Beyond Provable Security
Verifiable IND-CCA Security of OAEP

Gilles Barthe[1], Benjamin Grégoire[2],
Yassine Lakhnech[3], and Santiago Zanella Béguelin[1]

[1] IMDEA Software
[2] INRIA Sophia Antipolis-Méditerranée
[3] Université Grenoble 1, CNRS, Verimag

Abstract. OAEP is a widely used public-key encryption scheme based on trapdoor permutations. Its security proof has been scrutinized and amended repeatedly. Fifteen years after the introduction of OAEP, we present a machine-checked proof of its security against adaptive chosen-ciphertext attacks under the assumption that the underlying permutation is partial-domain one-way. The proof can be independently verified by running a small and trustworthy proof checker and fixes minor glitches that have subsisted in published proofs. We provide an overview of the proof, highlight the differences with earlier works, and explain in some detail a crucial step in the reduction: the elimination of indirect queries made by the adversary to random oracles via the decryption oracle. We also provide—within the limits of a conference paper—a broader perspective on independently verifiable security proofs.

1 Introduction

Optimal Asymmetric Encryption Padding (OAEP) [9] is a prominent public-key encryption scheme based on trapdoor permutations, most commonly used in combination with the RSA [29] and Rabin [28] functions. OAEP is widely deployed; many variants of OAEP are recommended by several standards, including IEEE P1363, PKCS, ISO 18033-2, ANSI X9, CRYPTREC and SET. Yet, the history of OAEP security is fraught with difficulties. The original 1994 paper of Bellare and Rogaway [9] proves that, under the hypothesis that the underlying trapdoor permutation family is one-way, OAEP is semantically secure under chosen-ciphertext attacks. Shoup [30] subsequently discovered in 2000 that this proof only established the security of OAEP against non-adaptive chosen-ciphertext attacks, and not (as was believed at that time) against the stronger version of IND-CCA that allows the adversary to adaptively obtain the decryption of ciphertexts of its choice. Shoup suggested a modified scheme, OAEP+, secure against adaptive attacks under the one-wayness of the underlying permutation, and gave a proof of the adaptive IND-CCA security of the original scheme when it is used in combination with RSA with public exponent $e = 3$. Simultaneously, Fujisaki, Okamoto, Pointcheval and Stern [15] proved that OAEP in its original formulation is indeed secure against adaptive attacks, but under the

assumption that the underlying permutation family is partial-domain one-way. Since for the particular case of RSA this latter assumption is no stronger than (full-domain) one-wayness, this finally established the adaptive IND-CCA security of RSA-OAEP. In 2004, Pointcheval [27] gave a different proof of the same result; this new proof fills several gaps in the reduction of [15], which results in a weaker bound than originally stated. Nonetheless, the inaccurate bound of [15] remains the reference bound used in practical analyses of OAEP, see e.g. [13]. Finally, Bellare, Hofheinz and Kiltz [8], recently pointed out some ambiguities in the definition of IND-CCA, leading to four possible formulations (all of them used in the literature), and question which definition is used in the statements and proofs of OAEP.

This paper reports on a machine-checked proof that OAEP is IND-CCA secure against adaptive attacks. For the sake of definitional clarity, we identify IND-CCA with the strongest definition in the taxonomy of [8], IND-CCA-SE. Let us first give a formal definition of OAEP:

Definition 1 (OAEP encryption scheme). *Let $(\mathcal{K}_f, f, f^{-1})$ be a family of trapdoor permutations on $\{0,1\}^k$, and*

$$G : \{0,1\}^{k_0} \to \{0,1\}^{k-k_0} \qquad H : \{0,1\}^{k-k_0} \to \{0,1\}^{k_0}$$

two hash functions, with $k = n + k_0 + k_1$. The Optimal Asymmetric Encryption Padding (OAEP) scheme is composed of the following triple of algorithms:

$$\begin{array}{lcl}
\mathcal{K}(\eta) & \stackrel{def}{=} & (pk, sk) \leftarrow \mathcal{K}_f(\eta);\ \textsf{return}\ (pk, sk) \\
\mathcal{E}(pk, m) & \stackrel{def}{=} & r \xleftarrow{\$} \{0,1\}^{k_0};\ s \leftarrow G(r) \oplus (m \,\|\, 0^{k_1});\ t \leftarrow H(s) \oplus r; \\
 & & \textsf{return}\ f(pk, s \,\|\, t) \\
\mathcal{D}(sk, c) & \stackrel{def}{=} & (s \,\|\, t) \leftarrow f^{-1}(sk, c);\ r \leftarrow t \oplus H(s);\ m \leftarrow s \oplus G(r); \\
 & & \textsf{if}\ [m]_{k_1} = 0^{k_1}\ \textsf{then}\ \textsf{return}\ [m]^n\ \textsf{else}\ \textsf{return}\ \perp
\end{array}$$

where $[x]_n$ (resp. $[x]^n$) denotes the n least (resp. most) significant bits of x.

