Computer-Aided Cryptographic Proofs

Gilles Barthe[1], Juan Manuel Crespo[1], Benjamin Grégoire[2], César Kunz[1,3], and Santiago Zanella Béguelin[4]

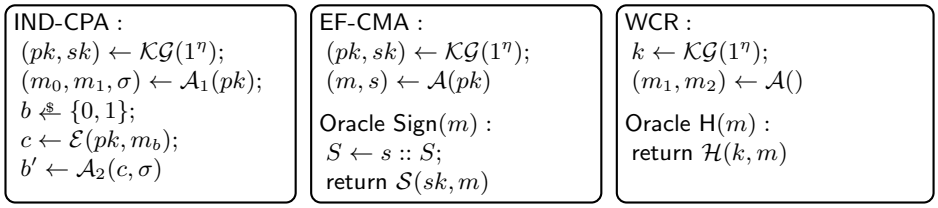
[1] IMDEA Software Institute
{Gilles.Barthe,Cesar.Kunz,JuanManuel.Crespo}@imdea.org
[2] Universidad Politécnica de Madrid
[3] INRIA Sophia Antipolis-Méditerranée
Benjamin.Gregoire@inria.fr
[4] Microsoft Research
santiago@microsoft.com

Abstract. EasyCrypt is an automated tool that supports the machine-checked construction and verification of security proofs of cryptographic systems, and that has been used to verify emblematic examples of public-key encryption schemes, digital signature schemes, hash function designs, and block cipher modes of operation. The purpose of this paper is to motivate the role of computer-aided proofs in the broader context of provable security and to illustrate the workings of EasyCrypt through simple introductory examples.

1 Introduction

The rigorous study of cryptographic systems as mathematical objects originates with the landmark article "Communication Theory of Secrecy Systems" [**?**], in which Shannon defines the notion of perfect secrecy for (symmetric) encryption systems, and shows that it can only be achieved if the size of keys equals or exceeds the size of messages. Shannon's article is often viewed as marking the beginning of modern cryptography, because it was the first to recognize the importance of rigorous mathematical definitions and proofs in the analysis of cryptographic systems.

In contrast to perfect secrecy, which yields unconditional, information-theoretic security, modern cryptography yields conditional guarantees that only hold under computational assumptions. Modern cryptography takes inspiration from complexity theory: rather than considering arbitrary adversaries against the security of cryptographic systems, security is established against adversaries with bounded computational resources. Moreover, the security guarantee itself is probabilistic and is expressed as an upper bound of the probability of an adversary with bounded resources breaking the security of the system. Typically, the computational security of a cryptographic system is proved *by reduction* to one or more assumptions about the hardness of computational problems. This reductionist approach originates from the seminal article "Probabilistic Encryption" [**?**], in which Goldwasser and Micali elaborate a three-step process for proving the security of a cryptographic system:



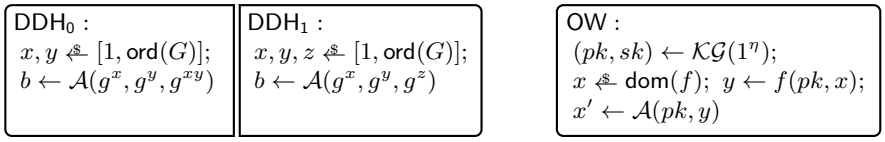
$$\mathbf{Adv}^{(\mathcal{KG},\mathcal{E},\mathcal{D})}_{\mathsf{IND\text{-}CPA}}(\mathcal{A}) \stackrel{\text{def}}{=} |\Pr[\mathsf{IND\text{-}CPA} : b = b'] - 1/2|$$

$$\mathbf{Adv}^{(\mathcal{KG},\mathcal{S},\mathcal{V})}_{\mathsf{EF\text{-}CMA}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[\mathsf{EF\text{-}CMA} : \mathcal{V}(pk, m, s) \wedge m \notin S]$$

$$\mathbf{Adv}^{(\mathcal{KG},\mathcal{H})}_{\mathsf{WCR}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[\mathsf{WCR} : \mathcal{H}(k, m_1) = \mathcal{H}(k, m_2) \wedge m_1 \neq m_2]$$

Fig. 1. Experiments corresponding to security notions for various cryptographic constructions (from left to right): *indistinguishability under chosen-plaintext attack* for encryption schemes, *existential unforgeability under chosen-message attack* for signature schemes, and *weak collision-resistance* for keyed hash functions. In these experiments $\mathcal{A}$ denotes an adversary that may have access to oracles; $\mathcal{A}$ has access to a signature oracle $\mathcal{S}(sk, \cdot)$ in experiment EF-CMA and to a hash oracle $\mathcal{H}(k, \cdot)$ in experiment WCR.

1. Formalize precisely the security goal and the adversarial model. A common manner of proceeding is to consider an experiment in which an adversary interacts with a challenger. The challenger sets up and runs the experiment, answers to adversary oracle queries, and determines whether the adversary succeeds. Figure 1 describes experiments corresponding to some typical security notions. Formally, an experiment EXP can be seen as a function that given as input a cryptographic system Π and an adversary $\mathcal{A}$, returns a distribution over some set of output variables. The advantage of an adversary $\mathcal{A}$ in a security experiment EXP, noted $\mathbf{Adv}^{\Pi}_{\mathsf{EXP}}(\mathcal{A})$, is defined in terms of this output distribution.
2. Formalize precisely the assumptions upon which the security of the system relies. Such assumptions assert the practical unfeasibility of solving a computational (or decision) problem believed to be hard. As security goals, they can also be formalized by means of experiments between a challenger and an adversary (an assumption could play the role of a security goal in a lower level proof). Figure 2 describes some assumptions used to realize cryptographic functionalities.
3. Define a cryptographic system Π and give a rigorous proof of its security by exhibiting a reduction from the experiment EXP, corresponding to the security goal, to one or more computational assumptions. Suppose for simplicity that the reductionist proof involves a single assumption, modelled by an experiment EXP′. Broadly construed, the reduction must show that for every efficient adversary $\mathcal{A}$ against EXP, there exists an efficient adversary $\mathcal{B}$ against EXP′ whose advantage is comparable to that of $\mathcal{A}$. In most cases, the proof is constructive and exhibits an adversary $\mathcal{B}$ against EXP′ that uses $\mathcal{A}$ as a sub-routine.



