through the definition of a probabilistic distribution.

\[ (3 \Rightarrow 2) \text{ For any valuations } \rho^1, \rho^2 \text{ and any } c \in A^k, \text{ we have that } \]
\[ |\{ \tau \in A^R | [\rho^1]_{\rho^1 \cup \tau} = c \} = |\{ \tau \in A^R | [\rho^2]_{\rho^2 \cup \tau} = c \} | \]
\[ = f^{-1}(\{ \tau \in A^R | [\rho^2]_{\rho^2 \cup \tau} = c \}) \]
\[ = f^{-1}(\{ \tau \in A^R | [\rho^1]_{\rho^1 \cup \tau} = c \}) \]
\[ \text{We thus have } |S_{[\rho^1], \rho^1, \tau, \cdot}| = |S_{[\rho^2], \rho^2, \tau, \cdot}|. \]

\[ (2 \Rightarrow 3) \text{ For any valuations } \rho^1, \rho^2 \text{ and any } c \in A^k, \]
\[ |\{ \tau \in A^R | [\rho^1]_{\rho^1 \cup \tau} = c \} = |\{ \tau \in A^R | [\rho^2]_{\rho^2 \cup \tau} = c \} | \]
\[ \text{This means that for each } c \text{ we can define a bijection } \}
\[ f_c : \{ \tau \in A^R | [\rho^1]_{\rho^1 \cup \tau} = c \} \mapsto \{ \tau \in A^R | [\rho^2]_{\rho^2 \cup \tau} = c \} \]
\[ \text{which verifies } \forall \tau \in \{ \tau \in A^R | [\rho^1]_{\rho^1 \cup \tau} = c \}, [\rho^2]_{\rho^2 \cup \tau} = c = \]
\[ [\rho^1]_{\rho^1 \cup \tau}, (\cdot). \text{ Moreover, } g : \tau \in A^R \mapsto [\rho^2]_{\rho^2 \cup \tau} \text{ is a function whose image is contained inside } A^R, \text{ so } \]
\[ A^R = \bigcup_{c \in A^R} \{ \tau \in A^R | [\rho^2]_{\rho^2 \cup \tau} = c \} \]
\[ \text{i.e., the domains of all the } f_c \text{ form a partition of } A^R. \text{ Therefore the function } f, \text{ defined as the union of all the } f_c, \text{ is a }
\text{bijection that verifies } \forall \tau \in A^R | [\rho^1]_{\rho^1 \cup \tau} = [\rho^2]_{\rho^2 \cup \tau} = c. \]

Corollary 27. If \( p \in C_k(X, R) \) with \( R = \{r_1, \ldots, r_k\} \), then \( p \simeq r_1, \ldots, r_k \iff p \text{ is R-bijective} \).

Proof. We prove both directions separately.

\[ (\Rightarrow) \text{ Let } r = (r_1, \ldots, r_k) \text{ We must prove that } \forall \rho \in A^X, \tau \mapsto [\rho]_{\rho \cup \tau} / r \text{ is bijective. Using } \text{Corollary 23} \text{ we have } \]
\[ \forall \rho \in A^X. \exists f \in \text{bij}_R, \forall \tau \in A^R. [\rho]_{\rho \cup \tau} = [r]_{f(\tau)} \]
\[ \text{As } f \text{ is a bijection, } \tau \mapsto [\rho]_{\rho \cup \tau} / r \text{ is bijective, and as } [\rho]_{\rho \cup \tau} = [\rho]_{f(\tau)} \text{, } \]
\[ \tau \mapsto (\tau \mapsto [\rho]_{\rho \cup \tau}) \text{ is the bijection that allows us to apply } \]
\[ \text{Corollary 23} \]

Proposition 31. \( (F_q)^m \) is a primal algebra.

