Verified Indifferentiable Hashing into Elliptic Curves

Gilles Barthe[1], Benjamin Grégoire[2], Sylvain Heraud[2], Federico Olmedo[1], and Santiago Zanella-Béguelin[3]

[1]*IMDEA Software Institute, Madrid, Spain,*
{gilles.barthe,federico.olmedo}@imdea.org
[2]*INRIA Sophia Antipolis-Méditerranée, France,*
benjamin.gregoire@inria.fr, sylvain.heraud@gmail.com
[3]*Microsoft Research*
santiago@microsoft.com

Submitted July 2012. Revised April 2013.

Abstract

Many cryptographic systems based on elliptic curves are proven secure in the Random Oracle Model, assuming there exist probabilistic functions that map elements in some domain (e.g. bitstrings) onto uniformly and independently distributed points in a curve. When implementing such systems, and in order for the proof to carry over to the implementation, those mappings must be instantiated with concrete constructions whose behavior does not deviate significantly from random oracles. In contrast to other approaches to public-key cryptography, where candidates to instantiate random oracles have been known for some time, the first generic construction for hashing into ordinary elliptic curves indifferentiable from a random oracle was put forward only recently by Brier et al. We present a machine-checked proof of this construction. The proof is based on an extension of the CertiCrypt framework with logics and mechanized tools for reasoning about approximate forms of observational equivalence, and integrates mathematical libraries of group theory and elliptic curves.

Keywords: *Provable security, indifferentiability, random oracle model, elliptic curve cryptography*

1 Introduction

Following an established trend [25], the prevailing methodology for building secure cryptosystems is to conduct a rigorous analysis that proves security under standard hypotheses. Sometimes this analysis is performed assuming that some components of the system have an ideal behavior. However, ideal functionalities are difficult or

even impossible to realize, leading to situations where provably secure systems have no secure implementation. An alternative methodology is to devise systems based on constructions that do not deviate significantly from ideal ones, and to account for these deviations in the security analysis. Statistical distance is a natural notion for quantifying the deviation between idealized functionalities and their implementations.

Verifiable security [4, 6] is an emerging approach that advocates the use of interactive proof assistants and automated provers to establish the security of cryptographic systems. It improves on the guarantees of provable security by delivering fully machine-checked and independently verifiable proofs. The CertiCrypt framework, built on top of the Coq proof assistant, is one prominent tool that realizes verifiable security by using standard techniques from programming languages and program verification. CertiCrypt is built around the central notion of observational equivalence of probabilistic programs, which unfortunately cannot model accurately other weaker, quantitative, forms of equivalence. As a result, CertiCrypt cannot be used as it is to reason about the statistical distance of distributions generated by probabilistic programs. More generally, the development of quantitative notions of equivalence is quite recent and rather limited; see Section 7 for an account of related work.

One main contribution of this article is the formalization of several quantitative notions of program equivalence and logics for reasoning about them. More specifically, we extend CertiCrypt with the notion of statistical distance and develop a logic to upper-bound the distance between distributions generated by probabilistic programs. Moreover, we introduce approximate and conditional variants of observational equivalence and develop equational theories for reasoning about them.

In a landmark article, Maurer et al. [33] introduce the concept of indifferentiability to justify rigorously the substitution of an idealized component in a cryptographic system by a concrete implementation. In a subsequent article, Coron et al. [17] argue that a secure hash function should be indifferentiable from a random oracle, i.e. a perfectly random function. Although the random oracle model has been under fierce criticism [14] and the indifferentiability framework turns out to be weaker than initially believed [22, 36], it is generally accepted that proofs in these models provide some evidence that a system is secure. Not coincidentally, all finalists in the NIST Cryptographic Hash Algorithm competition have been proved indifferentiable from a random oracle.

Elliptic curve cryptography allows to build efficient public-key cryptographic systems with comparatively short keys and as such is an attractive solution for resource-constrained applications. In contrast to other approaches to public-key cryptography, where candidates to instantiate random oracles into bitstrings, residue classes, or finite fields have been known for some time, constructions of random oracles into ordinary elliptic curves have remained elusive. Informally, hash functions into elliptic curves are typically built from a hash function $\mathcal{G}$ on the underlying field and a deterministic function f that maps elements of the finite field into the elliptic curve; examples of such mappings include Icart's function [30] and the Shallue-Woestijne-Ulas algorithm [39]. In general, and in particular for the aforementioned mappings, the function f is not surjective and only covers a fraction of points in the curve. Hence, the naive definition of a hash function $\mathcal{H}$ as $f \circ \mathcal{G}$ does not cover the whole curve, i.e. there are many points m for which there is no x s.t. $\mathcal{H}(x) = m$, contradicting the assumption that $\mathcal{H}$

behaves as a random oracle. In 2010, Brier et al. [13] proposed the first generic construction indifferentiable from a random oracle into elliptic curves. This construction is of practical significance since it allows to securely implement elliptic curve cryptosystems. We present a machine-checked and independently verifiable proof of the security of this construction. The proof involves the various notions of equivalence we develop in this article and is thus an excellent testbed for evaluating the applicability of our methods. Additionally, the proof builds on several large developments (including Théry's formalization of elliptic curves [44] and Gonthier et al.'s formalization of finite groups [26]) and demonstrates that CertiCrypt blends well with large and complex mathematical libraries, and is apt to support proofs involving advanced algebraic and number-theoretical reasoning.

Organization of the article. This article is an extended version of [7]. The remainder of the article is structured as follows. Section 2 provides a brief introduction to CertiCrypt. Section 3 introduces the notion of statistical distance between probabilistic programs and describes programming language techniques to bound it, whereas Section 4 defines weak forms of observational equivalence and their associated reasoning principles. Section 5 presents a machine-checked proof of the indifferentiability of a generalization of Brier et al.'s construction from a random oracle into an abelian finite group; its application to elliptic curves is discussed in Section 6. We survey prior art and conclude in Sections 7 and 8.

2 An Overview of CertiCrypt

CertiCrypt is a framework built on top of the Coq proof assistant [43] for building and verifying security proofs of cryptographic systems. CertiCrypt adopts the code-based game-playing approach, in which probabilistic programs are used to describe security goals and assumptions as experiments where an adversary interacts with a challenger—these experiments are called *games*. Proofs are structured as a sequence (in general, a tree) of games $\mathsf{G}_0, \cdots, \mathsf{G}_n$, and the overall goal is to show that the probability of some event in the initial game is related in some specified way to the probability of an event in the final game of the sequence. A proof proceeds by relating the probability of appropriate events in consecutive games, which typically boils down to proving that the games satisfy some form of program equivalence. One fundamental advantage of this approach, which lies at the heart of CertiCrypt, is that it enables justifying game transformations by means of semantic arguments.

This section summarizes the main components of CertiCrypt, namely the representation of probability distributions, the probabilistic programming language used to model games, and the relational Probabilistic Hoare Logic for reasoning about them. We refer the reader to [4] for further details.

2.1 Representation of Distributions

CertiCrypt adopts the monadic representation of distributions proposed by Audebaud and Paulin-Mohring [2]. Let $[0, 1]$ denote the unit interval. A distribution over a set A

is a monotonic, continuous and linear function of type

$$\mathcal{D}(A) \stackrel{\text{def}}{=} (A \to [0,1]) \to [0,1]$$

More formally, a distribution is modeled as a function μ of type $\mathcal{D}(A)$ satisfying the following (universally quantified) properties:

Monotonicity: $f \leq g \implies \mu\, f \leq \mu\, g$;

Compatibility with inverse: $\mu\,(\mathbb{1} - f) \leq 1 - \mu\, f$, where $\mathbb{1}$ is the constant function 1;

Homogeneity: $\mu\,(k \times f) = k \times \mu\, f$ for any $k \in [0,1]$;

Additivity: $f \leq \mathbb{1} - g \implies \mu\,(f + g) = \mu\, f + \mu\, g$;

Continuity: $\mu\,(\sup\ F) \leq \sup\ (\mu \circ F)$ for any monotonic $F : \mathbb{N} \to (A \to [0,1])$.

Observe that we do not require that $\mu\ \mathbb{1} = 1$; therefore our definition corresponds to sub-probability distributions. In particular, we let μ_0 denote the null sub-distribution, i.e. $\mu_0\ \mathbb{1} = 0$.

Intuitively, an element μ of type $\mathcal{D}(A)$ models the expectation operator of a sub-probability distribution over A: when A is a discrete set one has

$$\mu\, f = \sum_{a \in A} \mu(a) f(a) \tag{1}$$

where $\mu(a)$ denotes the probability mass function of μ at a. Thus, the probability that the distribution $\mu : \mathcal{D}(A)$ assigns to an event $X \subseteq A$ can be computed by measuring its characteristic function $\mathbb{1}_X$, i.e.

$$\Pr[\mu : X] \stackrel{\text{def}}{=} \mu\, \mathbb{1}_X$$

Distributions can be given the structure of a monad. The unit and bind operators are defined as follows:

$$\begin{array}{lll} \mathsf{unit} & : & A \to \mathcal{D}(A) \\ & \stackrel{\text{def}}{=} & \lambda x.\ \lambda f.\ f\ x \end{array} \qquad \begin{array}{lll} \mathsf{bind} & : & \mathcal{D}(A) \to (A \to \mathcal{D}(B)) \to \mathcal{D}(B) \\ & \stackrel{\text{def}}{=} & \lambda \mu.\ \lambda M.\ \lambda f.\ \mu\ (\lambda\ x.\ M\ x\ f) \end{array}$$

For a value $a \in A$, the expression $\mathsf{unit}\ a$ denotes the Dirac distribution on a, which assigns probability 1 to a and 0 to all other values in A. The bind operator composes a distribution μ over A and a conditioned distribution M over B given $a \in A$: when A and B are discrete sets, the probability mass function of $\mathsf{bind}\ \mu\ M$ evaluated at b is $\sum_{a \in A} \mu(a) M(a)(b)$.

We say that a distribution $\mu : \mathcal{D}(A)$ is *discrete* when (1) holds for any function $f : A \to [0,1]$. Note that if A is discrete, then any distribution on A is discrete. However μ might be discrete even if A is not; in particular, we prove that programs that only sample values from discrete sets output discrete distributions, even if the set of program states is not discrete. Finally, we define the *support* of a discrete distribution $\mu : \mathcal{D}(A)$ as $\mathsf{supp}\,(\mu) \stackrel{\text{def}}{=} \{a \in A \mid \mu(a) > 0\}$.

2.2 Programming Model

In this section we describe the programming language adopted by CertiCrypt to describe games. Roughly speaking, games are modeled as probabilistic imperative programs with procedure calls. The set of commands $\mathcal{C}$ is defined inductively by the clauses:

$\mathcal{C}$	::=	skip	nop
	|	$\mathcal{V} \leftarrow \mathcal{E}$	deterministic assignment
	|	$\mathcal{V} \xleftarrow{\$} \mathcal{D}\mathcal{E}$	random assignment
	|	$\mathsf{assert}\ \mathcal{E}$	runtime assertion
	|	$\mathsf{if}\ \mathcal{E}\ \mathsf{then}\ \mathcal{C}\ \mathsf{else}\ \mathcal{C}$	conditional
	|	$\mathsf{while}\ \mathcal{E}\ \mathsf{do}\ \mathcal{C}$	while loop
	|	$\mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E})$	procedure call
	|	$\mathcal{C};\ \mathcal{C}$	sequence

where $\mathcal{V}$ is a set of variables tagged with their scope (either local or global), $\mathcal{E}$ is a set of deterministic expressions, and $\mathcal{D}\mathcal{E}$ is a set of expressions that denote distributions from which values can be sampled in random assignments. In the remainder, we let $\mathsf{true} \oplus_\delta \mathsf{false}$ denote the Bernoulli distribution with success probability δ, so that the instruction $x \xleftarrow{\$} \mathsf{true} \oplus_\delta \mathsf{false}$ assigns true to x with probability δ, and we denote by $x \xleftarrow{\$} A$ the instruction that assigns to x a value uniformly chosen from a finite set A.

A program (or game) consists of a command c and an environment E that maps procedure identifiers to their declaration, specifying their formal parameters, their body, and a return expression that is evaluated upon exit.

$$\mathsf{decl} \stackrel{\text{def}}{=} \{\mathsf{args} : \mathsf{list}\ \mathcal{V};\ \mathsf{body} : \mathcal{C};\ \mathsf{re} : \mathcal{E}\}$$

Although procedures are single-exit, we often write games using explicit return expressions for the sake of readability. Declarations are subject to well-formedness and well-typedness conditions; these conditions are enforced using the underlying dependent type system of Coq.

Adversaries are procedures whose code and return expression are unknown. However, adversaries are required to respect basic interface conditions; such conditions enforce scoping, and may ensure for example that the adversary cannot read or write values that it has to guess. Formally, an interface is a triple $(\mathcal{O}, \mathcal{RW}, \mathcal{R})$, where $\mathcal{O}$ is a set of oracles, and $\mathcal{RW}$ and $\mathcal{R}$ are sets of variables. An adversary respects an interface $(\mathcal{O}, \mathcal{RW}, \mathcal{R})$ if it only reads variables in $\mathcal{RW} \cup \mathcal{R}$, only writes variables in $\mathcal{RW}$, and only call oracles in $\mathcal{O}$ or procedures that respect the same interface as itself. If this is the case, we say that $\mathcal{A}$ is *well-formed* w.r.t. interface $(\mathcal{O}, \mathcal{RW}, \mathcal{R})$ and note it $\vdash_{\text{wf}} \mathcal{A}$. This condition is defined inductively by the rules of Figure 1. We remark that this set of rules only aims at ensuring the correct use of variables and procedure calls by the adversary and are general enough as to capture the behavior of any legitimate adversary in standard cryptographic security models. Any additional constraints, such as conditions on the number or form of oracle calls may be stated as post-conditions of security experiments.

