2019 IEEE 32nd Computer Security Foundations Symposium (CSF)

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 $\mathcal{A}$ against the cryptographic construction under consideration into an adversary $\mathcal{B}$ against the computational assumption, and a proof that $p_{\mathcal{A}} \leq f(p_{\mathcal{B}})$, where $p_{\mathcal{B}}$ denotes the probability of $\mathcal{B}$ winning the experiment (against the construction), $p_{\mathcal{A}}$ denotes the probability of $\mathcal{A}$ 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 $p_{\mathcal{A}_i} \leq f_i(p_{\mathcal{A}_{i+1}})$, where $\mathcal{A}_i$ is an adversary and $p_{\mathcal{A}_i}$ 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

DOI 10.1109/CSF.2019.00017

primarily based on complex heuristics, making the tools difficult to use, maintain or extend. Similar specializations have been proposed to validate masking, a popular countermeasure used for protecting implementations against differential power analysis [11].

Problem statement. 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 wining events are the same).
2) independence: such steps require proving that some values in a given experiment are independent of some secrets. This is modelled 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, where non interpreted function symbols may capture attacker actions. Thanks to our framework, we can however derive the soundness and/or completeness of many different heuristics for those extended settings. For instance, to prove program equivalence over $\mathbb{F}_{2^n}$ for all n, it follows from our results that it is sound to prove their equivalence over a commutative ring of characteristic 2.

We demonstrate the usefulness of our approach in practice through the implementation of a library that we interfaced with two existing tools: EASYCRYPT [7], 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.

$$
\begin{array}{rcll}
t & ::= & x & \text{variables} \\
 & | & f(t_1, \ldots, t_n) & \text{operation of arity } n \\
 & & & \\
p & ::= & x := t_1 & \text{deterministic assignment} \\
 & | & r_1, \ldots, r_n \xleftarrow{\$} \mathcal{A} & \text{probabilistic assignments} \\
 & | & p_1; p_2 & \text{sequential composition} \\
 & | & \mathsf{return}\ \ (t_1, \ldots, t_n) & \text{return of arity } n
\end{array}
$$

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

$$
\begin{array}{l}
x, x_1, x_2, y_1, y_2, z_1, z_2 \xleftarrow{\$} \mathbb{F}_q; \\
sk := (g^x, x_1, x_2, y_1, y_2, z_1, z_2); \\
pk := (g^x, g^{x_1 + x x_2}, g^{y_1 + x y_2}, g^{z_1 + x z_2}); \\
\mathsf{return}\ \ sk
\end{array}
$$

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

2.1. Background

A signature Σ 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). A Σ-algebra $\underline{\mathcal{A}}$ for the signature $\Sigma = (\mathcal{F}_n)_{n\in\mathbb{N}}$ is given by a set $\mathcal{A}$ and the interpretation of Σ, which consists of a total function $f^{\underline{\mathcal{A}}} : \mathcal{A}^n \mapsto \mathcal{A}$ for each $f \in \mathcal{F}_n$.

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

$$
\begin{array}{rcl}
[x]_\rho & = & \rho x \\
[f(t_1, \ldots, t_n)]_\rho & = & f^{\underline{\mathcal{A}}}([(t_1, \ldots, t_n)]_\rho) \\
[(t_1, \ldots, t_n)]_\rho & = & ([t_1]_\rho, \ldots, [t_n]_\rho)
\end{array}
$$

2.2. Syntax and semantics of probabilistic programs

We consider a core probabilistic language over a signature Σ. Expressions of the language are built from variables using function symbols from Σ. 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*

$$r_1, \ldots, r_m \xleftarrow{\$} \mathcal{A}; \mathsf{return}\ \langle t_1, \ldots, t_k \rangle$$

with $R = \{r_1, \ldots, r_m\}$.

From now on, we identify commands as tuples of terms, and let $\mathcal{C}_k(X, R)$ denote the set of tuples $(t_1, \ldots, t_k)$ 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 $p \in \mathcal{C}_k(\emptyset, \{x, y, z, x_1, x_2, y_1, y_2, z_1, z_2\})$ 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 $+^{\underline{\mathcal{A}}} = \oplus$ (the xor operator) and $\times^{\underline{\mathcal{A}}} = \wedge$ (the and operator) $\underline{\mathcal{A}}$ defines Boolean expressions.

We often write uv for $u \times v$. We define the programs $p_1, p_2, p_3 \in \mathcal{C}_1(\{x\}, \{u, v, w\})$:

- $p_1 = x + v$,
- $p_2 = xv$,
- $p_3 = uv + vw + uw$.

Let $p = (t_1, \ldots, t_k)$ be a command in $\mathcal{C}_k(X, R)$. For every $\rho \in \mathcal{A}^X$, we let $[\![p]\!]_\rho \in \mathsf{Distr}(\mathcal{A}^k)$ be the distribution $[(t_1, \ldots, t_k)]_{\rho \cup \tau}$, where τ is sampled uniformly over $\mathcal{A}^R$.

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

Lemma 4. *Let $p \in \mathcal{C}_k(X, R)$, $\rho \in \mathcal{A}^X$ and $\vec{v} \in \mathcal{A}^k$. Then*

$$[\![p]\!]_\rho(\vec{v}) = \frac{1}{|\mathcal{A}^k|} \cdot \left|\mathcal{S}_{[\![p]\!], \rho, \vec{v}}\right|$$

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

$$\{\rho' \in \mathcal{A}^{X \cup R} \mid {\rho'}_{|X} = \rho \wedge [p]_{\rho'} = \vec{v}\}$$

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.

$[\![p_3]\!]$ only depends on the random variables. We can easily compute its distribution by writing the truth table of the program:

u	0	0	0	0	1	1	1	1
v	0	0	1	1	0	0	1	1
w	0	1	0	1	0	1	0	1
p_3	0	0	0	1	0	1	1	1

We see that $[\![p_3]\!]$ is actually the uniform distribution, i.e., we have that $[\![p_3]\!](0) = \frac{1}{2}$ and $[\![p_3]\!](1) = \frac{1}{2}$.

2.3. Probabilistic relations

In this section we introduce several probabilistic relations that allow us to express interesting properties over probabilistic programs. 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 6 (I-Equivalence)**.** Let $I \subseteq X$ and $p, q \in \mathcal{C}_k(X, R)$. p and q are I-equivalent, denoted $p \simeq_I q$, iff

$$\forall \rho^1, \rho^2 \in \mathcal{A}^X.\ {\rho^1}_{|I} = {\rho^2}_{|I} \Rightarrow [\![p]\!]_{\rho^1} = [\![q]\!]_{\rho^2}$$

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 $u \simeq u$, but $(u, v) \not\simeq (v, v)$.

