Fully Automated Analysis of Padding-Based Encryption in the Computational Model

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

ABSTRACT

Computer-aided verification provides effective means of analyzing the security of cryptographic primitives. However, it has remained a challenge to achieve fully automated analyses yielding guarantees that hold against computational (rather than symbolic) attacks. This paper solves this challenge for public-key encryption schemes built from trapdoor permutations and hash functions. Using a novel methodology to combine computational and symbolic cryptography, we present proof systems for analyzing the chosen-plaintext and chosen-ciphertext security of such schemes in the random oracle model, and a toolset that bundles together fully automated proof search and attack finding algorithms. Using the toolset in batch mode, we build a comprehensive database of encryption schemes that records attacks against insecure schemes, and proofs with concrete bounds for secure ones.

Keywords

Attack finding, automated proofs, provable security, public-key encryption, static equivalence

1. INTRODUCTION

Two different models for analyzing the security of cryptographic constructions have developed over the years and coexist until today, each with its own advantages and disadvantages: the computational model and the symbolic model.

The computational model has its roots in the work of Goldwasser and Micali [20] and views cryptographic primitives as functions on bitstrings. Adversaries are modeled as probabilistic algorithms and security is defined in terms of their success probability and computational resources. Security proofs are reductions, showing how a successful attack can be converted into an efficient algorithm that breaks the security of primitives.

The symbolic model, which originates from the seminal work of Dolev and Yao [17], views cryptographic primitives as function symbols in an algebra of expressions. The properties of primitives and the capabilities of adversaries are modeled using equational theories. This model enables automated analysis, but can miss attacks that are possible in the computational model.

A celebrated article by Abadi and Rogaway [2] bridges the gap between the two models: it gives sufficient conditions under which symbolic security implies computational security; symbolic methods that exhibit this property are called cryptographically sound. This result, originally proved for symmetric encryption, has since been extended to richer theories [15]. Cryptographic soundness opens the perspective of combining the best of both worlds: fully automated proofs of computational security. However, its applications have remained confined to protocols. Developing cryptographically sound symbolic methods for primitives remains a challenge.

A recent alternative to cryptographic soundness uses programming language techniques for reasoning about the security of cryptographic constructions directly in the computational model. This alternative is embodied in tools like CryptoVerif [11] and EasyCrypt [4], which have been used to verify emblematic cryptographic constructions. Proofs built using these tools yield reductions with concrete security bounds. However, these tools are primarily interactive and their use requires expert knowledge. Combining them with fully automated methods that apply to large classes of cryptographic primitives has remained an open problem.

A converse goal to proving security is finding attacks. Recent work [3] explores the use of symbolic methods to find attacks and estimate their success probability. Automating attack finding can provide valuable feedback during cryptographic design and be used to evaluate empirically the completeness of automated security analyses: how many constructions that may be secure cannot be proven so.

Contributions. We present new methods for automatically analyzing the security of padding-based encryption schemes, i.e. public-key encryption schemes built from hash functions and trapdoor permutations, such as OAEP [10], OAEP+ [28], SAEP [13], or PSS-E [**?**]. These methods rest on two main technical contributions.

First, we introduce specialized logics for proving chosen-plaintext and chosen-ciphertext security. Proof rules have a symbolic flavor and proof search can be automated, yet one can extract concrete security bounds from valid derivations. This is achieved through a novel combination of symbolic and computational methods; for instance, the logics use symbolic deducibility relations for computing bounds of the probability of events, and for checking the existence of reductions to computational assumptions.

*IMDEA Software Institute, Spain.
E-mail: {gilles.barthe,benedikt.schmidt,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

Second, we propose fully automated methods for finding attacks against chosen-plaintext and chosen-ciphertext security. Our methods are inspired by static equivalence [1], and exploit algebraic properties of trapdoor permutations to find attacks against realizations of schemes that are consistent with computational assumptions.

We demonstrate the strengths of our methods by implementing a toolset, called ZooCrypt, and by evaluating it extensively. The toolset can be used either in batch mode or in interactive mode:

- The batch mode allows to take a census of padding-based encryption schemes. We used it to build a database of more than a million entries. The database records for each scheme and set of assumptions on trapdoor permutations, either an adversary that breaks the security of the scheme or a formal derivation that proves its security.
- The interactive mode allows to build proofs or attacks for a selected scheme, and forms the backbone of a cryptographic tutor. The tutor guides users in building security proofs or exhibiting attacks for a scheme chosen by the user or selected from the database. In proof mode, the tutor provides the user with the set of applicable rules at each step of the derivation; upon completion of the derivation, the tutor outputs alternative derivations that deliver better bounds.

We stress that we focus on padding-based encryption primarily for illustrative purposes. Padding-based schemes include many practically relevant constructions, and provide an excellent testbed to evaluate the effectiveness of our methods and illustrate their working. However, our methods are applicable to other settings; see § 8 for a discussion.

Contents. § 2 provides background on public-key encryption schemes and their security; § 3 introduces symbolic tools that are the base of our analyses; § 4 describes logics for chosen-plaintext and chosen-ciphertext security, while § 5 covers attack finding techniques; § 6 reports on the implementation of ZooCrypt and on its evaluation. We conclude with a survey of related work and directions for future work.

Anonymized supplemental material, including the source code of the tool and a web interface for browsing results, is available at

`http://zoocrypt.no-ip.org/`

2. CRYPTOGRAPHIC BACKGROUND

A public-key encryption scheme is a triple of probabilistic 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 message m or a distinguished value $\perp$ denoting failure.

We require that for pairs of keys (pk, sk) generated by $\mathcal{KG}$, $\mathcal{D}_{sk}(\mathcal{E}_{pk}(m)) = m$ holds for any message m.

We study public-key encryption schemes built from trapdoor permutations, hash functions and basic bitstring operations, such as bitwise exclusive-or ($\oplus$), concatenation ($\|$) and projection. We let $\{0,1\}^n$ denote the set of bitstrings of length n and $[x]_n^\ell$ the bitstring obtained from x by dropping its n most significant bits and taking the ℓ most significant bits of the result.

Trapdoor permutations are functions that are easy to evaluate, but hard to invert without knowing a secret trapdoor. Formally, a family of trapdoor permutations on $\{0,1\}^n$ is a triple of algorithms $(\mathcal{KG}, f, f^{-1})$ such that for any pair of keys (pk, sk) output by $\mathcal{KG}$, f_{pk} and f_{sk}^{-1} are 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 f on a random input.

Definition 1 (One-Way Trapdoor Permutation) *Let $\Theta = (\mathcal{KG}, f, f^{-1})$ be a family of trapdoor permutations on $\{0,1\}^n$ and let k, ℓ be such that $k + \ell \leq n$. The probability of an algorithm $\mathcal{I}$ in q-inverting Θ on bits $[k+1 \ldots k+\ell]$ is*

$$\mathbf{Succ}_{\Theta}^{\mathsf{s\text{-}pd\text{-}OW}}(k, \ell, q, \mathcal{I}) \stackrel{def}{=} \Pr\left[\mathsf{OW}_k^\ell : [s]_k^\ell \in S \wedge |S| \leq q\right]$$

where OW_k^ℓ is the experiment:

$$(pk, sk) \leftarrow \mathcal{KG};\ s \xleftarrow{\$} \{0,1\}^n;\ S \leftarrow \mathcal{I}(f_{pk}(s))$$

This definition generalizes the set partial-domain one-wayness problem of Fujisaki et al. [18], which corresponds to $k = 0$. We define $\mathbf{Succ}_{\Theta}^{\mathsf{s\text{-}pd\text{-}OW}}(k, \ell, q, t)$ as the maximal success probability over all inverters executing in time t, and omit the first parameter when it is 0. When $q = 1$, $k = 0$, and $\ell = n$, the experiment corresponds to the standard one-wayness problem, and we note this quantity $\mathbf{Succ}_{\Theta}^{\mathsf{OW}}(t)$.

We consider two security notions for public-key encryption: chosen-plaintext security, or IND-CPA, and adaptive chosen-ciphertext security, or IND-CCA. Both can be described succinctly using the following 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:

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

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. Note that $\mathcal{A}_1$ and $\mathcal{A}_2$ have oracle access to all hash functions $\vec{H}$ used in the scheme and can communicate with each other through σ. We call this basic experiment CPA; we define the experiment CCA 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 $c^\star$. In the remainder, we note $\boldsymbol{L}_H$ the list of queries that the adversary makes to a hash oracle H in either experiment, and $\boldsymbol{L}_{\mathcal{D}}$ the list of decryption queries that $\mathcal{A}_2$ makes during the second phase of the CCA experiment.

Security is measured by the advantage of an adversary playing against a CPA or CCA challenger.