The system of Figure 1 yields an induction principle for well-formed adversaries of key importance in our development, since it allows to extend any proof system for

$$I \vdash \mathsf{skip} : I \qquad \frac{I \vdash i : I' \quad I' \vdash c : O}{I \vdash i; c : O}$$

$$\frac{\mathsf{writable}(x) \quad \mathsf{fv}(e) \subseteq I}{I \vdash x \leftarrow e : I \cup \{x\}} \qquad \frac{\mathsf{writable}(x)}{I \vdash x \xleftarrow{\$} T : I \cup \{x\}}$$

$$\frac{\mathsf{fv}(e) \subseteq I \quad I \vdash c_i : O_i,\ i = 1, 2}{I \vdash \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 : O_1 \cap O_2} \qquad \frac{\mathsf{fv}(e) \subseteq I \quad I \vdash c : I}{I \vdash \mathsf{while}\ e\ \mathsf{do}\ c : I}$$

$$\frac{\mathsf{fv}(\vec{e}) \subseteq I \quad \mathsf{writable}(x) \quad p \in \mathcal{O}}{I \vdash x \leftarrow p(\vec{e}) : I \cup \{x\}}$$

$$\frac{\mathsf{fv}(\vec{e}) \subseteq I \quad \mathsf{writable}(x) \quad p \notin \mathcal{O} \quad \vdash_{\mathrm{wf}} p}{I \vdash x \leftarrow p(\vec{e}) : I \cup \{x\}}$$

$$\frac{\mathcal{RW} \cup \mathcal{R} \cup \mathcal{A}.\mathsf{args} \vdash \mathcal{A}.\mathsf{body} : O \quad \mathsf{fv}(\mathcal{A}.\mathsf{re}) \subseteq O}{\vdash_{\mathrm{wf}} \mathcal{A}}$$

$$\mathsf{writable}(x) \stackrel{\text{def}}{=} \mathsf{local}(x) \vee x \in \mathcal{RW}$$

Figure 1: Rules for well-formedness of an adversary against interface $(\mathcal{O}, \mathcal{RW}, \mathcal{R})$. A judgment of the form $I \vdash c : O$ reads as follows: assuming variables in I may be read, the adversarial code fragment c respects the interface, and after its execution variables in O may be read. Thus, $I \vdash c : O \implies I \subseteq O$.

closed programs to programs with calls to well-formed adversaries. Specifically, in Section 3.1 we present a logic to bound the statistical distance between the output distributions of two (structurally similar) programs. The soundness of the rule for calling a well-formed adversary is proved by structural induction on the derivation of its well-formedness.

Program states (or memories) are dependently typed functions that map a variable of type T to a value in its interpretation $[\![T]\!]$; we let $\mathcal{M}$ denote the set of states. Expressions have a deterministic semantics: an expression $e \in \mathcal{E}$ of type T is interpreted as a function $[\![e]\!] : \mathcal{M} \to [\![T]\!]$. The semantics of a distribution expression $d \in \mathcal{DE}$ of type T is given by a function $[\![d]\!] : \mathcal{M} \to \mathcal{D}([\![T]\!])$. Finally the semantics of a command c in an environment E relates an initial memory to a sub-probability distribution over final memories: $[\![c, E]\!] : \mathcal{M} \to \mathcal{D}(\mathcal{M})$. We often omit the environment when it is irrelevant. The semantics of programs verifies the expected equations for language constructs (see Figure 2). In particular, the semantics of a while loop can be given in terms of its n-th unrolling $[\mathsf{while}\ e\ \mathsf{do}\ c]_n$, defined recursively by the equations

$$\begin{array}{lcl} [\mathsf{while}\ e\ \mathsf{do}\ c]_0 & \stackrel{\text{def}}{=} & \mathsf{assert}\ \neg e \\ [\mathsf{while}\ e\ \mathsf{do}\ c]_{n+1} & \stackrel{\text{def}}{=} & \mathsf{if}\ e\ \mathsf{then}\ c;\ [\mathsf{while}\ e\ \mathsf{do}\ c]_n \end{array}$$

By specializing the definition of probability $\Pr[\mu : X]$ from Section 2.1 to programs, we have that the probability $\Pr[G, m : X]$ of an event X w.r.t. the distribution obtained by running a game G with initial memory m is given by $[\![G]\!]\ m\ \mathbb{1}_X$. In particular, the probability the game terminates is $\Pr[G, m : \mathsf{true}]$. We say that a game is *lossless* if it terminates with probability 1 independently of the initial memory.

$$
\begin{array}{lcl}
[\![x \leftarrow e]\!]\, m & = & \mathsf{unit}\,(m\,\{[\![e]\!]\, m/x\}) \\
[\![x \xleftarrow{\$} d]\!]\, m & = & \mathsf{bind}\,([\![d]\!]\, m)\,(\lambda v.\ \mathsf{unit}\,(m\,\{v/x\})) \\
[\![\mathsf{assert}\ e]\!]\, m & = & \begin{cases} \mathsf{unit}\, m & \text{if } [\![e]\!]\, m = \mathsf{true} \\ \mu_0 & \text{if } [\![e]\!]\, m = \mathsf{false} \end{cases} \\
[\![\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2]\!]\, m & = & \begin{cases} [\![c_1]\!]\, m & \text{if } [\![e]\!]\, m = \mathsf{true} \\ [\![c_2]\!]\, m & \text{if } [\![e]\!]\, m = \mathsf{false} \end{cases} \\
[\![\mathsf{while}\ e\ \mathsf{do}\ c]\!]\, m & = & \lambda f.\ \sup_{n \in \mathbb{N}}\,([\![[\mathsf{while}\ e\ \mathsf{do}\ c]_n]\!]\, m\ f)
\end{array}
$$

Figure 2: Denotatonial semantics of programs

In order to reason about program complexity and define the class of probabilistic polynomial-time computations, the semantics of programs is indexed by a security parameter (a natural number) and instrumented to compute the time and memory cost of evaluating a command, given the time and memory cost of each construction in the expression language. We chose not to make this parameterization explicit to avoid cluttering the presentation.

2.3 Reasoning Tools

CertiCrypt provides several tools for reasoning about games. One main tool is a probabilistic relational Hoare logic. Its judgments are of the form $\models \mathsf{G}_1 \sim \mathsf{G}_2 : \Psi \Rightarrow \Phi$, where G_1 and G_2 are games, and Ψ and Φ are relations over states. We represent relations as first-order formulae over tagged program variables; we use the tags $\langle 1 \rangle$ and $\langle 2 \rangle$ to distinguish between the value of a variable or formula in the left and right-hand side program, respectively. A judgment

$$\models \mathsf{G}_1 \sim \mathsf{G}_2 : \Psi \Rightarrow \Phi$$

is valid, iff for all memories m_1 and m_2 we have

$$m_1\ \Psi\ m_2 \implies ([\![\mathsf{G}_1]\!]\, m_1)\, \mathcal{L}(\Phi)\, ([\![\mathsf{G}_2]\!]\, m_2),$$

where $\mathcal{L}(\Phi)$ denotes the lifting of Φ to distributions. The lifting operator $\mathcal{L}(\cdot)$ transforms a binary relation $R \subseteq A \times B$ into a binary relation $\mathcal{L}(R) \subseteq \mathcal{D}(A) \times \mathcal{D}(B)$. We adopt the following definition, from probabilistic process algebra [32]:

$$\mu_1\, \mathcal{L}(R)\, \mu_2 \stackrel{\text{def}}{=} \exists \mu : \mathcal{D}(A \times B).\ \begin{cases} \mu(a,b) > 0 \implies a\ R\ b & \text{for all } (a,b) \in A \times B \\ (\pi_1 \mu)(a) = \mu_1(a) & \text{for all } a \in A \\ (\pi_2 \mu)(b) = \mu_2(b) & \text{for all } b \in B \end{cases}$$

where $\pi_1\mu \in \mathcal{D}(A)$ and $\pi_2\mu \in \mathcal{D}(B)$ denote the first and second projections of μ respectively, i.e. $(\pi_1\mu)(a) = \mu(a, B)$ and $(\pi_2\mu)(b) = \mu(A, b)$ or, in monadic notation,

$$\pi_1(\mu) \stackrel{\text{def}}{=} \mathsf{bind}\ \mu\ (\mathsf{unit} \circ \mathsf{fst}) \qquad \pi_2(\mu) \stackrel{\text{def}}{=} \mathsf{bind}\ \mu\ (\mathsf{unit} \circ \mathsf{snd})$$

$$\models \mathsf{skip} \sim \mathsf{skip} : \Phi \Rightarrow \Phi \ [\mathsf{Skip}] \qquad \frac{\models c_1 \sim c_2 : \Psi \Rightarrow \Theta \quad \models c_1' \sim c_2' : \Theta \Rightarrow \Phi}{\models c_1; c_1' \sim c_2; c_2' : \Psi \Rightarrow \Phi}\ [\mathsf{Seq}]$$

$$\frac{m_1 \ \Psi \ m_2 = (m_1 \{[\![e_1]\!] m_1 / x_1\}) \ \Phi \ (m_2 \{[\![e_2]\!] m_2 / x_2\})}{\models x_1 \leftarrow e_1 \sim x_2 \leftarrow e_2 : \Psi \Rightarrow \Phi}\ [\mathsf{Assn}]$$

$$\frac{\begin{array}{c} m_1 \ \Psi \ m_2 \implies ([\![d_1]\!] \, m_1) \, \mathcal{L}(\Theta) \, ([\![d_2]\!] \, m_2) \\ \text{where } v_1 \ \Theta \ v_2 = (m_1 \{v_1/x_1\}) \ \Phi \ (m_2 \{v_2/x_2\}) \end{array}}{\models x_1 \xleftarrow{\$} d_1 \sim x_2 \xleftarrow{\$} d_2 : \Psi \Rightarrow \Phi}\ [\mathsf{Rnd}]$$

$$\frac{\begin{array}{c} m_1 \ \Psi \ m_2 \implies [\![e_1]\!] \, m_1 = [\![e_2]\!] \, m_2 \\ \models c_1 \sim c_2 : \Psi \wedge e_1\langle 1\rangle \Rightarrow \Phi \quad \models c_1' \sim c_2' : \Psi \wedge \neg e_1\langle 1\rangle \Rightarrow \Phi \end{array}}{\models \mathsf{if}\ e_1\ \mathsf{then}\ c_1\ \mathsf{else}\ c_1' \sim \mathsf{if}\ e_2\ \mathsf{then}\ c_2\ \mathsf{else}\ c_2' : \Psi \Rightarrow \Phi}\ [\mathsf{Cond}]$$

$$\frac{m_1 \ \Phi \ m_2 \implies [\![e_1]\!] \, m_1 = [\![e_2]\!] \, m_2 \quad \models c_1 \sim c_2 : \Phi \wedge e_1\langle 1\rangle \Rightarrow \Phi}{\models \mathsf{while}\ e_1\ \mathsf{do}\ c_1 \sim \mathsf{while}\ e_2\ \mathsf{do}\ c_2 : \Phi \Rightarrow \Phi \wedge \neg e_1\langle 1\rangle}\ [\mathsf{While}]$$

$$\frac{\Psi \subseteq \Psi' \quad \models c_1 \sim c_2 : \Psi' \Rightarrow \Phi' \quad \Phi' \subseteq \Phi}{\models c_1 \sim c_2 : \Psi \Rightarrow \Phi}\ [\mathsf{Sub}]$$

$$\frac{\models c_1 \sim c_2 : \Psi \wedge \Psi' \Rightarrow \Phi \quad \models c_1 \sim c_2 : \Psi \wedge \neg\Psi' \Rightarrow \Phi}{\models c_1 \sim c_2 : \Psi \Rightarrow \Phi}\ [\mathsf{Case}]$$

Figure 3: Selected rules of probabilistic Relational Hoare Logic

Figure 3 provides an excerpt of the set of rules of the relational Hoare logic. The logic can be used to prove (in)equalities between probability quantities; for instance, using the following rules:

$$\frac{m_1 \ \Psi \ m_2 \qquad \models \mathsf{G}_1 \sim \mathsf{G}_2 : \Psi \Rightarrow \Phi \qquad \Phi \implies (A\langle 1\rangle \Longrightarrow B\langle 2\rangle)}{\Pr[\mathsf{G}_1, m_1 : A] \leq \Pr[\mathsf{G}_2, m_2 : B]}$$

$$\frac{m_1 \ \Psi \ m_2 \qquad \models \mathsf{G}_1 \sim \mathsf{G}_2 : \Psi \Rightarrow \Phi \qquad \Phi \implies (A\langle 1\rangle \Longleftrightarrow B\langle 2\rangle)}{\Pr[\mathsf{G}_1, m_1 : A] = \Pr[\mathsf{G}_2, m_2 : B]}$$

Observational equivalence is defined by specializing judgments to relations Ψ and Φ corresponding to the equality relation on subsets of program variables. Formally, let X be a set of variables, $m_1, m_2 \in \mathcal{M}$ and $f_1, f_2 : \mathcal{M} \to [0,1]$. We define

$$\begin{array}{l} m_1 =_X m_2 \stackrel{\text{def}}{=} \forall x \in X.\, m_1(x) = m_2(x) \\ f_1 =_X f_2 \stackrel{\text{def}}{=} \forall m_1, m_2.\, m_1 =_X m_2 \implies f_1(m_1) = f_2(m_2) \end{array}$$

Then, two games G_1 and G_2 are observationally equivalent w.r.t. an input set of variables I and an output set of variables O, written $\models \mathsf{G}_1 \simeq_O^I \mathsf{G}_2$, iff

$$\models \mathsf{G}_1 \sim \mathsf{G}_2 : =_I \Rightarrow =_O$$

Equivalently, $\models \mathsf{G}_1 \simeq_O^I \mathsf{G}_2$ iff for all memories $m_1, m_2 \in \mathcal{M}$ and functions $f_1, f_2 : \mathcal{M} \to [0,1]$,

$$m_1 =_I m_2 \wedge f_1 =_O f_2 \implies [\![\mathsf{G}_1]\!]\, m_1\, f_1 = [\![\mathsf{G}_2]\!]\, m_2\, f_2$$

Observational equivalence is amenable to automation. CertiCrypt provides mechanized tactics based on dependency analyses to perform common program transformations and to prove that two programs satisfy an observational equivalence specification. Note that since observational equivalence is only a partial equivalence relation, it might be of interest to prove that a program is observational equivalent to itself—this actually encompasses information flow security. Relevant mechanized transformations used in the context of this article include dead code elimination, procedure call inlining, code motion and expression propagation.

We sometimes use a standard Hoare logic for reasoning about individual programs. Its judgments are of the form $\{P\}\ \mathsf{G}\ \{Q\}$, where G is a game and P and Q are predicates on states. Formally, a judgment $\{P\}\ \mathsf{G}\ \{Q\}$ is valid iff for every memory $m \in \mathcal{M}$ and function $f : \mathcal{M} \to [0,1]$,

$$P\ m \wedge (\forall m.\, Q\ m \implies f(m) = 0) \implies [\![\mathsf{G}]\!]\, m\, f = 0$$

3 Statistical Distance

Statistical distance quantifies the largest difference between the probability that two distributions assign to the same event, and underlies many concepts in cryptography, such as indifferentiability [33] and statistical zero-knowledge proofs [24]. We review its definition next, and provide below an alternative characterization that is more appropriate for reasoning about the monadic representation of distributions that we use in our development; we refer to Shoup [40] or Sahai and Vadhan [37] for an in-depth presentation of statistical distance and its properties.