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

Definition 9 (Uniformity)**.** $p \in \mathcal{C}_k(X, R)$ is uniform if and only if $p \simeq r_1, \ldots, r_k$ where $\{r_1, \ldots, r_k\} \subseteq R$.

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)**.** $p \in \mathcal{C}_k(X_s \uplus X_{pub}, R)$ is independent of X_s, denoted $p \perp X_s$, iff $p \simeq_{X_{pub}} p$.

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 $[\![x + v]\!]_{\{x \mapsto 1\}} = [\![x + v]\!]_{\{x \mapsto 0\}}$.

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

Definition 12 (Probabilistic non-interference)**.** Let $p \in \mathcal{C}_k(X, R)$, $I \subset X$ and $O \subset \{1, \ldots, k\}$. Probabilistic non-interference of p with respect to I, O is defined as

$$\pi_O(p) \simeq_I \pi_O(p)$$

where π_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 the long version [16].

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 $p \in \mathcal{C}_k(X_s \uplus X_{pub}, R)$. 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 $p \in \mathcal{C}_k(X, R)$ 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.

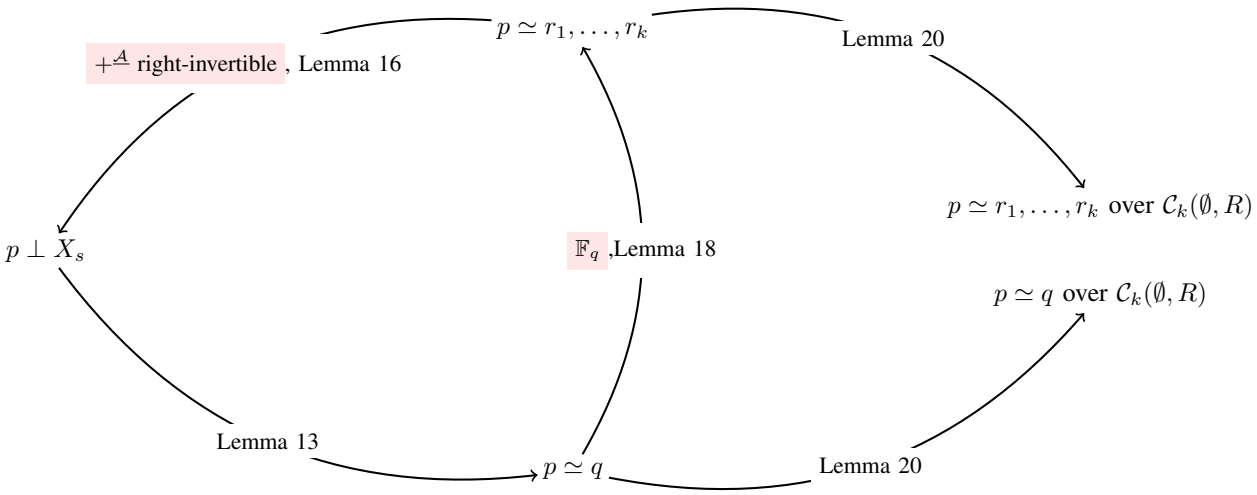
Lemma 16. *Let $\underline{\mathcal{A}}$ be a Σ-algebra that contains a binary symbol $+$ such that $+^{\mathcal{A}}$ is right invertible. Let $p = (p_1, \ldots, p_k) \in \mathcal{C}_k(X, R)$ with $\{r_1, \ldots, r_k\} \subseteq R$ and $x_1, \ldots, x_k$ fresh input variables. We have that*

$$p \simeq r_1, \ldots, r_k \Leftrightarrow (p_1 + x_1, \ldots, p_k + x_k) \perp \{x_1, \ldots, x_k\}$$

The reduction from uniformity to independence directly follows.

Proposition 17. *If $\underline{\mathcal{A}}$ contains a binary symbol $+$ such that $+^{\mathcal{A}}$ 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 Σ-algebra where



Lemma 13: $p \perp X_s \Leftrightarrow p \simeq p[^{X'_s}/_{X_s}]$
Lemma 16: if $+^{\mathcal{A}}$ is right-invertible then $p \simeq r \Leftrightarrow p + x_r \perp x_r$
Lemma 18: over $\mathbb{F}_q$, $p \simeq p' \Leftrightarrow$ (if $r = 0_k$ then $(p+x)^{q-1}$else if $r = 1_k$ then $(1_k - (p'+x)^{q-1})$else $r) \simeq r$
Lemma 20: $p \simeq q \Leftrightarrow (p[^{r_X}/_X], r_X) \simeq (q[^{r_X}/_X], r_X)$

Figure 3. Summary of reductions.

$\Sigma = \{+, \times, 0, 1\}$. $+$, $\times$, 0 and 1 are interpreted using the field composition laws and corresponding unit elements as expected. We also write

$$xy \text{ for } x \times y,\ x - y \text{ for } x + \underbrace{y + \cdots + y}_{p-1 \text{ times}},\ x^n \text{ for } \underbrace{x \times \cdots \times x}_{n \text{ times}}.$$

The encoding that we present below rely on the fact that for any element $x \in \mathbb{F}_q$,

$$x^{q-1} = \begin{cases} 0 \text{ if } x = 0 \\ 1 \text{ else} \end{cases}$$

which also allows us to introduce a syntactic sugar for the if then else construct that we will use when inside a finite field $\mathbb{F}_q$:

$$(\text{if } r = i \text{ then } p \text{ else } p') := \quad (1 - (r-i)^{q-1})p + (r-i)^{q-1}p'$$

Before giving a general encoding of program equivalence in terms of uniformity, we show the simpler case of Boolean algebras, i.e., the case where $q = 2$, and programs returning a single value. Let

$$t = (\text{if } r = 1 \text{ then } p \text{ else } 1 - p')$$

We show that

$$p \simeq p' \text{ iff } t \simeq r$$

where r is a fresh random variable. We fix a valuation $\rho \in \mathcal{A}^X$. By disjunction on the possible values of r, we have

$$[\![t]\!]_\rho(0) = \frac{1}{2}[\![p]\!]_\rho(0) + \frac{1}{2}[\![p']\!]_\rho(1) = \frac{1}{2} + \frac{[\![p]\!]_\rho(0) - [\![p']\!]_\rho(0)}{2}.$$

It follows that: $t \simeq r \Leftrightarrow \forall \rho. [\![t]\!]_\rho(0) = \frac{1}{2} \Leftrightarrow \forall \rho. [\![p]\!]_\rho(0) = [\![p']\!]_\rho(0) \Leftrightarrow p \simeq p'$

We can generalize the result to programs of arbitrary length over any finite field $\mathbb{F}_q$ where q is a prime power.