Definition 2 (Adversary Advantage) *The advantage of $\mathcal{A}$ against the* CPA *and* CCA *security of an encryption*

scheme $\Pi = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ *is defined respectively as:*

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

We define $\mathbf{Adv}^{\mathsf{CPA}}_{\Pi}(t, \vec{q})$ (resp. $\mathbf{Adv}^{\mathsf{CCA}}_{\Pi}(t, \vec{q})$) as the maximal advantage over all adversaries $\mathcal{A}$ that execute within time t and make at most $q_{\mathcal{O}}$ queries to oracle $\mathcal{O}$.

To define asymptotic security, we consider instead of a single scheme, a family of schemes Π_η indexed by a security parameter $\eta \in \mathbb{N}$. Probability quantities thus become functions of η. We say that a scheme is secure if the advantage of any adversary executing in time polynomial in η is a negligible function ν of η, i.e. $\forall c.\ \exists n_c.\ \forall n > n_c.\ \nu(n) < n^{-c}$.

A security proof for a padding-based encryption scheme is a reduction showing that any efficient adversary against the security of the scheme can be turned into an efficient inverter for the underlying trapdoor permutation. Security proofs are typically in the random oracle model, where hash functions are modeled as truly random functions. One seemingly artificial property of the random oracle model is that reductions to computational assumptions are allowed to adaptively choose the responses of oracles modeling hash functions; this property is called *programmability*. In contrast, we only consider proofs in the non-programmable random oracle model, where reductions are only allowed to observe oracle queries and responses.

For illustrative purposes, we use OAEP [10] as a running example. RSA-OAEP, which instantiates OAEP with RSA as trapdoor permutation is recommended by several international standards.

Definition 3 (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:*

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

3. SYMBOLIC TOOLS

We introduce algebraic expressions to model padding-based encryption schemes, and symbolic notions to model the knowledge that either an adversary or an honest user can derive from an expression.

3.1 Syntax and Semantics of Expressions

Expressions are built from a set $\mathcal{R}$ of random bitstrings and a set $\mathcal{X}$ of variables, using bitstring operations, hash functions drawn from a set $\mathcal{H}$, and trapdoor permutations drawn from a set $\mathcal{F}$. The grammar of expressions is given in Figure 1. For instance, the encryption algorithm of OAEP may be represented as the expression:

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

Expressions are typed using a size-base type system. A type $\ell \in \mathcal{S}$ is either a size variable $\imath$ or the sum $\ell + n$ of two types.

e	::=	x	variable
	|	r	uniformly random bitstring
	|	0^n	all-zero bitstring of length n
	|	$\mathsf{f}(e)$	permutation
	|	$\mathsf{f}^{-1}(e)$	inverse permutation
	|	$H(e)$	hash function
	|	$e \oplus e$	exclusive-or
	|	$e \,\|\, e$	concatenation
	|	$[e]^\ell_n$	drop n bits then take ℓ bits

where $x \in \mathcal{X}$, $r \in \mathcal{R}$, $n, \ell \in \mathcal{S}$, $\mathsf{f} \in \mathcal{F}$, and $H \in \mathcal{H}$.

Figure 1: Syntax of Expressions

$$(e_1 \oplus e_2) \oplus e_3 = e_1 \oplus (e_2 \oplus e_3) \qquad e_1 \oplus e_2 = e_2 \oplus e_1$$
$$[e_1 \oplus e_2]^{k+n}_\ell = [e_1 \oplus e_2]^k_\ell \,\|\, [e_1 \oplus e_2]^n_{\ell+k} \qquad [0^k]^n_\ell = 0^n$$
$$(e_1 \,\|\, e_2) \,\|\, e_3 = e_1 \,\|\, (e_2 \,\|\, e_3) \qquad [e_1 \,\|\, e_2]^{|e_1|}_0 = e_1$$
$$[e_1 \,\|\, e_2]^\ell_{|e_1|+n} = [e_2]^\ell_n \qquad [e_1 \oplus e_2]^\ell_n = [e_1]^\ell_n \oplus [e_2]^\ell_n$$
$$[e]^{|e|}_0 = e \qquad [[e]^\ell_n]^{\ell'}_{n'} = [e]^{\ell'}_{n+n'} \qquad [e]^\ell_n \,\|\, [e]^{\ell'}_{n+\ell} = [e]^{\ell+\ell'}_n$$
$$e \oplus 0^{|e|} = e \quad e \oplus e = 0^{|e|} \quad \mathsf{f}(\mathsf{f}^{-1}(e)) = e \quad \mathsf{f}^{-1}(\mathsf{f}(e)) = e$$

Figure 2: Equational theory of bitstrings

We note $|e|$ the type of an expression e, which intuitively represents the length of the bitstring it denotes. For instance, the type of $e_1 \,\|\, e_2$ is $|e_1| + |e_2|$ and $e_1 \oplus e_2$ is well-typed iff $|e_1| = |e_2|$. In an asymptotic setting, we interpret size variables as functions that grow polynomially with the security parameter.

We let $\mathcal{R}(e)$, $\mathcal{X}(e)$, $\mathcal{F}(e)$ and $\mathcal{H}(e)$ denote, respectively, the sets of random bitstrings, variables, trapdoor permutation, and hash function symbols occurring in an expression e. We note $e\{\vec{e}_1/\vec{e}_0\}$ the simultaneous substitution of $\vec{e}_0$ by $\vec{e}_1$ in e and $\mathcal{ST}_H(e)$ the set of sub-expressions of e that are of the form $H(e')$. Abusing notation, we write e.g. $\mathcal{R}(e_1, e_2)$ instead of $\mathcal{R}(e_1) \cup \mathcal{R}(e_2)$.

Given values for size variables in $|e|$ and trapdoor permutations with adequate domains, we interpret a well-typed expression e as a probabilistic algorithm $(\!|e|\!)$ that calls oracles in $\mathcal{H}(e)$ and takes as parameters values for variables in $\mathcal{X}(e)$. The algorithm is implicitly parametrized by public and secret keys (if required) of the trapdoor permutations that occur in e.

3.2 Equality and Deducibility

We model algebraic properties of bitstrings by the equational theory $=_E$, defined as the smallest congruence relation that contains all instances of the axioms of Figure 2. The relation $=_E$ is sound [8] for all trapdoor permutations and valid key pairs. This means that for all closed expressions s and t, $s =_E t$ implies that $(\!|s|\!)$ and $(\!|t|\!)$ return the same value when shared random bitstrings are jointly sampled; formally,

$$\Pr\left[y \leftarrow (\!|s \,\|\, t|\!) : [y]^{|s|}_0 = [y]^{|t|}_{|s|}\right] = 1.$$

We use the standard notion of deducibility from symbolic cryptography to reason about adversarial knowledge. Infor-

mally, an expression e' is deducible from e if there exists an efficient algorithm that computes e' from e. We use contexts to describe such algorithms symbolically. A context is an expression with a distinguished variable $*$. The application $C[e]$ of a context C to an expression e is defined by substitution.

Definition 4 (Deducibility) *An expression e' is deducible from another expression e, written $e \vdash e'$, if there is a context C such that*

1. *$C[e]$ is well-typed;*
2. *$\mathcal{R}(C) = \emptyset$;*
3. *f^{-1} does not occur in C, and*
4. *$C[e] =_E e'$.*

We define $\vdash^\star$ analogously, but dropping restriction 3.

We prove that $\vdash$ is sound in the following sense: $e \vdash e'$ implies that there is an efficient algorithm that computes e' from e and the public keys of the trapdoor permutations in e and e'. For $\vdash^\star$ we obtain an identical result, except that the corresponding algorithm can also use the secret keys of the trapdoor permutations. Note that we do not require $\vdash$-completeness, also known as $\not\vdash$-soundness [8], since we never use the deducibility relation to conclude that an expression is not computable from another.

3.3 Inequality and Static Distinguishability

Static equivalence [1] models symbolic indistinguishability of expressions. Informally, two expressions e_0 and e_1 are statically equivalent if all tests of the form $C[\cdot] =_E C'[\cdot]$ succeed for e_0 if and only if they succeed for e_1. Since the completeness of $=_E$ with respect to the computational interpretation of expressions is an open question, we use a relation inspired by static equivalence to model distinguishability of expressions. We call this relation static distinguishability, and define it in terms of an apartness relation $\sharp$ in place of $\neq_E$. The soundness of this relation implies that if $s \sharp t$ holds, then the probability that the results of $(\!|s|\!)$ and $(\!|t|\!)$ coincide when shared random bitstrings are jointly sampled is negligible. The inductive definition of $\sharp$ comprises axioms such as $r \sharp 0$ and $r \sharp r'$, and rules stating that $s \sharp t$ implies $H(s) \sharp H(t)$, that $C[s] \sharp C[t]$ implies $s \sharp t$, and that $s \sharp t$ and $t =_E u$ implies $s \sharp u$.