$$\mathbf{Adv}^{(G,g)}_{\mathsf{DDH}}(\mathcal{A}) \stackrel{\text{def}}{=} |\Pr[\mathsf{DDH}_0 : b] - \Pr[\mathsf{DDH}_1 : b]|$$
$$\mathbf{Adv}^{(\mathcal{KG},f)}_{\mathsf{OW}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[\mathsf{OW} : f(pk, x') = y]$$

Fig. 2. Experiments corresponding to security assumptions used to realize cryptographic goals: *Decision Diffie-Hellman* problem for a finite cyclic multiplicative group G with generator g (left) and *One-Wayness* of a trapdoor function $(\mathcal{KG}, f)$ (right).

Early works on provable security take an asymptotic approach to capture the notions of efficiency and hardness. In this setting, experiments and assumptions are indexed by a security parameter, typically noted η, which determines the size of objects on which computations are performed (e.g. keys, messages, groups). Asymptotic security equates the class of efficient computations to the class of *probabilistic polynomial-time* algorithms, so that security is established against adversaries whose memory footprint and computation time is bounded by a polynomial on the security parameter. Moreover, in an asymptotic security setting, a problem is considered hard when no efficient adversary can achieve a *non-negligible* advantage as a function of the security parameter. (A function is negligible on η when it is upper-bounded by $1/\eta^c$ for any $c > 0$.)

A more practically relevant approach to cryptographic proofs evaluates quantitatively the efficiency of reductions. This approach, known as practice-oriented provable security or concrete security, originates from the work of Bellare and Rogaway on authentication protocols [**?**] and the DES block cipher [**?**]. A typical concrete security proof reducing the security of a construction Π w.r.t. EXP to an assumption EXP' about some object Π', begins by assuming the existence of an adversary $\mathcal{A}$ against EXP that runs within time $t_{\mathcal{A}}$ (and makes at most $q_{\mathcal{A}}$ oracle queries). The proof exhibits a witness for the reduction in the form of an adversary $\mathcal{B}$ against EXP' that uses $\mathcal{A}$ as a sub-routine, and provides concrete bounds for its resources and its advantage in terms of those of $\mathcal{A}$, e.g.:

$$t_B \leq t_A + p(q_{\mathcal{A}})$$
$$\mathbf{Adv}^{\Pi'}_{\mathsf{EXP}'}(\mathcal{B}) \geq \mathbf{Adv}^{\Pi}_{\mathsf{EXP}}(\mathcal{A}) - \epsilon(q_{\mathcal{A}})$$

A concrete security proof can be used to infer sensible values for the parameters (e.g. key size) of cryptographic constructions. Based on an estimate of the resources and advantage of the best known method to solve EXP' (and a conservative bound on $q_{\mathcal{A}}$), one can choose the parameters of Π such that the reduction $\mathcal{B}$ would yield a better method, thus achieving a practical contradiction.

The game-based approach, as popularized by Shoup [**?**], and Bellare and Rogaway [**?**], is a methodology to structure reductionist proofs in a way that makes them easier to understand and check. A game-based proof is organized as a tree of games (equivalently, experiments). The root of the tree is the experiment

that characterizes the security goal, whereas the leaves are either experiments corresponding to security assumptions or experiments where the probability of an event of interest can be directly bounded. Edges connecting a game G at one level in the tree to its successors $G_1, \ldots, G_n$ correspond to *transitions*; a transition relates the probability of an event in one game to the probability of some, possibly different, event in another game. Put together, these transitions may allow to prove, for example, an inequality of the form

$$\Pr[G : E] \leq a_1 \Pr[G_1 : E_1] + \cdots + a_n \Pr[G_n : E_n]$$

By composing statements derived from the transitions in the tree, one ultimately obtains a bound on the advantage of an adversary against the experiment at the root in terms of the advantage of one of more *concrete* adversaries against assumptions at the leaves.

Whereas games can be formalized in the usual language of mathematics, Bellare and Rogaway [**?**] model games as probabilistic programs, much like we modelled experiments in Figures 1 and 2. This code-based approach allows giving games a rigorous semantics, and paves the way for applying methods from programming language theory and formal verification to cryptographic proofs. This view was further developed by Halevi [**?**], who argues that computer-aided verification of cryptographic proofs would be of significant benefit to improve confidence in their correctness, and outlines the design of a computer-aided framework for code-based security proofs.

Verified security [**?**, **?**] is an emerging approach to practice-oriented provable security: its primary goal is to increase confidence in reductionist security proofs through their computer-aided formalization and verification, by leveraging state-of-the-art verification tools and programming language techniques. CertiCrypt [**?**] realizes verified security by providing a fully machine-checked framework built on top of the Coq proof assistant, based on a deep embedding of an extensible probabilistic imperative language to represent games. CertiCrypt implements several verification methods that are proved sound (in Coq) w.r.t. the semantics of programs and inherits the expressive power and the strong guarantees of Coq. Unfortunately, it also inherits a steep learning curve and as a result its usage is time-consuming and requires a high level of expertise. EasyCrypt [**?**], makes verified security more accessible to the working cryptographer by means of a concise input language and a greater degree of automation, achieved by using off-the-shelf SMT solvers and automated theorem provers rather than an interactive proof assistant like Coq.