To generalize the encoding to arbitrary length programs, for any k, we denote $0_k = (0, \ldots, 0)$ and $1_k = (1, \ldots, 1)$ and extend $+$ and $\times$ to tuples homomorphically, i.e for programs $p = (p_1, \ldots, p_k)$,$p' = (p'_1, ..., p'_k)$, we will have that $p + p' = (p_1 + p'_1, \ldots, p_k + p'_k)$, and for any n, $p^n = (p_1^n, \ldots, p_k^n)$.

Lemma 18. *Let $p, p' \in \mathcal{C}_k(X, R)$ be programs over $\mathbb{F}_q$. 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*

$$p \simeq p' \Leftrightarrow T(p + x, p' + x) \simeq r$$

where

$$T(p, p') := \quad \textit{if } r = 0_k \textit{ then } p^{q-1} \textit{ else if } r = 1_k \textit{ then } 1_k - p'^{q-1} \textit{ else } r$$

Intuitively, $T(p, p')$ is a generalization of the previous encoding for Booleans, and is uniform if and only if $[\![p]\!]_\rho(0) = [\![p']\!]_\rho(0)$. To prove $p \simeq p'$, we need to have that for any value x, $[\![p]\!]_\rho(x) = [\![p']\!]_\rho(x)$, this is equivalent to prove that for a fresh variable x, $[\![p+x]\!]_\rho(0) = [\![p'+x]\!]_\rho(0)$.

From this encoding we immediately have the following proposition.

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 $p, q \in \mathcal{C}_k(X, R)$ and r_X a sequence of random variables such that $|X| = |r_X|$.*

$$p \simeq q \quad \Leftrightarrow \quad (p[^{r_X}/_X], r_X) \simeq (q[^{r_X}/_X], r_X)$$

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 the long version [16].

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 valuations 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 $p, q \in \mathcal{C}_k(X, R)$. For every $\rho_1, \rho_2 \in \mathcal{A}^X$, the following are equivalent:*

1) $[\![p]\!]_{\rho_1} = [\![q]\!]_{\rho_2}$,
2) $\forall \vec{v} \in \mathcal{A}^k.\ \left|\mathcal{S}_{[\![p]\!],\rho_1,\vec{v}}\right| = \left|\mathcal{S}_{[\![q]\!],\rho_2,\vec{v}}\right|$,
3) $\exists f \in \mathsf{bij}^R.\ \forall \tau \in \mathcal{A}^R.\ [p]_{\rho^1 \cup \tau} = [q]_{\rho^2 \cup f(\tau)}$.

The following corollary will be used extensively throughout the paper.

Corollary 23. *Let $p, q \in \mathcal{C}_k(X, R)$ and $I \subseteq X$. We have that*

$$p \simeq_I q \quad \textit{iff} \quad \begin{array}{c} \forall \rho^1, \rho^2 \in \mathcal{A}^X.\ \exists f \in \mathsf{bij}^R.\ \forall \tau \in \mathcal{A}^R. \\ {\rho^1}_{|I} = {\rho^2}_{|I} \Rightarrow [p]_{\rho^1 \cup \tau} = [q]_{\rho^2 \cup f(\tau)} \end{array}$$

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 $\forall \tau \in \mathcal{A}^{\{u,v,w\}}, [uv + vw + wu]_\tau = [u]_{f(\tau)}$

u	0	0	0	0	1	1	1	1
v	0	0	1	1	0	0	1	1
w	0	1	0	1	0	1	0	1
p_3	0	0	0	1	0	1	1	1
f_u	0	0	0	1	0	1	1	1

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 $p \in \mathcal{C}_k(X, R)$, it is possible if and only if $k = |R|$.

Definition 25. $p \in \mathcal{C}_k(X, R)$ is R-bijective if and only if

$$\forall \rho \in \mathcal{A}^X.\ \tau \mapsto (R \mapsto [p]_{\rho \cup \tau}) \in \mathsf{bij}^R$$

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

Corollary 27. *If $p \in \mathcal{C}_k(X, R)$ with $R = \{r_1, \ldots, r_k\}$, then*

$$p \simeq r_1, \ldots, r_k \quad \Leftrightarrow \quad p \textit{ is } R\textit{-bijective}$$

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 $x + u$ is $\{u\}$-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 Σ-algebras: in some cases the term algebra equipped with an equational theory E gives a fully abstract representation of the Σ-algebra.

3.2.1. Primal Algebra. A central notion of this section is primality introduced in [12]:

Definition 29. $\underline{\mathcal{A}}$ is said to be primal if and only if

$$\forall n.\ \forall f : A^n \mapsto A.\ \exists t \in \mathcal{T}(\Sigma, (x_1, ..., x_n)).\ f = t^{\underline{\mathcal{A}}}$$

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

Term algebra for $\mathbb{F}_q$. We denote by $\mathbb{F}_q$ a finite field, where $q = p^n$ and p is prime, and by P an irreducible polynomial over $\mathbb{F}_p[\alpha]$ of degree n. The $\underline{\mathbb{F}_q}$-algebra is defined by the signature

$$\Sigma_{\mathbb{F}_q} = \{0, 1, \alpha, +, *\}$$

and their usual mathematical interpretation with $\mathbb{F}_q$ seen as ${}^{\mathbb{F}_p[\alpha]}/_{(P(\alpha))}$. Nipkow [12] has shown that the $\mathbb{F}_q$-algebra is a primal algebra:

Proposition 30 ([12])**.** $\underline{\mathbb{F}_q}$ *is a primal algebra.*

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

$$\mathsf{if}\ x = 0\ \mathsf{then}\ f(0)\ \mathsf{else}\ \ldots \mathsf{if}\ x = i\ \mathsf{then}\ f(i)\ \mathsf{else}\ \ldots$$

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 $(\mathbb{F}_q)^m$. It is interesting to note that $(\mathbb{F}_q)^m$ can be made primal. We write tuples directly as sequences of elements, for example we denote $(0, 0, 0)$ with 000, or 0_3. The $(\mathbb{F}_q)^m$-algebra is then defined by the signature

$$\Sigma_{(\mathbb{F}_q)^m} = \{0_m, (0_k 1 0_{m-1-k})_{0\le k\le m-1}, (0_k \alpha 0_{m-1-k})_{0\le k\le m-1}, +, *\}$$