Definition 1. *The statistical distance (i.e. total variation distance) $\Delta(\mu_1, \mu_2)$ between two distributions μ_1 and μ_2 over a set A is defined as*

$$\Delta(\mu_1, \mu_2) \stackrel{def}{=} \sup_{f : A \to \{0,1\}} |\mu_1\ f - \mu_2\ f| \tag{2}$$

Note that for any f, the expression $|\mu_1 f - \mu_2 f|$ is upper-bounded by 1; hence the supremum exists and $\Delta(\mu_1, \mu_2)$ is well defined. Statistical distance satisfies the metric axioms, and is non-increasing under function application, i.e. for all sets A and B, distributions μ_1, μ_2, and μ_3 over $\mathcal{D}(A)$ and functions $f : \mathcal{D}(A) \to \mathcal{D}(B)$:

i) $\Delta(\mu_1, \mu_2) \geq 0$, and $\Delta(\mu_1, \mu_2) = 0$ iff $\mu_1 = \mu_2$;

ii) $\Delta(\mu_1, \mu_2) = \Delta(\mu_2, \mu_1)$;

iii) $\Delta(\mu_1, \mu_3) \leq \Delta(\mu_1, \mu_2) + \Delta(\mu_2, \mu_3)$;

iv) $\Delta(f\ \mu_1, f\ \mu_2) \leq \Delta(\mu_1, \mu_2)$.

For discrete distributions, an equivalent definition is obtained if one lets f in (2) range over the real interval $[0,1]$ rather than over Boolean values. This characterization of statistical distance is more convenient for reasoning about our monadic formalization of distributions.

Lemma 1. *For any pair of discrete distributions μ_1 and μ_2 over a set A,*

$$\Delta(\mu_1, \mu_2) = \sup_{f:A\to[0,1]} |\mu_1\ f - \mu_2\ f|$$

Proof. Inequality $\Delta(\mu_1,\mu_2) \leq \sup_{f:A\to[0,1]} |\mu_1 f - \mu_2 f|$ is trivial. For the proof of the reverse inequality, observe that distributions μ_1 and μ_2 partition A into two sets $A_o = \{a \in A \mid \mu_1(a) \geq \mu_2(a)\}$ and $A_1 = \{a \in A \mid \mu_1(a) < \mu_2(a)\}$.

We claim that for any pair of functions $g, g' : A \to [0,1]$ s.t. $g \geq g'$ and $g = g' = 0$ in A_1, we have $|\mu_1\ g - \mu_2\ g| \geq |\mu_1\ g' - \mu_2\ g'|$. Indeed, as $g \geq g'$, there exists a non-negative h s.t. $g = g' + h$. Moreover $h = 0$ in A_1. Hence:

$$\begin{array}{rcl} |\mu_1\ g - \mu_2\ g| & = & |\mu_1\ g' - \mu_2\ g' + \mu_1\ h - \mu_2\ h| \\ & = & |\mu_1\ g' - \mu_2\ g'| + |\mu_1\ h - \mu_2\ h| \\ & \geq & |\mu_1\ g' - \mu_2\ g'| \end{array}$$

The second equality holds because as g' and h are null in A_1, we have $\mu_1\ g' \geq \mu_2\ g'$ and $\mu_1\ h \geq \mu_2\ h$. This concludes the proof of the claim.

To prove the lemma, we define for every set X and $[0,1]$-valued function f, the function $f_X(x) \stackrel{\text{def}}{=} f(x)\ \mathbb{1}_X(x)$. Since $A = A_0 \uplus A_1$, we have that $f = f_{A_0} + f_{A_1}$ and hence:

$$\begin{array}{rl} |\mu_1\ f - \mu_2\ f| = & |(\mu_1\ f_{A_0} - \mu_2\ f_{A_0}) - (\mu_2\ f_{A_1} - \mu_1\ f_{A_1})| \\ & \leq \max\{|\mu_1\ f_{A_0} - \mu_2\ f_{A_0}|, |\mu_2\ f_{A_1} - \mu_1\ f_{A_1}|\} \\ & \leq \max\{|\mu_1\ \mathbb{1}_{A_0} - \mu_2\ \mathbb{1}_{A_0}|, |\mu_2\ \mathbb{1}_{A_1} - \mu_1\ \mathbb{1}_{A_1}|\} \\ & \leq \Delta(\mu_1, \mu_2) \end{array}$$

The first inequality holds because for non-negative a, b, one has $|a-b| \leq \max(|a|,|b|)$, whereas the second inequality holds by the above claim (and its dual). □

3.1 A Logic for Bounding Statistical Distance

In this section, we consider the problem of bounding the statistical distance between the distributions output by two games given the same initial memory. Formally, let G_1 and G_2 be two games, which we assume to be executed in two fixed environments E_1 and E_2, and let $\Delta_m(\mathsf{G}_1, \mathsf{G}_2)$ denote $\Delta([\![\mathsf{G}_1]\!]\ m, [\![\mathsf{G}_2]\!]\ m)$. We define a logic that allows upper-bounding $\Delta_m(\mathsf{G}_1, \mathsf{G}_2)$ by a function of the memory m. We consider judgments of the form $(\!|\mathsf{G}_1, \mathsf{G}_2|\!) \preceq g$; such a judgment is valid iff

$$\forall m.\ \Delta_m(\mathsf{G}_1, \mathsf{G}_2) \leq g(m)$$

Figure 4 presents the main rules of the logic; contrary to the logic of Barthe et al. [8], this logic is not restricted to constant functions g. The logic deals with programs that

$$\frac{}{(\!|\mathsf{skip},\mathsf{skip}|\!) \preceq \lambda m.\ 0} \qquad \frac{(\!|c_1,c_2|\!) \preceq g \qquad (\!|c_1',c_2'|\!) \preceq g'}{(\!|c_1;c_1',c_2;c_2'|\!) \preceq \lambda m.\ [\![c_1]\!]\ m\ g' + g(m)}$$

$$\frac{}{(\!|x \leftarrow e, x \leftarrow e|\!) \preceq \lambda m.\ 0} \qquad \frac{\forall m.\ \Delta\left([\![d_1]\!]\ m, [\![d_2]\!]\ m\right) \leq g(m)}{(\!|x \xleftarrow{\$} d_1, x \xleftarrow{\$} d_2|\!) \preceq g}$$

$$\frac{(\!|c_1,c_1'|\!) \preceq g_1 \qquad (\!|c_2,c_2'|\!) \preceq g_2}{(\!|\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, \mathsf{if}\ b\ \mathsf{then}\ c_1'\ \mathsf{else}\ c_2'|\!) \preceq \lambda m.\ \mathbf{if}\ [\![b]\!]\ m\ \mathbf{then}\ g_1(m)\ \mathbf{else}\ g_2(m)}$$

$$\frac{(\!|c_1,c_2|\!) \preceq g \quad g_0(m) = 0 \quad g_{n+1}(m) = \mathbf{if}\ [\![b]\!]\ m\ \mathbf{then}\ [\![c_1]\!]\ m\ g_n + g(m)\ \mathbf{else}\ 0}{(\!|\mathsf{while}\ b\ \mathsf{do}\ c_1, \mathsf{while}\ b\ \mathsf{do}\ c_2|\!) \preceq \lambda m.\ \sup_{n\in\mathbb{N}}(g_n(m))}$$

$$\frac{(\!|E_1(p),E_2(p)|\!) \preceq g \quad g =_X g \quad \forall x.\ x \in X \Rightarrow \mathsf{global}(x)}{(\!|y \leftarrow p(x), y \leftarrow p(x)|\!) \preceq g}$$

Figure 4: Logic to bound the statistical distance between two probabilistic programs

are structurally similar, and as shown by Lemma 2, supports a rule for reasoning about adversaries.

To prove the soundness, for instance, of the rule for sequential composition, we introduce an intermediate program $c_1; c_2'$ (where c_1 is executed in environment E_1 and c_2' in environment E_2) and prove that the distance between $[\![c_1; c_1']\!]\ m$ and $[\![c_1; c_2']\!]\ m$ is bounded by $[\![c_1]\!]\ m\ g'$, while the distance between $[\![c_1; c_2']\!]\ m$ and $[\![c_2; c_2']\!]\ m$ is bounded by $g(m)$. The rule for loops relies on the characterization of the semantics of a while loop as the least upper bound of its n-th unrolling $[\mathsf{while}\ e\ \mathsf{do}\ c]_n$ (see Figure 2), and on the auxiliary rule

$$\frac{(\!|[\mathsf{while}\ b\ \mathsf{do}\ c_1]_n, [\mathsf{while}\ b\ \mathsf{do}\ c_2]_n|\!) \preceq g_n}{(\!|\mathsf{while}\ b\ \mathsf{do}\ c_1, \mathsf{while}\ b\ \mathsf{do}\ c_2|\!) \preceq \lambda m.\ \sup_{n\in\mathbb{N}}(g_n(m))}$$

The rule for procedure calls builds on the fact that the semantics of a call $y \leftarrow p(x)$ in memory m is basically given by an unfolding of p's code, where the resulting command is executed in a memory m' that only differs from m in the set of (local) variables that correspond to p's formal parameters; global variables have the same values in m and m'. The last two premises of the rule are thus required to guarantee that $g(m') = g(m)$.

While the rules in Figure 4 are sufficient to reason about closed programs, they do not allow to reason about games in the presence of adversaries. We enhance the logic with a rule that allows to draw conclusions of the form $(\!|\mathcal{A},\mathcal{A}|\!) \preceq g$, i.e. to bound the statistical distance between calls to an adversary $\mathcal{A}$ executed in two different environments E_1 and E_2.[1] Although the code of $\mathcal{A}$ is unknown, the only statements in its code that can increase statistical distance are calls to oracles (since these are the only instructions whose semantics may vary between the two environments E_1 and E_2). The

[1] For the sake of readability, we write $(\!|\mathcal{A},\mathcal{A}|\!) \preceq g$ instead of $(\!|x \leftarrow \mathcal{A}(\vec{e}), x \leftarrow \mathcal{A}(\vec{e})|\!) \preceq g$, and likewise for oracles.

rule we formalize captures this intuition, by providing an upper bound for the statistical distance between the final distributions in terms of the statistical distance induced by individual oracle calls, and the number of oracle calls made by the adversary. In its simplest formulation, the rule assumes that oracles are instrumented with a counter that keeps track of the number of queries made by the adversary, and that the statistical distance between the distributions induced by a call to an oracle $x \leftarrow \mathcal{O}(\vec{e})$ in E_1 and E_2 is upper-bounded by a constant ϵ, i.e. $(\!|\mathcal{O}, \mathcal{O}|\!) \preceq \lambda m.\ \epsilon$. In this case, the statistical distance between calls to the adversary $\mathcal{A}$ in E_1 and E_2 is upper-bounded by $q \cdot \epsilon$, where q is an upper bound on the number of oracle calls made.

For the application presented in Section 5, we need to formalize a more expressive rule, in which the statistical distance between two oracle calls can also depend on the program state. Moreover, we allow the counter to be any integer expression, and only require that it does not decrease across oracle calls.

Lemma 2 (Adversary rule)**.** *Let $\mathcal{A}$ be an adversary and let* cntr *be an integer expression whose variables are global and cannot be written by $\mathcal{A}$. Let $h : \mathbb{N} \to [0, 1]$ and define*

$$\bar{h}_{\mathsf{cntr}}(m, m') \stackrel{def}{=} \min\left(1, \sum_{i=[\![\mathsf{cntr}]\!]m}^{[\![\mathsf{cntr}]\!]m'-1} h(i)\right)$$

Assume that for every oracle $\mathcal{O}$,

$$(\!|\mathcal{O}, \mathcal{O}|\!) \preceq \lambda m.\ [\![E_1(\mathcal{O})]\!]\ m\ (\lambda m'.\ \bar{h}_{\mathsf{cntr}}(m, m'))$$

and moreover $\{\mathsf{cntr} = i\}\ E_1(\mathcal{O})\ \{i \leq \mathsf{cntr}\}$. Then,

$$(\!|\mathcal{A}, \mathcal{A}|\!) \preceq \lambda m.\ [\![E_1(\mathcal{A})]\!]\ m\ \left(\lambda m'.\ \bar{h}_{\mathsf{cntr}}(m, m')\right).$$

The first hypothesis states that the distance $\Delta_m(\mathcal{O}, \mathcal{O})$ can be bounded in terms of the value of cntr before and after executing $\mathcal{O}$; for instance, if a call to $\mathcal{O}$ always increments cntr by 2, then the distance $\Delta_m(\mathcal{O}, \mathcal{O})$ is bounded by $h([\![\mathsf{cntr}]\!]m) + h([\![\mathsf{cntr}]\!]m + 1)$. The second hypothesis captures the monotonicity property of the counter.

Proof. As in [5], the rule is derived from the induction principle induced by the definition of well-formed adversary using the rules in Figure 4. □

3.2 Reasoning about Failure Events

Transitions based on failure events allow to transform a game into another game that is semantically equivalent unless some *failure* condition is triggered. This kind of game transformations rely on the following lemma:

Lemma 3 (Fundamental Lemma)**.** *Consider two games G_1, G_2 and let A, B, and F be events. For every initial memory m, if*

$$\Pr[\mathsf{G}_1, m : A \wedge \neg F] = \Pr[\mathsf{G}_2, m : B \wedge \neg F],$$

then

$$|\Pr[\mathsf{G}_1, m : A] - \Pr[\mathsf{G}_2, m : B]| \leq \max\{\Pr[\mathsf{G}_1, m : F], \Pr[\mathsf{G}_2, m : F]\}$$

Note that if, for instance, game G_2 *is lossless, then* $\Pr[\mathsf{G}_1, m : F] \leq \Pr[\mathsf{G}_2, m : F]$.

When $A = B$ and $F = \mathbf{bad}$ for some Boolean variable $\mathbf{bad}$, the hypothesis of the lemma can be automatically established by inspecting the code of both games: it holds if their code differs only after program points setting $\mathbf{bad}$ to true and $\mathbf{bad}$ is never reset to false.

As a corollary, and in view of of Lemma 1, one can establish that if two programs G_1 and G_2 satisfy the above conditions and, for instance, G_2 is lossless, then

$$(\!|\mathsf{G}_1, \mathsf{G}_2|\!) \preceq \lambda m.\ \Pr[\mathsf{G}_2, m : \mathbf{bad}]$$

4 Weak Equivalences

In this section we introduce quantitative notions of program equivalence and equational theories to reason about them.