Issues with verified security. Verified security is no panacea and inherits several of the issues of provable security and formal proofs in general. We only review briefly some key issues, and refer the interested reader to more detailed reviews of provable security [**?**, **?**, **?**], and formal proofs [**?**, **?**]. We stress that these issues do not undermine by any means the importance of verified security.

The first issue regards the interpretation of a verified security proof. As the proof is machine-checked, one can reasonably believe in its correctness without the need to examine the details of the proof. However, a careful analysis of

the statement is fundamental to understand the guarantees it provides. In the case of verified security, statements depend on unproven hardness assumptions, which are meaningful only when instantiated with a sensible choice of parameters. Thus, one must consider the security assumptions and convince oneself that they are adequately modelled and instantiated; proofs relying on flawed or inappropriately instantiated assumptions fail to provide any meaningful guarantee. In addition, cryptographic proofs often assume that some functionalities are ideal. As with security assumptions, one must convince oneself that modelling primitives as ideal functionalities is reasonable, and that instantiating these primitives does not introduce subtle attack vectors. Random oracles are a common instance of ideal functionality; in the Random Oracle Model (ROM) [**?**], some primitives used in a cryptographic system, such as hash functions, are modelled as perfectly random functions, i.e. as maps chosen uniformly from a function space. Proofs in the ROM are considered as providing strong empirical evidence of security, despite some controversy [**?**, **?**, **?**].

The second issue is the level of abstraction in security proofs. Security proofs reason about models rather than implementations. As a result, cryptographic systems, even though supported by a proof of security, may be subject to practical attacks outside the model. Prominent examples of practical attacks are padding oracle attacks [**?**, **?**], which exploit information leakage through error handling, and side-channel attacks [**?**, **?**, **?**], which exploit quantitative information such as execution time or memory consumption. There is a growing body of work that addresses these concerns; in particular, leakage-resilient security [**?**] gives the adversary access to oracles performing side-channel measurements. However, most of the provable security literature, and certainly all of the verified security literature, forego an analysis of side-channels.

Organization of the paper. Section 2 overviews the foundations of EasyCrypt. Subsequent sections focus on examples: One-Time Pad encryption (Section 3), the nested message authentication code NMAC (Section 4), and ElGamal encryption (Section 5). Section 6 reviews some topics deserving more attention.

2 Foundations

This section reviews the foundations of the code-based game-based approach, as implemented by EasyCrypt; more detailed accounts appear in [**?**, **?**].

Programming language. Games are represented as programs in the strongly-typed, probabilistic imperative language pWHILE:

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

The language includes deterministic and probabilistic assignments, conditionals, loops, and procedure calls. In the above grammar, $\mathcal{V}$ is a set of variable identifiers, $\mathcal{P}$ a set of procedure names, $\mathcal{E}$ is a set of expressions, and $\mathcal{DE}$ is a set of probabilistic expressions. The latter are expressions that evaluate to distributions from which values can be sampled. An assignment $x \xleftarrow{\$} d$ evaluates the expression d to a distribution μ over values, samples a value according to μ and assigns it to variable x. The base language of expressions (deterministic and probabilistic) can be extended by the user to better suit the needs of the verification goal. The rich base language includes expressions over Booleans, integers, fixed-length bitstrings, lists, finite maps, and option, product and sum types. User-defined operators can be axiomatized or defined in terms of other operators. In the following, we let $\{0,1\}^\ell$ denote the uniform distribution over bitstrings of length ℓ, $\{0,1\}$ the uniform distribution over Booleans, and $[a,b]$ the uniform distribution over the integer interval $[a,b]$.

A program in EasyCrypt is modelled as a set of global variables and a collection of procedures. The language distinguishes between defined procedures, used to describe experiments and oracles, and abstract procedures, used to model adversaries. Quantification over adversaries in cryptographic proofs is achieved by representing them as abstract procedures parametrized by a set of oracles.

Denotational semantics. A pWHILE program c is interpreted as a function $[\![c]\!]$ that maps an initial memory to a sub-distribution over final memories. As pWHILE is a strongly-typed language, a memory is a mapping from variables to values of the appropriate type. When the set of memories is finite, a sub-distribution over memories can be intuitively seen as a mapping assigning to each memory a probability in the unit interval $[0,1]$, so that the sum over all memories is upper bounded by 1. In the general case, we represent a sub-distribution over memories using the *measure monad* of Audebaud and Paulin [**?**]. Given a program c, a memory m, and an event E, we let $\Pr[c, m : E]$ denote the probability of E in the sub-distribution induced by $[\![c]\!]\ m$; we often omit the initial memory m when it is irrelevant.

Relational program logic. Common reasoning patterns in cryptographic proofs are captured by means of a probabilistic Relational Hoare Logic (pRHL). Its judgments are of the form

$$\models c_1 \sim c_2 : \Psi \Rightarrow \Phi$$

where c_1 and c_2 are probabilistic programs, and the pre- and post-conditions Ψ and Φ are relations over memories. Informally, a judgment $\models c_1 \sim c_2 : \Psi \Rightarrow \Phi$ is valid if for every two memories m_1 and m_2 satisfying the pre-condition Ψ, the sub-distributions $[\![c_1]\!]\ m_1$ and $[\![c_2]\!]\ m_2$ satisfy the post-condition Φ. As the post-condition is a relation on memories rather than a relation on sub-distributions over memories, the formal definition of validity relies on a lifting operator, whose definition originates from probabilistic process algebra [**?**, **?**].

