An abridged version of this paper is under submission. This is the full version.

Automated Analysis and Synthesis of Padding-Based Encryption Schemes

Gilles Barthe, Juan Manuel Crespo, César Kunz[*] Benjamin Grégoire[†]
Yassine Lakhnech[‡] Santiago Zanella-Béguelin[§]

December 2012

Abstract

Verifiable security is an emerging approach in cryptography that advocates the use of principled tools for building machine-checked security proofs of cryptographic constructions. Existing tools following this approach, such as EasyCrypt or CryptoVerif, fall short of finding proofs automatically for many interesting constructions. In fact, devising automated methods for analyzing the security of large classes of cryptographic constructions is a long-standing problem which precludes a systematic exploration of the space of possible designs. This paper addresses this issue for padding-based encryption schemes, a class of public-key encryption schemes built from hash functions and trapdoor permutations, which includes widely used constructions such as RSA-OAEP.

Firstly, we provide algorithms to search for proofs of security against chosen-plaintext and chosen-ciphertext attacks in the random oracle model. These algorithms are based on domain-specific logics with a computational interpretation and yield quantitative security guarantees; for proofs of chosen-plaintext security, we output machine-checked proofs in EasyCrypt. Secondly, we provide a crawler for exhaustively exploring the space of padding-based encryption schemes under user-specified restrictions (e.g. on the size of their description), using filters to prune the search space. Lastly, we provide a calculator that computes the security level and efficiency of provably secure schemes that use RSA as trapdoor permutation.

Using these three tools, we explore over 1.3 million encryption schemes, including more than 100 variants of OAEP studied in the literature, and prove chosen-plaintext and chosen-ciphertext security for more than 250,000 and 17,000 schemes, respectively.

[*] IMDEA Software Institute, Spain. E-mail: {gilles.barthe,juanmanuel.crespo,cesar.kunz}@imdea.org
[†] INRIA Sophia Antipolis – Méditerranée, France. E-mail: benjamin.gregoire@inria.fr
[‡] Université de Grenoble, VERIMAG, France. E-mail: yassine.lakhnech@imag.fr
[§] Microsoft Research, UK. E-mail: santiago@microsoft.com

Contents

1 Introduction 3

2 Cryptographic Background 5

2.1 The pWHILE language . . . 5

2.2 Public-Key Encryption and Security . . . 5

2.3 Trapdoor Permutations . . . 6

2.4 Paddings as Generic Transformations . . . 7

2.5 The OAEP Encryption Scheme . . . 7

3 Algebraic View of Padding-Based Schemes 8

3.1 Expressions . . . 8

3.2 Deducibility . . . 9

4 Chosen-Plaintext Security 9

4.1 Judgments . . . 9

4.2 Proof System . . . 10

4.3 Soundness and Generation . . . 11

4.4 Proof Search . . . 11

4.5 Example . . . 12

5 Chosen-Ciphertext Security 14

5.1 Algorithms . . . 14

5.2 Observational Equivalence . . . 14

5.3 Judgments . . . 15

5.4 Proof System . . . 15

5.5 Soundness . . . 16

5.6 Proof Search . . . 16

5.7 Example . . . 16

6 Exploration 18

6.1 Filtering Incorrect and Insecure Schemes . . . 18

6.2 Synthesis of Decryption Algorithms . . . 19

6.3 Proving (In)Security . . . 19

6.4 Synthesis of Candidate Schemes . . . 20

7 Practical Interpretation 21

8 Experimental Results 21

8.1 Analysis of Variants of OAEP and SAEP . . . 22

8.2 Analysis of the Crawler . . . 24

9 Related Work 25

10 Future Work and Conclusions 26

1 Introduction

Practice-oriented provable security [6,31] provides a variety of rigorous mathematical methods and heuristics to prove the security of cryptographic constructions. Proofs constructed following this approach can be used to measure the efficiency and level of resistance against attacks of concrete constructions and, more importantly, to guide the choice of practical parameters (e.g. how large a key should be used?). Yet, practice-oriented provable security has not realized its full potential, partially because analyzing each individual construction is a tedious, time-consuming process.

The vision driving our research is that practice-oriented provable security would benefit greatly from automated tools that allow cryptographers to undertake systematic analyses of large classes of cryptographic constructions, and help security practitioners to choose the construction that best fits their needs.

Our vision is partly influenced by Landin's celebrated account of the next 700 programming languages [24], which can be transposed, perhaps with greater vividness, to cryptographic constructions:

> The question arises, do the idiosyncracies reflect [...] the situations that are being catered for? Or are they accidents of history and personal background that may be obscuring fruitful developments? [...] We must think in terms, not of languages, but of families of languages. That is to say we must systematize their design so that a new language is a point chosen from a well-mapped space, rather than a laboriously devised construction.

To realize this vision, it is necessary to develop automated tools that systematize the existing body of knowledge in cryptography, whereby a large number of cryptographic constructions can be explored and classified according to their characteristics. However, the development of such tools raises significant challenges, mainly because the automated analysis of cryptographic constructions in the computational model is a long-standing problem. Even if automated analysis methods were readily available, devising fast and effective methods to explore large design spaces of cryptographic constructions is currently out of the reach of existing synthesis tools.

In this paper, we address these challenges for the relatively simple but practically relevant class of padding-based encryption schemes, a class of public-key encryption schemes built from one-way trapdoor permutations and hash functions. Concretely, we provide:

1. Domain-specific logics for proving security against chosen-plaintext and chosen-ciphertext attacks in the random oracle model (modeling hash functions as perfectly random functions);

2. Effective proof search algorithms that automatically output valid derivations for these logics and, in the case of chosen-plaintext security, a certificate in the form of an independently verifiable proof in EasyCrypt [3];

3. A crawler which explores the space of padding-based encryption schemes under user-specified restrictions (e.g. on the size of their descriptions). The crawler uses filters to prune large parts of the search space and applies the proof search algorithms to find proofs of the remaining;

4. A calculator that computes from a security bound output by the crawler, an estimate of the minimum key length needed to reach a target security level, and constraints on other parameters of the scheme (e.g. the output-length of hash functions).

Logics, proof search and proof generation

Our first contribution is a pair of domain-specific logics for proving chosen-plaintext and chosen-ciphertext security of padding-based encryption schemes in the random oracle model. Both logics are computationally sound and are supported by proof search algorithms that find automatically valid derivations.

The proof search algorithms are extremely simple, yet effective: they apply the rules of the logic according to a specific strategy and backtrack to the last point of choice when the strategy fails to find a proof. The key to their effectiveness is the focus of the logics, which feature a minimal set of rules to reason about a restricted class of judgments.

The minimalism of the logics stands in sharp contrast with the approach implemented by interactive tools like EasyCrypt and CertiCrypt [4,5], where judgments are arbitrary statements of a relational program logic, and arbitrary reasoning steps are allowed provided they can be justified by the logic. However, both approaches are complementary; to illustrate this point, we instrument the algorithm that searches for proofs of chosen-plaintext security to extract EasyCrypt proofs that can be verified independently. This instrumentation provides another machine-supported validation of the soundness of the proof search algorithm, and allows it to remain outside of the Trusted Computing Base.

Exploration

Our second contribution is a crawler to explore the design space of padding-based encryption schemes; the crawler combines the proof search algorithms described above with a lazy strategy for generating schemes. We use filters to prune large portions of the search space. Most of the filters we use are based on a deducibility relation that is reminiscent from symbolic cryptography. In particular, deducibility is used to test whether a scheme (or a collection of them) admits an efficient decryption algorithm and is resistant against various attacks. Put together, these filters are highly effective and cut off over 95% of the search space; the more expensive proof search algorithms are only applied to the remainder.

Practical interpretation

Whenever our algorithms can prove a scheme secure, they output a concrete security bound that can be used as a guide to choose practical parameters for implementations. As an example, we consider instances of secure schemes that use RSA as the underlying trapdoor permutation, and compute, for different choices of parameters, their level of security, bandwidth and ciphertext overhead (respectively, the ratio and difference between the length of an encryption of a message and the message itself).

Evaluation

We demonstrate the effectiveness of our tools by synthesizing over 1.3 million schemes, including more than 100 variants of OAEP previously studied in the literature. We apply the proof search algorithms and determine that over 250,000 of these are secure against chosen-plaintext attacks, whereas 17,000 are also secure against chosen-ciphertext attacks. Our tools perform well; in a modern workstation the proof search algorithms can process one scheme within seconds, while the average time for verifying a generated proof in EasyCrypt is around one minute. Our largest experiment, which uses all tools combined and processes about one million potential schemes, takes approximately one day.

Structure of the paper

The paper is structured as follows: Section 2 gives some background on public-key encryption schemes and their security; Section 3 introduces the algebraic language of randomized expressions and the deducibility relations that are the base of our logics and filters; Sections 4 and 5 describe logics for proving chosen-plaintext and chosen-ciphertext security and their automation; Sections 6 overviews the symbolic filters and the crawler, while Section 7 discusses the practical interpretation of security bounds; Section 8 evaluates the performance of our tools on synthesized schemes and on schemes from the literature. Finally, Section 9 surveys related work and Section 10 suggests directions for further work.

2 Cryptographic Background

We follow a code-based approach and represent cryptographic constructions and security experiments as programs in the procedural probabilistic programming language pWhile [5]. This section describes the pWhile language, as well as public-key encryption schemes, trapdoor permutations, and their security. We also introduce the OAEP padding-based encryption scheme, which serves as a running example throughout the rest of the paper.

2.1 The pWhile language

We consider the following restricted grammar of commands operating on bitstrings:

$$\begin{array}{rcll}
\mathcal{C} & ::= & \mathcal{V} \leftarrow \mathcal{T} & \text{deterministic assignment} \\
 & | & \mathcal{V} \xleftarrow{\$} \{0,1\}^{\ell} & \text{uniform random sampling} \\
 & | & \mathsf{if}\ \mathcal{T}\ \mathsf{then}\ \mathcal{C}\ \mathsf{else}\ \mathcal{C} & \text{conditional} \\
 & | & \mathsf{while}\ \mathcal{T}\ \mathsf{do}\ \mathcal{C} & \text{loop} \\
 & | & \mathcal{V} \leftarrow \mathcal{P}(\mathcal{T}, \dots, \mathcal{T}) & \text{procedure call} \\
 & | & \mathcal{C};\ \mathcal{C} & \text{sequence}
\end{array}$$

where $\mathcal{V}$ ranges over variable identifiers, $\mathcal{P}$ over procedure identifiers, and $\mathcal{T}$ over (deterministic) expressions. A program is composed of a set of constants and global variables together with a collection of procedures. Programs in pWhile are strongly typed: a typing environment assigns types to constants and global variables and signatures to procedures. Procedures may be given a definition of the form $P(\vec{\mathcal{V}}) \stackrel{\text{def}}{=} \mathcal{C}; \mathsf{return}\ \mathcal{T}$ or left abstract. Quantification over adversaries in proofs is achieved by representing them as undefined procedures parameterized by a set of oracles (i.e. other procedures) they may call.

Programs operate on typed memories, which map variables to values of the appropriate type. The semantics of a program is a function from memories to probability sub-distributions on memories; we refer the reader to [5] for a detailed description of the semantics of pWhile. Events are first-order formulae over constants and program variables; we denote $\Pr[c : E]$ the probability of event E w.r.t. the distribution resulting from executing c in an initial memory mapping variables to default values.

2.2 Public-Key Encryption and Security

A public-key encryption scheme is a triple of algorithms $(\mathcal{KG}, \mathcal{E}, \mathcal{D})$:

Key Generation The key generation algorithm $\mathcal{KG}$ outputs a pair of keys (pk, sk); pk is a *public-key* used for encryption, sk is a *secret-key* used for decryption;

Encryption Given a public-key pk and a message m, $\mathcal{E}_{pk}(m)$ outputs a ciphertext c;

Decryption Given a secret-key sk and a ciphertext c, $\mathcal{D}_{sk}(c)$ outputs either a message m or a distinguished value $\perp$ denoting failure.

We require that for every pair of keys (pk, sk) output by the key generation algorithm and every message m, $\mathcal{D}_{sk}(\mathcal{E}_{pk}(m)) = m$ holds. In the remainder, we refer to schemes for which this condition does not necessarily hold as pseudo encryption schemes. Moreover, we refer to pairs of algorithms $(\mathcal{KG}, \mathcal{E})$ as encryption algorithms.

We consider two standard security notions for public-key encryption schemes: chosen-plaintext security, or IND-CPA, and adaptive chosen-ciphertext security, or IND-CCA. Both can be described succinctly by the following probabilistic experiment, or game, in which an adversary $\mathcal{A}$ represented as a pair of procedures $(\mathcal{A}_1, \mathcal{A}_2)$ interacts with a challenger:

$$\begin{array}{l} (pk, sk) \leftarrow \mathcal{KG}(); \\ (m_0, m_1, \sigma) \leftarrow \mathcal{A}_1(pk); \\ b \xleftarrow{\$} \{0,1\};\ c^\star \leftarrow \mathcal{E}_{pk}(m_b); \\ \bar{b} \leftarrow \mathcal{A}_2(c^\star, \sigma) \end{array}$$

The challenger first generates a fresh pair of keys (pk, sk) and gives the public key pk to the adversary, which returns a pair of messages (m_0, m_1) of its choice. The challenger then samples uniformly a bit b, encrypts the message m_b and gives the resulting ciphertext $c^\star$ (the challenge) to the adversary. Finally, the adversary outputs a guess $\bar{b}$ for b; observe that $\mathcal{A}_1$ can communicate with $\mathcal{A}_2$ through the state variable σ. We call this basic experiment CPA; the CCA experiment is defined similarly, except that the adversary $\mathcal{A}$ is given access to a decryption oracle $\mathcal{D}_{sk}(\cdot)$, with the proviso that $\mathcal{A}_2$ cannot ask for the decryption of the challenge ciphertext $c^\star$.