4.1 Approximate Observational Equivalence

Approximate observational equivalence generalizes observational equivalence between two games by allowing that their output distributions differ up to some quantity ϵ. Informally, two games G_1 and G_2 are ϵ-observationally equivalent w.r.t. an input set of variables I and an output set of variables O iff for every pair of memories m_1, m_2 coinciding on I, the statistical distance between the quotient distributions $([\![\mathsf{G}_1]\!]\ m_1)/=_O$ and $([\![\mathsf{G}_2]\!]m_2)/=_O$ over $\mathcal{M}/=_O$ is upper-bounded by ϵ. For the purpose of formalization, it is more convenient to rely on the following alternative characterization that does not use quotient distributions.

Definition 2. *Two games* G_1 *and* G_2 *are* ϵ*-observationally equivalent w.r.t. an input set of variables* I *and an output set of variables* O*, written* $\models \mathsf{G}_1 \simeq_O^I \mathsf{G}_2 \preceq \epsilon$*, iff for all memories* $m_1, m_2 \in \mathcal{M}$ *and functions* $f_1, f_2 : \mathcal{M} \to [0, 1]$

$$m_1 =_I m_2 \wedge f_1 =_O f_2 \implies |[\![\mathsf{G}_1]\!]\ m_1\ f_1 - [\![\mathsf{G}_2]\!]\ m_2\ f_2| \leq \epsilon$$

Figure 5 provides an excerpt of an equational theory for approximate observational equivalence; further and more general rules appear in the formal development. Most rules generalize observational equivalence in the expected way. For instance, the rule for random assignments considers the case of two distribution expressions ϵ-away from each other. Let μ_1 and μ_2 be their interpretation. In case $\mu_1 = \mu_2$, one obtains $\epsilon = 0$. Furthermore, if μ_1 and μ_2 are uniform distributions over subsets $A_1, A_2 \subseteq T$ of some underlying type, one has

$$\Delta(\mu_1, \mu_2) = \max\left\{\frac{\#(A_1 \setminus A_2)}{\#A_1}, \frac{\#(A_2 \setminus A_1)}{\#A_2}\right\}.$$

$$\frac{\models c_1 \simeq_O^I c_2 \preceq \epsilon_1 \quad \models c_2 \simeq_O^I c_3 \preceq \epsilon_2}{\models c_1 \simeq_O^I c_3 \preceq \epsilon_1 + \epsilon_2} \qquad \frac{\models c_1 \simeq_{O'}^I c_2 \preceq \epsilon_1 \quad \models c_1' \simeq_O^{O'} c_2' \preceq \epsilon_2}{\models c_1; c_1' \simeq_O^I c_2; c_2' \preceq \epsilon_1 + \epsilon_2}$$

$$\frac{\models c_1 \simeq_{O'}^{I'} c_2 \preceq \epsilon' \quad I' \subseteq I \quad O \subseteq O' \quad \epsilon' \leq \epsilon}{\models c_1 \simeq_O^I c_2 \preceq \epsilon}$$

$$\frac{\models c_1 \simeq_O^I c_1' \preceq \epsilon \quad \models c_2 \simeq_O^I c_2' \preceq \epsilon \quad \forall m_1, m_2.\ m_1 =_I m_2 \implies [\![b]\!]\, m_1 = [\![b']\!]\, m_2}{\models \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \simeq_O^I \mathsf{if}\ b'\ \mathsf{then}\ c_1'\ \mathsf{else}\ c_2' \preceq \epsilon}$$

$$\frac{\forall m_1, m_2.\ m_1 =_I m_2 \implies \Delta\left([\![d_1]\!]\, m_1, [\![d_2]\!]\, m_2\right) \leq \epsilon}{\models x \xleftarrow{\$} d_1 \simeq_{I \cup \{x\}}^I x \xleftarrow{\$} d_2 \preceq \epsilon}$$

Figure 5: Selected rules for reasoning about approximate observational equivalence

As an illustrative example we sketch a proof of the soundness of the rule for random assignments.

Consider $m_1, m_2 \in \mathcal{M}$ and $f_1, f_2 : \mathcal{M} \to [0,1]$ such that $m_1 =_I m_2$ and $f_1 =_{I \cup \{x\}} f_2$. Define distributions $\mu_1 = [\![d_1]\!]\, m_1$ and $\mu_2 = [\![d_2]\!]\, m_2$ and let $S_1 = \mathsf{supp}\,(\mu_1)$, $S_2 = \mathsf{supp}\,(\mu_2)$, $A_0 = \{a \in S_1 \cap S_2 \mid \mu_1(a) \geq \mu_2(a)\} \cup (S_1 \setminus S_2)$ and $A_1 = \{a \in S_1 \cap S_2 \mid \mu_1(a) < \mu_2(a)\} \cup (S_2 \setminus S_1)$. We have

$$\begin{aligned}
&|[\![x \xleftarrow{\$} d_1]\!]\, m_1\, f_1 - [\![x \xleftarrow{\$} d_2]\!]\, m_2\, f_2| \\
&\quad \overset{(1)}{=} \left| \sum_{a \in S_1} \mu_1(a) f_1(m_1\,\{a/x\}) - \sum_{a \in S_2} \mu_2(a) f_2(m_2\,\{a/x\}) \right| \\
&\quad = \left| \sum_{a \in S_1 \cap S_2} \mu_1(a) f_1(m_1\,\{a/x\}) - \mu_2(a) f_2(m_2\,\{a/x\}) + \right. \\
&\qquad \left. \sum_{a \in S_1 \setminus S_2} \mu_1(a) f_1(m_1\,\{a/x\}) - \sum_{a \in S_2 \setminus S_1} \mu_2(a) f_2(m_2\,\{a/x\}) \right| \\
&\quad \overset{(2)}{=} \left| \sum_{a \in A_0} (\mu_1(a) - \mu_2(a)) f_1(m_1\,\{a/x\}) - \sum_{a \in A_1} (\mu_2(a) - \mu_1(a)) f_2(m_2\,\{a/x\}) \right| \\
&\quad \overset{(3)}{\leq} \max \left\{ \sum_{a \in A_0} \mu_1(a) - \mu_2(a), \sum_{a \in A_1} \mu_2(a) - \mu_1(a) \right\} \\
&\quad \leq \Delta\left(\mu_1, \mu_2\right)
\end{aligned}$$

Equation (1) follows from unfolding the semantics of random assignments; equality (2) follows from the definitions of A_0, A_1, S_1 and S_2, and the fact that $f_1(m_1\,\{a/x\}) = f_2(m_2\,\{a/x\})$ for all $a \in S_1 \cap S_2$. Finally inequality (3) holds since for all reals x, y satisfying $0 \leq x, y \leq \epsilon$ one has $|x - y| \leq \epsilon$ and functions f_1 and f_2 take values in the interval $[0,1]$.

4.2 A Conditional Variant

The application we describe in Section 5 requires reasoning about conditional approximate observational equivalence, a generalization of approximate observational equivalence. We define for a distribution μ and event P the conditional distribution $\mu \downarrow_P$ as

$$\mu \downarrow_P \ \stackrel{\text{def}}{=}\ \lambda f.\, \mu \left(\lambda a.\, \frac{f(a)\ \mathbb{1}_P(a)}{\mu\ \mathbb{1}_P} \right)$$

Intuitively, $\mu \downarrow_P \ \mathbb{1}_Q$ yields the conditional probability of Q given P.

Definition 3. *A game G_1 conditioned on predicate P_1 is ϵ-observationally equivalent to a game G_2 conditioned on P_2 w.r.t. an input set of variables I and an output set of variables O, written $\models [\mathsf{G}_1]_{P_1} \simeq^I_O [\mathsf{G}_2]_{P_2} \preceq \epsilon$, iff for any $m_1, m_2 \in \mathcal{M}$ and $f_1, f_2 : \mathcal{M} \to [0,1]$,*

$$m_1 =_I m_2 \ \wedge\ f_1 =_O f_2 \implies |([\![\mathsf{G}_1]\!]\, m_1) \downarrow_{P_1}\ f_1 - ([\![\mathsf{G}_2]\!]\, m_2) \downarrow_{P_2}\ f_2| \leq \epsilon$$

Conditional approximate observational equivalence subsumes classic approximate observational equivalence, which can be recovered by taking $P_1 = P_2 = \mathsf{true}$.

4.3 Generalization to Arbitrary Relations

In order to formalize the result of Brier et al. [13], it suffices to reason about approximate observational equivalence. However we can obtain a full-fledged relational Hoare logic for approximate reasoning by considering as pre- and post-conditions arbitrary relations instead of just equality over a set of program variables. This can be achieved by considering an approximate notion of lifting, which generalizes the notion of lifting defined in Section 2.3. For any relation $R \subseteq A \times B$ and any $\epsilon \in [0,1]$, the *ϵ-approximate lifting* of R, noted $\mathcal{L}^\epsilon(R)$, is defined as

$$\mu_1\, \mathcal{L}^\epsilon(R)\, \mu_2 \ \stackrel{\text{def}}{=}\ \exists \mu : \mathcal{D}(A \times B).\ \begin{cases} \mu(a,b) > 0 \implies a\ R\ b \quad \text{for all } (a,b) \in A \times B \\ \Delta\,(\pi_1 \mu, \mu_1) \leq \epsilon \wedge \Delta\,(\pi_2 \mu, \mu_2) \leq \epsilon \\ \pi_1 \mu \leq \mu_1 \wedge \pi_2 \mu \leq \mu_2 \end{cases}$$

where $\leq$ is the pointwise partial order on $\mathcal{D}(A)$.

Judgments in the proposed logic are of the form

$$\models \mathsf{G}_1 \simeq^\epsilon \mathsf{G}_2\ :\ \Psi \Longrightarrow \Phi.$$

where G_1 and G_2 are games, Ψ and Φ are binary relations on program memories, and $\epsilon \in [0,1]$. We say that such a judgment is valid iff for all memories $m_1, m_2 \in \mathcal{M}$ one has:

$$m_1\ \Psi\ m_2 \implies ([\![G_1]\!]\, m_1)\, \mathcal{L}^\epsilon(\Phi)\, ([\![G_2]\!]\, m_2).$$

This logic generalizes simultaneously approximate observational equivalence and the probabilistic Relational Hoare Logic of CertiCrypt. Most of the rules of Figure 3 admit a direct transposition. For instance, the rule for sequential composition reads:

$$\frac{\models c_1 \simeq^{\epsilon_1} c_2\ :\ \Psi \Longrightarrow \Theta \qquad \models c_1' \simeq^{\epsilon_2} c_2'\ :\ \Theta \Longrightarrow \Phi}{\models c_1; c_1' \simeq^{\epsilon_1+\epsilon_2} c_2; c_2'\ :\ \Psi \Longrightarrow \Phi}$$

The logic also allows inferring claims about probability quantities:

$$\frac{m_1 \ \Psi \ m_2 \qquad \models \mathsf{G}_1 \simeq^{\epsilon} \mathsf{G}_2 \ : \ \Psi \Longrightarrow \Phi \qquad \Phi \implies (A\langle 1\rangle \Longleftrightarrow B\langle 2\rangle)}{|\Pr\left[\mathsf{G}_1, m_1 : A\right] - \Pr\left[\mathsf{G}_2, m_2 : B\right]| \leq \epsilon}$$

The logic can be further generalized by considering a weaker notion of distance between distributions instead of statistical distance. For instance Barthe et al. [8, 9] define for $\alpha \geq 1$, the α-distance between μ_1 and μ_2 as

$$\Delta_{\alpha}(\mu_1, \mu_2) \stackrel{\text{def}}{=} \max_{f:A\to[0,1]} \{\mu_1 f - \alpha\ \mu_2 f, \mu_2 f - \alpha\ \mu_1 f, 0\}.$$

This generalization allows reasoning about differential privacy [20].

5 Construction of Indifferentiable Hash Functions

In this section we apply the techniques introduced above to prove the security of cryptographic constructions in the indifferentiability framework of Maurer et al. [33]. In particular, we consider the notion of indifferentiability from a random oracle. A random oracle is an ideal primitive that maps elements in some domain into uniformly and independently distributed values in a finite set; queries are answered consistently so that identical queries are given the same answer. A proof conducted in the random oracle model for a function $\mathcal{H}$ assumes that $\mathcal{H}$ is a random oracle and makes it publicly available to all parties. In games, we represent random oracles as stateful procedures.

Definition 4 (Indifferentiability)**.** *A construction $\mathcal{H}$ built from a primitive $\mathcal{G}$ is said to be $(t_{\mathcal{S}}, t_{\mathcal{D}}, q_1, q_2, \epsilon)$-indifferentiable from an ideal primitive $\mathcal{F}$ if there exists a simulator $\mathcal{S}$ with oracle access to $\mathcal{F}$ such that any distinguisher $\mathcal{D}$ running within time $t_{\mathcal{D}}$ has at most probability ϵ of distinguishing a scenario where it is given $\mathcal{H}$ and $\mathcal{G}$ as oracles from a scenario where it is given $\mathcal{F}$ and $\mathcal{S}$ instead:*

$$\left|\Pr\left[b \leftarrow \mathcal{D}^{\mathcal{H}^{\mathcal{G}},\mathcal{G}}() : b = \mathsf{true}\right] - \Pr\left[b \leftarrow \mathcal{D}^{\mathcal{F},\mathcal{S}^{\mathcal{F}}}() : b = \mathsf{true}\right]\right| \leq \epsilon$$

The distinguisher $\mathcal{D}$ is allowed to make at most q_1 queries to $\mathcal{H}$ (resp. $\mathcal{F}$) and at most q_2 queries to $\mathcal{G}$ (resp. $\mathcal{S}$).

Intuitively, $\mathcal{S}$ must simulate the primitive $\mathcal{G}$ so that no distinguisher can tell whether it is interacting with $\mathcal{H}^{\mathcal{G}}$ and $\mathcal{G}$ or with $\mathcal{F}$ and $\mathcal{S}^{\mathcal{F}}$ (see Figure 6). The simulator must do so without access to the internal state (if any) of $\mathcal{F}$ or to its interaction with the distinguisher.