Relational formulae are represented in EasyCrypt by the grammar:

$$\Psi, \Phi ::= e \mid \neg\Phi \mid \Psi \wedge \Phi \mid \Psi \vee \Phi \mid \Psi \implies \Phi \mid \forall x.\ \Phi \mid \exists x.\ \Phi$$

where e stands for a Boolean expression over logical variables and program variables tagged with either $\langle 1 \rangle$ or $\langle 2 \rangle$ to denote their interpretation in the left or right-hand side program; the only restriction is that logical variables must not occur free. The special keyword res denotes the return value of a procedure and can be used in the place of a program variable. We write $e\langle i \rangle$ for the expression e in which all program variables are tagged with $\langle i \rangle$. A relational formula is interpreted as a relation on program memories. For example, the formula $x\langle 1 \rangle + 1 \leq y\langle 2 \rangle$ is interpreted as the relation

$$\Phi = \{(m_1, m_2) \mid m_1(x) + 1 \leq m_2(y)\}$$

Reasoning about probabilities. Security properties are typically expressed in terms of probability of events, and not as pRHL judgments. Pleasingly, one can derive inequalities about probability quantities from valid judgments. In particular, assume that Φ is of the form $A\langle 1 \rangle \implies B\langle 2 \rangle$, i.e. relates pairs of memories m_1' and m_2' such that when m_1' satisfies the event A, m_2' satisfies the event B. Then, for any two programs c_1 and c_2 and pre-condition Ψ such that $\models c_1 \sim c_2 : \Psi \Rightarrow \Phi$ is valid, and for any two memories m_1 and m_2 satisfying the pre-condition Ψ, we have $\Pr[c_1, m_1 : A] \leq \Pr[c_2, m_2 : B]$. Other forms of pRHL judgments allow to derive more complex inequalities and capture other useful forms of reasoning in cryptographic proofs, including Shoup's Fundamental Lemma [**?**].

3 Perfect Secrecy of One-Time Pad

Shannon [**?**] defines perfect secrecy of an encryption scheme by the condition that learning a ciphertext does not change any *a priori* knowledge about the likelihood of messages. In other words, for any given distribution over messages, the distribution over ciphertexts (determined by the random choices of the key generation and encryption algorithms) must be independent of the distribution over messages. Shannon shows that perfect secrecy can only be achieved if the key space is at least as large as the message space, and that the One-Time Pad encryption scheme (also known as Vernam's cipher) is perfectly secret.

For any positive integer ℓ, One-Time Pad is a deterministic symmetric encryption scheme composed of the following triple of algorithms:

Key Generation The key generation algorithm $\mathcal{KG}$ outputs a uniformly distributed key k in $\{0,1\}^\ell$;

Encryption Given a key k and a message $m \in \{0,1\}^\ell$, $\mathcal{E}(k, m)$ outputs the ciphertext $c = k \oplus m$ ($\oplus$ denotes bitwise exclusive-or on bitstrings);

Decryption Given a key k and a ciphertext $c \in \{0,1\}^\ell$, the decryption algorithm outputs the message $m = k \oplus c$.

We represent the a priori distribution over messages by a user-defined probabilsitc operator $\mathcal{M}$. We prove perfect secrecy of One-Time Pad by showing that

encrypting a message m sampled according to $\mathcal{M}$ results in a ciphertext distributed uniformly and independently from m. We prove this by showing that the joint distribution of c, m in experiments OTP and Uniform below is the same:

$$\textbf{Game } \mathsf{OTP} : m \xleftarrow{\$} \mathcal{M};\ k \leftarrow \mathcal{KG}();\ c \leftarrow \mathcal{E}(k, m);$$
$$\textbf{Game } \mathsf{Uniform} : m \xleftarrow{\$} \mathcal{M};\ c \xleftarrow{\$} \{0,1\}^{\ell};$$

In a code-based setting, this is captured by the following relational judgment:

$$\models \mathsf{OTP} \sim \mathsf{Uniform} : \mathsf{true} \Rightarrow (c, m)\langle 1\rangle = (c, m)\langle 2\rangle \tag{1}$$

The OTP and Uniform experiments are formalized in EasyCrypt as follows:

```
game OTP = {
  var m : message
  var c : ciphertext
  fun KG() : key = { var k:key = {0,1}^ℓ; return k; }
  fun Enc(k:key, m:message) : ciphertext = { return (k ⊕ m); }
  fun Main() : unit = { var k:key; m = M(); k = KG(); c = Enc(k, m); }
}.

game Uniform = {
  var m : message
  var c : ciphertext
  fun Main() : unit = { m = M(); c = {0,1}^ℓ; }
}.
```

where the types `key`, `message` and `ciphertext` are all synonyms for the type of bitstrings of length ℓ.

The relational judgment (1) is stated and proved in EasyCrypt as follows:

```
equiv Secrecy : OTP.Main ~ Uniform.Main : true ==> (c,m)⟨1⟩ = (c,m)⟨2⟩.
proof.
 inline KG, Enc; wp.
 rnd (c ⊕ m); trivial.
save.
```

The proof starts by inlining the definition of the procedures `KG` and `Enc`, and applying the `wp` tactic to compute the relational weakest pre-condition over the deterministic suffix of the resulting programs. This yields the following intermediate goal:

```
pre = true
stmt1 = m = M(); k = {0,1}^ℓ;
stmt2 = m = M(); c = {0,1}^ℓ;
post = (k ⊕ m, m)⟨1⟩ = (c, m)⟨2⟩
```

At this point, we can apply the following pRHL rule for proving equivalence of two uniformly random assignments over the same domain:

$$\frac{f \text{ is a bijection} \qquad \Psi \implies \forall x \in \{0,1\}^\ell.\ \Phi\,\{x/k\langle 1\rangle\}\,\{f(x)/c\langle 2\rangle\}}{\models k \xleftarrow{\$} \{0,1\}^\ell \sim c \xleftarrow{\$} \{0,1\}^\ell : \Psi \Rightarrow \Phi}$$