Proof. We see f as a function \( (F_q)^m \mapsto (F_q)^m \) denoted \( f(x_1, \ldots, x_k) = f_1(x_1, \ldots, x_k) \) \( \ldots f_n(x_1, \ldots, x_k) \). For \( 1 \leq i \leq k \) we may also decompose \( x_i \) as \( x_1^{m_i}, \ldots, x_1^{m_i} \) and see each \( f_i \) as a function \( (F_q)^{k_m} \mapsto F_q \). Then, by primality of \( F_q \) \( \text{Proposition 30} \), there exists \( t \in T(\Sigma_{\Sigma F_q}, x_i) \) such that \( f_i = t_i \times t_i \). We define \( t'_i \) as \( t_i \) where we replace:

- 0 with \( 0_{m_i} \)
- 1 with \( 0_{m_i} \times 1_{m_i} \)
- \( \alpha \) with \( 0_{m_i} \times 0_{m_i} \)
- \( x_i \) with \( 0_{m_i} \times 1_{m_i} \)

We observe that \( t'_i(F_q)^m = (0_{m_i} - t_i)^{0_{m_i} - t_i} \) and we may have \( f = t(F_q)^m \) with \( t = \sum t_i' \).

Proposition 33. \( E_{\Sigma_{\Sigma F_q}} \) is sound and faithfull with respect to \( F_q \).

Proof. We consider the classical representation of \( F_q \) as \( F_{(\alpha \to P)} \). The Euclidean division by \( P(\alpha) \) provides a normal form for any element in \( F_q \).

Soundness: By definition of a field, every equation in \( E_{\Sigma_{\Sigma F_q}} \) holds for all field elements. Soundness is thus immediate.

Faithfulness: Let \( r_1, r_2 \) be two ground terms such that \( t_{r_1}^2 = t_{r_2}^2 \). There exists a polynomial \( T \in \Sigma F_{(P = P)} \) such that \( t_{r_1}^2 = t_{r_2}^2 \). We can see \( r_1 \) and \( r_2 \) as polynomials over \( F_{P = P} \). Using the equations capturing associativity, commutativity and distributivity, a polynomial can be written in the developed form, and then using the euclidian division, with \( i \in \{1, 2\} \) under the form \( t_i = Q_i + P_i + R_i \) with \( \deg(R_i) < \deg(P) \). As \( P(\alpha) = 0 \), both polynomials reduce to \( t_i = Q_i + R_i \) which by with \( x \cdot 0 = 0 \) reduces to \( t_i = Q_i + R_i \), which finally reduces to \( t_i = R_i \). Now, with the correctness of \( E \), we have that \( t_i = R_i \), which by transitivity implies \( R_i = \tau \). We thus have \( t_i = R_i = T \), which concludes the proof.

Lemma 35. Let \( (A, \Sigma, E) \) be an effective algebra and \( p, q \in C_k(X, R) \).

\[ p \simeq q \iff \exists T \in T(\Sigma, X \cup R)^R \]
\[ T \text{ R-bijective } p = q^{[T / R]} \]

Moreover, if \( E \) is sound, but not faithful, then the implication from right to left (\( \Leftarrow \)) still holds.

Proof.

\[ p \simeq q \]
\[ \Rightarrow \text{( Cor 23)} \]
\[ \text{We set point by point } s \text{ so that } \forall \rho, \mu, f(p \cup \mu) = f(s) \]
\[ \text{bijection, } \forall \rho \cup \mu \in A^X \cup R, \]
\[ [\rho]_{\rho \cup \tau} = [q]_{f(\rho \cup \tau)} \]
\[ (A \text{ is primal}) \]
\[ \exists T(X, R) \in T(\Sigma, X \cup R), T \text{ R-bijective, } \forall \mu \in A^X \cup R, \]
\[ [\mu]_{\mu \cup \tau} = [q]^{[T / R]} \]
\[ \text{E is sound and faithful} \]
\[ \exists T(X, R), T \text{ R-bijective, } p = q^{[T / R]} \]

Appendix C.