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. $\underline{(\mathbb{F}_q)^m}$ *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 $\{t_i = u_i\}_i$ where $t_i, u_i \in \mathcal{T}(\Sigma, V)$ 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 ($\Leftarrow$) and faithful ($\Rightarrow$) with respect to $\underline{\mathcal{A}}$ if and only if

$$\forall t_1, t_2 \in \mathcal{T}(\Sigma).\ t_1^{\mathcal{A}} = t_2^{\mathcal{A}} \Leftrightarrow t_1 =_E t_2$$

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

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

$$\begin{array}{cc} x + 0 = x & x * 0 = 0 \\ x * 1 = x & x + \cdots + x = 0 \textit{ (p times)} \\ x + y = y + x & x * \cdots * x = 1 \textit{ (q-1 times)} \\ x * y = y * x & x * (y + z) = x * y + x * z \\ x + (y + z) = (x + y) + z & x * (y * z) = (x * y) * z \\ P(\alpha) = 0 & \end{array}$$

Proposition 33. $E_{\mathbb{F}_q}$ *is sound and faithful with respect to* $\underline{\mathbb{F}_q}$*.*

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

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

We consider the examples where $\mathcal{A}$ is a finite field, denoted $(\mathbb{F}_q)^n$, where q is an explicit value, and n is a parameter. We may for instance study programs manipulating bitstrings of length n using $\mathbb{F}_2^n$.

Some interesting effective algebra are, for an explicit q and a parameter n, $(\underline{\mathbb{Z}_p}, \Sigma_{\mathbb{Z}_p}, E_{\mathbb{Z}_p})$ and $(\underline{\mathbb{F}_q}, \Sigma_{\mathbb{F}_q}, E_{\mathbb{F}_q})$.

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 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 $(\underline{\mathcal{A}}, \Sigma, E)$ be an effective algebra and $p, q \in \mathcal{C}_k(X, R)$*

$$p \simeq q \quad \textit{iff} \quad \begin{array}{c} \exists T \in \mathcal{T}(\Sigma, X \cup R)^{|R|}. \\ T \textit{ R-bijective} \wedge p =_E q[{}^T/_R] \end{array}$$

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

Example 36. Consider again $uv + vw + uw \simeq u$ from Example 5. A valid witness of this equivalence is

$$t(u, v, w) = (uv + wu + vw, u + v, u + w)$$

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 $t_2 t_3 = uv + uw + wv + u = t_1 + u$. Thus, we have $t_2 t_3 + t_1 = u$. Then, $t_2 + t_2 t_3 + t_1 = v$ and $t_3 + t_2 t_3 + t_1 = w$. Finally, if we set $g(x_1, x_2, x_3) := (x_2 x_3 + x_1, x_2 + x_2 x_3 + x_1, x_3 + x_2 x_3 + x_1)$, we find that $g(t(u, v, w)) = (u, v, w))$, i.e t is a bijection[1].

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 the long version [16].

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 Σ be a signature equipped with E, V a set of variables, and $t_1, \ldots, t_k, s \in \mathcal{T}(\Sigma, V)$. We say that s is deducible from $t_1, \ldots, t_k$, denoted $t_1, \ldots, t_k \vdash_E s$ if and only if:

$$\exists R \in \mathcal{T}(\Sigma, (x_1, \ldots, x_k)).\ R\sigma =_E s$$

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 $\Sigma_\oplus = \{0, +\}$ and the equational theory $E_\oplus$ defined as the subset of $E_{\mathbb{F}_q}$ with $q = 2$ defined over $\Sigma_\oplus$. We have that

$$u + v + w, v + w \vdash_{E_\oplus} u$$

witnessed by the term $R = x_1 + x_2$. However,

$$u + v + w, v + w \not\vdash_{E_\oplus} u + w.$$

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 given its outputs allows to recompute its inputs, it can be checked if a function is bijective using deduction.

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

$$p \simeq r_1, \ldots, r_k \quad \Leftrightarrow \quad \forall r_i \in R.\ p, X \vdash_E r_i$$

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

4.2. Deduction constraints and unification for program equivalence

In this section we show how deduction constraint as used in symbolic cryptography [17] 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 Σ be a signature equipped with E, and V a set of variables. A deduction constraint is an expression $T \vdash^?_E u$ where $T \subseteq \mathcal{T}(\Sigma, V)$ is a set of terms and $u \in \mathcal{T}(\Sigma, V)$ a term.

A deduction constraint system is either $\bot$ or a conjunction of deduction constraints of the form:

$$T_1 \vdash^?_E u_1 \wedge \ldots T_n \vdash^?_E u_n$$

where $T_1, \ldots, T_n$ are finite set of terms, $u_1, \ldots, u_n$ are terms.

A substitution σ with $dom(\sigma) = V$ is a solution over variables V of a deduction constraint system if and only if $\forall i.\ T_i\sigma \vdash_E u_i\sigma$.

A deduction constraint system may satisfy additional properties:

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

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_is) 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 the long version [16] 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 [18] 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 $\mathtt{mgu}_{\Sigma,E,\mathcal{V}}(u, v)$ the set of most general unifiers of u and v over Σ, equational theory E and variables $\mathcal{V}$. A set of unifiers is a most general set of unifiers if for any unifier σ, there exists a most general unifier μ, such that σ is an

1. Actually, we show that t has a left inverse g, which implies that t is injective. For a function over a finite set, injective implies bijective, so we conclude that t is a bijection.

instance of μ, i.e., there exists a substitution θ 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 $(\underline{\mathcal{A}}, \Sigma, E)$ be an effective algebra and $p, q \in \mathcal{C}_k(X, R)$. Let R' be a set of variables such that $|R'| = |R|$ and $R' \cap R = \emptyset$. We have that*

$$p \simeq q \Leftrightarrow \begin{array}{c} \exists \sigma \in mgu_{\Sigma \cup X^0 \cup R^0, E, R'}(p, q). \\ (\bigwedge_{r \in R}(X, R')\sigma \vdash^? r) \text{ has a solution over } R' \end{array}$$

Moreover, if E is only sound, but not faithful, the implication from right to left ($\Leftarrow$) 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 $(\underline{\mathcal{A}}, \Sigma, E)$ be an effective algebra and $p_1, \ldots, p_m \in \mathcal{C}_1(X, R)$ 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*

$$\begin{array}{c} p_1, \ldots, p_m \\ \simeq \\ r_1, \ldots, r_m \end{array} \Leftrightarrow \begin{array}{l} \exists p_{m+1}, \ldots, p_n \in \mathcal{C}_1(X, R). \\ \forall r \in R.\ (p_1, \ldots, p_n, X) \vdash r \end{array}$$

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

4.3. Static equivalence and non equivalence

The notion of static equivalence was introduced in [19] and its decidability has been studied in [20]. Static equivalence expresses the inability of an adversary to distinguish two sequences of messages.