This rule is automated in EasyCrypt by the tactic `rnd`. When given as argument a single expression f as in the above proof script, `rnd` yields a new goal where the post-condition is a conjunction of two formulas universally quantified:

pre = `true`
stmt1 = `m = `$\mathcal{M}$`();`
stmt2 = `m = `$\mathcal{M}$`();`
post = $\forall x \in \{0,1\}^\ell.\ (x \oplus \mathtt{m}\langle 2\rangle) \oplus \mathtt{m}\langle 2\rangle = x \wedge (x \oplus \mathtt{m}, \mathtt{m})\langle 1\rangle = (x \oplus \mathtt{m}, \mathtt{m})\langle 2\rangle$

The first formula in the post-condition asserts that f, seen as a function of `c`, is an involution (and thus bijective). The second formula is the outcome of substituting x for $\mathtt{k}\langle 1\rangle$ and $f(x) = x \oplus \mathtt{m}\langle 2\rangle$ for $\mathtt{c}\langle 2\rangle$ in the original post-condition. Combining these two formulas under a single quantification results in a more succint goal. A similar rule could be applied to prove an equivalence between the remaining (identical) random assignments. This would leave us with a goal where the statements in both programs are empty and for which it suffices to show that the pre-condition implies the post-condition; the tactic `trivial` does all this automatically using an external solver (e.g. Alt-Ergo [**?**]) to discharge the resulting proof obligation:

$$\mathsf{true} \implies \forall x, y \in \{0,1\}^\ell.\ (x \oplus y) \oplus y = x \wedge (x \oplus y, y) = (x \oplus y, y)$$

4 The NMAC Message Authentication Code

Message Authentication Codes (MACs) are cryptographic algorithms used to provide both authenticity and data integrity in communications between two parties sharing a secret key. At an abstract level, a MAC algorithm M takes as input a key $k \in K$ and a message m, and returns a short bitstring $M(k, m)$—a tag. Given a message m and a key k, a verification algorithm can determine the validity of a tag; for stateless and deterministic MACs, this can be simply done by re-computing the tag. A MAC algorithm is deemed secure if, even after obtaining many valid tags for chosen messages, it is unfeasible to forge a tag for a fresh message without knowing the secret key k. Formally, this can be expressed in terms of the experiment EF-MAC in Figure 3 by requiring that the advantage of an adversary $\mathcal{A}$ that makes at most q queries to a MAC oracle for a freshly sampled key be negligible, where:

$$\mathbf{Adv}^{M}_{\mathsf{EF\text{-}MAC}(q)}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[\mathsf{EF\text{-}MAC} : y = M(k, x) \wedge x \notin X \wedge n \leq q\right]$$

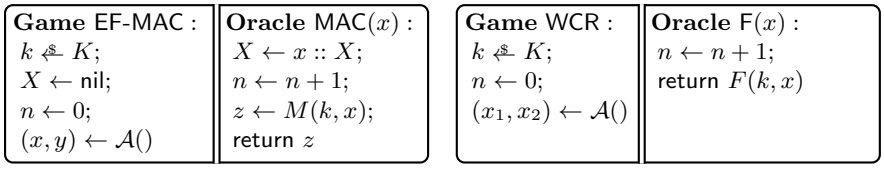


Fig. 3. Security experiments for MAC Forgery and Weak Collision Resistance

In the remainder of this section we overview the security proof of the NMAC construction [**?**]. Let ℓ and b be positive integers such that $\ell \leq b$, and let $\mathsf{pad} : \{0,1\}^* \to (\{0,1\}^b)^+$ be an injective function that pads an arbitrary length input message to a positive multiple of b. The NMAC construction transforms a secure fixed input-length MAC $f : \{0,1\}^\ell \times \{0,1\}^b \to \{0,1\}^\ell$ into a secure variable input-length MAC:

$$\begin{array}{lcl} \mathsf{NMAC} & : & (\{0,1\}^\ell \times \{0,1\}^\ell) \times \{0,1\}^* \to \{0,1\}^\ell \\ \mathsf{NMAC}((k_1,k_2),m) & \stackrel{\text{def}}{=} & F(k_1, F(k_2, m)) \end{array}$$

where $F(k,m) = f^*(k, \mathsf{pad}(m))$ and $f^* : \{0,1\}^\ell \times (\{0,1\}^b)^* \to \{0,1\}^\ell$ is the function that on input $k \in \{0,1\}^\ell$ and $x = x_1 \cdots x_n$ consisting of n b-bits blocks returns h_n, where $h_0 = k$ and $h_i = f(h_{i-1}, x_i)$ for $1 \leq i \leq n$.

The proof of security for NMAC establishes that it is no more difficult to forge a valid message for NMAC than forging a valid message for the underlying function f, viewed as a MAC, or finding a collision for the keyed function F. Formally, we define Weak Collision-Resistance for F in terms of the experiment WCR shown in Figure 3, and define the advantage of an adversary $\mathcal{A}$ making at most q queries to F as

$$\mathbf{Adv}^F_{\mathsf{WCR}(q)}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[\mathsf{WCR} : F(k,x_1) = F(k,x_2) \wedge x_1 \neq x_2 \wedge n \leq q\right]$$

Given an arbitrary adversary $\mathcal{A}$ against the security of NMAC, we exhibit two adversaries $\mathcal{A}_F$ and $\mathcal{A}_f$ such that

$$\mathbf{Adv}^{\mathsf{NMAC}}_{\mathsf{EF\text{-}MAC}(q)}(\mathcal{A}) \leq \mathbf{Adv}^F_{\mathsf{WCR}(q+1)}(\mathcal{A}_F) + \mathbf{Adv}^f_{\mathsf{EF\text{-}MAC}(q)}(\mathcal{A}_f) \tag{2}$$