Proofs of Section 4

Proposition 39. Let \( (A, \Sigma, E) \) be an effective algebra and \( p \in C_k(X, R) \) with \( R = \{r_1, \ldots, r_k\} \). We have that

\[ p \simeq r_1, \ldots, r_k \iff \forall r_i \in R, p, X \vdash E r_i \]

Moreover, if \( E \) is sound, but not faithful, the implication from right to left (\( \Leftarrow \)) still holds.
Proof. We have:

\[ p \simeq r_1, \ldots, r_k \]
\[ \Leftrightarrow^1 \exists t \in T(\Sigma, X \cup R), t \text{ R-bijective } \land p = E \ t \]
\[ \Leftrightarrow^2 \exists t \in T(\Sigma, X \cup R), t \text{ R-bijective } \land p_A = t \Delta \]
\[ \Leftrightarrow^3 \exists \tau : A^X \rightarrow A^R, \forall \rho, \tau, \]
\[ \tau \mapsto c([X]_\rho, [R]_\rho) \in \text{bij}^{R} \land c([X]_\rho, [p]_\rho, \tau) = [R]_\tau \]
\[ \Leftrightarrow^4 \forall \tau \in R, \exists \rho : A^X \rightarrow A^R, \forall \rho, \tau, \]
\[ \tau \mapsto c_r([X]_\rho, [R]_\rho), \ldots, c_r([X]_\rho, [R]_\rho)) \in \text{bij}^{R} \land c_r([X]_\rho, [p]_\rho, \tau) = [R]_\tau \]
\[ \Leftrightarrow^5 \forall \tau \in R, \exists c_r \in T(\Sigma, X \cup R), (c_r, \ldots, c_r) \text{ R-bijective } \]
\[ \land \forall \rho, \forall \tau, c_r([X], P(X, R)) = E \ t \]
\[ \Leftrightarrow^6 \forall \tau \in R, \exists c_r \in T(\Sigma, X \cup R), c_r([X], P(X, R)) = E \ t \]
\[ \Leftrightarrow^7 \forall \tau \in R, P(X, R), X \vdash E \ t \]

We detail below each equivalence.

\[ ^1 \text{By \textbf{Lemma 35}} \]
\[ ^2 \text{By the soundness and faithfulness of } E. \]
\[ ^3 \text{By } t \text{ is } R\text{-bijective, so for any } \rho \in A^X \text{ and } z \in A^R, \text{ we may take } c_r(z) = t^{-1}([X]_\rho, z) \text{ and we may then define } c([X]_\rho, z) = c_r(z) \text{ which is a bijection for any } \rho. \]
\[ ^4 \text{By splitting } c \text{ over the } k \text{ dimensions.} \]
\[ ^5 \text{By primality, soundness and faithfulness.} \]
\[ ^6 \text{We detail the difficult part:} \]
\[ \forall \tau \in R, \exists c_r \in T(\Sigma, X \cup R), c_r(X, P(X, R)) = E \ t \]
\[ \Rightarrow (c_r, \ldots, c_r) \text{ R-bijective} \]

For any \( \rho, \tau \mapsto [p]^{R'}_\rho \) has a left inverse which is \( \tau \mapsto [\rho^{R'}_\tau]_\rho \). \( \tau \mapsto [p]^{R'}_\rho \) is a function over a finite set, so if it has a left inverse, it is injective, which in a finite settings means that it is bijective (pigeon hole principle). Moreover, if a bijective function has both a right inverse and a left inverse, they are equals. Indeed, if \( f \) as for left inverse \( g \ (g \circ f = id) \) and for right inverse \( h \ (f \circ h = id) \), then \( g(x) = g(f \circ h(x)) = g \circ f(h(x)) = h(x) \). Thus, \( \forall \rho, \tau \mapsto [p]^{R'}_\rho \) is the inverse of \( \tau \mapsto [p]^{R'}_\rho \), and is in particular a bijection.

\[ \square \]

Lemma 41. Let \( (A, \Sigma, E) \) be an effective algebra and \( p, q \in C_k(X, R) \). Let \( R' \) be a set of variables such that \( |R'| = |R| \) and \( R' \cap R = \emptyset \). We have that

\[ p \simeq q \Leftrightarrow \exists \sigma \in \text{mgu}(\Sigma \cup \{X \mid X \in \Sigma \cup R', E, R\} \ (p, q) \text{ has a solution over } R' \]

Moreover, if \( E \) is only sound, but not faithful, the implication from right to left (\( \Leftarrow \)) still holds.