Definition 5 (Static Distinguishability) *The relation $\not\approx$ is the smallest symmetric relation between expressions such that $e_0 \not\approx e_1$ iff there are contexts C and C' such that*

1. *$C[e_0]$, $C'[e_0]$, $C[e_1]$, and $C'[e_1]$ are well-typed;*
2. *$\mathcal{R}(C) = \emptyset$;*
3. *f^{-1} does not occur in C, and*
4. *$C[e_0] =_E C'[e_0]$ and $C[e_1] \sharp C'[e_1]$.*

We use static distinguishability to find attacks on encryption schemes. For the attacks to be meaningful, we rely on the soundness of $\not\approx$, which follows from the soundness of $=_E$ and $\sharp$. Informally, if $e_0 \not\approx e_1$ holds, then there is an efficient algorithm $\mathsf{distinguish}$ that when given as input the public keys of trapdoor permutations and a value computed using $(\!|e_0|\!)$ returns 0, but when given instead a value computed using $(\!|e_1|\!)$ returns 1, except with negligible probability.

As an example, the expressions $e_0 = \mathsf{f}(r) \parallel (r \oplus 0)$ and $e_1 = \mathsf{f}(r) \parallel (r_1 \oplus r_2)$ are statically distinguishable, because for the contexts

$$C = [*]_0^{|\mathsf{f}(r)|} \text{ and } C' = \mathsf{f}\left([*]_{|\mathsf{f}(r)|}^{|r|}\right)$$

it holds that $C[e_0] =_E C'[e_0]$ and $C[e_1] \sharp C'[e_1]$. A simple distinguisher algorithm applies the algorithms corresponding to C and C' to its input and returns 0 if this yields equal bitstrings and 1 otherwise.

4. PROOF SYSTEMS

This section introduces logics for proving chosen-plaintext and chosen-ciphertext security of padding-based encryption schemes by reduction to computational assumptions on the underlying trapdoor permutations. For the sake of readability, our presentation separates the derivation of valid judgments (§ 4.1 and § 4.2) from the computation of concrete security bounds from valid derivations (§ 4.3). The soundness of the logics is stated in § 4.4.

4.1 Chosen-Plaintext Security

Judgments predicate over the probability of an event ϕ in the CPA experiment where the challenge ciphertext is computed using $(\!|c^\star|\!)$, which we denote $\mathsf{CPA}_{c^\star}$. Events are drawn from the grammar

$$\phi ::= \mathsf{Guess} \mid \mathsf{Ask}(H, e) \mid \phi \wedge \phi$$

where Guess corresponds to the adversary correctly guessing the hidden bit b, and $\mathsf{Ask}(H, e)$ corresponds to the adversary asking the query $H(e)$.

Judgments are of the form $\vDash_{\hat{p}} c^\star : \phi$, where $c^\star$ is a well-typed expression with $\mathcal{X}(c^\star) = \{m\}$, ϕ is an event, and $\hat{p} \in \{0, 1/2\}$ is a probability tag[1]. The CPA logic is designed to ensure that if $\vDash_{\hat{p}} c^\star : \phi$ is derivable, then the probability of ϕ in the experiment $\mathsf{CPA}_{c^\star}$ is upper bounded by $\hat{p}$ plus a negligible term. In particular, if $\vDash_{1/2} c^\star : \mathsf{Guess}$ is derivable, then any scheme Π with encryption algorithm $(\!|c^\star|\!)$ is asymptotically secure against chosen-plaintext attacks.

Derivability in the logic is parametrized by a set Γ of computational assumptions of the form (f, k, ℓ), where $\mathsf{f} \in \mathcal{F}$ and $k, \ell \in \mathcal{S}$. Such assumptions state that it is hard to compute $[r]_k^\ell$ from $\mathsf{f}(r)$. There is no restriction on Γ: it may contain assumptions for several permutations, and multiple assumptions for any given permutation.

Figure 3 presents the rules of the proof system. Note that the terminal rules [Rnd] and [OW] are decorated with information that is needed to compute concrete bounds (§ 4.3).

The rules [Opt], [Perm], [Merge], and [Split] correspond to bridging steps in proofs and allow reformulating judgments into equivalent forms. The rule [Opt] corresponds to *optimistic sampling*, which allows substituting $e \oplus r$ by r provided that $r \notin \mathcal{R}(e)$. The rule [Perm] replaces all occurrences of $\mathsf{f}(r)$ by r. The rule [Merge] replaces the concatenation of two random bitstrings r_1 and r_2 with a fresh random bitstring r. The rule [Split] performs the opposite transformation; it replaces a random bitstring r with the concatenation of two fresh random bitstrings r_1 and r_2.

Rule [Sub] can be used to weaken the event and replace expressions in judgments with equivalent expressions. This

[1]Probability tags increase the readability of rules, and support a more uniform interpretation of judgments. However, they are superfluous and can be deduced from the shape of the event: for every valid judgment $\vDash_{\hat{p}} c^\star : \phi$, the probability tag $\hat{p}$ is $1/2$ iff Guess appears in ϕ.

$$\frac{\vDash_{\hat{p}} c^\star : \phi \qquad r \notin \mathcal{R}(e)}{\vDash_{\hat{p}} c^\star\{e \oplus r/r\} : \phi\{e \oplus r/r\}}[\mathsf{Opt}] \qquad \frac{\vDash_{\hat{p}} c^\star : \phi}{\vDash_{\hat{p}} c^\star\{\mathsf{f}(r)/r\} : \phi\{\mathsf{f}(r)/r\}}[\mathsf{Perm}] \qquad \frac{\vDash_{\hat{p}} c^\star : \phi \qquad r_1, r_2 \notin \mathcal{R}(c^\star, \phi)}{\vDash_{\hat{p}} c^\star\{r_1 \,\|\, r_2/r\} : \phi\{r_1 \,\|\, r_2/r\}}[\mathsf{Merge}]$$

$$\frac{\vDash_{\hat{p}} c^\star\{r_1 \,\|\, r_2/r\} : \phi\{r_1 \,\|\, r_2/r\} \qquad r_1, r_2 \notin \mathcal{R}(c^\star, \phi)}{\vDash_{\hat{p}} c^\star : \phi}[\mathsf{Split}] \qquad \frac{\vDash_{\hat{p}} c_2^\star : \phi_2 \quad c_1^\star =_E c_2^\star \quad \phi_1 \Longrightarrow_E \phi_2}{\vDash_{\hat{p}} c_1^\star : \phi_1}[\mathsf{Sub}]$$

$$\frac{\vDash_{\hat{p}} c^\star : \phi \quad \vDash_0 c^\star : \mathsf{Ask}(H, e) \quad r \notin \mathcal{R}(e) \quad \mathcal{ST}_H(c^\star, \phi) = \emptyset}{\vDash_{\hat{p}} c^\star\{H(e)/r\} : \phi\{H(e)/r\}}[\mathsf{Fail}_1]$$

$$\frac{\vDash_{\hat{p}} c^\star : \phi \quad \vDash_0 c^\star\{H(e)/r\} : \phi\{H(e)/r\} \wedge \mathsf{Ask}(H, e) \quad r \notin \mathcal{R}(e) \quad \mathcal{ST}_H(c^\star, \phi) = \emptyset}{\vDash_{\hat{p}} c^\star\{H(e)/r\} : \phi\{H(e)/r\}}[\mathsf{Fail}_2] \qquad \frac{\mathcal{X}(c^\star) = \emptyset}{\vDash_{1/2} c^\star : \mathsf{Guess}}[\mathsf{Ind}]$$

$$\frac{\vec{e} \vdash^\star r \qquad r \notin \mathcal{R}(c^\star)}{\vDash_0 c^\star : \mathsf{Ask}(H_1, e_1) \wedge \cdots \wedge \mathsf{Ask}(H_n, e_n)}[\mathsf{Rnd}(|r|, \vec{H})] \qquad \frac{\vec{e} \,\|\, r_2 \,\|\, m \vdash [r_1]_k^\ell \quad \mathsf{f}(r_1) \,\|\, r_2 \,\|\, m \vdash c^\star \quad r_1 \neq r_2 \quad (\mathsf{f}, k, \ell) \in \Gamma}{\vDash_0 c^\star : \mathsf{Ask}(H_1, e_1) \wedge \cdots \wedge \mathsf{Ask}(H_n, e_n)}[\mathsf{OW}_k^\ell(f, \vec{H})]$$

Figure 3: Proof rules of the CPA logic ($\Longrightarrow_E$ denotes implication in first-order logic with respect to $=_E$)

rule is often needed to prepare the ground for the application of another rule: e.g. rule [Opt] can be applied to a judgment that contains $H(r)$ to obtain $H(e \oplus r)$ by first using [Sub] to replace $H(r)$ by $H(e \oplus (e \oplus r))$.