The security of a scheme is measured by the advantage of an adversary playing against the CPA or CCA game.

Definition 1 (Adversary Advantage)**.** *The advantage of an adversary $\mathcal{A}$ against the IND-CPA and IND-CCA security of an encryption scheme $\Pi = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ is defined respectively as:*

$$\begin{array}{rcl} \mathbf{Adv}_{\Pi}^{\mathsf{CPA}}(\mathcal{A}) & \stackrel{def}{=} & \left| \Pr\left[\mathsf{CPA} : \bar{b} = b\right] - \dfrac{1}{2} \right| \\ \mathbf{Adv}_{\Pi}^{\mathsf{CCA}}(\mathcal{A}) & \stackrel{def}{=} & \left| \Pr\left[\mathsf{CCA} : \bar{b} = b\right] - \dfrac{1}{2} \right| \end{array}$$

We define $\mathbf{Adv}_{\Pi}^{\mathsf{CPA}}(t)$ (resp. $\mathbf{Adv}_{\Pi}^{\mathsf{CCA}}(t)$) as the maximal CPA (resp. CCA) advantage over all adversaries $\mathcal{A}$ that execute within time t.

2.3 Trapdoor Permutations

Trapdoor permutations are functions that are easy to compute, but hard to invert without knowing the corresponding trapdoor.

Definition 2 (Trapdoor Permutation)**.** *A family of trapdoor permutations on $\{0,1\}^n$ is a triple of algorithms $(\mathcal{KG}, f, f^{-1})$. The key generation algorithm $\mathcal{KG}$ is probabilistic, whereas f and f^{-1} are deterministic. For any pair of keys (pk, sk) output by the key generation algorithm $\mathcal{KG}$, the algorithms f_{pk} and f_{sk}^{-1} must be permutations on $\{0,1\}^n$ and inverse of each other.*

The security of a family of trapdoor permutations is measured by the probability that an adversary (partially) inverts the image of a randomly sampled value.

Definition 3 (One-Way Trapdoor Permutation)**.** *Let $\Theta = (\mathcal{KG}, f, f^{-1})$ be a family of trapdoor permutations on $\{0,1\}^\ell$ and let $k \leq \ell$. The success probability of an algorithm $\mathcal{I}$ in partially q-inverting Θ on its k most significant bits is defined as*

$$\mathbf{Succ}_{\Theta}^{\mathsf{OW}_k^q}(\mathcal{I}) \overset{def}{=} \Pr\left[\mathsf{OW}_k : s \in S \wedge |S| \leq q\right]$$

where OW_k is the experiment:

$$\begin{array}{l} (pk, sk) \leftarrow \mathcal{KG}(); \\ s \overset{\$}{\leftarrow} \{0,1\}^k;\ t \overset{\$}{\leftarrow} \{0,1\}^{\ell-k}; \\ S \leftarrow \mathcal{I}(f_{pk}(s \,\|\, t)) \end{array}$$

We define $\mathbf{Succ}_{\Theta}^{\mathsf{OW}_k^q}(t)$ as the maximal value of $\mathbf{Succ}_{\Theta}^{\mathsf{OW}_k^q}(t)$ over all inverters $\mathcal{I}$ executing within time t. We omit the subscript k when $k = \ell$; likewise, we omit the superscript q when $q = 1$.

2.4 Paddings as Generic Transformations

Paddings are generic randomized transformations that convert trapdoor permutations into encryption schemes using hash functions and basic operations, such as bitwise exclusive-or $\oplus$ and concatenation $\|$ on bitstrings. Security proofs for paddings are traditionally in the random oracle model, and assume that hash functions are fixed input-length functions sampled uniformly at the onset of the experiment.

The main results of this paper (Theorems 1 and 3) are proved w.r.t. an interpretation of security experiments as programs. In this interpretation, trapdoor permutations are modeled by an abstract procedure $\mathcal{KG}_f$ together with two operators f and f^{-1}, whereas random oracles are modeled as stateful procedures that respond to queries by sampling answers on demand. A random oracle H maintains a list $\boldsymbol{L}_H$ of previous queries; adversaries are given indirect access to H via a wrapper that stores queries in a list $\boldsymbol{L}_H^{\mathcal{A}}$ and limits the number of distinct queries to some parameter q_H.

The denotational semantics of pWhile programs allows mapping the guarantees derived from these theorems to fully instantiated schemes, in which all parameters, including the trapdoor permutation and the allotted number of adversarial queries to each oracle are resolved.

2.5 The OAEP Encryption Scheme

We use the OAEP scheme as a running example throughout the paper. OAEP is a padding scheme introduced by Bellare and Rogaway [9]. To date, RSA-OAEP, which instantiates OAEP with RSA as trapdoor permutation, remains the most widely deployed public-key encryption scheme and is recommended by several international standards.

Definition 4 (OAEP)**.** *Let $(\mathcal{KG}_f, f, f^{-1})$ be a family of trapdoor permutations on $\{0,1\}^n$, $\rho \in \mathbb{N}$, and let*

$$G : \{0,1\}^k \to \{0,1\}^\ell \qquad H : \{0,1\}^\ell \to \{0,1\}^k$$

be two hash functions such that $n = k + \ell$ and $\rho < \ell$. OAEP is composed of the following triple of algorithms:

$$\begin{array}{lcl} \mathcal{KG}() & \overset{def}{=} & (pk, sk) \leftarrow \mathcal{KG}_f();\ \mathsf{return}\ (pk, sk) \\ \mathcal{E}_{pk}(m) & \overset{def}{=} & r \overset{\$}{\leftarrow} \{0,1\}^k;\ s \leftarrow G(r) \oplus (m \,\|\, 0^\rho);\ t \leftarrow H(s) \oplus r;\ \mathsf{return}\ f_{pk}(s \,\|\, t) \\ \mathcal{D}_{sk}(c) & \overset{def}{=} & s \,\|\, t \leftarrow f_{sk}^{-1}(c);\ r \leftarrow t \oplus H(s);\ m \leftarrow s \oplus G(r); \\ & & \mathsf{if}\ [m]_{\ell-\rho}^{\rho} = 0^\rho\ \mathsf{then\ return}\ [m]_0^{\ell-\rho}\ \mathsf{else\ return}\ \perp \end{array}$$

$$\begin{array}{rcll} e & ::= & x & \text{variable} \\ & | & r & \text{uniformly random bitstring} \\ & | & 0^s & \text{zero bitstring of length } s \\ & | & 1^s & \text{one bitstring of length } s \\ & | & \mathsf{f}(e) & \text{permutation} \\ & | & \mathsf{f}^{-1}(e) & \text{inverse permutation} \\ & | & H(e) & \text{hash function} \\ & | & e \oplus e & \text{exclusive-or} \\ & | & e \,\|\, e & \text{concatenation} \\ & | & [e]_n^\ell & \text{projection} \end{array}$$

where $s, \ell \in \mathcal{S}$, $x \in \mathcal{X}$, $r \in \mathcal{R}$, and $H \in \mathcal{H}$.

Figure 1: Expressions

where $[m]_n^\ell$ denotes the bitstring obtained from m by dropping the n most significant bits and taking the ℓ most significant bits of the result.

3 Algebraic View of Padding-Based Schemes

We introduce algebraic expressions to model padding-based encryption schemes, and provide an interpretation of expressions as pWhile procedures. Moreover, we introduce deducibility relations that model the knowledge that either an adversary or a legitimate user can derive from an expression.

3.1 Expressions

We assume given sets $\mathcal{S}_0$ of (strictly positive) size variables, $\mathcal{H}$ of hash function identifiers, $\mathcal{R}$ of random seeds, and $\mathcal{X}$ of variables. We let $\mathcal{S}$ denote the set of linear combinations over $\mathcal{S}_0$, and we assume given a distinguished variable m, called the plaintext. Expressions are drawn from the grammar of Figure 1.[1]

We let $\mathcal{R}(e)$ and $\mathcal{V}(e)$ denote, respectively, the set of random seeds and the set of variables occurring in e, and $e\,\{\vec{e}_1/\vec{e}_0\}$ denote the simultaneous substitution of $\vec{e}_0$ by $\vec{e}_1$ in e. Well-formedness of expressions is enforced using a size-based type system, whose types are elements of $\mathcal{S}$. Typing rules are as expected.

A well-formed expression e has a natural interpretation $\langle e \rangle$ as pWhile procedure. The interpretation of every expression is parameterized by a family of trapdoor permutations $\Theta = (\mathcal{KG}_f, f, f^{-1})$ and a pair of keys (pk, sk) generated using $\mathcal{KG}_f$; f is interpreted as f_{pk} while f^{-1} is interpreted as f_{sk}^{-1}. Moreover, the interpretation treats each $H \in \mathcal{H}$ as a random oracle. An expression e is translated into a pWhile procedure $\langle e \rangle$ by performing a left-to-right evaluation of expressions, and allowing sub-expressions to share their computations. For instance, the encryption algorithm of OAEP may be represented as the expression:

$$\mathsf{f}((G(r) \oplus (m \,\|\, 0)) \,\|\, H(G(r) \oplus (m \,\|\, 0)) \oplus r)$$

whose interpretation is the following procedure $\mathcal{E}(m)$:

$$r \xleftarrow{\$} \{0,1\}^k;\ g \leftarrow G(r);\ s \leftarrow g \oplus (m \,\|\, 0);\ h \leftarrow H(s);\ \mathtt{return}\ f_{pk}(s \,\|\, (h \oplus r))$$

The interpretation of e also induces an encryption algorithm $\Pi_{\langle e \rangle} \stackrel{\text{def}}{=} (\mathcal{KG}_f, \langle e \rangle)$.

[1] Although our presentation focuses on expressions that use a single trapdoor permutation and contain a single plaintext variable; our tools do not impose these restrictions.

$$\frac{}{e \vdash e}[\mathsf{Refl}] \qquad \frac{}{e \vdash 0}[\mathsf{Z}] \qquad \frac{}{e \vdash 1}[\mathsf{O}] \qquad \frac{e \vdash e'}{e \vdash H(e')}[\mathsf{H}] \qquad \frac{e \vdash e_1 \quad e \vdash e_2}{e \vdash e_1 \,\|\, e_2}[\mathsf{Conc}] \qquad \frac{e \vdash e_1 \qquad e \vdash e_2}{e \vdash e_1 \oplus e_2}[\mathsf{Xor}]$$

$$\frac{e \vdash e}{e \vdash [e]^{\ell}_{n}}[\mathsf{Proj}] \qquad \frac{e \vdash e_1 \qquad \vdash e_1 = e_2}{e \vdash e_2}[\mathsf{Conv}] \qquad \frac{e \vdash e'}{e \vdash \mathsf{f}(e')}[\mathsf{F}] \qquad \boxed{\frac{e \vdash e'}{e \vdash \mathsf{f}^{-1}(e')}[\mathsf{Finv}]}$$

Figure 2: Deducibility relations

$$(x \oplus y) \oplus z = x \oplus (y \oplus z) \quad x \oplus y = y \oplus x \quad x \oplus 0 = x \quad x \oplus x = 0$$

$$(x \,\|\, y) \,\|\, z = x \,\|\, (y \,\|\, z) \quad (x \,\|\, y) \oplus z = (x \oplus z) \,\|\, (y \oplus z) \quad \mathsf{f}(\mathsf{f}^{-1}(x)) = x \quad \mathsf{f}^{-1}(\mathsf{f}(x)) = x$$

$$[x \,\|\, y]^{|x|}_{0} = x \quad [x \,\|\, y]^{\ell}_{n+|x|} = [y]^{\ell}_{n} \quad [x \oplus y]^{\ell}_{n} = [x]^{\ell}_{n} \oplus [y]^{\ell}_{n} \quad [x]^{|x|}_{0} = x \quad [[x]^{\ell}_{n}]^{\ell'}_{n'} = [x]^{\ell'}_{n+n'} \quad [x]^{\ell}_{n} \,\|\, [x]^{\ell'}_{n+\ell} = [x]^{\ell+\ell'}_{n}$$

Figure 3: Equational theory of bitstrings

3.2 Deducibility

The knowledge that either an adversary or a legitimate recipient can derive from an algebraic expression is modeled by two deducibility relations $\vdash^{\mathcal{A}}$ and $\vdash$. These deducibility relations are used for multiple purposes. They are used by the CPA and CCA logics to capture side conditions in proof rules, and by the proof search algorithm for the CCA logic to decide the best strategy for applying rules. In addition, they are used by the crawler to detect schemes that are incorrect or vulnerable to simple attacks.