Proof. \( \Leftarrow \) We have a unifier of the form \( \sigma := \{r_i' \mapsto t_i(X, R, R')\} \) and a solution to the deduction constraints \( \tau := \{r_i' \mapsto u_i(X, R)\} \), we set \( T = R' \sigma \tau \). First, \( \sigma \tau \) is an instance of the mgu, thus we have \( p = E \ q[T]^{R'}/R \).

Secondly, we have that \( \forall \tau \in R, X, T \vdash r \), thus thanks to \textbf{Proposition 39} and \textbf{Corollary 27} we have that \( T \) is \textbf{R-bijective}. We conclude thanks to \textbf{Lemma 35}.

\( \Rightarrow \) Using \textbf{Lemma 35}, we have \( T, T \text{ R-bijective } \land p = E \ q[T]^{R'}/R \). We have \( T = \{t_1, \ldots, t_k\} \) with \( k = |R| \), and then \( \phi := \{r_i' \mapsto t_i\} \) is a valid unifier.

Thus the most general unifier exists \( (\text{\textbf{12}}) \) shows that unification in a primal algebra is unitary, so the existence of a unifier implies the existence of the mgu, let us call the mgu \( \sigma \), and we then have \( \tau \) such that \( \phi = \sigma \circ \tau \).

With \textbf{Proposition 39} we have \( \forall \tau \in R, T, X \vdash r \), and as \( T = R \tau \sigma \), this means that \( \tau \) is a solution to our deduction constraints.

\[ \square \]

\textbf{Proposition 45.} Let \( A \) be a finite primal algebra over \( \Sigma \) with a sound equational theory \( E \), and \( p, q \in C_k(X, R) \). We have that

\[ p \not\equiv E q \Rightarrow p \not\equiv q \]

Proof. We have that

\[ \exists R_1, R_2, \ R_1(p) = E \ R_2(q) \land R_1(p) \not\equiv E R_2(q) \]

Now let us assume by contradiction that \( p \) and \( q[R'/R] \) have a unifier \( \sigma \) such that \( p = E \ q[R'/R] \). Then, combined with \( R_1(q[R'/R]) \not\equiv E R_2(q[R'/R]) \), we obtain

\[ R_1(q[R'/R]) \not\equiv E R_2(q[R'/R]) \]

i.e. \( R_1(p) \not\equiv E R_2(p) \) which contradicts the hypothesis. Thus, there is no unifier, and by \textbf{Lemma 41} we have \( p \not\equiv q \). \( \square \)

\textbf{Appendix D.}

\textbf{Extending symbolic results}

We presented previously how several symbolic methods could be used to reason about probabilistic programs. Before providing a summary of the decidability of those methods in \textbf{Appendix E}, we provide a few extensions to handle our framework. First, we show how to encode one-turn deduction constraints into more classical deduction constraints. Then, we would like to use our methods in the framework of DH exponentiation, combined with bistrings. We then provide two extensions based on previous work, first how to obtain deducibility for combination of equational theories that might be typed (DH exponentiation is typed in the sense that variables may either be in the group or the field of exponents, and specific operations can only be applied to the correct object), and second a deducibility results for finite fields.

\textbf{D.1. From one-step deduction constraints to originated and monotone constraints.}

The one-step property we introduced has not been studied, we thus want to reduce it to more classical deduction constraints in order to leverage existing result. We note that a decision procedure was developed for deduction constraints without origination or monotonicity \textbf{35}, but they cover encryption theories, rather than the algebraic theory we are interested in.
**Definition 54.** Given an equational theory $E$, a symbol function $h$ is homomorphic if
\[
\forall n \geq 1. \forall f \in \Sigma^n. \forall x_1, \ldots, x_n \in V, h(f(x_1, \ldots, x_n)) = f(h(x_1), \ldots, h(x_n))
\]

**Definition 55.** Signatures may contain private symbol function. Then, deductibility under private function symbols $\mathcal{P}$ is defined as, given $E$ an equational theory over $\Sigma$, $t_1, \ldots, t_k \in T(\Sigma, V)$ and $(x_1, \ldots, x_k)$ disjoint variables, $t_1, \ldots, t_k \vdash_E s$ if and only if:
\[
\exists C(x_1, \ldots, x_k) \in T(\Sigma \setminus \mathcal{P}, (x_1, \ldots, x_k)). C(t_1, \ldots, t_n) = E \ s
\]

**Lemma 56.** We are given a theory $(E, \Sigma)$, and a one-step deduction constraint which can be written as any rule in the proofs.