Random oracles into elliptic curves over finite fields are typically built from a random oracle $\mathcal{G}$ on the underlying field and a deterministic encoding f that maps elements of the field into the elliptic curve. Examples of such encodings include Icart's function [30] and the Shallue-Woestijne-Ulas (SWU) algorithm [39]. In general (and for the aforementioned mappings) the function f is not surjective and only covers a fraction of points in the curve. Hence, the naive definition of a hash function H as $f \circ \mathcal{G}$ would not cover the whole curve, contradicting the assumption that H behaves as a

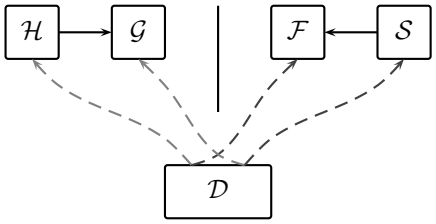

Figure 6: The two scenarios in the definition of indifferentiability of construction $\mathcal{H}$ from an ideal primitive $\mathcal{F}$

random oracle. In a recent paper, Brier et al. [13] show how to build hash functions into elliptic curves that are indifferentiable from a random oracle for a particular class of encodings, including both SWU and Icart's encodings.

Brier et al. [13] prove that if $(\mathbb{G}, \otimes)$ is a finite cyclic group of order N with generator g, a function into $\mathbb{G}$ indifferentiable from a random oracle can be built from any polynomially invertible function $f : A \to \mathbb{G}$ and hash functions $\mathcal{G}_1 : \{0,1\}^* \to A$ and $\mathcal{G}_2 : \{0,1\}^* \to \mathbb{Z}_N$ that behave as random oracles as follows:

$$\mathcal{H}(m) \stackrel{\text{def}}{=} f(\mathcal{G}_1(m)) \otimes g^{\mathcal{G}_2(m)} \tag{3}$$

Intuitively, the term $g^{\mathcal{G}_2(m)}$ behaves as a one-time pad and ensures that $\mathcal{H}$ covers all points in the group even if f covers only a fraction. This construction can be seen as the composition of the function $F(a, p) = f(a) \otimes g^p$ and a random oracle into $A \times \mathbb{Z}_N$.

We prove in CertiCrypt the indifferentiability of a generalization of Brier et al.'s construction to finitely generated abelian groups. The proof introduces two intermediate constructions and is structured in three steps:

1. We first prove that any efficiently invertible encoding f can be turned into a *weak encoding* (Theorem 1);
2. We then show an efficient construction to transform any weak encoding f into an *admissible encoding* (Theorem 2);
3. Finally, we prove that any admissible encoding can be turned into a hash function indifferentiable from a random oracle (Theorem 3).

Moreover, we show in Sect. 6 that Icart's encoding is efficiently invertible and thus yields a hash function indifferentiable from a random oracle when plugged in into the above construction.

We recall the alternative definitions of weak and admissible encoding from [31]. Note that these do not match the definitions of [13], but, in comparison, are better behaved: e.g. admissible encodings as we define them are closed under functional composition and Cartesian product.

Definition 5 (Weak encoding)**.** *A function $f : S \to R$ is an (α, ϵ)-weak encoding if it is computable in polynomial-time and there exists a probabilistic polynomial-time algorithm $\mathcal{I}_f : R \to S_\perp$ such that*

i) $\{\mathsf{true}\}\ r \xleftarrow{\$} R;\ s \leftarrow \mathcal{I}_f(r)\ \{s = \bot \vee f(s) = r\}$

ii) $\models [r \xleftarrow{\$} R;\ s \leftarrow \mathcal{I}_f(r)]_{s \neq \bot} \simeq^{\emptyset}_{\{s\}} [s \xleftarrow{\$} S] \preceq \epsilon$

iii) $\Pr[r \xleftarrow{\$} R;\ s \leftarrow \mathcal{I}_f(r) : s = \bot] \leq 1 - \alpha^{-1}$

Condition i) states that $\mathcal{I}_f$ either inverts f or fails; ii) states that given a random input, $\mathcal{I}_f$ returns a pre-image chosen almost uniformly when it does not fail; while iii) states that $\mathcal{I}_f$ does not fail too often.

Definition 6 (Admissible encoding)**.** *A function $f : S \to R$ is an ϵ-admissible encoding if it is computable in polynomial-time and there exists a probabilistic polynomial-time algorithm $\mathcal{I}_f : R \to S_\bot$ such that*

i) $\{\mathsf{true}\}\ r \xleftarrow{\$} R;\ s \leftarrow \mathcal{I}_f(r)\ \{s = \bot \vee f(s) = r\}$

ii) $\models r \xleftarrow{\$} R;\ s \leftarrow \mathcal{I}_f(r) \simeq^{\emptyset}_{\{s\}} s \xleftarrow{\$} S \preceq \epsilon$

Compared to a weak encoding, an admissible encoding may seem to impose no explicit bound on the probability of $\mathcal{I}_f$ failing. However, condition ii) requires the output of inverter $\mathcal{I}_f$ on a random input be statistically indistinguishable from the uniform distribution on S; the bound ϵ must account for the probability of $\mathcal{I}_f$ failing.

We begin our proof by showing that any efficiently invertible encoding is a weak encoding.

Theorem 1. *Let $f : S \to R$ be a function computable in polynomial-time such that for any $r \in R$, $\#f^{-1}(r) \leq B$. Assume there exists a polynomial-time algorithm $\mathcal{I}$ that given $r \in R$ outputs the set $f^{-1}(r)$. Then, f is an $(\alpha, 0)$-weak encoding, with $\alpha = B\ \#R/\#S$.*

Proof. Using $\mathcal{I}$, we build a partial inverter $\mathcal{I}_f : R \to S_\bot$ of f that satisfies the properties in Definition 5:

$$\begin{array}{ll} \mathcal{I}_f(r): & X \leftarrow \mathcal{I}(r);\ b \xleftarrow{\$} \mathsf{true} \oplus_{\#X/B} \mathsf{false}; \\ & \mathsf{if}\ b = \mathsf{true}\ \mathsf{then}\ s \xleftarrow{\$} X;\ \mathsf{return}\ s\ \mathsf{else}\ \mathsf{return}\ \bot \end{array}$$

First observe that $\mathcal{I}_f(r)$ fails with probability $1 - \#f^{-1}(r)/B$ or else returns an element uniformly chosen from the set of pre-images of r, and thus satisfies the first property trivially. In addition we have

$$\Pr[r \xleftarrow{\$} R;\ s \leftarrow \mathcal{I}_f(r) : s \neq \bot] = \sum_{r \in R} \frac{1}{\#R} \frac{\#f^{-1}(r)}{B} = \frac{\#S}{B\#R}$$

and for any $x \in S$,

$$\Pr[r \xleftarrow{\$} R;\ s \leftarrow \mathcal{I}_f(r) : s = x] = \sum_{r \in R} \frac{1}{\#R} \frac{\#f^{-1}(r)}{B} \frac{\mathbb{1}_{f^{-1}(r)}(x)}{\#f^{-1}(r)} = \frac{1}{B\#R}$$

Hence, for a uniformly chosen r, the probability of $\mathcal{I}_f(r)$ failing is exactly $1 - \alpha^{-1}$, and the probability of returning any particular value in S conditioned to not failing is uniform. □

We show next how to construct an admissible encoding $F : A \times P \to Q$ from a weak encoding $f : A \to Q$ for appropriate sets P and Q. This construction generalizes Brier et al.'s original result (see [13, Theorem 3]) that restricts the analysis to the case where Q is a cyclic group of order N and P is $\mathbb{Z}_N$.

The proof that we present relies on the application of two padding lemmas involving a pair of expressions of type P and Q. To capture these properties we introduce a novel algebraic structure coined *padding algebra*.

Definition 7 (Padding algebra)**.** *Let P and Q be two finite sets equipped with binary operations $\circledast, \odot : P \times Q \to Q$ and $\oslash : Q \times Q \to P$. We say that $(P, Q, \circledast, \odot, \oslash)$ is a* padding algebra *iff:*

i) $(p \circledast q) \oslash q = p \quad$ *for all* $p \in P, q \in Q$*;*

ii) $q \oslash (p \odot q) = p \quad$ *for all* $p \in P, q \in Q$*;*

iii) for all $q \in Q$, the function $\lambda p.\ p \circledast q$ is an isomorphism between P and Q; and

iv) for all $q \in Q$, the function $\lambda p.\ p \odot q$ is an isomorphism between P and Q.

For such a structure one can prove the following algebraic equivalences:

$$\models p \xleftarrow{\$} P;\ q_2 \leftarrow p \circledast q_1 \simeq^{\{q_1\}}_{\{q_1,q_2,p\}} q_2 \xleftarrow{\$} Q;\ p \leftarrow q_2 \oslash q_1 \tag{4}$$

$$\models q_2 \xleftarrow{\$} Q;\ p \leftarrow q_1 \oslash q_2 \simeq^{\{q_1\}}_{\{q_1,q_2,p\}} p \xleftarrow{\$} P;\ q_2 \leftarrow p \odot q_1 \tag{5}$$

To do so we need to rely on the rules for random assignments and sequential composition Rnd and Seq of CertiCrypt's logic (see Figure 3) and on the above set of axioms—axioms i) and iii) for the former equivalence and axioms ii) and iv) for the latter. In Section 6 we show that every finite abelian group induces a padding algebra and use this fact to instantiate the results presented in this section.

We now show how to turn a weak encoding $f : A \to Q$ into an admissible encoding $F : A \times P \to Q$ when P and Q can be given the structure of a padding algebra.

Theorem 2. *Let $(P, Q, \circledast, \odot, \oslash)$ be a padding algebra such that*

$$(q_1 \oslash q_2) \odot q_1 = q_2 \qquad \forall q_1, q_2 \in Q,$$

and operations $\circledast$ and $\odot$ can be computed in polynomial-time. Then, for any (α, ϵ)-weak encoding $f : A \to Q$, the function

$$\begin{array}{lcl} F & : & A \times P \to Q \\ F(a, p) & \stackrel{def}{=} & p \odot f(a) \end{array}$$

is an ϵ'-admissible encoding into Q, with $\epsilon' = \epsilon + \left(1 - \alpha^{-1}\right)^{T+1}$ for any value T polynomial in the security parameter.

Proof. Since f is a weak encoding, there exists a polynomial-time computable inverter $\mathcal{I}_f$ of f satisfying the conditions in Definition 5. Let T be polynomial in the security

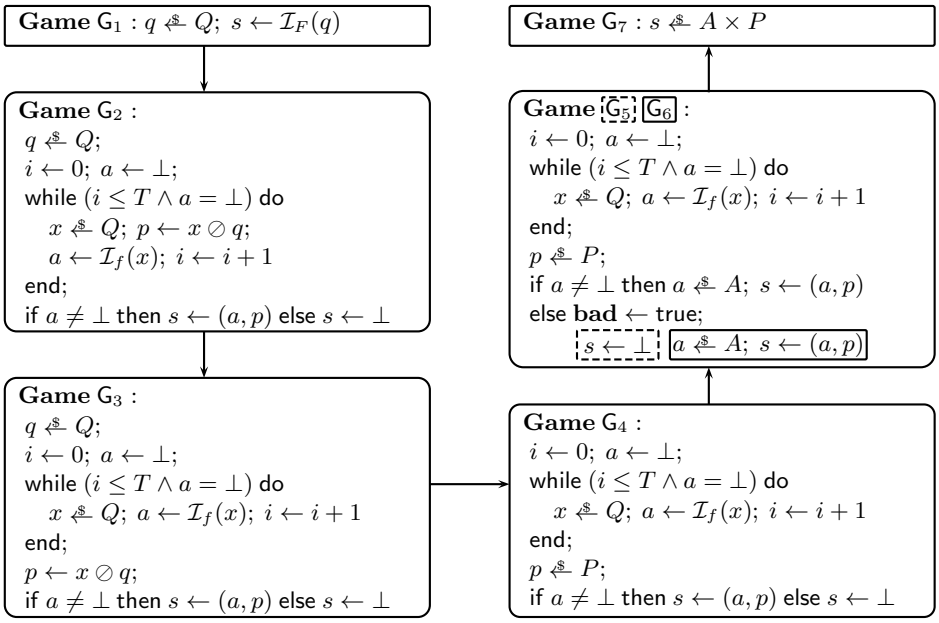

Figure 7: Sequence of games used in Theorem 2

parameter. Using $\mathcal{I}_f$, we build a partial inverter $\mathcal{I}_F$ of F that satisfies the properties in Definition 6:

$$
\begin{array}{ll}
\mathcal{I}_F(q): & i \leftarrow 0;\ a \leftarrow \bot; \\
& \mathsf{while}\ (i \leq T \wedge a = \bot)\ \mathsf{do} \\
& \quad p \xleftarrow{\$} P; \\
& \quad x \leftarrow p \circledast q; \\
& \quad a \leftarrow \mathcal{I}_f(x); \\
& \quad i \leftarrow i + 1 \\
& \mathsf{end}; \\
& \mathsf{if}\ a \neq \bot\ \mathsf{then\ return}\ (a, p)\ \mathsf{else\ return}\ \bot
\end{array}
$$

The partial inverter $\mathcal{I}_F$ runs in time $t_{\mathcal{I}_F} = (T + 1)\ t_{\mathcal{I}_f}$, where $t_{\mathcal{I}_f}$ is a bound on the running time of $\mathcal{I}_f$. Hence, $\mathcal{I}_F$ is polynomial-time.

We prove that

$$
\models q \xleftarrow{\$} Q;\ s \leftarrow \mathcal{I}_F(q) \simeq^{\emptyset}_{\{s\}} s \xleftarrow{\$} A \times P \preceq \epsilon' \tag{6}
$$

using the sequence of games $\mathsf{G}_1, \ldots, \mathsf{G}_7$ shown in Figure 7, the mechanized program transformations of CertiCrypt, and the proof rules for observational and approximate observational equivalence. We briefly describe the proof below.

We obtain game G_2 by first inlining the call to $\mathcal{I}_F$ in the initial game and then applying the algebraic equivalence (4) to transform the body of the while loop.

Game G_3 is obtained by moving the assignment to p outside the loop in game G_2. This transformation is semantics-preserving because p is never used inside the loop

and the value that it has when exiting the loop only depends on the value of x in the last iteration. Formally, this is proven by unfolding the first iteration of the loop and establishing that the relation

$$=_{\{i,x,a,q\}} \wedge (p = x \oslash q)\langle 1 \rangle$$

is a relational invariant between the loop in G_2 and the loop resulting from removing the assignment to p. By appending $p \leftarrow x \oslash q$ to the latter loop, we recover equivalence on p.

Observe that games G_2 and G_3 use operation $\oslash$, which might not be computable in polynomial-time (for elliptic curve groups, it would require computing a discrete logarithm). This is a valid proof technique that does not undermine the validity of the analysis; we prove all necessary equivalences.

Since q is no longer used inside the loop, we can postpone choosing it after the loop and use the algebraic equivalence (5) to sample p instead of q. We obtain G_4 by additionally removing the assignment to q, which is now dead code.