Definition 5 (Deducibility)**.** *The relation $\vdash$ (resp. $\vdash^{\mathcal{A}}$) is the smallest relation closed under the rules of Figure 2 (resp. all rules except* [Finv]*), where $\vdash e = e'$ is the smallest congruence relation that contains all instances of the axioms of Figure 3.*

Note that any derivation $e \vdash e'$ induces an algorithm that transforms an output of $\langle e \rangle$ into an output of $\langle e' \rangle$. This algorithm is deterministic except for oracle calls, and takes as input both the public and secret key of the underlying trapdoor permutation; for derivations of the form $e \vdash^{\mathcal{A}} e'$ the public key suffices. We write $e \vdash_t e'$ if the induced algorithm executes within time t, where t is drawn from the grammar in Figure 4.

4 Chosen-Plaintext Security

This section presents a computationally sound proof system for chosen-plaintext security, and a proof search algorithm that applies the rules of the proof system in a prescribed order. The proof search algorithm is certifying, in the sense that, when it finds a proof, it outputs an EasyCrypt script that can be verified independently.

4.1 Judgments

Judgments are of the form $\vdash c^{\star} :_p \varphi$, where $c^{\star}$ is an algebraic expression, φ is an event, and p a probability bound built from the grammar of Figure 4. Informally, such a judgment states that p is an upper bound for the probability of φ in the CPA game for the encryption algorithm $\Pi_{\langle c^{\star} \rangle}$.

$$
\begin{array}{lcl}
p & ::= & p+p \mid p \times p \mid q \mid 2^{-k} \mid \frac{1}{2} \mid \mathbf{Succ}_{\Theta}^{\mathsf{OW}_k^q}(t) \\
t & ::= & t+t \mid t \times t \mid q \mid t_{\mathcal{A}} \mid t_f \\
q & ::= & q+q \mid q \times q \mid q_H
\end{array}
$$

where $k \in \mathcal{S}$, $H \in \mathcal{H}$, and t_f and $t_{\mathcal{A}}$ represent the execution time of f_{pk} and of an adversary $\mathcal{A}$, respectively.

Figure 4: Probability and time bounds

$$
\frac{m \notin \mathcal{V}(c^\star)}{\vdash c^\star :_{\frac{1}{2}} \mathsf{Guess}}[\mathsf{Indep}] \qquad \frac{e \vdash \vec{r} \qquad \vec{r} \cap \mathcal{R}(c^\star) = \emptyset \qquad \vec{r} \text{ distinct}}{\vdash c^\star :_{q_H\ 2^{-|\vec{r}|}} e \in \boldsymbol{L}_H^{\mathcal{A}}}[\mathsf{Indom}]
$$

$$
\frac{e \vdash_t^{\mathcal{A}} [\vec{r}]_0^{\ell} \qquad e' = c^\star\,\{0/\mathsf{f}(\vec{r})\} \qquad \vec{r} \text{ distinct} \qquad \mathsf{f}(\vec{r}) \,\|\, \mathcal{R}(e') \,\|\, \mathcal{V}(c^\star) \vdash_{t'}^{\mathcal{A}} c^\star \qquad \vec{r} \cap \mathcal{R}(e') = \emptyset}{\vdash c^\star :_{\mathbf{Succ}_{\Theta}^{\mathsf{OW}_\ell^{q_H}}(t_{\mathcal{A}} + \sum_i q_{H_i} + t \cdot q_H + t')} e \in \boldsymbol{L}_H^{\mathcal{A}}}[\mathsf{OW}]
$$

$$
\frac{\vdash c^\star :_p \varphi \qquad r \notin \mathcal{R}(e)}{(\vdash c^\star :_p \varphi)\,\{e \oplus r, r/r, e \oplus r\}}[\mathsf{OptS}] \qquad \frac{\vdash c^\star :_p \varphi}{(\vdash c^\star :_p \varphi)\,\{\mathsf{f}^{-1}(r), r/r, \mathsf{f}(r)\}}[\mathsf{Perm}]
$$

$$
\frac{\vdash c^\star :_p \varphi \qquad \vdash c^\star :_{p'} e \in \boldsymbol{L}_H^{\mathcal{A}} \qquad r \text{ fresh in } e}{(\vdash c^\star :_{p+p'} \varphi)\,\{H(e)/r\}}[\mathsf{Fail}]
$$

where:

$$
(\vdash c^\star :_p \mathsf{Guess})\,\{\vec{e}_1/\vec{e}_0\} \overset{\text{def}}{=} \vdash c^\star\,\{\vec{e}_1/\vec{e}_0\} :_p \mathsf{Guess} \quad \text{and} \quad (\vdash c^\star :_p e' \in \boldsymbol{L}_H^{\mathcal{A}})\,\{\vec{e}_1/\vec{e}_0\} \overset{\text{def}}{=} \vdash c^\star\,\{\vec{e}_1/\vec{e}_0\} :_p e'\,\{\vec{e}_1/\vec{e}_0\} \in \boldsymbol{L}_H^{\mathcal{A}}
$$

Figure 5: Proof rules of the CPA logic

Only two kinds of events are considered: φ is either Guess, denoting the event that the CPA adversary guesses correctly the hidden bit b, or else is of the form $e \in \boldsymbol{L}_H^{\mathcal{A}}$, meaning that the adversary has issued the query $H(e)$.

4.2 Proof System

The rules of the logic are given in Figure 5. The logic features a minimal set of rules that embody the reasoning principles used to prove chosen-plaintext security of padding-based schemes.

The rules [OptS] and [Perm] correspond to bridging steps in a game-based setting, and allow transforming a judgment into an equivalent one. The former allows to swap all occurrences of a random seed r and an expression of the form $e \oplus r$, provided that $r \notin \mathcal{R}(e)$; whereas the latter allows to replace simultaneously all occurrences of r and $\mathsf{f}(r)$ by $\mathsf{f}^{-1}(r)$ and r, respectively. As random seeds are uniformly distributed, such modifications preserve the semantics of judgments.

The rules [Indep], [Indom], and [OW] provide a means of directly bounding the probability of an event. The first two rules are information-theoretic, whereas the third rule formalizes a reduction step. Rule [Indep] states that if the challenge is independent from m_b, and hence, from b, then the probability that an adversary correctly guesses the bit b in the CPA game is at most $1/2$ (it is actually exactly $1/2$). Rule [Indom] upper bounds the probability that an adversary queries some expression e to a random oracle H by q_H/ϵ, where ϵ measures the Shannon entropy of e given all random seeds in $c^\star$. Rule [OW] upper bounds the probability that a CPA adversary issues a query $H(e)$ in terms of the success probability of an inverter against the underlying trapdoor permutation Θ. Suppose an adversary $\mathcal{A}$ can compute from e the most significant ℓ bits of a vector of distinct random seeds $\vec{r}$, which only occur in $c^\star$ under f, and that it can reconstruct $c^\star$ given $f(\vec{r})$ and all other random seeds and plaintext variables in $c^\star$. We can then build an inverter that given $\mathsf{f}(\vec{r})$, simulates consistently a CPA experiment for $\mathcal{A}$, and succeeds in recovering the ℓ most significant bits of ℓ whenever $\mathcal{A}$ queries $H(e)$.

Finally, the rule [Fail], for *equivalence up to failure*, corresponds to a specialized form of Shoup's Fundamental Lemma [34]. Formally, this rule allows to replace a random seed r by an expression $H(e)$, incurring in a probability loss in a reduction corresponding to the probability of $\mathcal{A}$ querying $H(e)$.

4.3 Soundness and Generation

The logic is *sound* w.r.t. its game-based interpretation.

Theorem 1 (Soundness)**.** *If* $\vdash c^\star :_p \mathsf{Guess}$ *then* $\mathbf{Adv}^{\mathsf{CPA}}_{\Pi_{\langle c^\star \rangle}}(t_{\mathcal{A}}) \leq p - \frac{1}{2}$ *(Every valid derivation contains a single application of rule* [Indep]*, so p contains a single occurrence of $\frac{1}{2}$.)*

The proof rests on providing a code-based interpretation of $\vdash c^\star :_p e \in \boldsymbol{L}^{\mathcal{A}}_H$ and $e \vdash^{\mathcal{A}} e'$.

For each valid derivation in the logic, we generate a *proof* in $\mathsf{EasyCrypt}$ that can be verified independently. Generation is done in four steps:

1. build a context that declares all size variables, operators, constants and global variables required in the different games of the proof;
2. build the sequence of games, including the code of inverters for leaves in the proof tree corresponding to applications of rule [OW];
3. output judgments in the relational logic of $\mathsf{EasyCrypt}$ to justify all branches in the derivation tree, inferring the necessary probability inequalities from them;
4. prove the concluding claim by combining all previous derived inequalities.

4.4 Proof Search

The proof search algorithm tries to build a valid proof tree bottom-up. When proving the $\mathsf{IND\text{-}CPA}$ security of an encryption algorithm $\Pi_{\langle c^\star \rangle}$, at the onset the proof tree consists of the single node $\vdash c^\star :_p \mathsf{Guess}$, where p is a meta-variable that will be instantiated during the proof. At each step, the proof search algorithm attempts to resolve a pending goal in the proof tree. Once and if all goals are closed, the algorithm outputs a complete proof tree, and a security bound. The proof search algorithm consists of a strategy for applying rules, and a backtracking procedure.

Strategy At each step, the algorithm checks whether the goal can be discharged by applying the rules [Indep], [Indom], [OW]. If it is not the case, the algorithm tries to transform the goal by applying any of the rules [OptS], [Perm] and [Fail] (in this order), subject to some heuristics to enhance its efficiency—for instance, applications of rule [Fail] replace the sub-expression of the form $H(e)$ closest to an occurrence of m. Moreover, to ensure termination, the rule [OptS] is only applied when an expression of the form $e \oplus r$ appears in the goal.

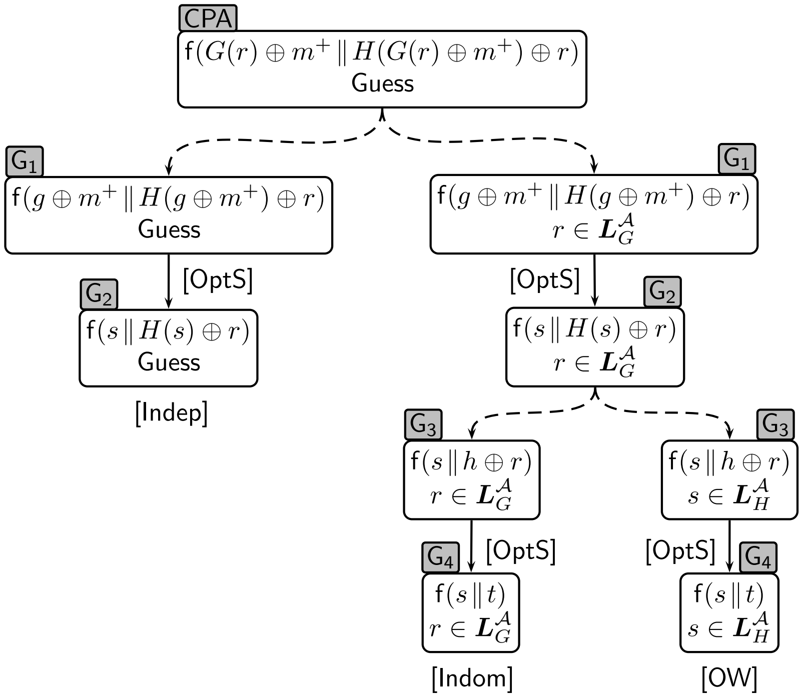


Figure 6: Proof tree output by the proof search algorithm for OAEP. Dashed arrows introduce failure events and correspond to applications of rule [Fail]; solid arrows correspond to semantics-preserving transformations. Tabs attached to nodes indicate the corresponding game in Fig. 7.

Backtracking If at some point the search reaches a situation where no more transformations can be applied and there are still unsolved goals, the algorithm backtracks to the latest point where there are unexplored choices; the search fails when all choices have been explored and no proof has been found.

4.5 Example

We illustrate the proof search procedure and the proof generation mechanism on OAEP. Figure 6 shows the proof tree found by our search algorithm, which reduces the IND-CPA security of OAEP to the hardness of inverting the trapdoor permutation on its ℓ most significant bits; for compactness, in the figure and in the remainder we write m^+ instead of $m \,\|\, 0$. This proof corresponds to the following theorem.