The rules [Fail_1] and [Fail_2], for *equivalence up to failure*, correspond to specialized forms of Shoup's Fundamental Lemma [29] and create two branches in a derivation. These rules can be used to substitute an expression $H(e)$ by a random bitstring r, incurring a probability loss in the reduction corresponding to the probability of the CPA adversary asking the query $H(e)$. One can choose to analyze this probability either after (as in rule [Fail_1]) or before applying the substitution (as in rule [Fail_2]).

Finally, the rules [Ind], [Rnd], and [OW] are terminal and provide a means of directly bounding the probability of an event. The first two rules are information-theoretic, whereas the third formalizes a reduction to a computational assumption. The rule [Ind] closes a branch when the challenge ciphertext no longer depends on a plaintext variable; in terms of the CPA experiment, this means that the challenge is independent from the hidden bit b, and hence the probability that an adversary guesses it is exactly $1/2$. The rule [Rnd] closes a branch in the case when the adversary must make a number of oracle queries that would require guessing random values that are independent from its view. The rule [OW] closes a branch when it is possible to find a reduction to the problem of partially inverting the underlying trapdoor permutation: this is the case when the image of a random element (the challenge of an inverter) can be embedded in the challenge ciphertext of the CPA experiment in such a way that its (partial) pre-image can be computed from the oracle queries made by the CPA adversary. To apply this rule, r_1 is usually obtained by searching for applications of f in $c^\star$ and r_2 is set to the concatenation of all random bitstrings in $\mathcal{R}(c^\star) \setminus \{r_1\}$. Additionally, one must check that the assumption (f, k, ℓ) is an assumption in Γ.

4.2 Chosen-Ciphertext Security

Judgments are of the form $\vDash_{\hat{p}} (c^\star, D) : \phi$, where $\hat{p}$ is a probability tag, $c^\star$ is an expression, D is a decryption oracle, and ϕ is an event. The challenge ciphertext $c^\star$ and the tag $\hat{p}$ can take the same values as in the CPA logic, whereas the decryption oracle is drawn from the following grammar

$$\begin{array}{lcl} F & ::= & \mathsf{find}\ x\ \mathsf{in}\ \boldsymbol{L}_H, F \ \mid\ \diamond \\ T & ::= & e = e' \ \mid\ \mathsf{Ask}(H, e) \ \mid\ T \wedge T \\ D & ::= & F : T \rhd e \end{array}$$

We assume that D does not contain random bitstrings and that all variables in expressions in D are bound by find, except for a distinguished parameter c denoting the ciphertext queried to the oracle. The above grammar is sufficiently expressive to encode decryption algorithms of padding-based schemes, as well as plaintext-simulators used in reductions.[2]

Informally, given as input a ciphertext c, an algorithm $F : T \rhd e$ searches among the oracle queries made by an adversary for values $\vec{v}$ satisfying T. If such values are found, it returns the value of $e\{\vec{v}/\vec{x}\}$; otherwise, it returns $\perp$. Conditions in T are interpreted as equality checks on bitstrings and membership tests in the lists of oracle queries made by the adversary. If the sequence of find expressions is empty, we just write $T \rhd e$. We note $(\!|D|\!)$ the interpretation of D as an algorithm.

Events of the logic are as in the CPA logic or of the form $\exists x \in \boldsymbol{L}_\mathcal{D}.\ T$ (when x does not occur in T, we write just T). Events Guess and $\mathsf{Ask}(H, e)$ are defined as in the CPA logic but for the CCA experiment. For $\exists x \in \boldsymbol{L}_\mathcal{D}.\ T$, the quantified variable ranges over the ciphertexts queried to the decryption oracle during the second phase of the CCA experiment; tests are interpreted as above.

A judgment $\vDash_{\hat{p}} (c^\star, D) : \phi$ predicates over the probability of ϕ in a CCA experiment where the challenge ciphertext is computed using $(\!|c^\star|\!)$ and decryption queries are answered by $(\!|D|\!)$. The judgment states that the probability of ϕ in this experiment is upper bounded by $\hat{p}$ plus a negligible term.

Figure 4 presents the rules of the logic. The rule [Bad] allows to transform the decryption oracle so that it rejects ciphertexts whose decryption requires oracle queries that have not yet been made by the adversary. The resulting decryption oracle no longer makes queries to the given oracle. It is therefore sufficient to consider two events that can lead the adversary to distinguish between the original and transformed oracles: 1. when the adversary makes a de-

[2] A plaintext-simulator is an algorithm that extracts the plaintext from a ciphertext by reconstructing it from oracle queries made by an adversary, without using the trapdoor to the underlying permutation; a plaintext-simulator that does not query any hash oracle is called a plaintext-extractor.

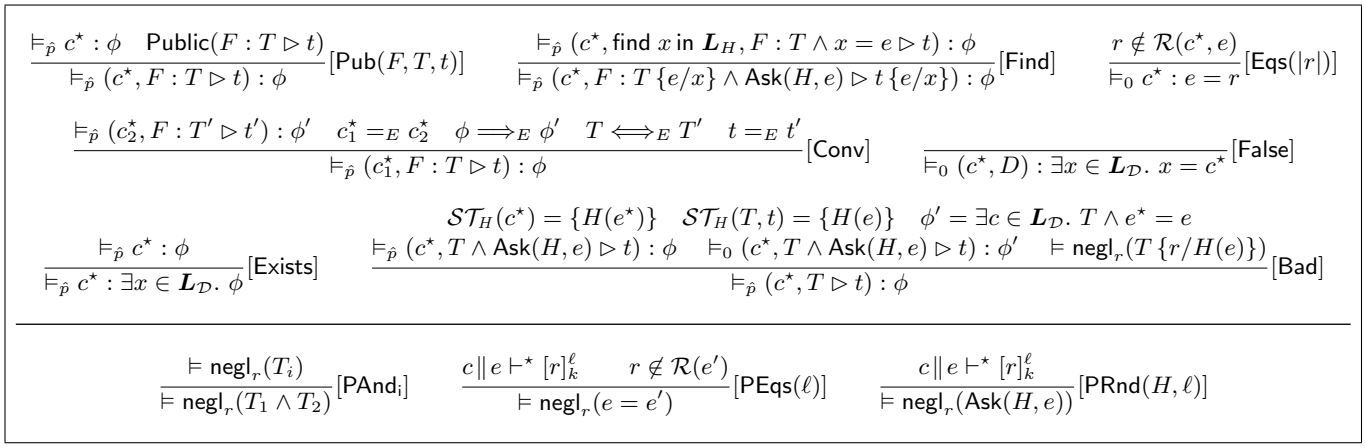

Figure 4: CCA logic: proof rules for CCA judgments, extended rules for CPA logic, and proof rules for tests

cryption query c such that T succeeds after making a query $H(e)$ that is also needed to compute the challenge ciphertext; 2. when the test T succeeds, even though it gets a random answer from H. The probability of this last event can be proven negligible using the rules [PAnd], [PEqs], and [PRnd] in Fig. 4.

The rule [False] captures the fact that the adversary cannot ask for the decryption of $c^\star$ during the second phase of the CCA experiment; the probability of this happening is 0. Rule [Conv] allows switching between observationally equivalent decryption oracles and weakening the event considered; it is usually used to transform a test that requires f^{-1} into an equivalent test that only requires f.

The rule [Find] allows replacing an oracle that computes a value explicitly by one that computes it from the oracle queries made by the adversary.

The rule [Pub] links the CCA with the CPA logic, and captures the intuition that an adversary does not gain any advantage from getting access to a publicly simulatable decryption oracle. Here, $\mathsf{Public}(D)$ holds if f^{-1} is not used in D. Note that the second premise may be of the form $\vDash_{\hat{p}} c^\star : \phi$, where ϕ is an event of the CCA logic. This kind of judgments are handled by the rules [Exists] and [Eqs] given in Figure 4, which extend the CPA logic to the additional events used in the CCA logic.

4.3 Concrete Bounds

In this section we show how to extract concrete security bounds from derivations in our logics. Security bounds p are drawn from the grammar

$$p ::= \epsilon \mid \tfrac{1}{2} + \epsilon \quad q ::= 1 \mid q \times q \mid q_H \quad t ::= t_{\mathcal{A}} \mid q \times t_{\mathsf{f}} \mid t + t$$

$$\epsilon ::= 2^{-k} \mid \mathbf{Succ}^{\mathsf{s\text{-}pd\text{-}OW}}_{\Theta_{\mathsf{f}}}(k, \ell, q, t) \mid q \times \epsilon \mid \epsilon + \epsilon$$

where $k, \ell \in \mathcal{S}$, $H \in \mathcal{H}$, t_f is a worst-case bound for the time required to evaluate f, and $t_{\mathcal{A}}$ is the execution time of $\mathcal{A}$.