Definition 43. Let Σ be a signature equipped with E, V a set of variables, and $t_1^1, \ldots, t_k^1, t_1^2, \ldots, t_k^2 \in \mathcal{T}(\Sigma, V)$. 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

$$\forall u_1, u_2 \in \mathcal{T}(\Sigma, (x_1, \ldots, x_k)). \quad \begin{array}{c} u_1\sigma^1 =_E u_2\sigma^1 \\ \Leftrightarrow \\ u_1\sigma^2 =_E u_2\sigma^2 \end{array}$$

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 $\underline{\mathcal{A}}$ be a finite primal algebra over Σ with a sound equational theory E, and $p, q \in \mathcal{C}_k(X, R)$. We have that*

$$p \not\sim_E q \quad \Rightarrow \quad p \not\simeq q$$

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

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

Corollary 47. *Let $\underline{\mathcal{A}}$ be a finite primal algebra over Σ with a sound equational theory E, $p, q \in \mathcal{C}_k(X_{pub} \uplus X_s, R)$, and X_s' such that $|X_s| = |X_s'|$.*

$$p, X_s \not\sim_E p[{}^{X_s'}/_{X_s}], X_s \quad \Rightarrow \quad p \not\perp X_s$$

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 uniformity in the linear case, where only the xor is used. It was previously explored and shown useful in [21], 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 A for our results on computational complexity) developing such heuristics is particularly important.

Our techniques rely on well established symbolic methods. Novel extensions of these techniques and a summary of related, existing results are given in the long version [16].

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 $\underline{\mathbb{F}_q}$, as we can prove that it is uniformly distributed in a ring theory, which is a sound theory for any $\mathbb{F}_q$;
- $u \times x$ is never uniform for any $\underline{\mathbb{F}_{2^n}}$, as we can prove that it is not uniform in $E_{\mathbb{F}_2}$, which is a faithful theory for $\mathbb{F}_{2^n}$;
- $uv + vw + wu$ is uniform over $\underline{\mathbb{F}_2}$ but not $\underline{\mathbb{F}_4}$, while $E_{\mathbb{F}_2}$ is faithful for $\underline{\mathbb{F}_4}$;
- $u \times u$ is not uniform over a ring, but is over $\underline{\mathbb{F}_2}$, while a ring theory is sound for $\underline{\mathbb{F}_2}$.

5.2. Boolean algebras: the linear case

Consider programs only built on Booleans and the 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 [22]. 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 (see the long version [16]).

This problem is decidable in polynomial time [23]. Note that [23] 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 [21] 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 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 $p = uv+vw+vu$. We cannot directly show that $p \simeq u$. However, if we extend p into a program of size 3, we may decide that

$$(uv + vw + uv, u + w, u + v) \simeq (u, v, w)$$

using deduction in finite fields (cf the long version [16]).

Example 50. Let u, v, w be three random variables. We can show using deduction in a commutative ring [24] that:

$$(u, v + uw, w - uw - v) \simeq u, v, w$$

Indeed, $u, v + uw, w - uw - v \vdash u, v, w$ 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 $R' \mapsto R$ 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 $x(u + s) \simeq x(u + s')$, by Lemma 41 with the unifier $u' \mapsto u + s' + s$ on $x(u + s) =_E x(u' + s')$ 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_v, s be input variables. We may solve $v(u + s) \perp s$ by replacing v with the input variable x_v and prove $x_v(u + s) \perp s$, 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 $p = uv + vw + vu$. Applying Lemma 41 directly we obtain the deducibility constraint

$$uv + vw + vu, v', w' \vdash^? u, v, w$$

which admits no obvious solution. Let s, s' be two input variables. Applying Proposition 17, uniformity of p is equivalent to $uv + vw + vu + s \perp s$, which in turn (Lemma 13) holds if and only if

$$uv + vw + vu + s \simeq u'v' + v'w' + v'u' + s'$$

By setting s to $s + s'$, this is directly equivalent to

$$uv + vw + vu + s \simeq u'v' + v'w' + v'u'$$

On this equivalence, Lemma 41 provides a unifier $u', v', w' \mapsto u + s, v + s, w + s$ for which the deducibility constraint has a trivial solution.

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 $p \in \mathcal{C}_m(X, R)$ where $R = \{r_1, \ldots, r_n\}$. Using Proposition 39, uniformity is equivalent to the existence of a program $\alpha \in \mathcal{C}_{n-m}(X, R)$ such that P, α is uniform, which can be verified using deducibility by Proposition 39. Such an α 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 [25] 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 the long version [16]. 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 [26], and deducibility in the bilinear setting was studied in [24] (which we extend to the case where the finite field is of explicit size in the long version [16]).

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 $p = (p_1, \ldots, p_k) \in \mathcal{C}_k(X, R)$, 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 $p \not\perp \{X'\}$.

The techniques for deciding static equivalence developed in [24] 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 $\mathcal{C}_k(X, R_1, \ldots, R_n)$ 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], [27], 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 [24]. 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 a subset of the input I such that $p \simeq_I p$. Furthermore, there is a cardinality constraint on I but this is

independent of the property and not relevant here. The three properties are checked using a common algorithm which require to be able to check the base property for a single tuple.

MASKVERIF provides a very efficient procedure to check the base property. However it is incomplete and, in case of failure, does not provide a witness of interference. 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 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 [28] 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

$$((a_1b_1 + r_1) + a_1b_0 + a_0b_1 + r_0, r_0, r_1)$$

A simple witness of interference that we obtain is then the equation $t_1 + t_2 + t_3 = a_1b_1 + a_1b_0 + a_0b_1$.

Our procedure does output a witness demonstrating that the masked implementation of the Verilog implementation of the AES Sbox as designed in [29] 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 the long version [16].

6.3. Integration in EASYCRYPT

6.3.1. The 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, on may replaced it by a variable that is sampled at random.