Figure 4 shows the tree of games used in the proof. We start from the game encoding an attack against the security of NMAC. We then define another game EF-MAC′ that just introduces a list Y to store the intermediate values of $F(k_2, x)$ computed to answer to oracle queries, and simplifies the definition of NMAC using the identity

$$\mathsf{NMAC}((k_1,k_2),m) \stackrel{\text{def}}{=} f(k_1, \mathsf{pad}(F(k_2,m)))$$

whose validity stems from the fact that the outer application of the function F is on a message of $\ell \leq b$ bits. We prove the following judgment:

$$\models \mathsf{EFMAC} \sim \mathsf{EFMAC}' : \mathsf{true} \Rightarrow \begin{array}{l} (y = \mathsf{NMAC}((k_1,k_2),x) \wedge x \notin X \wedge n \leq q)\langle 1\rangle \iff \\ (y = f(k_1, \mathsf{pad}(F(k_2,x))) \wedge x \notin X \wedge n \leq q)\langle 2\rangle \end{array}$$

From which we have

$$\mathbf{Adv}^{\mathsf{NMAC}}_{\mathsf{EF\text{-}MAC}_q}(\mathcal{A}) = \Pr\left[\mathsf{EF\text{-}MAC}' : y = f(k_1, \mathsf{pad}(F(k_2, x))) \wedge x \notin X \wedge n \leq q\right] \quad (3)$$

We now make a case analysis on whether, when the experiment $\mathsf{EF\text{-}MAC}'$ finishes and $\mathcal{A}$ succeeds, there is a value $x' \in X$ s.t. $F(k_2, x) = F(k_2, x')$ or not. Since we are interested only in executions where $x \notin X$, to make this case analysis it suffices to check whether the value $F(k_2, x)$ is in the list Y.

- If there exists $x' \in X$ such that $F(k_2, x) = F(k_2, x')$, we exhibit an adversary $\mathcal{A}_F$ against the WCR of F that finds a collision. This is trivial: x and x' collide and are necessarily distinct because one belongs to X while the other does not;
- If there is no $x' \in X$ such that $F(k_2, x) = F(k_2, x')$, we exhibit an adversary against the MAC-security of the function f that successfully forges a tag. Indeed, if (x, y) is a forgery for NMAC, then $(\mathsf{pad}(F(k_2, x)), y)$ is a forgery for f.

We prove the following judgments:

$$\begin{aligned}
&\vDash \mathsf{EF\text{-}MAC}' \sim \mathsf{WCR}_F : \mathsf{true} \Rightarrow \\
&\quad (y = f(k_1, \mathsf{pad}(F(k_2, x))) \wedge x \notin X \wedge n \leq q \wedge F(k_2, x) \in Y)\langle 1\rangle \implies \\
&\quad (F(k, x_1) = F(k, x_2) \wedge x_1 \neq x_2 \wedge n \leq q + 1)\langle 2\rangle
\end{aligned}$$

$$\begin{aligned}
&\vDash \mathsf{EF\text{-}MAC}' \sim \mathsf{EF\text{-}MAC}_f : \mathsf{true} \Rightarrow \\
&\quad (y = f(k_1, \mathsf{pad}(F(k_2, x))) \wedge x \notin X \wedge n \leq q \wedge F(k_2, x) \notin Y)\langle 1\rangle \implies \\
&\quad (y = f(k, x) \wedge x \notin X \wedge n \leq q)\langle 2\rangle
\end{aligned}$$

From which follows

$$\begin{aligned}
&\Pr\left[\mathsf{EF\text{-}MAC}' : y = f(k_1, \mathsf{pad}(F(k_2, x))) \wedge x \notin X \wedge n \leq q\right] \leq \\
&\quad \Pr\left[\mathsf{WCR}_F : F(k, x_1) = F(k, x_2) \wedge x_1 \neq x_2 \wedge n \leq q + 1\right] + \\
&\quad \Pr\left[\mathsf{EF\text{-}MAC}_f : y = f(k, x) \wedge x \notin X \wedge n \leq q\right]
\end{aligned} \quad (4)$$

We conclude from (3) and (4) that the bound (2) holds.

We observe that the bound in [**?**, Theorem 4.1] is off-by-one: the adversary against the WCR-security of F must call the iterated hash function one more time in order to find another value x' that collides with x among the queries made by the adversary against NMAC. Thus, one must assume that the function F is secure against adversaries that make $q + 1$ rather than just q queries.

5 ElGamal Encryption

ElGamal is a public-key encryption scheme based on the Diffie-Hellman key exchange. Given a cyclic group G of order q and a generator g, its key generation, encryption, and decryption algorithms are defined as follows:

$$\begin{array}{lcl}
\mathcal{KG}() & \stackrel{\text{def}}{=} & x \xleftarrow{\$} [1, q]; \mathsf{return}\ (g^x, x) \\
\mathcal{E}(\alpha, m) & \stackrel{\text{def}}{=} & y \xleftarrow{\$} [1, q]; \mathsf{return}\ (g^y, \alpha^y \times m) \\
\mathcal{D}(x, (\beta, \zeta)) & \stackrel{\text{def}}{=} & \mathsf{return}\ (\zeta \times \beta^{-x})
\end{array}$$