**Lemma 59.** $(\Sigma, \chi, E)$ a $T$-typed equational theory and its untyped instance $(\Sigma_t, E_t, \phi)$, we have for any $\alpha$ WT substitution on $\chi$ and $t \in T(\Sigma, \chi)$ WT:
\[
t \alpha \phi = E_t u \phi
\]

**Definition 57.** We consider a set of types $T$. A $T$-typed signature $\Sigma$ consists of a finite set of functions symbols each with an arity and a type: $f/n : t_1 \times \cdots \times t_n \rightarrow s, \chi$ is the union for each type of infinite sets of the given type. For any $t \in T(\Sigma, \chi)$, we define in the classical sense that $t$ is well-typed, and we can then have $\tau : T(\Sigma, \chi) \rightarrow T$ which gives the type of a term. $(\Sigma, \chi, E)$ is a $T$-typed equational theory if the equations in $E$ are built over $T(\Sigma, \chi)$ and are well typed: $\forall (t_1, t_2) \in E, t_1 WT \land t_2 WT \land \tau(t_1) = \tau(t_2)$.

In the following, we full use the deducibility based on inference system (classically known to be equivalent to our previous definition), as shown in Figure 3.

**Definition 58.** Given $(\Sigma, \chi, E)$ a $T$-typed equational theory and its untyped instance $(\Sigma_t, E_t, \phi)$, we have for any $\alpha$ WT substitution on $\chi$ and $t \in T(\Sigma, \chi)$ WT:
\[
t \alpha \phi = E_t u \phi
\]

**Lemma 60.** $(\Sigma, \chi, E)$ a $T$-typed equational theory and its untyped instance $(\Sigma_t, E_t, \phi)$, we have for all $t, u \in T(\Sigma, \chi)$ WT:
\[
t = E_t u \Leftrightarrow t \phi = E_t u \phi
\]
We finally have the following results.

**Proposition 61.** \((\Sigma, X, E)\) a \(T\)-typed equational theory and its untyped instance \((\Sigma_t, E_t, \phi)\). Let Let \(\psi = M_1, \ldots, M_n\) be a WT frame and \(t\) a WT term.

\[\psi \vdash_E t \iff \psi \vdash_{E_t} t\phi\]

**Proof.** We prove by induction on the size of the proofs that \(\psi \vdash_E t \iff \psi \vdash_{E_t} t\phi\). If the proof is of size one, \(t\) belongs to the frame \((ax)\), and this is stable by substitution.

Let us assume that the result is true for any proofs smaller than some \(n > 1\).

\(\Rightarrow\) We consider the last rule of the proof of \(\psi \vdash_E t\).

\[\frac{\psi \vdash_{E_t} M_1 \quad \ldots \quad \psi \vdash_{E_t} M_l}{\psi \vdash_E f(M_1, \ldots, M_l)} f \in \Sigma\]

With \(t = f(M_1, \ldots, M_l)\). By induction hypothesis, we have proofs of \(\psi \vdash_{E_t} M_i\). and because \(t\phi = s(f(t_1(M_1\phi), \ldots, t_n(M_n\phi)))\), we conclude by multiple \(fa\) application.

\[eq \quad \frac{\psi \vdash_{E_t} M_1}{\psi \vdash_E M =_{E_t} M'}\]

\(\Leftarrow\) We consider the last rule of the proof of \(\psi \vdash_{E_t} t\phi\).

\[\frac{\psi \vdash_{E_t} M}{\psi \vdash_{E_t} t\phi} \quad \frac{\phi \vdash_{E_t} M_1}{M =_{E_t} t\phi} \quad \frac{M_1}{\phi \vdash_{E_t} s(M_1)} \quad \frac{s(T)}{t\psi = s(M_1) \quad s \in T \quad M_1 \quad \psi \vdash_{E_t} M} \quad \frac{t\psi = s(M_1)}{t\psi = s(M_1)} \quad \frac{\phi \vdash_{E_t} M_1}{\phi \vdash_{E_t} s(M_1)} \quad \frac{t\psi = s(T)}{t\psi = s(T)}

We can then conclude by induction.