Theorem 2 (IND-CPA Security of OAEP)**.**

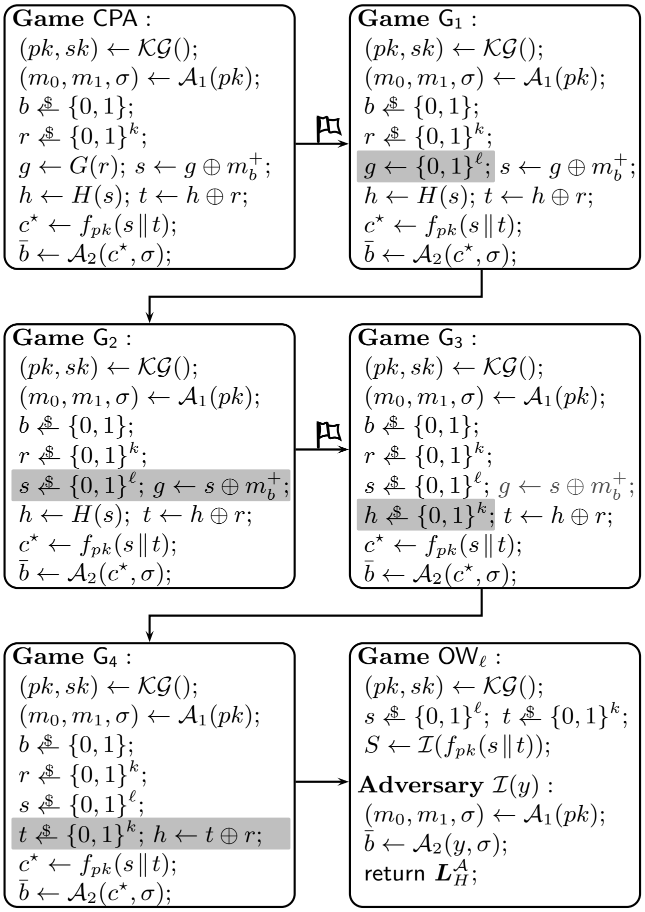
$$\mathbf{Adv}^{\mathsf{CPA}}_{\mathsf{OAEP}}(t_{\mathcal{A}}) \leq \frac{q_G}{2^k} + \mathbf{Succ}_{\Theta}^{\mathsf{OW}_\ell^{\mathcal{A}H}}(t_{\mathcal{A}} + q_H + q_G)$$

The proof search algorithm starts by applying rule [Fail] to replace the expression $G(r)$ with a fresh random bitstring g. To bound the probability of the original event Guess, the algorithm applies first [OptS], and then [Indep]. To bound the probability of the failure event $r \in \boldsymbol{L}_G^{\mathcal{A}}$, the algorithm applies first rule [OptS] and then once again [Fail]. This generates two sub-goals, after applying [OptS] to each of them, one can be discharged by rule [Indom] and the other by rule [OW].

Figure 7 shows the sequence of games that is automatically generated from the proof in the CPA logic. The corresponding EasyCrypt proof establishes the theorem, except that it does not verify the time complexity of the reduction. The EasyCrypt proof is around 1,000 lines long and takes just 15s to machine-check in a modern workstation.

The sequence starts with the standard CPA experiment for adversary $\mathcal{A}$, and ends with the experiment OW_ℓ, in which an inverter $\mathcal{I}$ uses $\mathcal{A}$ as a sub-procedure to invert the trapdoor permutation f on its most significant ℓ bits.

Intuitively, given a challenge ciphertext $c^\star = f_{pk}(s \,\|\, (H(s) \oplus r))$, an adversary $\mathcal{A}$ must query H on s to have any chance of guessing the hidden bit b. Otherwise, any value for r and therefore for s and $f(s\|t)$ would be equally likely. All that the inverter $\mathcal{I}$ has to do then is to consistently simulate a CPA experiment for $\mathcal{A}$ and return the list of queries that $\mathcal{A}$ made to H.



```
Game CPA :
  $(pk, sk) \leftarrow \mathcal{KG}()$;
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1(pk)$;
  $b \overset{\$}{\leftarrow} \{0,1\}$;
  $r \overset{\$}{\leftarrow} \{0,1\}^k$;
  $g \leftarrow G(r)$; $s \leftarrow g \oplus m_b^+$;
  $h \leftarrow H(s)$; $t \leftarrow h \oplus r$;
  $c^\star \leftarrow f_{pk}(s \,\|\, t)$;
  $\bar{b} \leftarrow \mathcal{A}_2(c^\star, \sigma)$;
```

```
Game G1 :
  $(pk, sk) \leftarrow \mathcal{KG}()$;
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1(pk)$;
  $b \overset{\$}{\leftarrow} \{0,1\}$;
  $r \overset{\$}{\leftarrow} \{0,1\}^k$;
  $g \leftarrow \{0,1\}^\ell$; $s \leftarrow g \oplus m_b^+$;
  $h \leftarrow H(s)$; $t \leftarrow h \oplus r$;
  $c^\star \leftarrow f_{pk}(s \,\|\, t)$;
  $\bar{b} \leftarrow \mathcal{A}_2(c^\star, \sigma)$;
```

```
Game G2 :
  $(pk, sk) \leftarrow \mathcal{KG}()$;
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1(pk)$;
  $b \overset{\$}{\leftarrow} \{0,1\}$;
  $r \overset{\$}{\leftarrow} \{0,1\}^k$;
  $s \overset{\$}{\leftarrow} \{0,1\}^\ell$; $g \leftarrow s \oplus m_b^+$;
  $h \leftarrow H(s)$; $t \leftarrow h \oplus r$;
  $c^\star \leftarrow f_{pk}(s \,\|\, t)$;
  $\bar{b} \leftarrow \mathcal{A}_2(c^\star, \sigma)$;
```

```
Game G3 :
  $(pk, sk) \leftarrow \mathcal{KG}()$;
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1(pk)$;
  $b \overset{\$}{\leftarrow} \{0,1\}$;
  $r \overset{\$}{\leftarrow} \{0,1\}^k$;
  $s \overset{\$}{\leftarrow} \{0,1\}^\ell$; $g \leftarrow s \oplus m_b^+$;
  $h \overset{\$}{\leftarrow} \{0,1\}^k$; $t \leftarrow h \oplus r$;
  $c^\star \leftarrow f_{pk}(s \,\|\, t)$;
  $\bar{b} \leftarrow \mathcal{A}_2(c^\star, \sigma)$;
```

```
Game G4 :
  $(pk, sk) \leftarrow \mathcal{KG}()$;
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1(pk)$;
  $b \overset{\$}{\leftarrow} \{0,1\}$;
  $r \overset{\$}{\leftarrow} \{0,1\}^k$;
  $s \overset{\$}{\leftarrow} \{0,1\}^\ell$;
  $t \overset{\$}{\leftarrow} \{0,1\}^k$; $h \leftarrow t \oplus r$;
  $c^\star \leftarrow f_{pk}(s \,\|\, t)$;
  $\bar{b} \leftarrow \mathcal{A}_2(c^\star, \sigma)$;
```

```
Game OW_ℓ :
  $(pk, sk) \leftarrow \mathcal{KG}()$;
  $s \overset{\$}{\leftarrow} \{0,1\}^\ell$; $t \overset{\$}{\leftarrow} \{0,1\}^k$;
  $S \leftarrow \mathcal{I}(f_{pk}(s \,\|\, t))$;

Adversary $\mathcal{I}(y)$ :
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1(pk)$;
  $\bar{b} \leftarrow \mathcal{A}_2(y, \sigma)$;
  return $\boldsymbol{L}_H^{\mathcal{A}}$;
```

Figure 7: Game-based proof of IND-CPA security of OAEP

In the second game in the sequence, G_1, the value of $G(r)$ used to compute the challenge ciphertext is replaced with a uniformly random value; this only makes a difference if $\mathcal{A}$ queries G on r. The generated proof uses the mechanization of Shoup's Fundamental Lemma in EasyCrypt, as described in [3], to obtain

$$\left|\Pr\left[\mathsf{CPA} : \bar{b} = b\right] - \Pr\left[\mathsf{G}_1 : \bar{b} = b\right]\right| \leq \Pr\left[\mathsf{G}_1 : r \in \boldsymbol{L}_G^{\mathcal{A}}\right]$$

Then, applying the *optimistic sampling* transformation, instead of sampling a value g uniformly and defining $s = g \oplus m_b^+$, the next game samples s uniformly and defines $g = s \oplus m_b^+$ (the assignment to g can be removed as dead code). This transformation removes the syntactic dependency of the output of $\mathcal{A}_2$ from the hidden bit b, and thus

$$\Pr\left[\mathsf{G}_1 : \bar{b} = b\right] = \Pr\left[\mathsf{G}_2 : \bar{b} = b\right] = \frac{1}{2}$$
$$\Pr\left[\mathsf{G}_1 : r \in \boldsymbol{L}_G^{\mathcal{A}}\right] = \Pr\left[\mathsf{G}_2 : r \in \boldsymbol{L}_G^{\mathcal{A}}\right]$$

In game G_3, $H(s)$ is replaced with a uniformly random value. Again, this only makes a difference if $\mathcal{A}$ queries $H(s)$ and hence:

$$\Pr\left[\mathsf{G}_2 : r \in \boldsymbol{L}_G^{\mathcal{A}}\right] \leq \Pr\left[\mathsf{G}_3 : r \in \boldsymbol{L}_G^{\mathcal{A}}\right] + \Pr\left[\mathsf{G}_3 : s \in \boldsymbol{L}_H^{\mathcal{A}}\right]$$

Game G_4 is obtained by applying again *optimistic sampling* to randomly sample t instead of h. This makes r obviously independent of the adversary's view, obtaining

$$\Pr\left[\mathsf{G}_3 : r \in \boldsymbol{L}_G^{\mathcal{A}}\right] = \Pr\left[\mathsf{G}_4 : r \in \boldsymbol{L}_G^{\mathcal{A}}\right] \leq \frac{q_G}{2^k}$$

Moreover, the challenge ciphertext is now computed as the value of f on a uniformly random value. This matches exactly the scenario of OW_ℓ; for the synthesized inverter $\mathcal{I}$, one obtains:

$$\begin{aligned}\Pr\left[\mathsf{G}_3 : s \in \boldsymbol{L}_H^{\mathcal{A}}\right] &= \Pr\left[\mathsf{G}_4 : s \in \boldsymbol{L}_H^{\mathcal{A}}\right] \\ &\leq \Pr\left[\mathsf{OW}_\ell : s \in S \wedge |S| \leq q_H\right]\end{aligned}$$

The proof concludes by putting all the above equations together. □

5 Chosen-Ciphertext Security

This section presents a proof system for chosen-ciphertext security, and a proof search algorithm that covers a significant fraction of examples from the literature. The logic is inspired by existing proofs of padding-based encryption schemes, and allows transforming the decryption oracle of the CCA experiment into a plaintext-extractor, i.e. an algorithm that recovers the plaintext from a ciphertext without using the secret key, by inspecting the oracle queries made by the adversary. Since a plaintext-extractor can be implemented efficiently by a CCA adversary, one can then continue the proof by ignoring the decryption oracle and conclude using an extension of the CPA logic.

5.1 Algorithms

Decryption oracles are modeled as algorithms of the form:

$$x_1 \overset{c}{\leftarrow} \boldsymbol{L}_{H_1}^{\mathcal{A}}, \ldots, x_n \overset{c}{\leftarrow} \boldsymbol{L}_{H_n}^{\mathcal{A}} : T \rhd e$$

where T is a boolean-valued test that depends on a distinguished variable c, called the ciphertext, and e is an expression that may depend on c and on variables $\vec{x}$. When $\vec{x}$ is empty, we simply write $T \rhd e$. Tests are built from the following grammar:

$$T ::= e = e \mid e \in \boldsymbol{L}_H \mid e \in \boldsymbol{L}_H^{\mathcal{A}} \mid T \wedge T$$

Given a ciphertext c, an algorithm $\vec{x} \overset{c}{\leftarrow} \vec{\boldsymbol{L}}_H^{\mathcal{A}} : T \rhd e$ searches for values of $\vec{x}$ satisfying T among the oracle queries made by the adversary. If such values are found, the algorithm computes the corresponding value of e; otherwise, it returns a distinguished value $\perp$. Likewise, an algorithm $T \rhd e$ returns the value of e if T holds, and $\perp$ otherwise. Similarly to expressions, every algorithm D can be interpreted as a pWhile procedure $\langle D \rangle$; for instance, the algorithm:

$$[\mathsf{f}^{-1}(c)]_0^{\ell} \in \boldsymbol{L}_H \rhd [\mathsf{f}^{-1}(c)]_{\ell}^{n} \oplus G(r)$$

is interpreted as the procedure:

```
$z \leftarrow \mathsf{f}^{-1}(c);\ x \leftarrow [z]_0^{\ell};\ y \leftarrow [z]_{\ell}^{n};$
if $x \in \boldsymbol{L}_H$ then $g \leftarrow G(r);$ return $y \oplus g$ else return $\perp$
```

An algebraic expression $c^\star$ and an algorithm D induce a pseudo encryption scheme $\Pi_{\langle c^\star, D\rangle} = (\mathcal{KG}_f, \langle c^\star \rangle, \langle D \rangle)$.