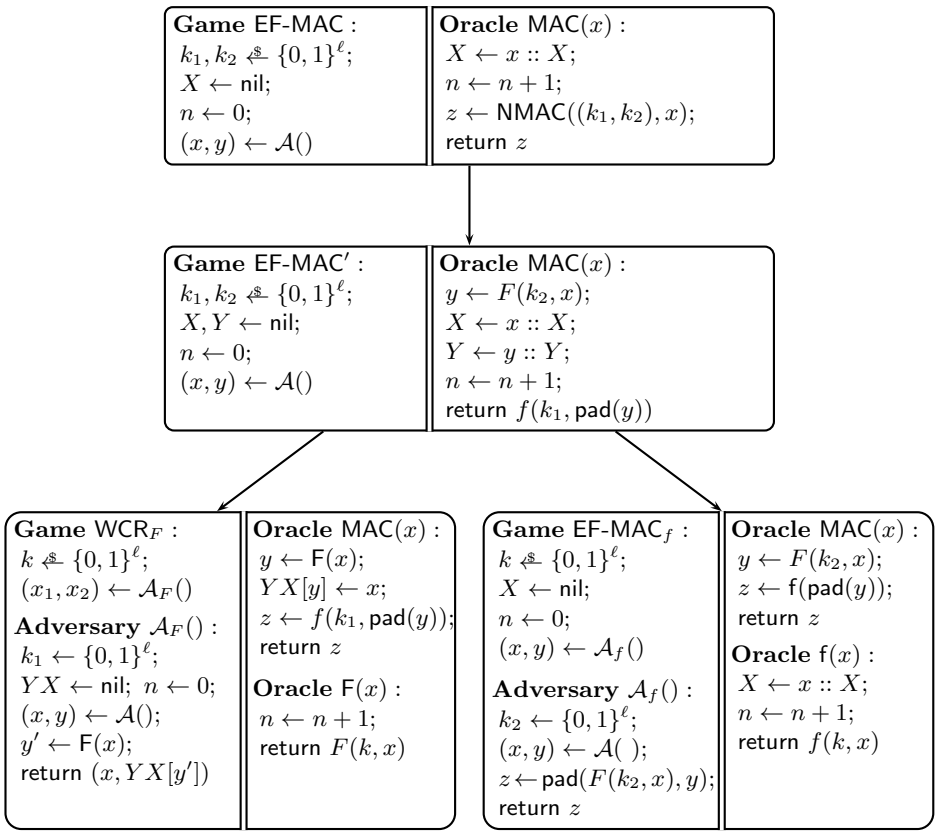


Fig. 4. Tree of games in the proof of the NMAC construction

Shoup [**?**] uses ElGamal as a running example to review some interesting points in game-based proofs. We outline a code-based proof of the indistinguishability under chosen-plaintext attacks of ElGamal by reduction to the Decision Diffie-Hellman (DDH) assumption on the underlying group G. The experiments encoding both the security goal and the assumption, were introduced before in Figures 1 and 2 and are instantiated for ElGamal in Figure 5.

Indistinguishability under chosen-plaintext attacks requires that an efficient adversary cannot distinguish, except with small probability, between two ciphertexts produced from messages of its choice. In the experiment IND-CPA, the challenger samples a fresh pair of keys using the algorithm $\mathcal{KG}$ and gives the public key pk to the adversary, who returns two plaintexts m_0, m_1 of his choice. The challenger then tosses a fair coin b and gives the encryption of m_b under pk back to the adversary, whose goal is to guess which message has been encrypted. We model an IND-CPA adversary $\mathcal{A}$ in EasyCrypt as two unspecified procedures that share state by means of an explicit state variable σ. By keeping the type of this variable abstract, we obtain a generic reduction. Using the

keyword res that denotes the return value of a procedure in EasyCrypt, we define the IND-CPA-advantage of $\mathcal{A}$ as in Fig. 1:

$$\mathbf{Adv}^{\mathsf{ElGamal}}_{\mathsf{IND\text{-}CPA}}(\mathcal{A}) \stackrel{\text{def}}{=} \left| \Pr\left[\mathsf{IND\text{-}CPA} : \mathsf{res}\right] - \frac{1}{2} \right|$$

The DDH problem consists in distinguishing between triples of the form (g^x, g^y, g^{xy}) and (g^x, g^y, g^z), where the exponents x, y, z are uniform and independently sampled from the interval $[1..\mathsf{ord}(G)]$. The DDH-advantage of an adversary $\mathcal{B}$ is defined as:

$$\mathbf{Adv}^{(G,g)}_{\mathsf{DDH}}(\mathcal{B}) \stackrel{\text{def}}{=} \left| \Pr\left[\mathsf{DDH}_0 : \mathsf{res}\right] - \Pr\left[\mathsf{DDH}_1 : \mathsf{res}\right] \right|$$

Figure 5 presents the overall structure of the reduction, showing a concrete DDH distinguisher $\mathcal{B}$ that achieves exactly the same advantage as an arbitrary IND-CPA adversary $\mathcal{A}$, with constant resource overhead.

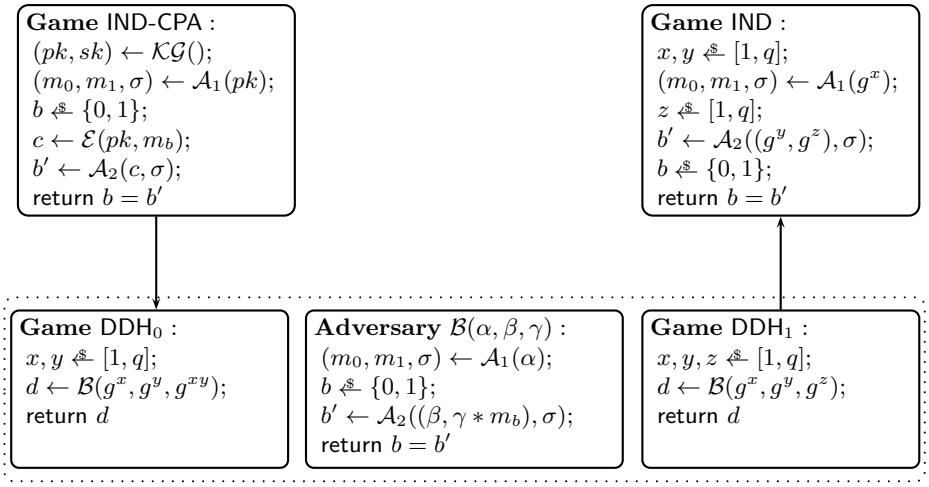


Fig. 5. Game-based proof of IND-CPA-security of ElGamal. Games DDH_0 and DDH_1, enclosed in a dotted box, share the same definition for the concrete adversary $\mathcal{B}$.