In EASYCRYPT, this is done via the proof tactic `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 `rnd` tactic, where the bijection inverse is specified by hand.

```
> examples/elgamal.ec
rnd (fun z, z + log (if b then m1 else m0){2})
    (fun z, z - log (if b then m1 else m0){2}).

> examples/cramer-shoup/cramer_shoup.ec
rnd (fun x2 ⇒  (x2 + G2.v * G1.y2)
         * (G1.w * (G1.u' - G1.u))
         + G1.u * (G1.x + G2.v * G1.y)){2}
    (fun r ⇒  (r - G1.u * (G1.x + G2.v * G1.y))
         / (G1.w * (G1.u' - G1.u))
         - G2.v * G1.y2){2}.

> examples/incomplete/oaep/OAEP.eca
rnd (fun x ⇒  x + pad (if b then m0 else m1){2})
    (fun x ⇒  x - pad (if b then m0 else m1){2}).
```

Listing 1. Examples of the EASYCRYPT rnd tactic

Using our library, it has been possible to enhance the `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 `rnd` tactic in the EASYCRYPT standard libraries and examples.

6.3.2. Simultaneous 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 that appears in the EASYCRYPT proof of the Cramer-Shoup encryption scheme, at the point where one has to apply the DDH assumption. We present in Figure 4 the pseudo-code of the goal, i.e., the two games and the (simplified) post-condition for these games. For readability, we omit the variable prefixes and suffixes used by EASYCRYPT, and simply write w for $G1.w2$. We also omit some variables assignments that do not affect the post condition.

This goal amounts to prove that the secret keys provided by the actual game or the simulator are the same. We directly used our previous notations to capture this goal. If we expend some variable bindings, we obtain the following goal:

$$(k, g, g^x, x_1, x_2, y_1, y_2, z_1, z_2) \simeq (k, g, g^w, x - w * x_2, x_2, y - w * y_2, y_2, z - w * z_2, z_2)$$

We can then conclude by proving that the following map:

$$(k, g, x, x_1, x_2, y_1, y_2, z_1, z_2) \mapsto (k, g, w, x - w * x_2, x_2, y - w * y_2, y_2, z - w * z_2, z_2)$$

Left game:

$x \xleftarrow{\$} \mathbb{F}_q \setminus \{0\}$
$y, z \xleftarrow{\$} \mathbb{F}_q$
$gx, gy, gz \leftarrow g^x, g^y, g^z$
$x_1, x_2, y_1, y_2, z_1, z_2 \xleftarrow{\$} \mathbb{F}_q$
$g\_, a, a\_ \leftarrow gx, gy, gz$
$k \xleftarrow{\$} dk$
$e \leftarrow g^{x1} * g\_^{x2}$
$f \leftarrow g^{y1} * g\_^{y2}$
$h \leftarrow g^{z1} * g\_^{z2}$
$pk \leftarrow (k, g, g\_, e, f, h)$
$sk \leftarrow (k, g, g\_, x_1, x_2, y_1, y_2, z_1, z_2)$

Right game:

$w \xleftarrow{\$} \mathbb{F}_q \setminus \{0\}$
$u, x_0 \xleftarrow{\$} \mathbb{F}_q$
$g\_ \leftarrow= g^w$
$k \xleftarrow{\$} dk$
$x, x_2 \xleftarrow{\$} \mathbb{F}_q$
$x_1, e \leftarrow x - w * x_2, g^x$
$y, y_2 \xleftarrow{\$} \mathbb{F}_q$
$y_1, f \leftarrow y - w * y_2, g^y$
$z, z_2 \xleftarrow{\$} \mathbb{F}_q$
$z_1 \leftarrow z - w * z_2$

Post-condition:

$(k, g, g\_, x_1, x_2, y_1, y_2, z_1, z_2) \simeq (k, g, g\_, x_1, x_2, y_1, y_2, z_1, z_2))$

Figure 4. (Abstracted) EASYCRYPT Goal for Cramer-Shoup

is a bijection - for example, the inverse of $x_1 \mapsto x - w * x2$ being $r \mapsto r + w * x2$.

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

```
swap{1} 16 -9;  wp; swap -1; swap -1.
rnd (fun z ⇒  z + G1.w{2} * G1.z2{2})
    (fun z ⇒  z - G1.w{2} * G1.z2{2}).
rnd.
wp; swap -1.
rnd (fun z ⇒  z + G1.w{2} * G1.y2{2})
    (fun z ⇒  z - G1.w{2} * G1.y2{2}).
rnd.
wp; swap -1.
rnd (fun z ⇒  z + G1.w{2} * G1.x2{2})
    (fun z ⇒  z - G1.w{2} * G1.x2{2}).
rnd; wp; rnd; wp.
```

Listing 2. EASYCRYPT proof script

Using our techniques, we were able to replace it with a single line tactic, using the new tactic `rndmatch`.

```
rndmatch
  (k, g, x, x_1, x_2, y_1, y_2, z_1, z_2)
  (G.k, G.g, G.w, G.x - G.w * G.x2, G.x2,
     G.y - G.w * G.y2, G.y2, G.z - G.w * G, z2, G.z2))
```

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 `rnd` tactic:

```
rnd (fun (v1, v2, v3, v4, v5, v6, v7, v8, v9) ⇒
       (v1, v3, v3, v4 - v3 * v5, v5,
          v6 - v3 * v7, v7, v8 - v2 * v9, v9).
```

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 [30], 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 [31]. 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 [32] show decidability of equality for probabilistic programs (without conditionals or oracle calls) over fixed-length bitstrings. Jutla and Roy [33] show decidability of simulatability for programs (with conditionals but no oracle calls) over finite-length bistrings.

Applications of symbolic methods to masking were considered by Barthe et al in [11], [34], 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 [35]. 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 the long version [16].

Finally, it would be interesting to obtain a better understanding of the decidability and computational complexity of equivalence of programs over finite fields $\mathbb{F}_{2^n}$ for all n. 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

[1] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984.

[2] V. Shoup, "Sequences of games: a tool for taming complexity in security proofs," Cryptology ePrint Archive, Report 2004/332, 2004.

[3] M. Bellare and P. Rogaway, "The security of triple encryption and a framework for code-based game-playing proofs," in *Advances in Cryptology – EUROCRYPT 2006*, ser. Lecture Notes in Computer Science, vol. 4004. Heidelberg: Springer, 2006, pp. 409–426.

[4] S. Halevi, "A plausible approach to computer-aided cryptographic proofs," Cryptology ePrint Archive, Report 2005/181, 2005.

[5] B. Blanchet, "A computationally sound mechanized prover for security protocols," in *27th IEEE Symposium on Security and Privacy, S&P 2006*. IEEE Computer Society, 2006, pp. 140–154.