Security bounds are computed from a derivation ∇ in the CPA or CCA logic using a function $\mathcal{B}$ parametrized by the computational resources of an adversary, namely, its execution time $t_{\mathcal{A}}$ and the number of oracle queries it can make to each oracle, $\vec{q}$. The definition of the bound function $\mathcal{B}$ is given in Figure 5 using the following conventions. We omit the resource parameters when they remain unchanged. Moreover, for a derivation ∇ with rule R at its root, we use L_∇ to refer to R's label and ∇_i to refer to the derivation of the i-th premise of R.

$$\mathcal{B}_{(t_{\mathcal{A}}, \vec{q})}(\nabla) = \begin{cases} \mathcal{B}(\nabla_1) + \mathcal{B}(\nabla_2) & \text{if } L_\nabla = [\mathsf{Fail}_{1,2}] \\ \mathcal{B}(\nabla_1) + \mathcal{B}(\nabla_2) + q_D \times \mathcal{B}(\nabla_3) & \text{if } L_\nabla = [\mathsf{Bad}] \\ \mathcal{B}_{(t'_{\mathcal{A}}, q'_{H_1}, \ldots, q'_{H_n})}(\nabla_1) & \text{if } L_\nabla = [\mathsf{Pub}(F,T,u)] \\ 1/2 & \text{if } L_\nabla = [\mathsf{Ind}] \\ 0 & \text{if } L_\nabla = [\mathsf{False}] \\ q_{\vec{H}} \times 2^{-\ell} & \text{if } L_\nabla = [\mathsf{Rnd}(\vec{H}, \ell)] \\ q_H \times 2^{-\ell} & \text{if } L_\nabla = [\mathsf{PRnd}(H, \ell)] \\ 2^{-\ell} & \text{if } L_\nabla = [\mathsf{Eqs}(\ell)] \\ 2^{-\ell} & \text{if } L_\nabla = [\mathsf{PEqs}(\ell)] \\ \mathbf{Succ}^{\mathsf{s\text{-}pd\text{-}OW}}_{\Theta_{\mathsf{f}}}(k, \ell, q_{\vec{H}}, t_{\mathcal{A}} + q_{\vec{H}}) & \text{if } L_\nabla = [\mathsf{OW}^\ell_k(\mathsf{f}, \vec{H})] \\ \mathcal{B}(\nabla_1) & \text{otherwise} \end{cases}$$

where $q_{\vec{H}} = \prod_{H_i \in \vec{H}} q_{H_i}$ and

$$t'_{\mathcal{A}} = t_{\mathcal{A}} + q_D \times (\mathsf{T}(u) + \mathsf{T}(T) \times \prod_{H_j \in F} q_{H_j})$$

$$q'_{H_i} = q_{H_i} + q_D \times (\mathsf{Q}_i^{F,T}(u) + \mathsf{Q}_i^F(T) \times \prod_{H_j \in F} q_{H_j})$$

Figure 5: Function $\mathcal{B}$ for bound computation

For all bridging rules, the bound is inherited from their single premise. For rules $[\mathsf{Fail}_1]$ and $[\mathsf{Fail}_2]$, the bound is computed as $p_1 + p_2$ where p_1 is the probability of the original event in the transformed experiment and p_2 bounds the probability of failure. Rule [Bad] is similar except that bounds come from the probability of two different failure events; the second one can be triggered in any decryption query and its bound must be multiplied by the factor q_D.

The [Pub] case represents a reduction from CCA to CPA, where a simulator $\mathcal{S}$ uses a CCA adversary $\mathcal{A}$, to win in the CPA experiment. We therefore have to compute bounds for the computational resources $(t'_{\mathcal{A}}, \vec{q}')$ used by $\mathcal{S}$ in terms of the resources of $\mathcal{A}$ and the plaintext-simulator $F : T \rhd u$. We first define a function T that computes a bound for the time required to evaluate a test T or an expression u. Then, $t'_{\mathcal{A}}$ can be defined as $t_{\mathcal{A}} + q_D \times t_{Dec}$ where t_{Dec} is the the time required to evaluate the test T on each combination of queries traversed by F plus the time required to evaluate u once. Similarly, we define a function Q_i that bounds the

number of queries made to H_i during the simulation of the decryption oracle, and use it to compute q'_{H_i}.

The rules [Ind] and [False], yield exact bounds that can be readily used. The cases [Eqs], [PEqs], [Rnd], and [PRnd] correspond to the probability of guessing a random bitstring of length ℓ, respectively in $q_{\vec{H}}$, q_H, or just 1 tries. In the case [OW], $\mathcal{B}$ directly returns the success probability for set partial-domain one-wayness for the appropriate parameters. If $k = 0$ and $\ell = |f|$, we can use the standard reduction from set one-wayness to one-wayness to obtain the bound $\mathbf{Succ}^{\mathsf{OW}}_{\Theta_f}(t_{\mathcal{A}} + q_{\vec{H}} + q_{\vec{H}} \times t_f)$. Here, the adjusted time bound accounts for the fact that the simulator may have to apply f to every element in the set to find the pre-image.

4.4 Soundness

Let Π be a scheme with encryption algorithm $(\!|c^\star|\!)$ and decryption algorithm $(\!|D|\!)$. Assume that the interpretations of trapdoor permutations satisfy the assumptions in Γ.

Theorem 1 *If ∇ is a derivation of $\vDash_{\frac{1}{2}} c^\star : \mathsf{Guess}$ then*

$$\mathbf{Adv}^{\mathsf{CPA}}_{\Pi}(t_{\mathcal{A}}, \vec{q}) \leq 2\ \mathcal{B}_{(t_{\mathcal{A}}, \vec{q})}(\nabla) - 1$$

Moreover, this bound is negligible in the security parameter.

Theorem 2 *If ∇ is a derivation of $\vDash_{\frac{1}{2}} (c^\star, D) : \mathsf{Guess}$ then*

$$\mathbf{Adv}^{\mathsf{CCA}}_{\Pi}(t_{\mathcal{A}}, \vec{q}) \leq 2\ \mathcal{B}_{(t_{\mathcal{A}}, \vec{q})}(\nabla) - 1$$

Moreover, this bound is negligible in the security parameter.

The proofs of these theorems rest on showing the soundness of each individual rule using game-based techniques.

4.5 Example

In this paragraph, we illustrate the use of the CPA logic to prove the chosen-plaintext security of OAEP under one-wayness (OW) and partial-domain one-wayness (PDOW) assumptions on the underlying permutation. Additionally, we show the concrete security bounds extracted from the corresponding derivations. A similar illustration of the CCA logic will appear in the full version of the paper.

The proofs under one-wayness and partial-domain one-wayness share a common part, depicted at the bottom of Figure 6. This first part proceeds by applying [Fail_1] to replace the expression $G(r)$ with a fresh random bitstring r'. To close the branch corresponding to the Guess event, we apply optimistic sampling, replacing $r' \oplus (m \,\|\, 0)$ by r'. In the judgment thus obtained, the challenge ciphertext does not depend on the plaintext, so we conclude by applying [Ind]. We continue with the branch corresponding to the failure event $\mathsf{Ask}(G, r)$ applying optimistic sampling. At this point there are two different ways to proceed, that yield proofs w.r.t. different hypotheses.

The derivation for PDOW is depicted in the middle of Figure 6. The proof proceeds by applying rule [Fail_1] to replace $H(r')$ by a fresh random bitstring r''. The rule [Opt] is then applied in both premises to eliminate r. The branch corresponding to the event $\mathsf{Ask}(G, r)$ can be closed by applying rule [Rnd] because the challenge ciphertext does not depend on r anymore. The branch corresponding to the event $\mathsf{Ask}(H, r')$ is closed by merging the random bitstrings r' and r as r and then reformulating the event as $\mathsf{Ask}(H, [r]^\ell_0)$. At this point, it is clear that querying H on the ℓ most significant bits of r amounts to inverting f in its ℓ most significant bits, so we conclude applying [$\mathsf{OW}^\ell_0(\mathsf{f}, H)$].