This Theorem allows us to transform a typed theory into a classical theory, and then use the result of [24] to combine it with other theories.

**D.3. Deduction for DH exponentiation**

As the size of the field is often abstracted in the security proofs, we may consider that we have a finite field of infinite size that we may abstract as a commutative ring. This particular case for deduction was solved in [23], where the deduction problem for DH exponentiation is reduced to deduction in a commutative ring, which is solved using Gröbner Basis. This approach is sound, but not complete. We provide a slight extension to make it complete when \(q\), the size of the field, is explicitly known.

**Theorem 62.** Deduction in a \(\mathbb{Z}_q\) for a given \(q\) is decidable.

**Proof.** With \(x_1, \ldots, x_n\) the set of variables in the deduction problem, we use the reduction from [23] to compute the Gröbner basis \(GB\) of the module corresponding to the attacker knowledge in the abstract commutative ring 

\[Z[x_1, \ldots, x_n].\]

Then, we may compute the module inside \(Z_q[x_1, \ldots, x_n]\), by computing the Gröbner basis of \(GB \cup (x_i^q - x_i)_{1 \leq i \leq n}\), and then removing the \((x_i^q - x_i)_{1 \leq i \leq n}\) \(\square\)

**D.3.2. Deduction in a \((\mathbb{F}_q)^m\).** We use the result of [24] to combine the result for \(\mathbb{F}_q\) with tuples and projection of length \(m\).

**Appendix E. Complexity results**

**E.1. Classical complexity classes**

Program equivalence for a given \(\Sigma\)-algebra \(A\) yields a natural decision problem \(EQUIV^k_A\) defined as follows.

**INPUT:** \(p, q \in C_k(X, R)\) defined over \(A\).

**QUESTION:** \(p \sim q\).

We first show a coNP lower and PSPACE upper bound.

**Proposition 63.** \(EQUIV^k_{F_p}\) is in PSPACE and coNP-hard.

**Proof.** We first show that \(EQUIV^k_{F_p}\) is in PSPACE. For this consider the following decision procedure where \(conv\) converts a tuple of \(k\) elements in \(F_p\), into a corresponding number in \([1, q^k]\).
For $p$ in $A^X$:

initialize arrays $t_p, t_q$ of size $q^k$ to 0 for $x$ in $A^R$:

$t_p \{ \text{conv} \{ p_{|\mu \tau} \} \} \oplus +$
$t_q \{ \text{conv} \{ q_{|\mu \tau} \} \} \oplus +$

if $t_p \neq t_q$ return False

The time complexity of this algorithm is in $\text{EXP}(|X| + |R|)$. However the space complexity is polynomial as the maximum value for each element of the arrays $t_p, t_q$ is $q^{k|R|}$, which can be represented using $\log_2(q) \cdot |R|$ bits.

We next show $\text{coNP}$-hardness by reduction from $\text{UNSAT}$. Given a formula CNF $\phi$ over variables in $R$ and $(\lor, \land)$ we transform $\phi$ into a formula $\phi'$ over $R$ and $\lor, \land$ in polynomial time w.r.t the size of the formula. Indeed, given a clause of $\phi$ of the form $x \lor y \lor z$, we have that $x \lor y \lor z = (x \lor y) \lor z = (x \lor y) \lor z$. With this transformation, we have $|\phi'| \leq 5 \times |\phi|$. Let $p = \phi'$. We have that $p \in C_1(\emptyset, R)$ defined over $F_2$ and $2p \equiv 0 \iff p \in \text{UNSAT}$

Indeed, if $\phi$ is unsatisfiable, then for any values taken by the variables in $X$, $\phi$ evaluates to 0 and $p$ equals 0. This means that $p$ is equal to the zero distribution. Conversely, if $\phi$ is satisfiable, then there exists a valuation of the variables so that $p$ is equal to 1, and thus $p$ cannot be the zero distribution.