For the next step in the proof we use the fact that f is a weak encoding and therefore the distribution of a after a call $a \leftarrow \mathcal{I}_f(x)$ conditioned to $a \neq \perp$ is ϵ-away from the uniform distribution. This allows us to re-sample the value of a after the loop, provided $a \neq \perp$, incurring a penalty ϵ on the statistical distance of the distribution of s between G_4 and G_5. To prove this formally, let b be the condition of the loop and c its body. Observe that the semantics of the loop coincides with the semantics of its $(T+1)$-unrolling $[\mathsf{while}\ b\ \mathsf{do}\ c]_{T+1}$. We show by induction on T that for any $[0,1]$-valued functions f, g s.t. $f =_{\{a'\}} g$,

$$m_1 =_{\{a,i\}} m_2 \ \wedge\ m_1(a) = \perp \implies |[\![c_1]\!]\, m_1\, f' - [\![c_2]\!]\, m_2\, g'| \leq \epsilon$$

where

$$\begin{array}{lcl} c_1 & = & [\mathsf{while}\ b\ \mathsf{do}\ c]_{T+1};\ \mathsf{if}\ a \neq \perp\ \mathsf{then}\ a' \leftarrow a \\ c_2 & = & [\mathsf{while}\ b\ \mathsf{do}\ c]_{T+1};\ \mathsf{if}\ a \neq \perp\ \mathsf{then}\ a' \xleftarrow{\$} A \\ f'(m) & = & \mathbf{if}\ m(a) \neq \perp\ \mathbf{then}\ f(m)\ \mathbf{else}\ 0 \\ g'(m) & = & \mathbf{if}\ m(a) \neq \perp\ \mathbf{then}\ g(m)\ \mathbf{else}\ 0 \end{array}$$

and use this to conclude the ϵ-approximate equivalence of G_4 and G_5.

Since G_5 and G_6 are syntactically equivalent except for code appearing after the flag $\mathbf{bad}$ is set, we apply the corollary of the Fundamental Lemma in Section 3.2 to obtain the bound

$$(\!|\mathsf{G}_5, \mathsf{G}_6|\!) \preceq \Pr[\mathsf{G}_5 : \mathbf{bad}]$$

Since the probability of failure of $\mathcal{I}_f$ on a uniformly chosen input is upper-bounded by $1 - \alpha^{-1}$, we can show by induction on T that

$$\Pr[\mathsf{G}_5 : \mathbf{bad}] \leq \left(1 - \alpha^{-1}\right)^{T+1},$$

from which we conclude $\models \mathsf{G}_5 \simeq^{\emptyset}_{\{s\}} \mathsf{G}_6 \preceq \left(1 - \alpha^{-1}\right)^{T+1}$.

By coalescing the branches in the conditional at the end of G_6 and removing dead code, we prove that the game is observational equivalent w.r.t. a and p to the game $a \xleftarrow{\$} A;\ p \xleftarrow{\$} P;\ s \leftarrow (a, p)$, which is trivially equivalent to G_7.

By composing the above results, we conclude

$$\models \mathsf{G}_1 \simeq^{\emptyset}_{\{s\}} \mathsf{G}_7 \preceq \epsilon + \left(1 - \alpha^{-1}\right)^{T+1} \tag{7}$$

We must also show that $s = \bot \vee F(s) = q$ is a post-condition of G_1. As G_1 and G_3 are observationally equivalent with respect to s and q, it is sufficient to establish the validity of the post-condition for G_3. We show that $a \neq \bot \Rightarrow x = f(a)$ is an invariant of the loop. When the loop finishes, either $a = \bot$ and in this case $s = \bot$, or $a \neq \bot$ and we have $F(s) = p \odot f(a) = (x \oslash q) \odot x = q$. □

Finally, we show that the composition of an admissible encoding $f : S \to R$ and a random oracle into S is indifferentiable from a random oracle into R.

Theorem 3. *Let $f : S \to R$ be an ϵ-admissible encoding with inverter algorithm $\mathcal{I}_f$ and let $\mathcal{G} : \{0,1\}^* \to S$ be a random oracle. Then, $f \circ \mathcal{G}$ is $(t_{\mathcal{S}}, t_{\mathcal{D}}, q_1, q_2, \epsilon')$-indifferentiable from a random oracle into R, where $t_{\mathcal{S}} = q_1\, t_{\mathcal{I}_f}$ and $\epsilon' = 2(q_1 + q_2)\epsilon$.*

Before moving to the proof of Theorem 3, we prove the following useful result.

Lemma 4. *Let $f : S \to R$ be an ϵ-admissible encoding with inverter algorithm $\mathcal{I}_f$. Then*

$$\models s \xleftarrow{\$} S;\ r \leftarrow f(s) \simeq^{\emptyset}_{\{r,s\}} r \xleftarrow{\$} R;\ s \leftarrow \mathcal{I}_f(r) \preceq 2\epsilon$$

Proof. Define

$$\begin{array}{lll}
c_i & \stackrel{\text{def}}{=} & s \xleftarrow{\$} S;\ r \leftarrow f(s) \\
c_f & \stackrel{\text{def}}{=} & r \xleftarrow{\$} R;\ s \leftarrow \mathcal{I}_f(r) \\
c_1 & \stackrel{\text{def}}{=} & c_i;\ \mathsf{if}\ s = \bot\ \mathsf{then}\ r \xleftarrow{\$} R\ \mathsf{else}\ r \leftarrow f(s) \\
c_2 & \stackrel{\text{def}}{=} & c_f;\ \mathsf{if}\ s = \bot\ \mathsf{then}\ \mathbf{bad} \leftarrow \mathsf{true};\ r \xleftarrow{\$} R\ \mathsf{else}\ r \leftarrow f(s) \\
c_3 & \stackrel{\text{def}}{=} & c_f;\ \mathsf{if}\ s = \bot\ \mathsf{then}\ \mathbf{bad} \leftarrow \mathsf{true}\ \mathsf{else}\ r \leftarrow f(s)
\end{array}$$

Since the first branch of the conditional in c_1 is never executed, we have:

$$\models c_i \simeq^{\emptyset}_{\{r,s\}} c_1$$

Due to the second property of Definition 6, the distributions of s after executing c_i and c_f are ϵ-away. Using the rules for approximate observational equivalence, we obtain

$$\models c_1 \simeq^{\emptyset}_{\{r,s\}} c_2 \preceq \epsilon$$

The corollary to the Fundamental Lemma in Section 3.2 implies that $(\!|c_2, c_3|\!) \preceq \Pr[c_2 : \mathbf{bad}]$. Moreover,

$$\Pr[c_2 : \mathbf{bad}] = 1 - \Pr[c_f : s \neq \bot] = \Pr[s \xleftarrow{\$} S : s \neq \bot] - \Pr[c_f : s \neq \bot] \leq \epsilon$$

where the last inequality holds again because of the second property of Definition 6. Since the final values of r and s in programs c_2 and c_3 are independent of the initial memory, we have

$$\models c_2 \simeq^{\emptyset}_{\{r,s\}} c_3 \preceq \epsilon$$

Because $\mathcal{I}_f$ is a partial inverter for f, the else branch of the conditional in c_3 has no effect and can be removed, and thus $\models c_3 \simeq^{\emptyset}_{\{r,s\}} c_f$. We conclude by transitivity of approximate observational equivalence. □

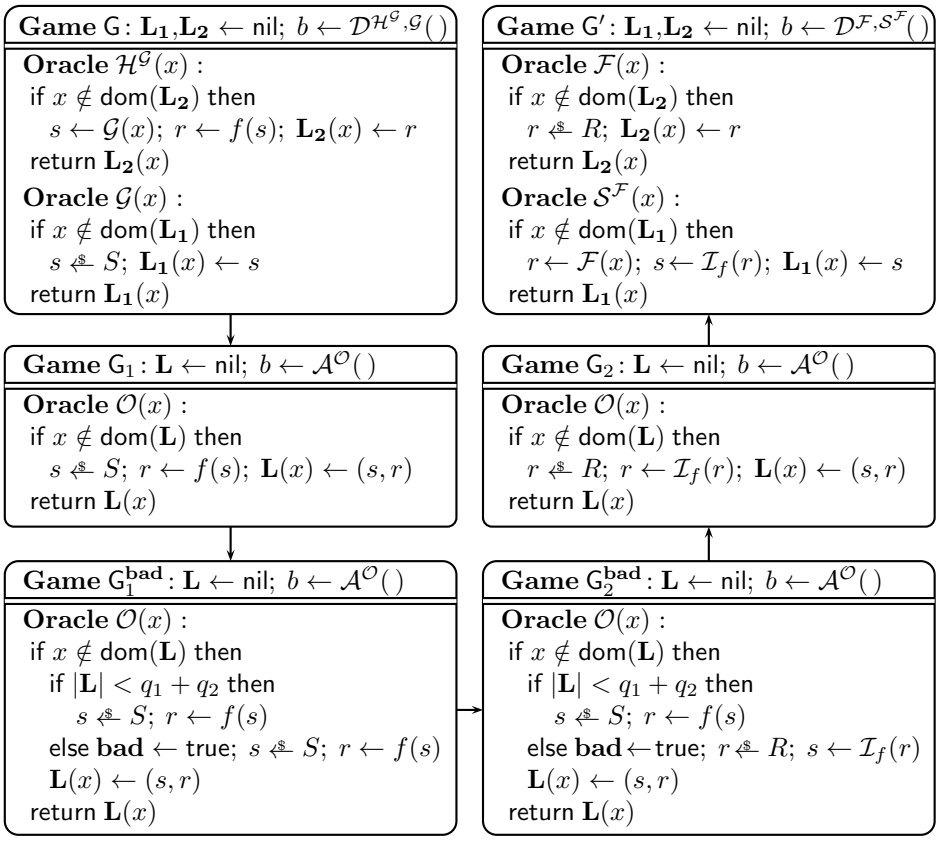

Figure 8: Games used in the proof of Theorem 3

Proof of Theorem 3. Let $\mathcal{D}$ be a distinguisher against the indifferentiability of $\mathcal{H} = f \circ \mathcal{G}$ from a random oracle $\mathcal{F}$ into R. We show that the simulator $\mathcal{S}$ constructed as $\mathcal{I}_f \circ \mathcal{F}$ is good enough, i.e. $\mathcal{D}$ cannot distinguish with probability greater than ϵ' between a game G where it is given $\mathcal{H}$ and $\mathcal{G}$ as oracles and a game G' where it is given $\mathcal{F}$ and $\mathcal{S}$ instead. An overview of the proof, including these two games is shown in Figure 8.

Our goal is to prove

$$|\Pr[\mathsf{G} : b = \mathsf{true}] - \Pr[\mathsf{G}' : b = \mathsf{true}]| \leq 2(q_1 + q_2)\epsilon \tag{8}$$

The crux of the proof is an application of Lemma 2. In order to apply it, we need first to transform game G (resp. G') to replace oracles $\mathcal{H}$ and $\mathcal{G}$ (resp. $\mathcal{F}$ and $\mathcal{S}$) by a single joint oracle that simultaneously returns the responses of both. Using $\mathcal{D}$, we construct an adversary $\mathcal{A}$ with access to a single joint oracle, such that game G (resp. G') is equivalent to game G_1 (resp. G_2) in the figure. Adversary $\mathcal{A}$ simply calls the distinguisher $\mathcal{D}$ and forwards the value it returns; it simulates $\mathcal{H}$ and $\mathcal{G}$ (resp. $\mathcal{F}$ and $\mathcal{S}$) by using its own oracle $\mathcal{O}$.

We assume, as Brier et al. [13] do, that whenever the distinguisher makes a query x to one of its oracles, it also makes the same query to its other oracle. Under this assumption, game G is equivalent to G_1, and game G' is equivalent to G_2. We thus have:

$$\Pr\left[\mathsf{G} : b = \mathsf{true}\right] = \Pr\left[\mathsf{G}_1 : b = \mathsf{true}\right] \qquad \Pr\left[\mathsf{G}' : b = \mathsf{true}\right] = \Pr\left[\mathsf{G}_2 : b = \mathsf{true}\right]$$

Furthermore, since $\mathcal{D}$ makes at most q_1 and q_2 queries to each of its oracles, $\mathcal{A}$ makes at most $q = q_1 + q_2$ queries to its joint oracle.

We next transform the implementation of oracle $\mathcal{O}$ in game G_1 so that its behavior after the first q queries is the same as in G_2. This transformation will pave the way to applying Lemma 2. The desired behavior of oracle $\mathcal{O}$ is represented in game $\mathsf{G}_2^{\mathbf{bad}}$. Observe that $\mathsf{G}_2^{\mathbf{bad}}$ is annotated with a flag $\mathbf{bad}$ that is set to true when the allotted number of queries is reached. This is because we essentially rely on Lemma 3 to justify the equivalence between G_1 and $\mathsf{G}_2^{\mathbf{bad}}$. To apply this lemma we need however to introduce an intermediate game $\mathsf{G}_1^{\mathbf{bad}}$. Since G_1 and $\mathsf{G}_1^{\mathbf{bad}}$ are trivially equivalent, we have

$$\Pr\left[\mathsf{G}_1 : b = \mathsf{true}\right] = \Pr\left[\mathsf{G}_1^{\mathbf{bad}} : b = \mathsf{true}\right]$$

An application of Lemma 3 between games $\mathsf{G}_1^{\mathbf{bad}}$ and $\mathsf{G}_2^{\mathbf{bad}}$ gives

$$\Pr\left[\mathsf{G}_1^{\mathbf{bad}} : b = \mathsf{true} \wedge \neg\mathbf{bad}\right] = \Pr\left[\mathsf{G}_2^{\mathbf{bad}} : b = \mathsf{true} \wedge \neg\mathbf{bad}\right]$$

But since $\mathbf{bad} \implies q < |\mathbf{L}|$ is an invariant and $|\mathbf{L}| \leq q$ a post-condition of both $\mathsf{G}_1^{\mathbf{bad}}$ and $\mathsf{G}_2^{\mathbf{bad}}$, we have

$$\Pr\left[\mathsf{G}_1^{\mathbf{bad}} : b = \mathsf{true}\right] = \Pr\left[\mathsf{G}_2^{\mathbf{bad}} : b = \mathsf{true}\right]$$