5.2 Observational Equivalence

The CCA logic relies on a proof system for observational equivalence between algorithms. Since observational equivalence requires algorithms to make the same sequence of oracle queries, we build this proof system on a fragment of the equational theory of Figure 3 that only equates expressions whose interpretation results in the same sequence of queries. Formally, we let $\Vdash e \sim e'$ iff $\vdash e = e'$ is derivable from the equational theory without using instances of the axioms $y \oplus y = 0$ or $[x \,\|\, y]_0^{|x|} = x$ with expressions y that use the $H(\cdot)$ constructor. The observational equivalence relations $\Vdash T \sim T'$ and $\Vdash D \sim D'$ (on tests and algorithms, respectively) are defined inductively; some selected rules are shown in Figure 8.

$$\frac{}{\Vdash T \wedge f(e) = f(e') \sim T \wedge e = e'} \qquad \frac{T \Vdash e \sim e'}{\Vdash T \wedge e \in \boldsymbol{L}_H^{\mathcal{A}} \sim T \wedge e' \in \boldsymbol{L}_H^{\mathcal{A}}} \qquad \frac{\Vdash T \sim T' \qquad T \Vdash e \sim e'}{\Vdash \vec{x} \xleftarrow{c} \vec{\boldsymbol{L}}_H^{\mathcal{A}} : T \rhd e \sim \vec{x} \xleftarrow{c} \vec{\boldsymbol{L}}_H^{\mathcal{A}} : T' \rhd e'}$$

Figure 8: Selected rules for observational equivalence

$$\frac{D = \vec{x} \xleftarrow{c} \vec{\boldsymbol{L}}_H^{\mathcal{A}} : T \rhd u \quad \vdash c^\star :_p \varphi \quad \mathsf{Public}(D) \quad \mathsf{Guarded}(D) \quad t'_{\mathcal{A}} = t_{\mathcal{A}} + q_D\, \mathsf{T}(u) + \left(\prod_i q_{H_i}\right) \mathsf{T}(T)}{\vdash (c^\star, D) :_{p\{t'_{\mathcal{A}}/t_{\mathcal{A}}\}} \varphi}\,[\mathsf{Pub}]$$

$$\frac{D = \vec{x} \xleftarrow{c} \vec{\boldsymbol{L}}_H^{\mathcal{A}} : T \wedge \vec{x} = \vec{e} \rhd t \quad \vdash (c^\star, D) :_p \varphi \qquad \mathsf{Guarded}(D)}{\vdash (c^\star, T\,\{\vec{e}/\vec{x}\} \wedge \vec{e} \in \vec{\boldsymbol{L}}_H^{\mathcal{A}} \rhd t\,\{\vec{e}/\vec{x}\}) :_p \varphi}\,[\mathsf{Find}]$$

$$\frac{\vdash (c^\star, D') :_p \varphi' \qquad \varphi \Rightarrow \varphi' \qquad \Vdash D \sim D'}{\vdash (c^\star, D) :_p \varphi}\,[\mathsf{Conv}] \qquad \frac{T \wedge e\,\{c^\star/c\} = e \vdash c^\star = c}{\vdash (c^\star, D) :_0 \mathsf{Qry}(T, e)}\,[\mathsf{False}]$$

$$\frac{\vdash (c^\star, T \wedge e \in \boldsymbol{L}_H \rhd t) :_p \varphi \qquad \Pr[T\,\{r/H(e)\}] \leq p'}{\vdash (c^\star, T \rhd t) :_{p + q_D p'} \varphi}\,[\mathsf{Eps}]$$

$$\frac{\vdash (c^\star, T \wedge e \in \boldsymbol{L}_H^{\mathcal{A}} \rhd t) :_p \varphi \quad \vdash (c^\star, T \wedge e \in \boldsymbol{L}_H^{\mathcal{A}} \rhd t) :_{p'} \mathsf{Qry}(T, e)}{\vdash (c^\star, T \wedge e \in \boldsymbol{L}_H \rhd t) :_{p+p'} \varphi}\,[\mathsf{Bad}]$$

Figure 9: Proof rules of the CCA logic

5.3 Judgments

Judgments are of the form $\vdash c^\star :_p \varphi$, or $\vdash (c^\star, D) :_p \varphi$, where $c^\star$ is an algebraic expression, D is an algorithm, φ is an event, and p is a probability expression drawn from the grammar of Figure 4. The former judgment is interpreted as in Section 4, whereas the latter states that p is an upper bound on the probability of φ in the CCA game for the pseudo encryption scheme $\Pi_{\langle c^\star, D\rangle}$.

In addition to the events of the CPA logic, the logic considers a new form of event $\mathsf{Qry}(T, e)$. Informally, an event $\mathsf{Qry}(T, e)$ holds whenever the adversary $\mathcal{A}_2$ queries the decryption oracle with a ciphertext c such that T holds and $e = e\,\{c^\star/c\}$, where $c^\star$ is the challenge ciphertext. Note that, by definition of the CCA game, we necessarily have $c \neq c^\star$.

5.4 Proof System

Figure 9 presents the rules of the logic; we describe them briefly below.

The rules [Bad] and [Eps] allow to make the decryption oracle reject ciphertexts whose decryption requires oracle queries that have not been made by the adversary (rule [Bad]) or are entirely fresh (rule [Eps]). Note that rule [Eps] requires computing an upper bound for the probability of a test, which is resolved to an expression from Figure 4 using rules similar to [Indom] in Figure 5. The rule [False] states that the probability of $\mathsf{Qry}(T, e)$ is null when there is no ciphertext $c \neq c^\star$ such that T holds and $e = e\,\{c^\star/c\}$.

The rules [Find], [Conv], and [Pub] correspond to the construction of a plaintext-extractor. The rule [Pub] interfaces CPA judgments with CCA judgments, and captures the intuition that an adversary does not gain any advantage from getting access to a publicly simulatable decryption oracle. Its soundness rests on the assumption that all oracle calls are guarded in D, written $\mathsf{Guarded}(D)$. We say that an oracle call $H(e)$ is guarded in $\vec{x} \overset{c}{\leftarrow} \vec{\boldsymbol{L}}_H^{\mathcal{A}} : T \rhd u$ iff it appears in T and is derivable from $e = x_i$, or there exists a conjunct $e' \in \boldsymbol{L}_H$ or $e' \in \boldsymbol{L}_H^{\mathcal{A}}$ in T such that $T \Vdash e \sim e'$.[2] Note that the probability in the conclusion involves the execution time $\mathsf{T}(T)$ and $\mathsf{T}(u)$ of T and u, respectively; these are resolved to an expression from Figure 4 using a simple analysis that considers that all operations except f can be executed in unit time. The rule [Find] allows replacing a decryption oracle by one that performs a lookup on the list of oracle queries made by the adversary; its soundness also rests on the fact that the decryption algorithm is guarded. Finally, the rule [Conv] allows switching between observational equivalent decryption oracles and weakening the event considered.

As a final remark, we observe that applications of rule [Pub] can result on proof obligations for judgments of the form $\vdash c^\star :_p \mathsf{Qry}(T, e)$. These should be interpreted as p being an upper bound on the probability of a CPA adversary computing a list of q_D values containing a value c such that T holds and $e = e\,\{c^\star/c\}$. We use a suitable extension of the CPA logic to prove these judgments.

5.5 Soundness

The logic is *sound* w.r.t. its game-based interpretation.

Theorem 3 (Soundness). *If* $\vdash (c^\star, D) :_p \mathsf{Guess}$ *then* $\mathbf{Adv}^{\mathsf{CCA}}_{\Pi_{\langle c^\star, D\rangle}}(t_{\mathcal{A}}) \leq p - \frac{1}{2}$

Generating $\mathsf{EasyCrypt}$ proofs from valid derivations is complex, but certainly feasible, and is left for future work.

5.6 Proof Search

The proof search algorithm tries to build a valid proof tree bottom-up. When proving the $\mathsf{IND\text{-}CCA}$ security of an encryption scheme $\Pi_{\langle c^\star, D\rangle}$, at the onset the proof tree consists of the single node $\vdash (c^\star, D) :_p \mathsf{Guess}$, where p is a meta-variable that will be instantiated during the proof. Contrary to the case of CPA, where the strategy is fixed, the proof search algorithm for CCA is based on the structure of D, and repeatedly applies the rules [Bad] and [Eps] until D is guarded. Then, it tries to apply the rule [False], or the sequence of rules [Find], [Conv] and [Pub] to all remaining sub-goals. When all the remaining sub-goals are of the form $\vdash c^\star :_p \varphi$, the (enhanced) proof search algorithm for CPA is launched on each of them. No backtracking occurs prior to this last step.

5.7 Example

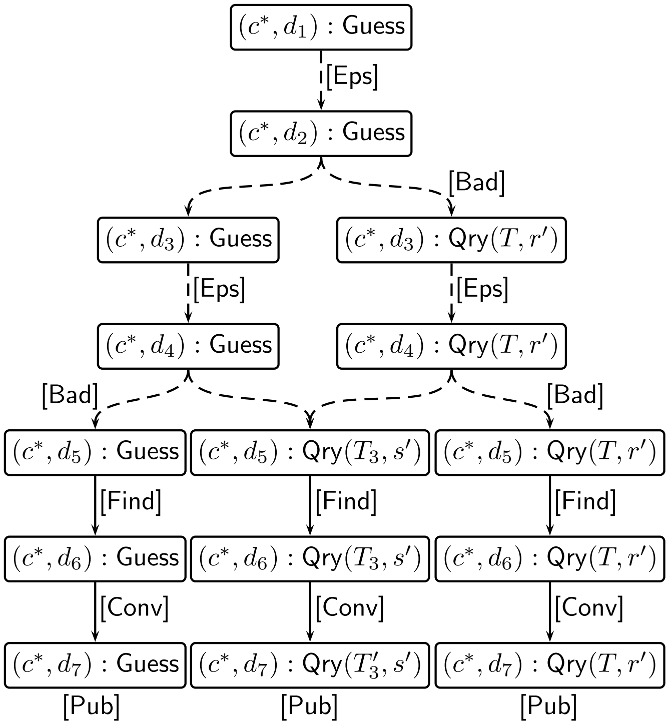
We illustrate the proof search procedure on OAEP. Figure 10 shows the (partial) proof search tree that reduces the $\mathsf{IND\text{-}CCA}$ security of OAEP to the hardness of partially inverting the underlying trapdoor permutation on its ℓ most significant bits. The complete proof corresponds to the following theorem.

[2] Our implementation uses a more liberal definition of guardedness. For the sake of clarity, we present specialized rules sufficient for most examples, including all proofs where the simulation of the decryption oracle does not make any oracle calls. More general rules have a similar shape, but the definition of $t'_{\mathcal{A}}$ is significantly more complex. Moreover, the number of additional queries made by the decryption oracle must be tracked explicitly.

Theorem 4 (IND-CCA Security of OAEP).

$$\mathbf{Adv}^{\mathsf{CCA}}_{\mathsf{OAEP}}(t_{\mathcal{A}}) \leq \frac{q_D}{2^\rho} + \frac{2q_D(q_G+1)+q_G}{2^k} + \epsilon$$

where $\epsilon = \mathbf{Succ}_{\Theta}^{\mathsf{OW}_\ell^{q_H}}(t_{\mathcal{A}} + q_H q_G(t_f+1) + q_H + q_G)$


(c^*, d_1) : Guess
[Eps]
(c^*, d_2) : Guess
[Bad]
(c^*, d_3) : Guess
(c^*, d_3) : Qry(T, r')
[Eps]
[Eps]
(c^*, d_4) : Guess
(c^*, d_4) : Qry(T, r')
[Bad]
[Bad]
(c^*, d_5) : Guess
(c^*, d_5) : Qry(T_3, s')
(c^*, d_5) : Qry(T, r')
[Find]
[Find]
[Find]
(c^*, d_6) : Guess
(c^*, d_6) : Qry(T_3, s')
(c^*, d_6) : Qry(T, r')
[Conv]
[Conv]
[Conv]
(c^*, d_7) : Guess
(c^*, d_7) : Qry(T_3', s')
(c^*, d_7) : Qry(T, r')
[Pub]
[Pub]
[Pub]

Defining equations for encryption

$s^\star \stackrel{\text{def}}{=} G(r^\star) \oplus (m \,\|\, 0^\rho)$ $\qquad$ $c^\star \stackrel{\text{def}}{=} \mathsf{f}(s^\star \,\|\, r^\star \oplus H(s^\star))$

Defining equations for decryption

$u \stackrel{\text{def}}{=} \mathsf{f}^{-1}(c)$ $\qquad$ $n \stackrel{\text{def}}{=} \ell - \rho$
$s' \stackrel{\text{def}}{=} [u]_0^\ell$ $\qquad$ $t' \stackrel{\text{def}}{=} [u]_\ell^k$
$h' \stackrel{\text{def}}{=} H(s')$ $\qquad$ $r' \stackrel{\text{def}}{=} h' \oplus t'$
$g' \stackrel{\text{def}}{=} G(r')$ $\qquad$ $m' \stackrel{\text{def}}{=} [g' \oplus s']_0^n$

Intermediate tests and algorithms

$T \stackrel{\text{def}}{=} [s']_n^\rho = [g']_n^\rho$ $\qquad$ $T_3 \stackrel{\text{def}}{=} r' \in \boldsymbol{L}_G^{\mathcal{A}} \wedge T$
$d_1 \stackrel{\text{def}}{=} T \rhd m'$ $\qquad$ $T_3' \stackrel{\text{def}}{=} H(s^\star) \oplus t' \in \boldsymbol{L}_G^{\mathcal{A}}$
$d_2 \stackrel{\text{def}}{=} r' \in \boldsymbol{L}_G \wedge T \rhd m'$
$d_3 \stackrel{\text{def}}{=} r' \in \boldsymbol{L}_G^{\mathcal{A}} \wedge T \rhd m'$
$d_4 \stackrel{\text{def}}{=} s' \in \boldsymbol{L}_H \wedge r' \in \boldsymbol{L}_G^{\mathcal{A}} \wedge T \rhd m'$
$d_5 \stackrel{\text{def}}{=} s' \in \boldsymbol{L}_H^{\mathcal{A}} \wedge r' \in \boldsymbol{L}_G^{\mathcal{A}} \wedge T \rhd m'$
$d_6 \stackrel{\text{def}}{=} s', r' \stackrel{c}{\leftarrow} \boldsymbol{L}_H^{\mathcal{A}}, \boldsymbol{L}_G^{\mathcal{A}} : T \wedge s' = [u]_0^\ell \wedge r' = h' \oplus t' \rhd m'$
$d_7 \stackrel{\text{def}}{=} s', r' \stackrel{c}{\leftarrow} \boldsymbol{L}_H^{\mathcal{A}}, \boldsymbol{L}_G^{\mathcal{A}} : c = \mathsf{f}(s' \,\|\, r' \oplus h') \wedge [s']_n^\rho = [g']_n^\rho \rhd m'$

Figure 10: Partial proof tree output by the CCA proof search algorithm for OAEP. We omit probability bounds in judgments for clarity.