We next show that program equivalence for programs returning a single value is most unlikely to be in $\text{coDP}$. Recall that $\text{coDP} = \{ L_1 \cup L_2 | L_1 \in \text{NP}, L_2 \in \text{coNP} \}$.

**Proposition 64.** If $\text{EQUIV}_2^1 \in \text{coDP}$ then $PH$ collapses.

**Proof.** Recall that unique satisfiability $\text{USAT}$ is the set of satisfiability problems that admit exactly one satisfying assignment. By [36], we have that $\text{USAT} \in \text{coDP}$ implies that $PH$ collapses. We thus reduce $\text{USAT}$ to $\text{program equivalence in polynomial time}$. We are given a CNF formula $\phi$ over variables $X$ (all used) and $(\lor, \land)$. We can transform in polynomial time w.r.t the size of the formula $\phi$ into a formula $\phi'$ over $(\lor, \land)$. Then, let $p = \phi'$ which is a program in $C_1(\emptyset, X)$ over $F_2$. We have that

$$p \equiv \bigwedge_{x \in X} x \iff p \in \text{USAT}$$

The reduction is polynomial as $|\bigwedge_{x \in X} x| \leq |p|$. 

This result implies that it is unlikely that our problem is either in $\text{coNP}$ or $\text{NP}$, i.e it is unlikely that an efficient decision procedure exists.

**E.1.1. Exact counting and quantum classes.** We now draw links to quantum complexity classes, where we are able to precisely characterize the complexity of $R \rightarrow \text{EQUIV}_2^1$ is the version without input variables defined as follows.

**INPUT:** $p, q \in C_1(\emptyset, R)$ defined over $A$.

**QUESTION:** $p \sim q$.

The problem halfSAT is defined as follows.

**INPUT:** CNF boolean formula $\phi$.

**QUESTION:** Exactly half of the valuations satisfy $\phi$.

**Definition 65.** $C=P$ is the class of decision problems solvable by a NP Turing Machine such that the number of accepting paths exactly equals the number of rejecting paths, if and only if the answer is ‘yes.’

Cook’s theorem allow us to derive:

**Theorem 66.** halfSAT is complete for $C=P$

We can know link uniformity and halfSAT.

**Proposition 67.** $R \rightarrow \text{EQUIV}_2^1$ is $C=P$-complete.

**Proof.** Hardness:

As in the reduction showing $\text{coNP}$-hardness, given $\phi$ we have that there exists $p \in C_1(\emptyset, R)$ such that

$$p \sim r \iff \phi \in \text{halfSAT}$$

where $r$ is a fresh random variable.

**Membership:**

We are given $p \in C_1(\emptyset, R)$. Consider the Turing Machine $M$ which sample a valuation for $R$, and accepts if and only if $p$ evaluates to 1 for this valuation. The number of accepting path of $M$ is equal to the number of values in $R$ such that $p$ is equal to 1.

$$p \sim R \iff p \text{ equals 1 for half of the valuations} \iff M \text{ accepts half of the time}$$

From this, we can obtain links to quantum computation, as $\text{coC}=P$ is equal to $\text{NQP}$ [37], a quantum analog of NP, and we also have that $\text{coC}=P$ is hard for PH under randomized reductions [38]. We also have $\text{coC}=P \subset \text{PP}$, and if $\text{coC}=P = \text{PP}$ then $\text{PH} \subset \text{PP}$ [39], which is unlikely.

**Appendix F. Summary**

We provide in Figure 6 a survey of the decidability results that currently exists, along with some extensions of our own.