We can now apply Lemma 2 to the games G_2 and $\mathsf{G}_2^{\mathbf{bad}}$, defining $\mathsf{cntr} = |\mathbf{L}|$ and $h(i) = \mathbf{if}\ i < q\ \mathbf{then}\ 2\epsilon\ \mathbf{else}\ 0$. The second hypothesis of the lemma, i.e. that a call to $E_2(\mathcal{O})$ cannot decrease $|\mathbf{L}|$, is immediate. We can assume that $2q\epsilon < 1$ (otherwise the theorem is trivially true). Then for any pair of initial and final memories m and m',

$$\sum_{[\![\mathsf{cntr}]\!]m \leq i < [\![\mathsf{cntr}]\!]m'} h(i) \leq 2q\epsilon < 1, \qquad \text{and} \qquad \bar{h}_{\mathsf{cntr}}(m, m') = \sum_{[\![\mathsf{cntr}]\!]m \leq i < [\![\mathsf{cntr}]\!]m'} h(i)$$

We are only left to prove that

$$(\!|E_2(\mathcal{O}), E_2^{\mathbf{bad}}(\mathcal{O})|\!) \preceq \lambda m.\ [\![E_2(\mathcal{O})]\!]\ m\ (\lambda m'.\ \bar{h}_{\mathsf{cntr}}(m, m'))$$

A case analysis on the conditions $x \in \mathsf{dom}(\mathbf{L})$ and $|\mathbf{L}| < q$ yields three cases; two of them yield a null distance and are immediate. The remaining case, where $x \notin \mathsf{dom}(\mathbf{L})$ and $|\mathbf{L}| < q$, yields a distance of 2ϵ and follows from Lemma 4. We finally obtain $(\!|\mathsf{G}_2, \mathsf{G}_2^{\mathbf{bad}}|\!) \preceq 2(q_1 + q_2)\epsilon$, which entails

$$\left|\Pr\left[\mathsf{G}_2 : b = \mathsf{true}\right] - \Pr\left[\mathsf{G}_2^{\mathbf{bad}} : b = \mathsf{true}\right]\right| \leq 2(q_1 + q_2)\epsilon$$

This, combined with the previous results implies the desired inequality. □

6 Application to Elliptic Curves

This section discusses the instantiation of the proof presented in the previous section to hashing into elliptic curves. We proceed in two steps. First, we show that every finite abelian group with an efficiently computable law can be given the structure of a padding algebra that satisfies the hypotheses of Theorem 2. It follows that any efficiently invertible encoding into such a group induces a hash function onto the group that is indifferentiable from a random oracle. Second, we show that the set of points of an elliptic curve form a finite abelian group with an efficiently computable law, and that Icart's function is an efficiently invertible encoding. The formalization of both steps in the Coq proof assistant relies on independently developed libraries of mathematics. In addition, it assumes standard mathematical results that are not formalized in Coq (e.g. Cassels' theorem).

We begin with some mathematical background, before describing the Coq formalization.

6.1 Mathematical Background

6.1.1 Fundamental theorem of finite groups

The fundamental theorem of finite groups states that every finite abelian group $(\mathbb{G}, \otimes)$ is isomorphic to a product of cyclic groups $\mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k}$. Moreover the decomposition can be made unique by fixing additional conditions on $n_1 \ldots n_k$. Assume that g_i is a generator of the group $\mathbb{Z}_{n_i}$, for all $1 \leq i \leq k$. Then for every group element $x \in \mathbb{G}$ there exists a unique vector $(z_1, \ldots, z_k) \in \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k}$ such that

$$x = g_1^{z_1} \otimes \cdots \otimes g_k^{z_k}$$

In the sequel, we use $\log\ x$ to denote $(z_1, \ldots, z_k)$ and $\vec{g}^{\,\vec{z}}$ to denote $g_1^{z_1} \otimes \cdots \otimes g_k^{z_k}$, as above.

6.1.2 Elliptic Curves over Finite Fields

Recall that the *order* of a finite field is of the form p^m, where p is a prime number called the *characteristic* of the field, and that finite fields of the same order are unique up to isomorphism. In the following, we let $\mathbb{F}_n$ refer to any finite field of order n (i.e. containing n elements).

For our purposes, it is sufficient to consider elliptic curves over fields of order p^m with $p > 3$, that is, finite fields of characteristic different from 2 and 3. For the sake of readability, we specialize all our definitions to this case. Let $a, b \in \mathbb{F}_{p^m}$ such that $4a^3 + 27b^2 \neq 0$. The elliptic curve induced by a and b contains all points $(X, Y) \in \mathbb{F}_{p^m} \times \mathbb{F}_{p^m}$ that satisfy the *Weierstrass equation*

$$Y^2 = X^3 + aX + b. \tag{9}$$

Note that the condition on a and b is equivalent to requiring that the polynomial $X^3 + aX + b$ has distinct roots, and ensures that the curve is non-singular (i.e. it has no cusps or self-intersections) and is thus a proper elliptic curve.

The set of points of an elliptic curve can be given the structure of an abelian group by adding an additional "idealized" *point at infinity* $\mathcal{O}$:

$$\mathbb{E}_{a,b}(\mathbb{F}_{p^m}) = \{(X,Y) \in \mathbb{F}_{p^m} \times \mathbb{F}_{p^m} \mid Y^2 = X^3 + aX + b\} \cup \{\mathcal{O}\}$$

The point $\mathcal{O}$ behaves as the identity; the inverse of an element is given by equations

$$-\mathcal{O} \stackrel{\text{def}}{=} \mathcal{O} \qquad \text{and} \qquad -(X,Y) \stackrel{\text{def}}{=} (X,-Y).$$

The definition of the group law rests on the following property of elliptic curves, which follows from Bézout's theorem [28, Corollary 7.8]: The line defined by two points P_1 and P_2 in a curve[2] intersects the curve at a third point P_3 (which might coincide with P_1 or P_2). The group law is defined as

$$P_1 \oplus P_2 \stackrel{\text{def}}{=} -P_3.$$

For an in-depth algebraic description of elliptic curves we refer the interested reader to Hankerson et al. [27] or Silverman [41].

The construction of hash functions onto elliptic curves exploits the particular group structure of this kind of curves; Cassels' theorem [27, Theorem 3.12] states that every elliptic curve $\mathbb{E}_{a,b}(\mathbb{F}_n)$ over a finite field $\mathbb{F}_n$ is isomorphic to the product of the two cyclic groups $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$ where n_1 and n_2 are uniquely determined; moreover n_2 divides both n_1 and $n-1$. As an immediate corollary, observe that $\#\mathbb{E}_{a,b}(\mathbb{F}_n) = n_1 n_2$, and that the group is cyclic when $n_2 = 1$.

6.1.3 Encodings into elliptic curves

Brier et al. [13] present two encodings into elliptic curves: Icart's function [30] and (a simplified version of) the Shallue-Woestijne-Ulas (SWU) algorithm [39]. We review their definition and prove that they can be computed and inverted in polynomial-time.

Icart's encoding. Let $p > 3$ be a prime such that $p^m \equiv 2 \pmod 3$. Icart's function $f_{a,b} : \mathbb{F}_{p^m} \to \mathbb{E}_{a,b}(\mathbb{F}_{p^m})$ is defined as:

$$f_{a,b}(t) \stackrel{\text{def}}{=} \begin{cases} (x, tx + v) & \text{if } t \neq 0 \\ ((-b)^{\frac{1}{3}}, 0) & \text{if } t = 0 \wedge a = 0 \\ \mathcal{O} & \text{if } t = 0 \wedge a \neq 0 \end{cases}$$

$$\text{where} \qquad x = \left(v^2 - b - \frac{t^6}{27}\right)^{\frac{1}{3}} + \frac{t^2}{3} \qquad v = \frac{3a - t^4}{6t}$$

As a side remark, observe that the original definition only deals with the case $a \neq 0$; the definition for the case $a = 0$ was suggested to us by Icart in a private communication.

One can prove by computation that the image of t through $f_{a,b}$ belongs to the curve $Y^2 = X^3 + aX + b$ for every t in $\mathbb{F}_{p^m}$. Moreover, the set of pre-images under Icart's

[2] If $P_1 = P_2 = (X_0, Y_0)$ we let $\{(X,Y) \in \mathbb{F}_n \times \mathbb{F}_n \mid \alpha(X - X_0) + \beta(Y - Y_0) = 0\}$ be the line defined by P_1 and P_2, where $\alpha = 3X_0^2 + a$ and $\beta = -2Y_0$.

function of a point in the curve can be characterized as the set of roots of a polynomial over $\mathbb{F}_{p^m}$:

$$f_{a,b}^{-1}(\mathcal{O}) = \begin{cases} \{0\} & \text{if } a \neq 0 \\ \emptyset & \text{if } a = 0 \end{cases}$$

$$f_{a,b}^{-1}(X,Y) = \begin{cases} \{t \mid t^3 - 6tX + 6Y = 0\} & \text{if } a = 0 \\ \{t \mid t^4 - 6t^2X + 6tY = 3a\} & \text{if } a \neq 0 \end{cases}$$

Since the polynomials are of degree at most 4, every point in the curve has at most 4 pre-images. Moreover, the pre-images can be computed in polynomial-time using algorithms for factoring polynomials over finite fields, e.g. Berlekamp's algorithm. Thus, Icart's encoding is polynomially invertible.

The Shallue-Woestijne-Ulas (SWU) encoding. Let $p > 3$ be a prime such that $p^m \equiv 3 \pmod 4$ and let $a, b \neq 0$ belong to $\mathbb{F}_{p^m}$. We use $g(X)$ as a shorthand for the polynomial $X^3 + aX + b$. For every $t \in \mathbb{F}_{p^m}$ we let

$$X_1(t) = -ba^{-1}\left(1 + (t^4 - t^2)^{-1}\right) \qquad X_2(t) = -t^2 X_1(t) \qquad U(t) = t^3 g(X_1(t))$$

Note that $U(t)^2 = -g(X_1(t))g(X_2(t))$. As -1 is a quadratic non-residue in $\mathbb{F}_{p^m}$, it follows that exactly one of $g(X_1(t))$ and $g(X_2(t))$ is a square and thus either $X_1(t)$ or $X_2(t)$ is the abscissa of a point on the curve $Y^2 = g(X)$ [23]. This motivates the definition of the SWU function $f'_{a,b}$ as follows:

$$f'_{a,b}(t) \stackrel{\text{def}}{=} \begin{cases} (X_1(t), g(X_1(t))^{\frac{1}{2}}) & \text{if } g(X_1(t)) \text{ is a square} \\ (X_2(t), g(X_2(t))^{\frac{1}{2}}) & \text{otherwise} \end{cases}$$

To compute the pre-images t of a point (X, Y) we need to solve equations $X_1(t) = X$ and $X_2(t) = X$, keeping the solutions that verify $g(X_1(t)) = Y^2$ and $g(X_2(t)) = Y^2$ respectively. Each of the constraints $X_1(t) = X$ and $X_2(t) = X$ can be reduced to a polynomial equation and the inverse of the SWU function can be computed as

$$\begin{aligned} {f'}_{a,b}^{-1}(\mathcal{O}) &= \emptyset \\ {f'}_{a,b}^{-1}(X,Y) &= \left\{t \mid -\alpha_1 t^4 + \alpha_1 t^2 - ba^{-1} = 0 \ \wedge \ g(X_1(t)) = Y^2\right\} \cup \\ &\quad \left\{t \mid -ba^{-1}t^4 + \alpha_2 t^2 - \alpha_2 = 0 \ \wedge \ g(X_2(t)) = Y^2\right\} \end{aligned}$$

where $\alpha_1 = ba^{-1} + X$ and $\alpha_2 = ba^{-1} - X$. Since each polynomial has degree 4 and for every t, exactly one of $g(X_1(t))$ and $g(X_2(t))$ is a square, every point in the curve admits at most 4 pre-images. As with Icart's function, they can be computed in polynomial-time using, for instance, the Berlekamp's algorithm. Therefore, the SWU encoding can be inverted in polynomial time.

Finally we claim that both encodings can be computed in polynomial time. This basically amounts to verifying that square (in the case of the SWU function) and cubic roots (in the case of Icart's function) can be computed in polynomial-time, which is in turn entailed by the restrictions imposed on the order p^m of the field. More precisely, for every $x \in \mathbb{F}_{p^m}$, condition $p^m \equiv 2 \pmod 3$ entails formula $x^{1/3} = x^{2p^m - 1}$ while condition $p^m \equiv 3 \pmod 4$ implies that $x^{1/2} = x^{(p^m+1)/4}$.

6.2 Formalization in Coq

To instantiate our generic proof of indifferentiability to Icart's encoding, we proceeded as follows:

1. We showed that every finite abelian group $\mathbb{G}$ can be given the structure of a padding algebra. The formalization exploits the fundamental theorem of finite groups to decompose $\mathbb{G}$ as a product of cyclic groups $\mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k}$. We set $P = \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k}$, $Q = \mathbb{G}$ and:

$$\vec{z} \circledast x = x \otimes \vec{g}^{-\vec{z}} \qquad \vec{z} \odot x = x \otimes \vec{g}^{\vec{z}} \qquad x_1 \oslash x_2 = \log\left({x_1}^{-1} \otimes x_2\right).$$

   Our formalization relies on the SSREFLECT standard library [26], which provides a wealth of results on finite abelian groups, including the fundamental theorem of finite groups.

   It follows from Theorems 1, 2 and 3 that one can build a hash function onto $\mathbb{G}$ from a polynomially invertible function $f : A \to \mathbb{G}$ and random oracles $\mathcal{G}_1 : \{0,1\}^* \to A$ and $\mathcal{G}_2 : \{0,1\}^* \to \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k}$.

   Lemma 5 (Indifferentiable hashing into finite abelian groups)**.** *Let $\mathbb{G}$ be a finite abelian group with an efficiently computable law. Let $\mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k}$ be the decomposition of $\mathbb{G}$ as a product of cyclic groups and let g_i be a generator of $\mathbb{Z}_{n_i}$ for $i = 1 \ldots k$. Assume that $f : A \to \mathbb{G}$ is a polynomial-time function such that for each x, the set $f^{-1}(x)$ can be computed by a probabilistic polynomial-time algorithm $\mathcal{I}_f$ and its size is bounded by B. Then, for any pair of random oracles $\mathcal{G}_1 : \{0,1\}^* \to A$ and $\mathcal{G}_2 : \{0,1\}^* \to \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k}$ and any value T polynomial in the security parameter, the construction*

$$\mathcal{H}(m) \stackrel{def}{=} f(\mathcal{G}_1(m)) \otimes \vec{g}^{\,\mathcal{G}_2(m)}$$

   is $(t_{\mathcal{S}}, t_{\mathcal{D}}, q_1, q_2, \epsilon)$-indifferentiable from a random oracle into $\mathbb{G}$, where $t_{\mathcal{S}} = q_1\,(T+1)t_{\mathcal{I}_f}$ and $\epsilon = 2(q_1+q_2)\,(1 - \#A/(B\,\#\mathbb{G}))^{T+1}$

2. We showed that the set of points of the elliptic curve $\mathbb{E}_{a,b}(\mathbb{F}_{p^m})$ can be construed as a finite abelian group. To carry out this step, we adapted Théry's formalization of elliptic curves [44] to match the definition of finite group used by SSREFLECT. As a result, one obtains an instantiation of Lemma 5 to elliptic curves (the fact that the group law is efficiently computable is assumed).