The proof requires showing the validity of two pRHL judgments. The first judgment relates the experiment IND-CPA to the instantiation of game DDH_0 with the concrete adversary $\mathcal{B}$ defined in Figure 5. We prove that the distribution of the result of the comparison $b = b'$ in game IND-CPA coincides with the distribution of d in game DDH_0, i.e.

$$\models \mathsf{IND\text{-}CPA} \sim \mathsf{DDH}_0 : \mathsf{true} \Rightarrow \mathsf{res}\langle 1\rangle = \mathsf{res}\langle 2\rangle$$

From this, we can derive the equality

$$\Pr\left[\mathsf{IND\text{-}CPA} : \mathsf{res}\right] = \Pr\left[\mathsf{DDH}_0 : \mathsf{res}\right] \tag{5}$$

The second judgment relates the game DDH_1 instantiated with the same adversary $\mathcal{B}$ to a game IND, where the guess b' of the adversary $\mathcal{A}$ no longer depends on the challenge bit b:

$$\models \mathsf{DDH}_1 \sim \mathsf{IND} : \mathsf{true} \Rightarrow \mathsf{res}\langle 1\rangle = \mathsf{res}\langle 2\rangle$$

We state and prove this judgment in EasyCrypt using the following proof script:

```
equiv DDH1_IND : DDH1.Main ~ IND.Main : true ==> res<1> = res<2>.
proof.
 inline B; swap<1> 3 2; swap<1> [5-6] 2; swap<2> 6 -2.
 auto.
 rnd ((z + log(b ? m0 : m1)) % q), ((z - log(b ? m0 : m1)) % q); trivial.
 auto.
 trivial.
save.
```

The `inline` tactic expands calls to procedures by replacing them with their definitions, performing appropriate substitutions and renaming variables if necessary. The tactic `swap` pushes a single instruction or a block of instructions down if its second arguments is positive, or up if it is negative. Dependencies are checked to verify these transformations are semantics-preserving. The tactic `rnd` f, f^{-1} applies the same rule for random assignments that we described in Section 3; except that this time we provide a function f and its inverse f^{-1} by means of justification of its bijectivity. The tactics `auto` and `trivial` implement heuristics to combine simpler tactics. For instance, the above applications of `auto` apply the `wp` transformer and tactics that implement rules for deterministic and random assignments and calls to abstract procedures. This suffices to prove the goal without any user intervention.

It follows from the above judgment that

$$\Pr\left[\mathsf{DDH}_1 : \mathsf{res}\right] = \Pr\left[\mathsf{IND} : \mathsf{res}\right] \tag{6}$$

The right-hand side of this equality is exactly $1/2$, i.e.

$$\Pr\left[\mathsf{IND} : \mathsf{res}\right] = \frac{1}{2} \tag{7}$$

This can be proven by direct computation:

```
claim Fact : IND.Main[res] = 1%r / 2%r by compute.
```

We conclude putting the above equations (5)–(7) together that

$$\mathbf{Adv}^{(G,g)}_{\mathsf{DDH}}(\mathcal{B}) = \mathbf{Adv}^{\mathsf{ElGamal}}_{\mathsf{IND\text{-}CPA}}(\mathcal{A})$$

6 Conclusion

EasyCrypt is a framework for computer-aided cryptographic proofs. It improves confidence in cryptographic systems by delivering formally verified proofs that they achieve their purported goals. In this paper, we have illustrated how EasyCrypt can be used to verify elementary examples. In other works, we have applied EasyCrypt to a range of emblematic examples, including asymmetric encryption schemes [**?**], signature schemes [**?**], hash function designs [**?**, **?**], and modes of operation for block ciphers. We conclude this article with a review of some topics that deserve further attention. Other topics, not developed below, include the automated synthesis of cryptographic schemes, and the development of more expressive relational program logics for probabilistic programs.

Compositionality. Compositionality and abstraction are fundamental principles in programming language semantics. They are supported by notions such as modules, which are key to structure large software developments. In contrast, it has proved extremely intricate to design general and sound abstraction and compositionality mechanisms for cryptographic proofs. For instance, a recent analysis [**?**] of the limitations of the indifferentiability framework [**?**] illustrates the difficulty of instantiating generic proofs to specific constructions. We believe that the code-based approach provides an excellent starting point for developing sound compositional reasoning methods, and that these methods can be incorporated into EasyCrypt.

Automation. EasyCrypt provides automated support to prove the validity of pRHL judgments and to derive inequalities about probability quantities. However, it does not implement any sophisticated mechanism to help users discover or build intermediate games in a game-based proof. It would be interesting to investigate whether one can develop built-in strategies that capture common patterns of reasoning in cryptographic proofs, and generate proof skeletons including the corresponding games and pRHL judgments. A more ambitious goal would be to enhance EasyCrypt with a language for programming strategies, in the way proof assistants such as Coq allow users to program their own tactics.

Certification and mathematical libraries. EasyCrypt was conceived as a front-end to the CertiCrypt framework. In [**?**], we report on a proof-producing mechanism that converts EasyCrypt files into Coq files that can be machine-checked in the CertiCrypt framework. Certification remains an important objective, although the proof-producing mechanism may fall temporarily out of sync with the development of EasyCrypt. As cryptographic constructions and proofs rely on a wide range of mathematical concepts, the further development of extensive libraries of formalized mathematics is an essential stepping stone towards this goal.