**Appendix G. Perspectives**

In this section, we consider two possible perspectives of interest. The first one is to try to obtain a decision result for program equivalence in the field setting using mathematical notions. The second one is to improve yet again the automation of EasyCrypt, by constructing simulators automatically.
Theorem 71. For every orthogonal system $f_1, \ldots, f_m \in \mathbb{F}_q[x_1, \ldots, x_n], 1 \leq m \leq n$, over $\mathbb{F}_q$ and every $r, 1 \leq r \leq n - m$, there exist $f_{m+1}, \ldots, f_{m+r} \in \mathbb{F}_q[x_1, \ldots, x_n]$, so that $f_1, \ldots, f_{m+r}$ forms an orthogonal system in $m$ variables over $\mathbb{F}_q$.

Proof. Using Proposition 21 we reduce program equivalence over a finite field to program equivalence without input variables, which we reduce to uniformity without input variables using Proposition 19, which we reduce to bijection testing of polynomials over random variables using Corollary 27, which directly reduces to deciding if a set of polynomials is an orthogonal system.

However, deciding efficiently if a set of polynomials is an orthogonal system, is to the best of our knowledge, still an open problem. This is a question of great interest for our problems.

G.2. Automatic construction of simulator

In the computational model, a classical proof technique is the proof by reduction, where given an assumption and a game, we show how part of the game may be viewed as an attacker against the assumption. To perform a reduction in EASYCRYPT, one has to match variables in the game to variables in the assumption, and then check that everything that appears in the game can be computed from the input of the attacker in the assumption. Moreover, the reduction might not be possible directly on the game, but may require previous steps, such as some application previously. In [9], reductions and rules were automated through heuristics without theoretical foundations, thus providing no information in case of failure, and making them difficult to construct upon and improve.

If we abstract from the notion of game and cryptographic proofs, the existence of a simulator captures the fact that from a given distribution (the input of the attacker) one can produce a specific distribution (the output of the game). It would be interesting to try to capture this in our framework.

This approach seems promising, because by considering abstract distributions, we can consider in a single step the matching of the variables, the application of some rules.
and the construction of the simulator. Thus, we believe that
developing further our framework to provide decision or
well understood heuristics for this problem could be of great
interest for the automation of cryptographic proofs.

G.2.1. Multi-party computation. We consider the multi-
party computations performed in [44]. In their setting, we
have a secret value \( s \) which is shared between three parties
in three shares \( s_1, s_2, s_3 \) that are uniformly distributed and
such that \( s_1 + s_2 + s_3 \).

Then, several algorithms are provided to perform com-
putations on shared secrets. The security of an algorithm can
be verified if the view of a party is independent of the secrets
shares of the two other party. We can then use independence
to verify the security of a protocol.

Let us consider as an example the multiplication al-
gorithm given in [44], Algorithm 2. We have two shared
value \((u_1, u_2, u_3)\) and \((v_1, v_2, v_3)\), which can be represented
as input variables. The view \( V_1 \) of the first agent can be
expressed as the following expression:

\[
V_1 := (u_1, v_1, r_{u12}, r_{u31}, r_{u12}, r_{v12}, r_{v31}, u_3 + r_{u31} - r_{u23}, v_3 + r_{u31} - r_{v23})
\]

where \( r_{ij}^k \) are uniformly distributed random variables. We
can only consider one view, as the three views are symmet-
ric. The security of the algorithm is then

\[
V_1 \perp \{u_2, v_2, u_3, v_3\}
\]

We may solve it using our framework, for instance by
using the technique described in [Section 5.3.1] and proving:

\[
\begin{align*}
&u_1, v_1, r_{u12}, r_{u31}, r_{u12}, r_{v12}, r_{v31}, u_3 + r_{u31} - r_{u23}, v_3 + r_{u31} - r_{v23} \\
\cong &u_1, v_1, r_{u12}, r_{u31}, r_{u12}, r_{v12}, r_{v31}, u_3 + r_{u31} - r_{u23}, v_3 + r_{u31} - r_{v23}
\end{align*}
\]

This is achieved through the unifier

\[
\begin{align*}
r_{u23} &\mapsto r_{u3} + u_3 - u_3 \\
r_{v23} &\mapsto r_{v3} + v_3 - v_3
\end{align*}
\]

which yields a trivial deduction constraint.