The derivation for OW is depicted at the top of Figure 6. It proceeds by applying rule [Opt] to move the expression $H(r')$ from the challenge expression to the event, that becomes $\mathsf{Ask}(G, H(r') \oplus r)$. We then apply [$\mathsf{Fail}_2$] to replace $H(r')$ by a fresh random bitstring r''. Note that this rule performs a substitution only on one of the branches. The branch corresponding to event $\mathsf{Ask}(G, r'' \oplus r)$ is closed by performing an optimistic sampling that replaces $r'' \oplus r$ by r''. At this point, the event is no longer dependent on the challenge, so we conclude using [Rnd]. Finally, the last branch is closed by merging $r' \,\|\, r$ into a fresh random bitstring r''. We adapt the event accordingly and obtain

$$\mathsf{Ask}(H, [r'']^\ell_0) \wedge \mathsf{Ask}(G, H([r'']^\ell_0) \oplus [r'']^k_\ell).$$

If the adversary manages to query $[r'']^{|r'|}_0$ to H and $H([r'']^\ell_0) \oplus [r'']^k_\ell$ to G, it can easily obtain r'': the first part is directly queried to H, the second can be obtained by taking the first part, hashing it with H and computing the exclusive-or with the value queried to G. Hence, the adversary can completely invert function f and we conclude by applying [$\mathsf{OW}^0_{|r''|}(\mathsf{f}, H, G)$].

Applying the bound function $\mathcal{B}_{(t_{\mathcal{A}}, (q_G, q_H))}$ to the derivation $\nabla_{\mathsf{OAEP}}, \nabla_{\mathsf{PDOW}}$ returns

$$1/2 + q_G \times 2^{-|r|} + \mathbf{Succ}^{\mathsf{s\text{-}pd\text{-}OW}}_{\Theta}(\ell, q_H, t_{\mathcal{A}} + q_H).$$

Applying the bound function $\mathcal{B}_{(t_{\mathcal{A}}, (q_G, q_H))}$ to the derivation $\nabla_{\mathsf{OAEP}}, \nabla_{\mathsf{OW}}$ returns (after applying the standard reduction from set one-wayness to one-wayness)

$$1/2 + q_G \times 2^{-|r|} + \mathbf{Succ}^{\mathsf{OW}}_{\Theta}(t_{\mathcal{A}} + q_H \times q_G + q_H \times q_G \times t_f).$$

5. ATTACKS

We present our approach for finding attacks against chosen-plaintext and chosen-ciphertext security of padding-based encryption schemes. Since our logics are incomplete, we use attack finding to obtain negative results and additional data points to evaluate schemes for which we cannot obtain proofs.

We distinguish between universal attacks and existential attacks relative to a set of one-wayness assumptions. An attack is universal if it works against every possible instantiation of the trapdoor permutations used by a scheme. An attack is existential if it relies on specific properties of the employed trapdoor permutation.

5.1 Universal attacks

To find universal attacks against CPA security of an encryption algorithm e, we search for closed expressions m_0 and m_1 that do not contain symbols of the form f^{-1} such that $e\,\{m_0/m\} \not\approx e\,\{m_1/m\}$. By soundness of $\not\approx$, there exists an efficient algorithm distinguish that returns 0 for input $(\!|m_0|\!)$ and 1 for input $(\!|m_1|\!)$ with overwhelming probability. To mount an attack against CPA security using this algorithm, an adversary chooses plaintexts $(\!|m_0|\!)$ and $(\!|m_1|\!)$, receives the challenge ciphertext $c^\star$, and returns $\mathsf{distinguish}(c^\star)$.

To analyze the CCA security of schemes, we search for malleability attacks using deducibility. More precisely, we search for closed expressions m_0 and $\Delta \neq 0^n$ such that

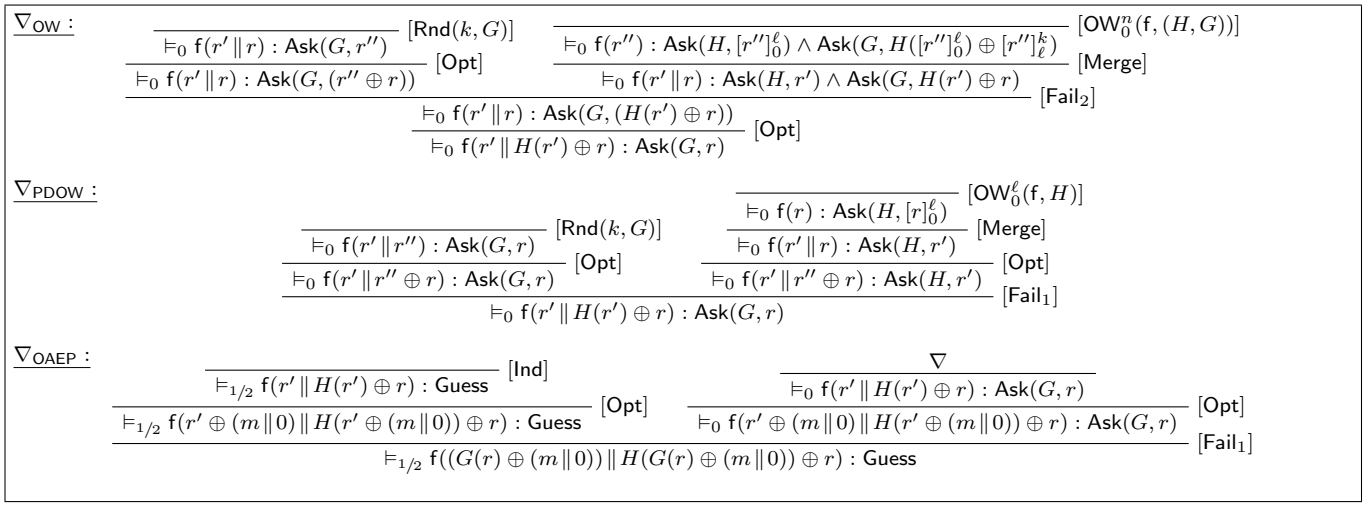

Figure 6: Derivations for CPA security of OAEP.

$e\{m_0/m\} \vdash e\{m_0 \oplus \Delta/m\}$; denote maul the corresponding algorithm. This would imply an effective attack against IND-CCA: first choose plaintexts m_0 and $m_1 \neq m_0$ and obtain the challenge $c^\star$; then query the decryption oracle on $\mathsf{maul}(pk, c^\star)$, xor the result with Δ, and return 0 if it equals $(\!|m_0|\!)$ and 1 otherwise. An example of a scheme vulnerable to a universal attack of this form is the Zheng-Seberry scheme [31], whose encryption algorithm is given by $e = \mathsf{f}(r) \,\|\, (G(r) \oplus m) \,\|\, H(m)$. The adversary can use the distinguisher corresponding to $C = [*]_{|r|+|m|}^{|H(m)|}$ and $C' = H(0^{|m|})$ to tell appart $e\{0/m\}$ from $e\{1/m\}$.

5.2 Existential attacks

To find existential attacks against an encryption algorithm e w.r.t. a set of assumptions Γ, we find universal attacks against a modified e' of it. An example can elucidate better the point.

Consider the encryption algorithm of ZAEP [7], $e = \mathsf{f}(r \,\|\, G(r) \oplus m)$. To show that there is no blackbox reduction from the CPA security of ZAEP to the assumption $(f, 0, |\mathsf{f}|)$, we use the instantiation $\mathsf{f}(a \,\|\, b) = a \,\|\, \mathsf{f}_2(b)$ for f. If f_2 is a one-way permutation, the permutation f satisfies the assumption $(f, 0, |\mathsf{f}|)$. Using static distinguishability, we find an attack on $e' = r \,\|\, \mathsf{f}_2(G(r) \oplus m)$ given by the contexts

$$C = [*]_{|r|}^{|m|} \text{ and } C' = \mathsf{f}_2\left(G\left([*]_{|m|}^{|0|}\right)\right)$$

which can be used to distinguish $e\{0/m\}$ and $e\{1/m\}$. To show that there is no blackbox reduction of the CCA security of ZAEP to an arbitrary Γ, we use the instantiation

$$\mathsf{f}(a) = \mathsf{f}'\left([a]_0^{|a|-c}\right) \Big\| [a]_{|a|-c}^c$$

Assuming f' satisfies the assumptions Γ (accounting for the size reduction by c), this instance of f satisfies Γ. It is easy to see that the resulting e' is malleable.

More generally, to prove that there is no blackbox reduction for a fixed set of one-wayness assumptions Γ, we must either find a universal attack or instantiations of the trapdoor permutations that yield attacks and are compatible with *all* assumptions in Γ. For example, the instantiation above is compatible with all one-wayness assumptions and the instantiation we used for ZAEP is compatible with all one-wayness assumptions except those of the form (f, k, ℓ), where $0 < \ell$ and $k + \ell \leq |a|$. In addition to the aforementioned instantiations, we also use instantiations of the form

$$\mathsf{f}(a \,\|\, b \,\|\, c) = \mathsf{f}_1(a) \,\|\, b \oplus a \,\|\, c$$

which allow us to find attacks if f is used in such a way that part of its input are leaked or interdependent.