[6] G. Barthe, B. Grégoire, and S. Zanella-Béguelin, "Formal certification of code-based cryptographic proofs," in *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009*. New York: ACM, 2009, pp. 90–101.

[7] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella-Béguelin, "Computer-aided security proofs for the working cryptographer," in *Advances in Cryptology – CRYPTO 2011*, ser. Lecture Notes in Computer Science, vol. 6841. Heidelberg: Springer, 2011, pp. 71–90.

[8] G. Barthe, J. M. Crespo, B. Grégoire, C. Kunz, Y. Lakhnech, B. Schmidt, and S. Z. Béguelin, "Fully automated analysis of padding-based encryption in the computational model," in *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, A. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM, 2013, pp. 1247–1260. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516663

[9] G. Barthe, B. Grégoire, and B. Schmidt, "Automated proofs of pairing-based cryptography," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, I. Ray, N. Li, and C. Kruegel, Eds. ACM, 2015, pp. 1156–1168. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813697

[10] G. Barthe, X. Fan, J. Gancher, B. Grégoire, C. Jacomme, and E. Shi, "Symbolic proofs for lattice-based cryptography," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 538–555. [Online]. Available: http://doi.acm.org/10.1145/3243734.3243825

[11] G. Barthe, S. Belaïd, F. Dupressoir, P. Fouque, B. Grégoire, and P. Strub, "Verified proofs of higher-order masking," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, ser. Lecture Notes in Computer Science, E. Oswald and M. Fischlin, Eds., vol. 9056. Springer, 2015, pp. 457–485. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-46800-5_18

[12] T. Nipkow, "Unification in primal algebras, their powers and their varieties," *J. ACM*, vol. 37, no. 4, pp. 742–776, Oct. 1990. [Online]. Available: http://doi.acm.org.ins2i.bib.cnrs.fr/10.1145/96559.96569

[13] "Solveq github repository," https://github.com/EasyCrypt/solveq.

[14] "Easycyrpt github repository," https://github.com/EasyCrypt/easycrypt/tree/deploy-solveeq.

[15] "Maskverif source files," https://sites.google.com/site/symbolicforcrypto/.

[16] G. Barthe, B. Grégoire, C. Jacomme, S. Kremer, and P.-Y. Strub, "Symbolic methods in computational cryptography proofs." [Online]. Available: https://hal.inria.fr/hal-02117794

[17] J. K. Millen and V. Shmatikov, "Constraint solving for bounded-process cryptographic protocol analysis," in *CCS 2001, Proc. 8th ACM Conference on Computer and Communications Security (CCS'01)*. ACM, 2001, pp. 166–175. [Online]. Available: https://doi.org/10.1145/501983.502007

[18] K. Knight, "Unification: A multidisciplinary survey," *ACM Comput. Surv.*, vol. 21, no. 1, pp. 93–124, Mar. 1989. [Online]. Available: http://doi.acm.org.ins2i.bib.cnrs.fr/10.1145/62029.62030

[19] M. Abadi and C. Fournet, "Mobile values, new names, and secure communication," in *28th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, POPL 2001*. New York: ACM, 2001, pp. 104–115.

[20] M. Abadi and V. Cortier, "Deciding knowledge in security protocols under equational theories," *Theor. Comput. Sci.*, vol. 367, no. 1-2, pp. 2–32, 2006.

[21] G. Barthe, M. Daubignard, B. Kapron, Y. Lakhnech, and V. Laporte, "On the Equality of Probabilistic Terms," in *Logic for Programming, Artificial Intelligence, and Reasoning*, ser. Lecture Notes in Computer Science, E. M. Clarke and A. Voronkov, Eds. Springer Berlin Heidelberg, 2010, pp. 46–63.

[22] Q. Guo, P. Narendran, and D. Wolfram, "Complexity of nilpotent unification and matching problems," *Information and Computation*, vol. 162, no. 1, pp. 3 – 23, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0890540199928493

[23] S. Delaune, S. Kremer, and D. Pasaila, "Security protocols, constraint systems, and group theories," in *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR'12)*, ser. Lecture Notes in Artificial Intelligence, B. Gramlich, D. Miller, and U. Sattler, Eds., vol. 7364. Manchester, UK: Springer, Jun. 2012, pp. 164–178. [Online]. Available: https://members.loria.fr/skremer/files/Papers/CKP-ijcar12.pdf

[24] G. Barthe, X. Fan, J. Gancher, B. Grégoire, C. Jacomme, and E. Shi, "Symbolic proofs for lattice-based cryptography," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 538–555.

[25] M. Arnaud, V. Cortier, and S. Delaune, "Combining algorithms for deciding knowledge in security protocols," in *Proceedings of the 6th International Symposium on Frontiers of Combining Systems (FroCoS'07)*, ser. Lecture Notes in Artificial Intelligence, F. Wolter, Ed., vol. 4720. Liverpool, UK: Springer, Sep. 2007, pp. 103–117.

[26] S. Delaune, "Easy intruder deduction problems with homomorphisms," *Information Processing Letters*, vol. 97, no. 6, pp. 213 – 218, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020019005003248

[27] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P. Strub, "Easycrypt: A tutorial," in *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, ser. Lecture Notes in Computer Science, A. Aldini, J. López, and F. Martinelli, Eds., vol. 8604. Springer, 2013, pp. 146–166. [Online]. Available: https://doi.org/10.1007/978-3-319-10082-1_6

[28] G. Cassiers and F. Standaert, "Improved bitslice masking: from optimized non-interference to probe isolation," *IACR Cryptology ePrint Archive*, vol. 2018, p. 438, 2018. [Online]. Available: https://eprint.iacr.org/2018/438

[29] H. Groß, S. Mangard, and T. Korak, "An efficient side-channel protected AES implementation with arbitrary protection order," in *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, ser. Lecture Notes in Computer Science, H. Handschuh, Ed., vol. 10159. Springer, 2017, pp. 95–112. [Online]. Available: https://doi.org/10.1007/978-3-319-52153-4_6

[30] M. Abadi and P. Rogaway, "Reconciling two views of cryptography (The computational soundness of formal encryption)," *J. Cryptology*, vol. 15, no. 2, pp. 103–127, 2002.

[31] V. Cortier, S. Kremer, and B. Warinschi, "A survey of symbolic methods in computational analysis of cryptographic systems," *J. Autom. Reasoning*, pp. 1–35, 2010.