The proof begins by applying rule [Eps] with $e = r'$, triggering the computation of an upper bound of $\Pr[T\,\{r_1/g'\}]$, which yields $2^{-\rho}$. Next, the tool applies the rule [Bad] taking $e = r'$; this yields two sub-goals that only differ on the event of interest and that are treated similarly. On each sub-goal, first the rule [Eps] is applied with $e = s'$, triggering the computation of an upper bound of $\Pr[r_1 \oplus t' \in \boldsymbol{L}_G^{\mathcal{A}}]$, which yields $q_G\ 2^{-k}$. Then, the rule [Bad] is applied with $e = s'$, followed by rule [Find] on both resulting sub-goals; observe that all oracle calls are guarded in d_6.

Making the decryption algorithm public requires applying the rule [Conv], which allows to remove all uses of f^{-1} from the decryption oracle (and to reformulate and weaken event T_3). At this point, the tool applies the rule [Pub] at each of the three remaining sub-goals and continues using the proof search algorithm for the enhanced CPA logic. For the judgment on the leftmost leaf, where the event of interest is Guess, the proof is the one given in Figure 6. The judgments on the two remaining sub-goals are harder to prove, but follow the same pattern: first calls to G and H are eliminated using the rules [Fail] and [OptS]; this allows to replace the challenge $c^\star$ by $f(s \,\|\, t)$, where s and t are random bitstrings. We focus on the goal Qry(T, r'); recall that Qry(T, r') holds iff the adversary queries the decryption oracle with a ciphertext c such that T holds and $r' = r'\,\{c^\star/c\}$. Since $r'\,\{c^\star/c\} = r^\star$, i.e. the random value used to compute the challenge ciphertext, it follows that $r^\star$ can be obtained from a query in $\boldsymbol{L}_D^{\mathcal{A}}$ when Qry(T, r') holds. Since, at this point the challenge is of the form $f(s \parallel t)$ and independent from $r^\star$, one can apply a generalization of the rule [Indom] to prove that this occurs with probability at most $q_D\,2^{-k}$. The remaining judgment, whose event of interest is Qry(T_3', s'), can be proved similarly, obtaining a probability bound of $q_D\,(q_G+1)\,2^{-k}$.

6 Exploration

This section describes how we explore large design spaces of encryption schemes. We first present filters that use the deducibility relations $\vdash$ and $\vdash^{\mathcal{A}}$ of Section 3 to discard incorrect or insecure candidate schemes. We then present a crawler that builds on these filters, and on the proof search algorithms of Sections 4 and 5.

6.1 Filtering Incorrect and Insecure Schemes

We build our filters on algorithmic approximations of the relations $\vdash$ and $\vdash^{\mathcal{A}}$. The fact that our filters only discard insecure schemes depends just on the soundness of these approximations and not on their completeness. Moreover, filters that determine that a scheme is insecure yield concrete attacks. Indeed, proving that $\vdash$ and $\vdash^{\mathcal{A}}$ are decidable, let alone implementing efficient decision procedures, is non-trivial (see [13] for results in this direction). Given an expression e with a single plaintext variable m we consider the following filters:

Invertibility We verify that an encryption algorithm $\Pi_{\langle e \rangle}$ can be completed to a full scheme by checking that $e \vdash m$. This implies the existence of an unfailing decryption algorithm, which never rejects a ciphertext and always gives some plaintext as answer. We discuss in the next section how to synthesize a stricter algorithm that rejects most invalid ciphertexts.

Plaintext leakage attacks against IND-CPA Let e' be an expression containing m (and not convertible to an expression not containing m) such that $e \vdash^{\mathcal{A}} e'$ and $m \vdash^{\mathcal{A}} e'$. If this is the case, there exist distinct messages m_0, m_1 such that with high probability $\langle e' \rangle(m_0)$ and $\langle e' \rangle(m_1)$ give different results. A CPA adversary can submit m_0, m_1 to the challenger, compute independently the value of e' from the challenge ciphertext and from m_1, and return 1 if the results match and 0 otherwise. An example of a scheme vulnerable to this attack is the Zheng-Seberry scheme [38], whose encryption algorithm is given by $\mathsf{f}(r) \,\|\, (G(r) \oplus m) \,\|\, H(m)$, taking $e' = H(m)$.

Re-encryption attacks against IND-CPA Suppose that $e \,\|\, m \vdash^{\mathcal{A}} \mathcal{R}(e)$ and denote $\mathcal{Z}$ the resulting algorithm. Consider a CPA adversary that chooses two distinct messages m_0, m_1, gets the challenge ciphertext $c^\star$, and then encrypts m_0 using $\mathcal{Z}(c^\star \,\|\, m_0)$ and m_1 using $\mathcal{Z}(c^\star \,\|\, m_1)$ as randomness. If the scheme passes the invertibility filter, then one and only one of these values matches $c^\star$, allowing the adversary to win the CPA game. In particular, all deterministic schemes for which $\mathcal{R}(e) = \emptyset$ are vulnerable to this attack.

Malleability attacks against IND-CCA Suppose that $e \,\|\, m \vdash^{\mathcal{A}} e\,\{0/m\}$ and denote $\mathcal{Z}$ the resulting algorithm. Consider a CCA adversary that chooses two distinct messages m_0, m_1 different from the zero bitstring, gets the challenge ciphertext $c^\star$, and then queries the decryption oracle on $\mathcal{Z}(c^\star \,\|\, m_1)$, returning 1 if the result is the zero bitstring and 0 otherwise. An example of a scheme vulnerable to this attack is $\mathsf{f}(r) \,\|\, (G(r) \oplus (m \,\|\, H(m)))$. We only apply this filter to IND-CPA schemes, for which this attack succeeds with high probability. Note that the attack would not work for e.g. the non-IND-CPA secure scheme $H(r) \,\|\, \mathsf{f}(H(r) \oplus m)$, because there exists an algorithm $\mathcal{Z}$ such that $\mathcal{Z}(e \,\|\, m_0) = \mathcal{Z}(e \,\|\, m_1)$.

Homomorphic attacks Since our goal is to prove security for a generic trapdoor permutation, we can assume additional algebraic properties that do not contradict the assumptions under which we prove security. Concretely, for the purpose of finding attacks we extend the equational theory of Fig. 3 with the axioms $\mathsf{f}(x \oplus y) = \mathsf{f}_1(x) \oplus \mathsf{f}_2(y)$, and $\mathsf{f}(x \,\|\, y) = \mathsf{f}_1(x) \,\|\, \mathsf{f}_2(y)$. For instance, the scheme

$$(G(r) \oplus (m \,\|\, 0)) \,\|\, \mathsf{f}(r \oplus H(G(r) \oplus (m \,\|\, 0)))$$

(corresponding to $\mathsf{OAEPx\text{-}A0}$ in Fig. 7) is vulnerable to a malleability attack when $\mathsf{f}(r \oplus t) = \mathsf{f}_1(r) \oplus \mathsf{f}_2(t)$.

6.2 Synthesis of Decryption Algorithms

A pseudo scheme $\Pi_{\langle e \rangle}$ typically admits more than one correct decryption algorithm, because the consistency condition does not specify what must be the result of decrypting an invalid ciphertext. However, the more invalid ciphertexts the algorithm rejects, the better the chances are the scheme is $\mathsf{IND\text{-}CCA}$ secure. Rather than completing a pseudo scheme $\Pi_{\langle e \rangle}$ with the unfailing decryption algorithm induced by $e \vdash m$, which seldom results in an $\mathsf{IND\text{-}CCA}$ secure scheme, we synthesize a stricter algorithm by analyzing the *redundancy* built into ciphertexts.

We illustrate how we check the validity of a ciphertext c for OAEP, whose encryption scheme is given by:

$$e = \mathsf{f}(G(r) \oplus (m \,\|\, 0^\rho) \,\|\, (H(G(r) \oplus (m \,\|\, 0^\rho)) \oplus r))$$

Observe that we have both $e \vdash r$ and $e \vdash m$. Let r' and m' respectively denote the result of applying the induced algorithms to c. Letting $u = \mathsf{f}^{-1}(c)$, we have

$$r' = H([u]_0^\ell) \oplus [u]_\ell^k \qquad m' = [G(r') \oplus [u]_0^\ell]_0^{\ell-\rho}$$

We derive sufficient conditions for the validity of a ciphertext c from the identity $c = e\,\{m', r'/m, r\}$,

$$c = \mathsf{let}\ s = G(r') \oplus (m' \,\|\, 0^\rho)\ \mathsf{in}\ \mathsf{f}(s \,\|\, (H(s) \oplus r'))$$

which is equivalent to

$$[u]_0^{\ell-\rho} \,\|\, [u]_{\ell-\rho}^\rho \,\|\, [u]_\ell^k = [u]_0^{\ell-\rho} \,\|\, [G(r')]_{\ell-\rho}^\rho \,\|\, (H(s) \oplus r')$$

where $s = [u]_0^{\ell-\rho} \,\|\, [G(r')]_{\ell-\rho}^\rho$. For this to hold, it suffices to check $[u]_n^\rho = [G(r')]_{\ell-\rho}^\rho$. Thus, we obtain the standard decryption algorithm of OAEP:

$$[u]_{\ell-\rho}^\rho = [G(r')]_{\ell-\rho}^\rho \rhd m'$$

The tool implements a generalization of the above procedure allowing one to deal with encryption algorithms that yield ciphertexts with unrecoverable randomness or that use more than one random bitstring. To obtain equations that any valid ciphertext must satisfy, one may consider any sub-expression $\tilde{e}$ of e deducible from e and solve for $c = e\,\{m', \tilde{e}'/m, \tilde{e}\}$.

6.3 Proving (In)Security

When considering an expression denoting an encryption algorithm that passes the invertibility, re-encryption and plaintext leakage filters of Section 6.1, we first search for a proof of $\mathsf{IND\text{-}CPA}$ security using the algorithm of Section 4. Only if this succeeds, we check that it passes the malleability filter, synthesize a decryption algorithm as described in the previous section and search for a proof of $\mathsf{IND\text{-}CCA}$ security using the algorithm of Section 5. When either of the proof search algorithms fails to determine the security of a scheme, we try to find homomorphic attacks in enriched equational theories to determine its insecurity. For $\mathsf{IND\text{-}CPA}$, we do this by considering all sensible equations of the form $\mathsf{f}(e_1 \,\|\, e_2) = \mathsf{f}_1(e_1) \,\|\, e_2$; the scheme is insecure if we can find attacks for all of them. Similarly, for schemes that we prove $\mathsf{IND\text{-}CPA}$ secure assuming the permutation is one-way on the most significant ℓ bits, we try to find attacks against $\mathsf{IND\text{-}CCA}$ in an equational theory extended with the axiom $\mathsf{f}(e) = \mathsf{f}_1([e]_0^{\ell}) \,\|\, [e]_{\ell}^{|e|-\ell}$.

6.4 Synthesis of Candidate Schemes

We implemented a synthesis algorithm that searches for candidate schemes within a budget-constrained space of algebraic expressions. The budget is specified by the user as the number of concatenation, exclusive-or, hash and trapdoor permutation constructors.

The synthesis algorithm generates candidate encryption schemes following a top-down approach. Crucial to this approach is the representation of a partially specified expression by interpreting variables (different from m) as holes. Generation starts from a fully unspecified expression $e = x$, and iteratively picks a hole in e and replaces it with either:

- An expression of the form $\mathsf{f}(x)$, $H(x)$, $x \oplus y$ or $x \,\|\, y$, for fresh variables x and y, if the budget permits;
- A hole-free sub-expression of e or one of 0, 1, m, r; (this does not consume the budget).

We trim large parts of the search space by implementing an early pruning strategy (in the style of [28]). At each iteration, the synthesis algorithm checks whether the (partially specified) expression e is ill-typed, incorrect, insecure, or essentially equivalent to a previously synthesized expression.

Type checking We check size constraints imposed by bitstring operations. For example, we assume that trapdoor permutations and hash function have fixed input and output length, and that exclusive-or can only be applied to bitstrings of the same length. The synthesis algorithm maintains a typing environment as a system of linear constraints on size variables, and updates it according to unification constraints when an expression is substituted for a variable. If unification fails, the system of constraints has no solution and the expression is discarded. For instance, any expression of the form $e \oplus (e \,\|\, x)$ is immediately discarded because e cannot be assigned a length regardless of any substitution for x.

Filtering Rather than applying all the filters of Section 6.1 in their full generality, during synthesis we apply only the invertibility and re-encryption filters, and a very simple version of the plaintext leakage filter that just checks whether $e \vdash^{\mathcal{A}} m$. Instead of running the deducibility procedures from scratch at each iteration, we implement them using memoization, storing and updating results obtained in previous iterations. This reduces the cost drastically.

Minimality checking We avoid generating expressions for which there exists a smaller expression with the same security guarantees. This is the case, for instance, of expressions

$$\mathsf{f}(r) \,\|\, (H(r) \oplus m) \,\|\, \mathsf{f}(r) \qquad (H(r) \oplus m) \,\|\, G(0) \,\|\, \mathsf{f}(r)$$

whose security is equivalent to $\mathsf{f}(r) \,\|\, (m \oplus H(r))$. We also avoid generating two different expressions that are equivalent modulo commutativity and associativity of exclusive-or. We do this by defining a total order $\prec$ between expressions, and only filling a hole with an expression of the form $e_1 \oplus e_2$ if $e_1 \prec e_2$.