6. EXPERIMENTAL VALIDATION

We implemented the proof search and attack finding methods described in § 4 and § 5 in a toolset that we coin ZooCrypt. ZooCrypt can prove the CPA and CCA security of a scheme under different assumptions on the trapdoor permutations it uses, or attacks consistent that are consistent with these assumptions.

6.1 Security Analysis

To analyze the security of a scheme with encryption algorithm given by an expression $c^\star$ under a set of assumptions Γ, the toolset follows the workflow depicted in Figure 6.1:

1. Checks that $c^\star$ is well-typed and that encryption is invertible, i.e. $c^\star \vdash^\star m$;
2. Searches for attacks against CPA security;
3. Searches for proofs of CPA security. If a proof is found, computes the corresponding concrete security bound;
4. Searches for malleability attacks against CCA security;
5. If a CPA proof has been found, synthesizes a decryption algorithm D and searches for a proof of CCA security. If a proof is found, computes the corresponding security bound.

The results of this security analysis are an adversary for each attack found, and derivations for all security proofs together with the set of assumptions effectively used and the corresponding concrete security bound. Steps 3 and 5 implement proof search algorithms for the logics. These algorithms try to build a valid derivation bottom up, by applying rules in a prescribed order. Simple heuristics allow to improve the efficiency of the search and to ensure termination.

A crucial step in the above workflow is synthesizing a decryption algorithm that reject as many invalid ciphertexts as possible, with the aim of easing the construction of a plaintext simulator during a CCA analysis. Indeed, an encryption algorithm typically admits several correct decryption

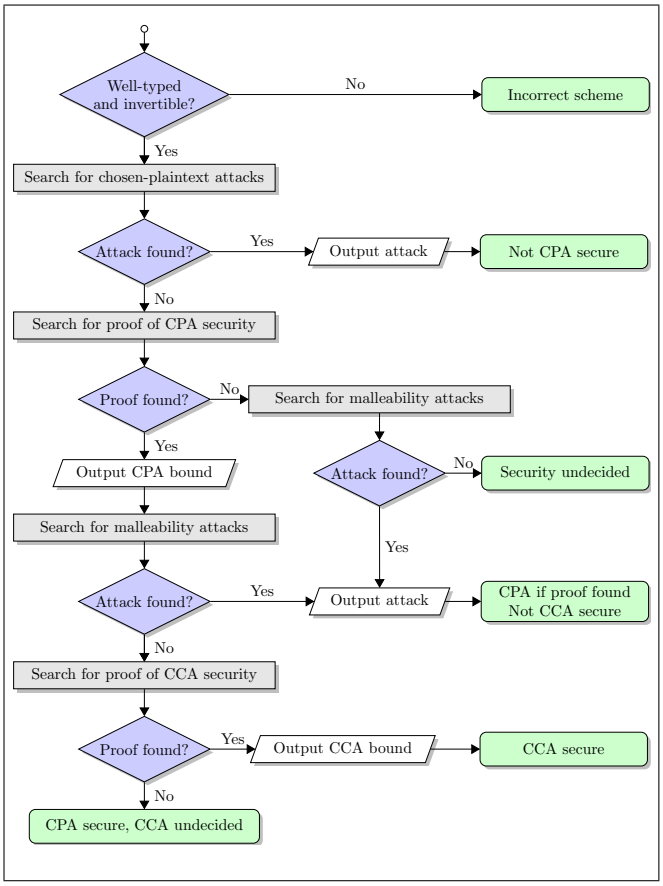

Figure 7: Security analysis workflow

algorithms, because the consistency condition gives complete freedom of choice as to what should be the result of decrypting an invalid ciphertext. The tool infers such algorithms using a method inspired by [? ?] to analyze the *redundancy* built into ciphertexts. We exploit the fact that our algorithm for checking static equivalence computes as a sub-routine non-trivial equations that hold for an expression; when applied to an expression denoting an encryption algorithm, this sub-routine yields tests for checking the validity of ciphertexts.

6.2 Practical Interpretation for RSA

Concrete security bounds can be used to guide the choice of practical parameters for realizations of encryption schemes based on RSA. In order to validate the quality of the bounds output by our tool, we implemented a method based on an extrapolation [24] of the estimated cost of factoring the RSA-768 integer from the RSA Factoring Challenge [22], and on the lattice basis reduction of set partial-domain one-wayness to one-wayness for RSA [19].

Fixing a maximum number of queries to oracles and an admissible advantage p, our tool estimates from the security bound obtained from a derivation, 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 may be that no CCA adversary executing in 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, the estimated cost of inverting RSA, and the bound found for the CCA security of OAEP, our tool estimates a minimum modulus length of 4864 bits, and a ciphertext overhead of around 200 bits.

6.3 Generation of Encryption Schemes

Our tool also implements an algorithm that generates expressions denoting encryption algorithms within budget constraints specified as the number of concatenation, exclusive-or, hash and trapdoor permutation constructors.

Candidate encryption schemes are generated following a top-down approach that uses variables to represent holes in partially specified expressions. Starting from a fully unspecified expression, i.e. just a variable x, at each iterative step the tool picks a hole 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, m, r; this does not consume the budget.

An incremental type-checker is used at each step to discard partially specified expressions that do not have any well-typed instance. For example, $e \oplus (e \,\|\, x)$ is immediately discarded because e cannot be assigned a size regardless of any substitution for the hole x.

We trim large parts of the search space by implementing an early pruning strategy in the style of [26]. Concretely, we apply some simple filters during generation. For instance, given an expression e with holes, we check for the existence of a substitution σ for holes respecting the budget constraints such that $e\sigma \vdash^{\star} m$, and that it is not the case that for all such substitutions $e\sigma \vdash m$ or $e\sigma \,\|\, m \vdash \mathcal{R}(e)$. These filters can be implemented efficiently using memoization to drastically reduce their computational cost.

6.4 Experiments

We evaluate our tools on encryption schemes generated under different budget constraints. Figure 1 summarizes the results of the automated security analysis of §6.1. In the figure, schemes are classified in rows by their size, measured by the total number of operators used in the expression denoting their encryption algorithm.

The reported experiment has been conducted under two classes of assumptions:

1. $\Gamma_1 = \{(\mathsf{f}, 0, |\mathsf{f}|) \mid \mathsf{f} \in \mathcal{F}(c^{\star})\}$, i.e. assuming that all trapdoor permutations are one-way;
2. $\Gamma_2 = \{(\mathsf{f}, k_f, n_f) \mid \mathsf{f} \in \mathcal{F}(c^{\star})\}$ such that $0 \leq k_f$ and $k_f + n_f \leq |\mathsf{f}|$ for all $\mathsf{f} \in \mathcal{F}(c^{\star})$, i.e. one arbitrary one-wayness assumption for each trapdoor permutation;

The columns grouped under OW CPA report the results obtained when analyzing CPA security under Γ_1. Column Proof indicates the number of schemes proved secure, column Attack the number of schemes for which some attack (existential or universal) was found, and column Undecided the number of schemes for which security could not be decided. Similarly, the columns grouped under CPA and CCA report the results when analyzing CPA and CCA security under all assumptions of the form Γ_2. In this case, column Proof indicates the number of schemes proved secure under *some* assumption of the form Γ_2, column Attack the number of schemes for which an attack was found for *all* assumption of the form Γ_2, and column Undecided the number of schemes for which security could not be decided. The attack and proof search algorithms are extremely efficient,

Size	Total	OW CPA (% of Total)			CPA (% of Total)			CCA (% of CPA Proof + CPA Undecided)			
		Proof	Attack	Undecided	Proof	Attack	Undecided	Proof	Attack	NR	Undecided
4	2	1 (50.00%)	1 (50.00%)	0 (0.00%)	2 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	2 (100.00%)	0 (0.00%)	0 (0.00%)
5	44	8 (18.18%)	36 (81.82%)	0 (0.00%)	12 (27.27%)	32 (72.73%)	0 (0.00%)	0 (0.00%)	13 (100.00%)	0 (0.00%)	0 (0.00%)
6	335	65 (19.40%)	270 (80.60%)	0 (0.00%)	93 (27.76%)	241 (71.94%)	1 (0.30%)	1 (0.98%)	96 (94.12%)	5 (4.90%)	0 (0.00%)
7	3263	510 (15.63%)	2735 (83.82%)	18 (0.55%)	750 (22.98%)	2475 (75.85%)	38 (1.16%)	45 (5.05%)	739 (82.94%)	45 (5.05%)	62 (6.96%)
8	32671	4430 (13.56%)	27894 (85.38%)	347 (1.06%)	6718 (20.56%)	25336 (77.55%)	617 (1.89%)	536 (6.26%)	6531 (76.25%)	306 (3.57%)	1192 (13.92%)
9	350111	43556 (12.44%)	301679 (86.17%)	4876 (1.39%)	66775 (19.07%)	274813 (78.49%)	8523 (2.43%)	7279 (8.16%)	62356 (69.93%)	3035 (3.40%)	16496 (18.50%)
10	644563	67863 (10.53%)	569314 (88.33%)	7386 (1.15%)	133476 (20.71%)	491189 (76.20%)	19898 (3.09%)	20140 (11.29%)	112993 (63.36%)	12794 (7.17%)	32397 (18.17%)
Total	1030989	116433 (11.29%)	901929 (87.48%)	12627 (1.22%)	207826 (20.16%)	794086 (77.02%)	29077 (2.82%)	28001 (10.11%)	182730 (65.95%)	16185 (5.84%)	50147 (18.10%)