[32] G. Barthe, M. Daubignard, B. M. Kapron, Y. Lakhnech, and V. Laporte, "On the equality of probabilistic terms," in *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, ser. Lecture Notes in Computer Science, E. M. Clarke and A. Voronkov, Eds., vol. 6355. Springer, 2010, pp. 46–63. [Online]. Available: https://doi.org/10.1007/978-3-642-17511-4_4

[33] C. S. Jutla and A. Roy, "A completeness theorem for pseudo-linear functions with applications to uc security," *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 17, p. 92, 2010.

[34] G. Barthe, S. Belaïd, F. Dupressoir, P. Fouque, B. Grégoire, P. Strub, and R. Zucchini, "Strong non-interference and type-directed higher-order masking," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 116–129. [Online]. Available: https://doi.org/10.1145/2976749.2978427

[35] G. L. Mullen and D. Panario, *Handbook of finite fields*. Chapman and Hall/CRC, 2013.

[36] R. Chang, J. Kadin, and P. Rohatgi, "On unique satisfiability and the threshold behavior of randomized reductions," *Journal of Computer and System Sciences*, vol. 50, no. 3, pp. 359 – 373, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000085710288

[37] S. Fenner, F. Green, S. Homer, and R. Pruim, "Determining acceptance possibility for a quantum computation is hard for the polynomial hierarchy," *Proceedings of The Royal Society A Mathematical Physical and Engineering Sciences*, vol. 455, 02 1999.

[38] S. Toda and M. Ogiwara, "Counting classes are at least as hard as the polynomial-time hierarchy," in *[1991] Proceedings of the Sixth Annual Structure in Complexity Theory Conference*, June 1991, pp. 2–12.

[39] M. N. Vyalyi, "Qma = pp implies that pp contains ph," in *In ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 2003.

Appendix A. Complexity results

A.1. Classical complexity classes

Program equivalence for a given Σ-algebra $\underline{\mathcal{A}}$ yields a natural decision problem $\mathsf{EQUIV}^k_{\underline{\mathcal{A}}}$ defined as follows.

INPUT: $p, q \in \mathcal{C}_k(X, R)$ defined over $\underline{\mathcal{A}}$.
QUESTION: $p \sim q$.

We first show a coNP lower and PSPACE upper bound.

Proposition 54. $\mathsf{EQUIV}^k_{\mathbb{F}_q}$ *is in* PSPACE *and* coNP*-hard.*

Proof. We first show that $\mathsf{EQUIV}^k_{\mathbb{F}_q}$ is in PSPACE. For this consider the following decision procedure where conv converts a tuple of k elements in $\mathbb{F}_q$, into a corresponding number in $[1, q^k]$.

```
For ρ in A^X:
  initialise arrays t_p, t_q of size q^k to 0
  for τ in A^R:
    t_P[conv([p]_{ρ∪τ})] ++
    t_Q[conv([q]_{ρ∪τ})] ++
  if t_p ≠ t_q return False
return True
```

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

We next show coNP-hardness by reduction from UNSAT. Given a formula CNF ϕ over variables in R and $(\vee, \wedge)$ we transform ϕ into a formula ϕ' over R and $\oplus, \wedge$ in polynomial time w.r.t the size of the formula. Indeed, given a clause of ϕ of the form $x \vee y \vee z$, we have that $x \vee y \vee z = (x \oplus y \oplus xy) \vee z = (x \oplus y \oplus xy) \oplus z \oplus (x \oplus y \oplus xy)z = x \oplus y \oplus xy \oplus z \oplus xz \oplus yz \oplus xyz = x \oplus y \oplus z \oplus xy \oplus yz \oplus xz \oplus xyz$. With this transformation, we have $|\phi'| \leq 5 \times |\phi|$. Let $p = \phi'$. We have that $p \in \mathcal{C}_1(\emptyset, R)$ defined over $\mathbb{F}_2$ and

$$p \simeq 0 \Leftrightarrow \phi \in \mathsf{UNSAT}$$

Indeed, if ϕ is unsatisfiable, then for any values taken by the variables in X, ϕ evaluates to 0 and p equals 0. This means that p is equal to the zero distribution. Conversely, if ϕ 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 coDP. Recall that $\mathsf{coDP} = \{L_1 \cup L_2 | L_1 \in \mathsf{NP}, L_2 \in \mathsf{coNP}\}$).

Proposition 55. *If* $\mathsf{EQUIV}^1_{\mathbb{F}_2} \in \mathsf{coDP}$ *then* PH *collapses.*

Proof. Recall that unique satisfiability USAT is the set of satisfiability problems that admit exactly one satisfying assignment. By [36], we have that $\mathsf{USAT} \in \mathsf{coDP}$ implies that PH collapses. We thus reduce USAT to program equivalence in polynomial time. We are given a CNF formula ϕ over variables X (all used) and $(\vee, \wedge)$. We can transform in polynomial time w.r.t the size of the formula ϕ into a formula ϕ' over $(\oplus, \wedge)$. Then, let $p = \phi'$ which is a program in $\mathcal{C}_1(\emptyset, \mathcal{X})$ over $\mathbb{F}_2$. We have that

$$p \simeq \bigwedge_{x \in X} x \Leftrightarrow \phi \in \mathsf{USAT}$$

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 coNP or NP, i.e it is unlikely that an efficient decision procedure exists.

A.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 $\mathsf{R-EQUIV}^1_{\mathbb{F}_2}$ is the version without input variables defined as follows.

INPUT: $p, q \in \mathcal{C}_k(\emptyset, R)$ defined over $\underline{\mathcal{A}}$.
QUESTION: $p \sim q$.

The problem halfSAT is defined as follows.

INPUT: CNF boolean formula ϕ.
QUESTION: Exactly half of the valuations satisfy ϕ.

Definition 56. $\mathsf{C_{=}P}$ 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 57. halfSAT *is complete for* $\mathsf{C_{=}P}$

We can know link uniformity and halfSAT.

Proposition 58. $\mathsf{R-EQUIV}^1_{\mathbb{F}_2}$ *is* $\mathsf{C_{=}P}$*-complete.*

Proof. **Hardness**:
As in the reduction showing coNP-hardness, given ϕ we have that there exists $p \in \mathcal{C}_1(\emptyset, R)$ such that

$$p \simeq r \Leftrightarrow \phi \in \mathsf{halfSAT}$$

where r is a fresh random variable.
Membership:
We are given $p \in \mathcal{C}_1(\emptyset, R)$. 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.

$$\begin{aligned} p \simeq R \quad &\Leftrightarrow p \text{ equals 1 for half of the valuations} \\ &\Leftrightarrow \text{M accepts half of the time} \end{aligned}$$

□

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