   Given its complexity, we have not attempted to formalize the proof of Cassels' theorem. However, Bartzia [10] are working towards completing a formalization of Cassels' theorem. Since their formalization is based on the SSREFLECT library of finite groups, it would be immediate to use it for specializing the lemma further. Specifically, one could define $\mathcal{G}_{2,1}$ and $\mathcal{G}_{2,2}$ as the first and second projections of $\mathcal{G}_2$ and let

$$\mathcal{H}(m) \stackrel{\text{def}}{=} f(\mathcal{G}_1(m)) \otimes {g_1}^{\mathcal{G}_{2,1}(m)} \otimes {g_2}^{\mathcal{G}_{2,2}(m)}$$

   Under the previous assumptions, the function H is indifferentiable from a random oracle;

3. We defined Icart's function $f_{a,b}$, and showed that it generates points in the curve $\mathbb{E}_{a,b}$. This required showing the existence of cubic roots in the field $\mathbb{F}_{p^m}$ when $p^m \equiv 2 \pmod 3$. Moreover, we defined the inverse of Icart's function, and assumed that it is polynomially computable. Discharging this assumption would require to show the existence of an efficient method for factoring polynomials over the underlying field, and is left as future work (Berlekamp's algorithm, for instance, computes the roots of a polynomial of degree d over field $\mathbb{F}_n$ in time $\mathcal{O}(d^2 \log^3 n)$). It then follows from Lemma 5 that $f_{a,b}$ induces a hash function onto $\mathbb{E}_{a,b}(\mathbb{F}_{p^m})$ that is $(t_{\mathcal{S}}, t_{\mathcal{D}}, q_1, q_2, 2(q_1 + q_2)\epsilon)$-indifferentiable from a random oracle into $\mathbb{E}_{a,b}(\mathbb{F}_{p^m})$, where $t_{\mathcal{S}} = q_1\ t_{\mathcal{I}_F} = q_1\ (T + 1)\ t_{f^{-1}}$ and $t_{f^{-1}}$ is an upper bound on the time needed to compute the pre-image of a point under Icart's function, i.e. to solve a polynomial of degree 4 in $\mathbb{F}_{p^m}$.

We could also instantiate Lemma 5 to the SWU encoding. However, this would require showing that equality $U(t)^2 = -g(X_1(t))g(X_2(t))$ entails that either $X_1(t)$ or $X_2(t)$ is the abscissa of point on the curve $Y^2 = g(X)$, which involves some reasoning about quadratic residues. Nowak's formalization of quadratic residues [34] would be a good starting point.

Overall, the formalization consists of over 65,000 lines of Coq (without counting components reused from the standard libraries of Coq and SSReflect), which break down as follows: 45,000 lines corresponding to the original CertiCrypt framework, 3,500 lines of extensions to CertiCrypt, 7,000 lines written originally for our application to indifferentiability, and 10,000 lines of a slightly adapted version of Théry's elliptic curve library [44].

We conclude this section with the observation that our work points to some underdeveloped areas in formalized mathematics. Although there have been substantial efforts to develop machine-checked libraries of mathematics, covering relevant topics such as polynomials and finite fields, the libraries lack many important results. For instance, we are not aware of any formalization of factorization algorithms for polynomials over finite fields. Since such algorithms, and in particular Berlekamp's algorithm, are used by many computer algebra systems, we believe that it would be of general interest to provide a machine-checked proof of their correctness. Moreover, elliptic curve theory is a fascinating area of mathematics, and it would be particularly appealing to develop formalizations of some of the most important results in the area.

7 Related Work

Weak Equivalences. Approximate observational equivalence can be construed as a quantitative hyperproperty [16], and is closely related to approximate notions of probabilistic bisimulations, see e.g. [18, 38, 45], and to quantitative information flow policies, see e.g. [15, 16, 35, 42]. Approximate observational equivalence can be further generalized by adding a multiplicative skew to statistical distance, as sketched at the end of Section 4. The resulting notion is able to characterize differential privacy [19], a notion of privacy that enjoys good composition results and enforces strong privacy guarantees. Barthe et al. [8] report on extending the CertiCrypt framework with an approximate probabilistic relational logic for reasoning about differential privacy.

Hashing into Elliptic Curves. A number of highly relevant cryptographic constructions, including identity based schemes [12] and password based key exchange protocols [11], require hashing into elliptic curves. Indeed, there have been a number of proposals for such hash functions, e.g. [23, 30, 39]. Recently, Farashahi et al. [21] developed powerful techniques to show the indifferentiability of hash function constructions based on deterministic encodings. Their results improve on [13], in the sense that they apply to a larger class of encodings, including encodings to hyperelliptic curves, and that they provide tighter bounds for encodings that are covered by both methods.

Formalization and Verification of Cryptography Despite their widespread use, hash functions and elliptic curves have received little attention from the formal verification community. To our best knowledge, our work provides the first machine-checked proof of security for a cryptographic primitive based on elliptic curves, and the first proof of security for a cryptographic hash function.

Previous works on the formalization of elliptic curves include [29, 44]. Hurd et al. [29] report on the verification in HOL of the group axioms and an application to the functional correctness of ElGamal encryption. Théry and Hanrot [44] use Coq to formalize the group axioms, and show how the formalization of elliptic curves can be used to build efficient reflective tactics for testing primality.

Other works on the formalization of hash functions include [3, 46]. Toma and Borrione [46] use ACL2 to reason about functional properties of SHA-1. Backes et al. [3] prove that the Merkle-Damgård construction is collision-resistant and indifferentiable from a random oracle, provided the underlying compression function is collision-resistant and input messages are padded using a prefix-free function.

8 Conclusion

This article reports on a machine-checked proof of a recent construction to build hash functions that are indifferentiable from a random oracle into an elliptic curve. The example is singular among other examples that have been formalized using CertiCrypt, because it builds on large libraries of formalized mathematics.

Our work paves the way for further developments around CertiCrypt and EasyCrypt. We have already used approximate observational equivalence to verify in CertiCrypt statistical zero-knowledge for the so-called Generalized Schnorr Protocols, and used the formalization as a back-end for a certified zero-knowledge compiler [1]. We have also enhanced EasyCrypt [6], an SMT-based verification tool based on the same logic as CertiCrypt, so that it can manipulate the notions of approximate equivalence considered in this article and in [8].

Acknowledgments. This work was partially funded by the European Projects FP7-256980 NESSoS and FP7-229599 AMAROUT, Spanish project TIN2009-14599 DESAFIOS 10, Madrid Regional project S2009TIC-1465 PROMETIDOS and French project ANR SESUR-012 SCALP.

References

[1] J. B. Almeida, M. Barbosa, E. Bangerter, G. Barthe, S. Krenn, and S. Zanella-Béguelin. Full proof cryptography: verifiable compilation of efficient zero-knowledge protocols. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM Conference on Computer and Communications Security*, pages 488–500. ACM, 2012. ISBN 978-1-4503-1651-4.

[2] P. Audebaud and C. Paulin-Mohring. Proofs of randomized algorithms in Coq. *Sci. Comput. Program.*, 74(8):568–589, 2009.

[3] M. Backes, G. Barthe, M. Berg, B. Grégoire, C. Kunz, M. Skoruppa, and S. Zanella-Béguelin. Verified security of Merkle-Damgård. In *25rd IEEE Computer Security Foundations Symposium, CSF 2012*. IEEE Computer Society, 2012.

[4] 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*, pages 90–101, New York, 2009. ACM.

[5] G. Barthe, B. Grégoire, and S. Zanella-Béguelin. Programming language techniques for cryptographic proofs. In *1st International Conference on Interactive Theorem Proving, ITP 2010*, volume 6172 of *Lecture Notes in Computer Science*, pages 115–130, Heidelberg, 2010. Springer.

[6] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella-Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90, Heidelberg, 2011. Springer.

[7] G. Barthe, B. Grégoire, S. Heraud, F. Olmedo, and S. Zanella-Béguelin. Verified indifferentiable hashing into elliptic curves. In *1st Conference on Principles of Security and Trust – POST 2012*, volume 7215 of *Lecture Notes in Computer Science*, Heidelberg, 2012. Springer.

[8] G. Barthe, B. Köpf, F. Olmedo, and S. Zanella-Béguelin. Probabilistic relational reasoning for differential privacy. In *39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012*, pages 97–110, New York, 2012. ACM.

[9] G. Barthe, G. Danezis, B. Grégoire, C. Kunz, and S. Zanella-Béguelin. Verified computational differential privacy with applications to smart metering. In *26th IEEE Computer Security Foundations Symposium, CSF 2013*, Los Alamitos, 2013. IEEE Computer Society.

[10] E.-I. Bartzia. A formalization of elliptic curves. Master's thesis, Université de Vincennes-Saint Denis – Paris VIII, 2011.

[11] S. Bellovin and M. Merritt. Encrypted key exchange: password-based protocols secure against dictionary attacks. In *13th IEEE Symposium on Security and Privacy, S&P 1992*, pages 72 –84, Los Alamitos, 1992. IEEE Computer Society.

[12] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17:297–319, 2004.

[13] E. Brier, J.-S. Coron, T. Icart, D. Madore, H. Randriam, and M. Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 237–254, Heidelberg, 2010. Springer.

[14] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[15] D. Clark, S. Hunt, and P. Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321–371, 2007.

[16] M. R. Clarkson and F. B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

[17] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448, Heidelberg, 2005. Springer.

[18] J. Desharnais, F. Laviolette, and M. Tracol. Approximate analysis of probabilistic processes: Logic, simulation and games. In *5th International Conference on Quantitative Evaluation of Systems, QEST 2008*, pages 264–273. IEEE Computer Society, 2008.

[19] C. Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, ICALP 2006*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12, Heidelberg, 2006. Springer.

[20] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 486–503, Heidelberg, 2006. Springer.

[21] R. R. Farashahi, P.-A. Fouque, I. Shparlinski, M. Tibouchi, and J. F. Voloch. Indifferentiable deterministic hashing to elliptic and hyperelliptic curves. *Mathematics of Computation*, 2011.

[22] E. Fleischmann, M. Gorski, and S. Lucks. Some observations on indifferentiability. In *Information Security and Privacy*, volume 6168 of *Lecture Notes in Computer Science*, pages 117–134, Heidelberg, 2010. Springer.

[23] P.-A. Fouque and M. Tibouchi. Deterministic encoding and hashing to odd hyperelliptic curves. In *4th International Conference on Pairing-Based Cryptography, Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 265–277, Heidelberg, 2010. Springer.

[24] O. Goldreich. Zero-knowledge twenty years after its invention. Technical Report TR02-063, Electronic Colloquium on Computational Complexity, 2002.

[25] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28 (2):270–299, 1984.

[26] G. Gonthier, A. Mahboubi, L. Rideau, E. Tassi, and L. Théry. A modular formalisation of finite group theory. In *20th International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2007*, volume 4732 of *Lecture Notes in Computer Science*, pages 86–101, Heidelberg, 2007. Springer.

[27] D. Hankerson, S. Vanstone, and A. Menezes. *Guide to Elliptic Curve Cryptography*. Springer Professional Computing. Springer, 2004. ISBN 9780387952734.

[28] R. Hartshorne. *Algebraic Geometry*. Graduate Texts in Mathematics. Springer-Verlag, 1977. ISBN 9780387902449.

[29] J. Hurd, M. Gordon, and A. Fox. Formalized elliptic curve cryptography. In *High Confidence Software and Systems, HCSS 2006*, 2006.

[30] T. Icart. How to hash into elliptic curves. In *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 303–316, Heidelberg, 2009. Springer.

[31] T. Icart. *Algorithms Mapping into Elliptic Curves and Applications*. PhD thesis, Université du Luxembourg, 2010.

[32] B. Jonsson, W. Yi, and K. G. Larsen. Probabilistic extensions of process algebras. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 685–710. Elsevier, Amsterdam, 2001.

[33] U. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *1st Theory of Cryptography Conference, TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39, Heidelberg, 2004. Springer.

[34] D. Nowak. On formal verification of arithmetic-based cryptographic primitives. In *11th International Conference on Information Security and Cryptology, ICISC 2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 368–382, Heidelberg, 2009. Springer.

[35] A. D. Pierro, C. Hankin, and H. Wiklicky. Approximate non-interference. *Journal of Computer Security*, 12(1):37–82, 2004.

[36] T. Ristenpart, H. Shacham, and T. Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 487–506, Heidelberg, 2011. Springer.

[37] A. Sahai and S. Vadhan. Manipulating statistical difference. In *Randomization Methods in Algorithm Design, DIMACS Workshop, 1997*, volume 43 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 251–270. American Mathematical Society, 1999.

[38] R. Segala and A. Turrini. Approximated computationally bounded simulation relations for probabilistic automata. In *20th IEEE Computer Security Foundations Symposium, CSF 2007*, pages 140–156. IEEE Computer Society, 2007.

[39] A. Shallue and C. van de Woestijne. Construction of rational points on elliptic curves over finite fields. In *7th International Symposium on Algorithmic Number Theory, ANTS-VII*, volume 4076 of *Lecture Notes in Computer Science*, pages 510–524, Heidelberg, 2006. Springer.

[40] V. Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, second edition, 2009.

[41] J. H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer, Heidelberg, 2nd edition, 2009.

[42] G. Smith. On the foundations of quantitative information flow. In *12th International Conference on Foundations of Software Science and Computational Structures - FoSSaCS 2009*, pages 288–302, Heidelberg, 2009. Springer.

[43] The Coq development team. The Coq Proof Assistant Reference Manual Version 8.3. Online – `http://coq.inria.fr`, 2010.

[44] L. Théry and G. Hanrot. Primality proving with elliptic curves. In *20th International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2007*, volume 4732 of *Lecture Notes in Computer Science*, pages 319–333, Heidelberg, 2007. Springer.

[45] S. Tini. Non-expansive ϵ-bisimulations for probabilistic processes. *Theoretical Computer Science*, 411(22-24):2202–2222, 2010.

[46] D. Toma and D. Borrione. Formal verification of a SHA-1 circuit core using ACL2. In *18th International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2005*, volume 3603 of *Lecture Notes in Computer Science*, pages 326–341, Heidelberg, 2005. Springer.