7 Practical Interpretation

The bounds delivered by our logics can be used to guide the choice of practical parameters for implementations of padding-based encryption schemes that use RSA as the underlying trapdoor permutation.

We base our analysis on an extrapolation of the estimated cost of factoring the RSA-768 integer from the RSA Factoring Challenge. Let t_N be the cost of factoring an N-bit RSA modulus: Kleinjung [22] estimates that $t_{768} = 2^{67}$, whereas Lenstra [25] suggests to extrapolate based on the ratio

$$\frac{t_N}{t_M} \approx \frac{\exp((1.9229)\log(N)^{1/3}\log(\log(N))^{2/3}}{\exp((1.9229)\log(M)^{1/3}\log(\log(M))^{2/3}}$$

Security bounds delivered by our logics are sometimes expressed relative to the computational cost of partially inverting RSA. However, $\mathbf{Succ}_{\mathsf{RSA}}^{\mathsf{OW}_k^q}(t)$ can be upper bounded by a function of $\mathbf{Succ}_{\mathsf{RSA}}^{\mathsf{OW}}(t')$, where t' depends on t.

Proposition 5 ([18])**.** *Let Θ be RSA with an ℓ-bit modulus such that $\ell < 2k$. Then,*

$$\mathbf{Succ}_{\Theta}^{OW_k^q}(t)\left(\mathbf{Succ}_{\Theta}^{OW_k^q}(t) - 2^{\ell-2k+6}\right) \leq \mathbf{Succ}_{\Theta}^{OW}(2t + q^2\ell^3)$$

When $k = \ell$, a better bound is achieved by converting any inverter returning a set of values S of size at most q into an inverter that returns a single value s (applying RSA to find the preimage among the elements of S):

$$\mathbf{Succ}_{\Theta}^{\mathsf{OW}^q}(t) \leq \mathbf{Succ}_{\Theta}^{\mathsf{OW}}(t + q\ \ell^3)$$

Fixing a maximum number of queries to oracles and an admissible advantage p, the tool can estimate from a security bound the minimum RSA modulus length N such that no adversary executing within time $t_{\mathcal{A}}$ achieves either a CPA or CCA advantage greater than p.

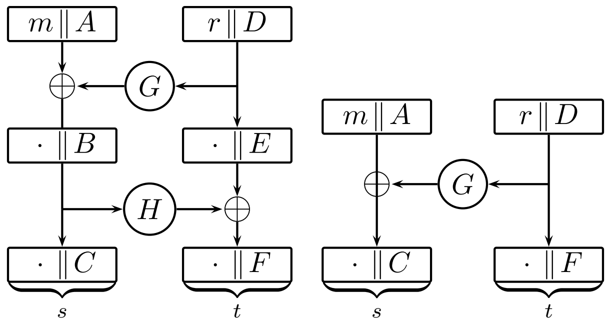
For instance, a reasonable security target might be that no IND-CCA adversary executing within time 2^{128} and making at most 2^{60} hash queries and 2^{30} decryption queries achieves an advantage of more than 2^{-20}. From these values and the security bound found for the scheme whose encryption algorithm is given by the expression:

$$\mathsf{f}(H(r \,\|\, G(r)) \oplus m) \,\|\, ((r \,\|\, G(r)) \oplus H'(H(r \,\|\, G(r)) \oplus m))$$

(that is, xOAEP-$\mathsf{2D}$ in next section) the calculator estimates a minimum modulus length of 3304 bits. Moreover, for a 3304-bit modulus, the calculator gives a lower bound of 130 bits for the length of the random bitstring r and constraints on the output-length of hash functions. These constraints can be used to compute the ciphertext overhead and bandwidth of the scheme. In this case, the calculator computes a ratio of 1.07 between the length of the ciphertext and the length of the plaintext.

8 Experimental Results

In this section we report on the performance of our tool on around 1.6 million synthesized schemes, and on variants of OAEP, SAEP and other schemes from the literature.



(a) OAEP (b) SAEP

Figure 11: The OAEP and SAEP templates

8.1 Analysis of Variants of OAEP and SAEP

The analysis of the security of OAEP is fraught with difficulties. In 1998, Bellare et al. [7] observed that the variant of plaintext-awareness used in [9] to prove the security of OAEP only yields a weaker form of non-adaptive chosen-ciphertext security (IND-CCA1), rather than the standard IND-CCA notion from Rackoff and Simon [30]. In spite of this observation, it was long believed that OAEP was IND-CCA secure. In 2001, Shoup [33] dismissed this belief by showing that a proof of IND-CCA security of OAEP was unlikely without additional assumptions on the trapdoor permutation, and proposed a new scheme, which he called OAEP+ that achieves chosen-ciphertext security without any further assumption. Shoup's discovery triggered new analyses [18,29] that establish the IND-CCA security of OAEP under the stronger assumption that the trapdoor permutation is partial-domain one-way. Subsequent scrutiny revealed ambiguities in the definition of IND-CCA security and gaps in proofs, leading to a machine-checked proof in the CertiCrypt framework [4].

In [23], Komano and Ohta introduce over one hundred variants of OAEP and SAEP, and analyze their security. In both cases, the schemes are obtained by adding redundancy to the padding transformations given by the Feistel network templates shown in Figure 11. For each template, they consider 4 different types of redundancy: 0 (type-0), $H'(m)$ (type-1), $H'(r)$ (type-2), and $H'(r \,\|\, m)$ (type-3), and all possible positions to introduce it: A through F for OAEP, and A, C, D, F for SAEP. Furthermore, for each type of redundancy and each possible position, they consider three different ways of computing a ciphertext given the output $s \,\|\, t$ of the network: by applying a trapdoor permutation just to s (variants xOAEP, xSAEP), just to t (variants OAEPx, SAEPx), or to the concatenation of both (variants OAEP, SAEP). For instance, the encryption algorithm of the scheme OAEP-B3, which corresponds to OAEP+, is given by the expression

$$\mathsf{f}((G(r) \oplus m) \,\|\, H'(r \,\|\, m) \,\|\, (H((G(r) \oplus m) \,\|\, H'(r \,\|\, m)) \oplus r)$$

obtained from the OAEP template by adding type-3 redundancy, i.e. $H'(r \,\|\, m)$, at position B, and applying the trapdoor permutation to the entire output of the Feistel network.

Tables 1 and 2 report on the results of applying our tool to all resulting schemes. For each scheme, we indicate in column CPA whether the tool finds a CPA attack (×), a proof of IND-CPA security (✓), or is unable to decide (?). For each scheme proved IND-CPA secure, column Time indicates the time in seconds needed to verify the generated proof in EasyCrypt, while column CCA indicates whether the tool finds a (malleability) CCA attack (×), a proof of IND-CCA security (✓), or is unable to decide (?). Finally, column Hyp indicates the hypothesis on the underlying trapdoor permutation needed to prove the

Name	CPA	Time	CCA	Hyp
OAEPx-A0	✓	12	×	(■)
OAEPx-B0	✓	11	×	(■)
OAEPx-C0	✓	12	×	(■)
OAEPx-D0	×			
OAEPx-E0	×			
OAEPx-F0	×			
OAEPx-A1	✓	24	×	(■)
OAEPx-B1	×			
OAEPx-C1	×			
OAEPx-D1	×			
OAEPx-E1	×			
OAEPx-F1	×			
OAEPx-A2	✓	26	×	(■)
OAEPx-B2	✓	25	×	(■)
OAEPx-C2	✓	25	×	(■)
OAEPx-D2	×			
OAEPx-E2	×			
OAEPx-F2	✓	26	×	(■, □)
OAEPx-A3	✓	35	✓	(■)
OAEPx-B3	✓	26	✓	(■)
OAEPx-C3	✓	25	✓	(■)
OAEPx-D3	?(×)		?(×)	
OAEPx-E3	×			
OAEPx-F3	✓	26	✓	(■, □)

Name	CPA	Time	CCA	Hyp
xOAEP-A0	✓	15	✓	(■)
xOAEP-B0	×			
xOAEP-C0	×			
xOAEP-D0	✓	16	?(✓)	(■)
xOAEP-E0	✓	14	?(✓)	(■)
xOAEP-F0	✓	14	?(✓)	(■)
xOAEP-A1	✓	29	✓	(■)
xOAEP-B1	×			
xOAEP-C1	×			
xOAEP-D1	✓	30	✓	(■)
xOAEP-E1	✓	30	✓	(■)
xOAEP-F1	×			
xOAEP-A2	✓	28	✓	(■)
xOAEP-B2	✓	30	✓	(■)
xOAEP-C2	✓	30	✓(?)	(■, □)
xOAEP-D2	✓	30	✓	(■)
xOAEP-E2	✓	27	✓	(■)
xOAEP-F2	✓	28	✓	(■)
xOAEP-A3	✓	38	✓	(■)
xOAEP-B3	✓	40	✓	(■)
xOAEP-C3	✓	41	✓(?)	(■, □)
xOAEP-D3	✓	45	✓	(■)
xOAEP-E3	✓	31	✓	(■)
xOAEP-F3	✓	33	✓	(■)

Name	CPA	Time	CCA	Hyp
OAEP-A0	✓	15	✓	(■, ■, □)
OAEP-B0	?(×)		?(×)	
OAEP-C0	?(×)		?(×)	
OAEP-D0	✓	15	?(×)	(■, □, □)
OAEP-E0	✓	15	?(×)	(■, □, □)
OAEP-F0	×			
OAEP-A1	✓	28	✓	(■, ■, □)
OAEP-B1	×			
OAEP-C1	×			
OAEP-D1	✓	30	✓	(■, □, □)
OAEP-E1	✓	30	✓	(■, □, □)
OAEP-F1	×			
OAEP-A2	✓	29	✓	(■, ■, □)
OAEP-B2	✓	30	✓	(■, ■, □)
OAEP-C2	✓	29	✓	(■, □, □)
OAEP-D2	✓	30	✓	(■, □, □)
OAEP-E2	✓	28	✓	(■, □, □)
OAEP-F2	✓	29	✓	(■, □, □)
OAEP-A3	✓	39	✓	(■, ■, □)
OAEP-B3	✓	41	✓	(■, ■, □)
OAEP-C3	✓	38	✓	(■, □, □)
OAEP-D3	✓	45	✓	(■, □, □)
OAEP-E3	✓	31	✓	(■, □, □)
OAEP-F3	✓	38	✓	(■, □, □)

Table 1: Results on variants of OAEP

security of the scheme. When the trapdoor permutation is applied to the concatenation of several algebraic expressions, we consider each of them as a different parameter of the permutation; black squares represent the parameters on which the permutation must be hard to invert. For instance, the trapdoor permutation in OAEP-F3, whose encryption algorithm is given by

$$\mathsf{f}((m \oplus G(r)) \,\|\, (r \oplus H(m \oplus G(r))) \,\|\, H'(r \,\|\, m))$$

can be seen as a function of three parameters, one of the same length as m, one of the same length as r, and one of the same length as the output of H'. We prove the IND-CCA security of this scheme under the hypothesis that the underlying trapdoor permutation is hard to invert on its first parameter; this hypothesis is depicted as (■, □, □).

Our results are consistent with the taxonomy of Komano and Ohta [23]. For comparison, we indicate between parentheses the results of [23] on schemes for which our tool cannot decide security; gladly, these are scarce. Our tool fails to decide the chosen-plaintext security of only 4 schemes, all of them insecure according to [23]. When it comes to chosen-ciphertext security, our tool fails to prove the security of only 3 variants of OAEP. Sometimes, we find proofs under different hypotheses than in [23]; for instance, some variants of OAEP are proved under an assumption of the form (■, □, □), whereas Komano and Ohta only consider assumptions (■), (■, □) and (■, ■, □). Proving security under the same assumptions on the underlying trapdoor permutation (and proving the IND-CCA security of the three schemes for which our tool fails to find a proof) would require to program random oracles, a proof technique that our logic does not currently support. For completeness, we note that we can prove that two variants of xOAEP (tagged with ✓(?)) are secure against chosen-plaintext attacks under an assumption of the form (■, □); these results are new, as Komano and Ohta [23] do not analyze xOAEP under this assumption.