Table 1: Evaluation of the tool on generated encryption schemes

e.g. proof search for schemes of size 7 takes on average 0.1ms for CPA and 0.5ms for CCA on a modern workstation

Observe that the figures in the first two groups suggest the separation between one-wayness and partial-domain one-wayness: the stronger the assumption, the more schemes can be proven secure and the less attacks can be found.

Finally, column NR in the CCA group counts the number of schemes that are CPA secure but non-redundant, meaning that all ciphertexts are valid. Non-redundant schemes can be CCA secure [6, 23], but their proofs require random oracle *programmability*, which is out of the scope of our logics.

Validation.

We also evaluated our automated proof search and attack finding algorithms on a number of schemes from the literature, including the over one hundred variants of OAEP and SAEP surveyed by Komano and Ohta [23]. In all cases, our results are consistent with published results, and in most we are able to prove security under exactly the same assumptions and obtain the same security bounds. As evidence of the effectiveness of our methodology, we observe that our analyses decide the CPA security of all 72 variants of OAEP in the taxonomy of [23]. The results for CCA security are more nuanced: for about 20% of schemes, we fail to find proofs or attacks when they exist.

For CPA security our methods seem to achieve empirical completeness, which suggests that completeness may be provable for some class of schemes or for some mild extension. A closer examination of the schemes on which our CCA analysis is unable to decide security reveals that this is either due to the fact that our approximation of inequality in rules [Eqs] and [PEqs] is too coarse, or because the schemes are non-redundant. Non-redundancy complicates enormously the task of simulating the decryption oracle in CCA reductions, because a meaningful response must be returned in all cases. Proving CCA security of non-redundant schemes, requires *programming* random oracles in order to maintain consistency during simulation, something that we have intentionally avoided in our proof systems. Extending our proof systems to embody some form of *programmability* would reduce the number of schemes for which security cannot be decided, and would be a step towards closing the empirical completeness gap.

7. RELATED WORK

Our work lies at the intersection between symbolic and computational cryptography, and draws on verification techniques from both areas. We refer to [12, 15] for a recent account of symbolic verification and focus on verification tools and methods for the computational model, and cryptographic soundness.

Since its inception [2], cryptographic soundness was extended to many settings [15]. Despite this success, cryptographic soundness for constructions based on exclusive or and one-way trapdoor permutations has remained elusive. Negative results [30] show that soundness may indeed be very difficult to achieve for the setting of this paper.

CryptoVerif [11] and EasyCrypt [4] support the construction and verification of cryptographic proofs in the computational model. Both CryptoVerif and EasyCrypt provide a high level of automation, and CryptoVerif supports fully automated verification of many cryptographic protocols. In addition, EasyCrypt has been applied to verify security of several padding-based schemes. However, fully automated proofs of padding-based schemes are out of reach of these tools. [16] reports on a Hoare-like logic and an automated tool for proving CPA security of padding-based schemes. The tool can verify the security of several schemes, such as BR [9] and REACT [25], but fails to verify most other examples. In our view, the limitations of the approach are inherited from using a Hoare logic, which favors local reasoning. In contrast, we use a more global approach in which one reasons about the probability of an event in an experiment.

In addition, a number of formalisms have been developed to reason about security of cryptographic primitives and constructions in the computational model [5, 21]. However, these systems reason about constructions described in mathematical vernacular, and are thus not readily amenable to automation. Similar formalisms exist for cryptographic protocols [27], but these are not automated either.

Finally, the batch mode of ZooCrypt is similar in spirit to the AVGI toolkit [26]. 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.

8. CONCLUSION

We have defined, implemented and evaluated proof systems to reason about security of public-key encryption schemes built from trapdoor permutations and hash functions. Our work sets a new methodology for analyzing cryptographic constructions in the computational model. Preliminary investigations suggest that our methodology extends well to other settings, and we intend to apply it for building a comprehensive database that includes classical cryptographic constructions such as digital signature schemes, modes of operation, and message authentication codes, and to build on top of ZooCrypt an online interactive tutor that can generate exercises, help users finding an attack or a security proof, and check that a solution input by users is correct. Based on our experience with the interactive mode of ZooCrypt, we are convinced that, if well-executed, such a tutor could provide a very attractive vector for introducing students to cryptographic proofs!

We have also started to connect ZooCrypt with EasyCrypt, and implemented a prototype mechanism that turns successful derivations in our logics into EasyCrypt proofs. While generation of EasyCrypt proofs provides independent evidence of the soundness of the logics, our primary interest for this connection is to increase automation in EasyCrypt proofs. We expect that combining reflection[3] and proof generation will deliver significant benefits for EasyCrypt.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *28th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, POPL 2001*, pages 104–115, New York, 2001. ACM. doi: 10.1145/360204.360213.

[2] M. Abadi and P. Rogaway. Reconciling two views of cryptography (The computational soundness of formal encryption). *J. Cryptology*, 15(2):103–127, 2002.

[3] G. Bana and H. Comon-Lundh. Towards unconditional soundness: Computationally complete symbolic attacker. In *1st Conference on Principles of Security and Trust – POST 2012*, volume 7215 of *Lecture Notes in Computer Science*, pages 189–208, Heidelberg, 2012. Springer.

[4] 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.

[5] 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.

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

[7] G. Barthe, D. Pointcheval, and S. Zanella-Béguelin. Verified security of redundancy-free encryption from Rabin and RSA. In *19th ACM Conference on Computer and Communications Security, CCS 2012*, pages 724–735, New York, 2012. ACM. doi: 10.1145/2382196.2382272.

[8] M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. *Inf. Comput.*, 207(4):496–520, Apr. 2009. ISSN 0890-5401. doi: 10.1016/j.ic.2008.12.005.

[9] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security, CCS 1993*, pages 62–73, New York, 1993. ACM.

[10] 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.

[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] D. Boneh. Simplified OAEP for the RSA and Rabin functions. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 275–291, Heidelberg, 2001. Springer.

[14] J.-S. Coron. Security proof for partial-domain hash signature schemes. In *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 271–287, Heidelberg, 2002. 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] D. Dolev and A. C.-C. Yao. On the security of public key protocols. In *22nd Annual IEEE Symposium on Foundations of Computer Science, FOCS 1981*, pages 350–357. IEEE Computer Society, 1981.

[3] Reflection is a technique widely used in proof assistants. It would allow us to reify EasyCrypt proof goals into judgments of our logics that can be proved automatically by our tool; the initial proof goals could then be discharged replaying the script output by the proof generation mechanism.

[18] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2001.

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

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

[21] R. Impagliazzo and B. M. Kapron. Logics for reasoning about cryptographic constructions. *J. Comput. Syst. Sci.*, 72(2):286–320, 2006.

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

[25] T. Okamoto and D. Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In *4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001*, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118, Heidelberg, 2001. Springer.

[26] A. Perrig and D. Song. Looking for diamonds in the desert – extending automatic protocol generation to three-party authentication and key agreement protocols. In *13th IEEE Workshop on Computer Security Foundations, CSFW 2000*, pages 64–76, Los Alamitos, 2000. IEEE Computer Society. doi: 10.1109/CSFW.2000.856926.

[27] A. Roy, A. Datta, A. Derek, and J. C. Mitchell. Inductive trace properties for computational security. *Journal of Computer Security*, 18(6):1035–1073, 2010.

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

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

[30] D. Unruh. The impossibility of computationally sound XOR. Cryptology ePrint Archive, Report 2010/389, 2010.

[31] Y. Zheng and J. Seberry. Practical approaches to attaining security against adaptively chosen ciphertext attacks. In *Advances in Cryptology – CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 292–304, Heidelberg, 1993. Springer. doi: 10.1007/3-540-48071-4_20.