Name	CPA	Time	CCA	Hyp
SAEPx-A0	✓	5	×	(■)
SAEPx-F0	×			
SAEPx-C0	✓	6	×	(■)
SAEPx-D0	×			
SAEPx-A1	✓	15	×	(■)
SAEPx-F1	×			
SAEPx-C1	×			
SAEPx-D1	×			
SAEPx-A2	✓	14	×	(■)
SAEPx-F2	✓	14	×	(■, □)
SAEPx-C2	✓	14	×	(■)
SAEPx-D2	✓	14	×	(■, □)
SAEPx-A3	✓	16	✓	(■)
SAEPx-F3	✓	16	✓	(■, □)
SAEPx-C3	✓	17	✓	(■)
SAEPx-D3	✓	16	?(×)	(■, □)

Name	CPA	Time	CCA	Hyp
xSAEP-A0	×			
xSAEP-F0	×			
xSAEP-C0	×			
xSAEP-D0	×			
xSAEP-A1	×			
xSAEP-F1	×			
xSAEP-C1	×			
xSAEP-D1	×			
xSAEP-A2	×			
xSAEP-F2	×			
xSAEP-C2	×			
xSAEP-D2	×			
xSAEP-A3	×			
xSAEP-F3	×			
xSAEP-C3	×			
xSAEP-D3	×			

Name	CPA	Time	CCA	Hyp
SAEP-A0	✓	6	×	(□, ■)
SAEP-F0	×			
SAEP-C0	?(×)		×	
SAEP-D0	×			
SAEP-A1	✓	15	×	(□, ■)
SAEP-F1	×			
SAEP-C1	×			
SAEP-D1	×			
SAEP-A2	✓	15	×	(□, ■)
SAEP-F2	✓	14	×	(□, ■, □)
SAEP-C2	✓	13	×	(□, □, ■)
SAEP-D2	✓	14	×	(□, ■, □)
SAEP-A3	✓	16	✓	(□, ■)
SAEP-F3	✓	17	✓	(□, ■, □)
SAEP-C3	✓	16	✓	(□, □, ■)
SAEP-D3	✓	17	?(×)	(□, ■, □)

Table 2: Results on variants of SAEP

8.2 Analysis of the Crawler

We explored the design space of padding-based schemes under different budgets, limiting only the total number of operations. Table 3 provides a summary of the output of the crawler for each budget: the column GEN provides the number of synthesized schemes. Columns CPA and CCA respectively indicate the number of schemes proved IND-CPA or IND-CCA secure. Columns ¬CPA and ¬CCA indicate the number of schemes for which a CPA or CCA attack is found. In our largest experiment, with a budget that limits the number of hash, concatenation, and exclusive-or operations to three of each, our tool synthesizes over 925,000 schemes, finds a proof of IND-CPA security for around 175,000 of them, and a proof of IND-CCA security for around 13,000.

Ops	GEN	¬ CPA	CPA	¬ CCA	CCA
4	2	0	2	2	0
5	44	27	12	9	0
6	419	244	104	68	1
7	4131	2392	883	537	39
8	41860	24166	7850	4424	436
9	275318	155669	54884	27697	3750

Table 3: Results on synthesized candidates

The smallest (in number of operations) IND-CCA scheme found by the tool is:

$$\mathsf{f}(r \,\|\, (m \oplus H(r))) \,\|\, G(r \,\|\, (m \oplus H(r)))$$

Which uses 1 exclusive-or, 2 concatenations, 2 hash computations, and applies f once. For the same security target considered in Section 7, the tool estimates a minimum RSA modulus length of 4864 bits, and requires the output length of G to be at least 70 bits and the length of the random bitstring r to be at least $(|m| + 46)/2$ bits.

Our benchmarks suggest that an early pruning strategy is effective in eliminating the majority of incorrect or insecure candidate schemes. The computationally more expensive proof search algorithms succeed in proving the security of more than half of the schemes on which they are applied. To confirm this observation, we evaluated the performance of the crawler disabling pruning and limiting the budget to two hash function, two concatenation, and two exclusive-or operations. This results in over 1.6 million candidate schemes, only 0.5% of which are well-typed and pass the minimality filter and the filters of Section 6.1. Disabling pruning increases computation time by a factor of over 100. Figure 12 compares the effectiveness of the different filters in pruning the search space. The area of each circle in the figure is proportional to the number of schemes that pass each filter as a fraction of the total 1.6 million candidate schemes generated. These correspond respectively to: well-typed expressions (around 74%), expressions that pass the minimality filter (13%), and expressions that pass all the filters of Section 6.1 (4%). Naturally, the proof search algorithms are applied only on schemes in the intersection of these three circles (0.5%).

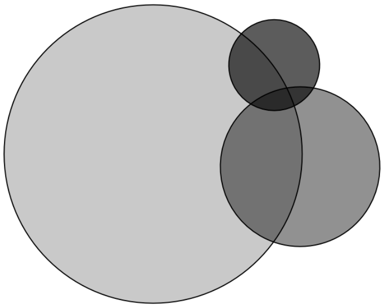


Figure 12: Comparative diagram of early pruning strategy

When using pruning, the crawler is reasonably efficient. It takes less than 17 minutes to analyze the more than 275,000 schemes that use at most 9 operations. On average, generating an EasyCrypt proof of the IND-CPA security of one of these schemes takes less than one second, while verifying it takes less than one minute.

9 Related Work

Our work spans over two major areas: tool-assisted cryptographic proofs and program synthesis. We briefly review relevant work in each area.

Tool-assisted cryptographic proofs.

We restrict our attention to works that address the verification of cryptographic systems in the computational model, and refer the reader to [12, 15] for a more broad account of symbolic verification and computational soundness.

CryptoVerif [11] is the first tool to have supported the automated construction and verification of cryptographic proofs in the computational model; one of the earliest applications of CryptoVerif is a verified proof of unforgeability of Full Domain Hash signatures. However, security proofs of OAEP-like encryption schemes has thus far remained out of reach of CryptoVerif.

The starting point of our investigation is the code-based approach to verified security embodied by CertiCrypt [5] and EasyCrypt [3]. These tools provide rigorously justified support for building and verifying code-based security proofs in the style advocated by Bellare and Rogaway [10] and Halevi [21]. Both tools share as a foundation a relational program logic, but differ in their realization: while CertiCrypt is based on the Coq proof assistant, EasyCrypt implements a verification condition generator that computes first order formulae from logic judgments, and discharges them using SMT solvers. Although EasyCrypt can reconstruct and verify automatically proofs from compact transcripts (called proof sketches), it does not provide any means to build these transcripts automatically.

Generation of automated proofs of security for OAEP-like constructions was first explored in [16], which proposes a specialized Hoare logic for proving that such encryption schemes are IND-CPA secure. The logic yields concrete security proofs for several schemes, including BR [8] and REACT [27], but excluding OAEP. This Hoare-like logic was later generalized into Computational Indistinguishability Logic (CIL) [2,14], which allows reasoning about oracles and adaptive adversaries and was used to prove the IND-CCA security of OAEP. CIL is not specialized to a specific class of schemes and reasoning is performed in the vernacular language of mathematics; hence, it cannot be used for automated proofs and synthesis.

Program synthesis.

Program synthesis aims at automatically generating programs that achieve a desired purpose. This purpose can be specified formally as a relation between inputs and outputs [26], as a temporal logic formula [17], or as a function in higher-order logic [19]. SMT-based program synthesis is a recent alternative [35, 37] that leverages the generality and effectiveness of SMT solvers to provide efficient synthesis methods across multiple application domains. Successful applications of SMT-based program synthesis methods include ciphers [35], bit-vector algorithms [20], and program inversion [36]. Such applications are directly relevant to cryptography: for instance, synthesizing decryption algorithms can be construed as an instance of program inversion, and synthesizing bit-vectors programs could have applications to optimized cryptographic implementations. However, these works focus on proving functional properties, rather than the much harder problem of security.

The AVGI toolkit [28] pioneered the application of synthesis to cryptographic protocols. The toolkit allows users to state security requirements, for instance authentication or secrecy, and non-functional requirements such as message length and available bandwidth. The AVGI toolkit is composed of a protocol generator, that applies pruning techniques to curb the state space explosion problem, and a protocol screener that applies symbolic methods to verify whether generated protocols comply with the desired properties. Moreover, the toolkit allowed the discovery of new secure protocols. Saidi [32] also outlines the design of a synthesizer for protocols. However, both works assume perfect cryptography and their underlying technical artifacts cannot be easily adapted to obtain computational guarantees.

10 Future Work and Conclusions

Automated verification and synthesis of cryptographic constructions are long-standing problems. In this paper, we devised and instrumented a layered methodology that successfully addresses both problems for the practically important class of padding-based encryption schemes. Our methodology is broadly applicable, and we intend to apply it for revisiting classical cryptographic constructions, with the ambitious goal of building a formally verified cryptographic atlas. We believe that a systematic exploration of the design space of cryptographic constructions is likely to lead to the discovery of new practical schemes, notably in areas where the quest for concrete constructions remains relatively open.

More generally, there exist other opportunities for applying automated synthesis techniques to cryptography. We expect that SMT-based synthesis methods [1, 20, 37] can be adapted to synthesize the code of simulators in cryptographic proofs, and to generate platform-dependent, optimized implementations of cryptographic primitives and protocols.

References

[1] G. Barthe, J. Crespo, S. Gulwani, C. Kunz, and M. Marron. From Relational Verification to SIMD Loop Synthesis. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM Press, 2013. To appear.

[2] G. Barthe, M. Daubignard, B. Kapron, and Y. Lakhnech. Computational indistinguishability logic. In *17th ACM Conference on Computer and Communications Security, CCS 2010*, pages 375–386, New York, 2010. ACM.

[3] 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.

[4] G. Barthe, B. Grégoire, Y. Lakhnech, and S. Zanella Béguelin. Beyond provable security. Verifiable IND-CCA security of OAEP. In *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 180–196, Heidelberg, 2011. Springer.

[5] 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.

[6] M. Bellare. Practice-oriented provable-security. In *1st International Workshop on Information Security*, volume 1396 of *Lecture Notes in Computer Science*, pages 221–231. Springer, 1998.

[7] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer, Aug. 1998.

[8] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73. ACM Press, Nov. 1993.

[9] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology – EUROCRYPT 1994*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111, Heidelberg, 1994. Springer.

[10] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426, Heidelberg, 2006. Springer.

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

[12] B. Blanchet. Security protocol verification: Symbolic and computational models. In P. Degano and J. D. Guttman, editors, *Principles of Security and Trust - First International Conference, POST 2012*, volume 7215 of *Lecture Notes in Computer Science*, pages 3–29. Springer, 2012.

[13] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *18th IEEE Symposium on Logic in Computer Science – LICS 2003*, pages 271–280. IEEE Computer Society, 2003.

[14] P. Corbineau, M. Duclos, and Y. Lakhnech. Certified security proofs of cryptographic protocols in the computational model: An application to intrusion resilience. In *1st International Conference on Certified Programs and Proofs - CPP 2011*, volume 7086 of *Lecture Notes in Computer Science*, pages 378–393, Heidelberg, 2011. Springer.

[15] V. Cortier, S. Kremer, and B. Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *J. Autom. Reasoning*, 46(3-4):225–259, 2011.

[16] J. Courant, M. Daubignard, C. Ene, P. Lafourcade, and Y. Lakhnech. Towards automated proofs for asymmetric encryption schemes in the random oracle model. In *15th ACM Conference on Computer and Communications Security, CCS 2008*, pages 371–380, New York, 2008. ACM.

[17] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[18] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. *J. Cryptology*, 17(2):81–104, 2004.

[19] M. Gordon, J. Iyoda, S. Owens, and K. Slind. Automatic formal synthesis of hardware from higher order logic. *Electr. Notes Theor. Comput. Sci.*, 145, 2006.

[20] S. Gulwani, S. Jha, A. Tiwari, and R. Venkatesan. Synthesis of loop-free programs. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011*, pages 62–73, 2011.

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

[22] T. Kleinjung, K. Aoki, J. Franke, A. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. Montgomery, D. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann. Factorization of a 768-bit RSA modulus. In *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 333–350, Heidelberg, 2010. Springer.

[23] Y. Komano and K. Ohta. Taxonomical security consideration of OAEP variants. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E89-A(5):1233–1245, 2006.

[24] P. J. Landin. The next 700 programming languages. *Communications of the ACM*, 9(3):157–166, Mar. 1966.

[25] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. *J. Cryptology*, 14(4):255–293, 2001.

[26] Z. Manna and R. Waldinger. Fundamentals of deductive program synthesis. *IEEE Transactions on Software Engineering*, 18:674–704, 1992.

[27] T. Okamoto and D. Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. In D. Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 159–175. Springer, Apr. 2001.

[28] A. Perrig and D. Song. Looking for diamonds in the desert — extending automatic protocol generation to three-party authentication and key agreement protocols. In *Computer Security Foundations Workshop, CSFW 13*, July 2000.

[29] D. Pointcheval. Provable security for public key schemes. In *Advanced Courses on Contemporary Cryptology*, chapter D, pages 133–189. Birkhäuser Basel, 2005.

[30] C. Rackoff and D. R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In J. Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444. Springer, Aug. 1992.

[31] P. Rogaway. Practice-oriented provable security and the social construction of cryptography. Unpublished essay, 2009.

[32] H. Saidi. Towards automated synthesis of security protocols. In *AAAI Spring Symposium on Logic-based program synthesis*, 2002.

[33] V. Shoup. OAEP reconsidered. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 239–259, Heidelberg, 2001. Springer.

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

[35] A. Solar-Lezama, R. M. Rabbah, R. Bodík, and K. Ebcioglu. Programming by sketching for bit-streaming programs. In V. Sarkar and M. W. Hall, editors, *PLDI*, pages 281–294. ACM, 2005.

[36] S. Srivastava, S. Gulwani, S. Chaudhuri, and J. S. Foster. Path-based inductive synthesis for program inversion. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011*, pages 492–503, 2011.

[37] S. Srivastava, S. Gulwani, and J. S. Foster. From program verification to program synthesis. In *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010*, pages 313–326, 2010.

[38] Y. Zheng and J. Seberry. Immunizing public key cryptosystems against chosen ciphertext attack. *IEEE Journal on Selected Areas in Communications*, 11(5):715–724, 1993.