Our main result is:

Theorem 1 (IND-CCA security of OAEP). *Let $\mathcal{A}$ be an adversary against the adaptive IND-CCA security of OAEP that makes at most q_G and q_H queries to the hash oracles G and H, respectively, and at most $q_{\mathcal{D}}$ queries to the decryption oracle $\mathcal{D}$. Suppose this adversary achieves an IND-CCA advantage ϵ within time t. Then, there exists an inverter $\mathcal{I}$ that finds a partial preimage (the most significant $k - k_0$ bits) of an element uniformly drawn from the domain of the underlying permutation f with probability ϵ' within time t', where*

$$\begin{aligned}
\epsilon' &\geq \frac{1}{q_H}\left(\frac{\epsilon}{2} - \frac{3q_{\mathcal{D}}q_G + q_{\mathcal{D}}^2 + 4q_{\mathcal{D}} + q_G}{2^{k_0}} - \frac{2q_{\mathcal{D}}}{2^{k_1}}\right) \\
t' &\leq t + q_{\mathcal{D}}\ q_G\ q_H\ (T_f + O(1))
\end{aligned}$$

and where T_f is an upper bound on the time needed to compute the image of a bitstring under f. Moreover, if the underlying permutation family is partial-domain one-way and adversary $\mathcal{A}$ runs in probabilistic polynomial-time (on some

security parameter η), then the advantage of $\mathcal{A}$ is negligible, provided parameters k_0, k_1 are at least linear on η.

The formal statement is given in Fig. 1. The proof is built using CertiCrypt [6], a general framework for building game-based cryptographic proofs in the Coq proof assistant [32], and yields an independently verifiable certificate. Said otherwise, an external verifier can examine the statement to convince herself that it faithfully captures the definitions of OAEP and IND-CCA security and can delegate the verification of the proof to an automated checker. Our exact security bound unveils minor glitches in the proof of [27], and marginally improves on its exact security bound by performing an aggressive analysis of oracle queries earlier in the sequence of games. Beyond its individual merits, the proof is highly emblematic and provides tangible evidence of the onset of tools to build and verify cryptographic proofs.

2 A Primer on Formal Proofs

Proof assistants are programs designed to support interactive construction and automatic verification of mathematical statements (understood in a broad sense). Initially developed by logicians to experiment with the expressive power of their foundational formalisms, proof assistants are now emerging as a mature technology that can be used effectively for verifying intricate mathematical proofs, such as the Four Color theorem [16] or the Kepler conjecture [18,19], or complex software systems, such as operating systems [21], virtual machines [22] and optimizing compilers [24]. In the realm of cryptography, proof assistants have been used to formally verify secrecy and authenticity properties of protocols [26].

Proof assistants rely on expressive specification languages that allow formalizing arbitrary mathematical notions, and that provide a formal representation of proofs as proof objects. Their architecture is organized into two layers: a kernel, and a proof engine.

- The kernel is the cornerstone for correctness. Its central component is a checker for verifying the consistency of formal theories, including definitions and proofs. In particular, the checker guarantees that definitions and proofs are well-typed, that there are no missing cases or undefined notions in definitions, and that all proofs are built from valid elementary logical steps and make a correct use of assumptions.
- In contrast, the proof engine helps proof construction. The proof engine embraces a variety of tools. The primary tools are a set of pre-defined tactics, and a language for writing user-defined tactics. Tactics allow to reduce a proof goal to simpler ones. When invoked on a proof goal A, a tactic will compute a new set of goals $A_1 \dots A_n$, and a proof that $A_1 \wedge \dots \wedge A_n \implies A$. At the end of each demonstration, the proof engine outputs a proof object.

Proof objects are independently checked by the kernel. Therefore, the proof engine need not be trusted, and the validity of a formal proof—beyond the

accuracy of the statement itself—only depends on the correctness of the kernel. Pleasingly, kernels are extremely reliable programs with restricted functionalities and solid logical foundations.

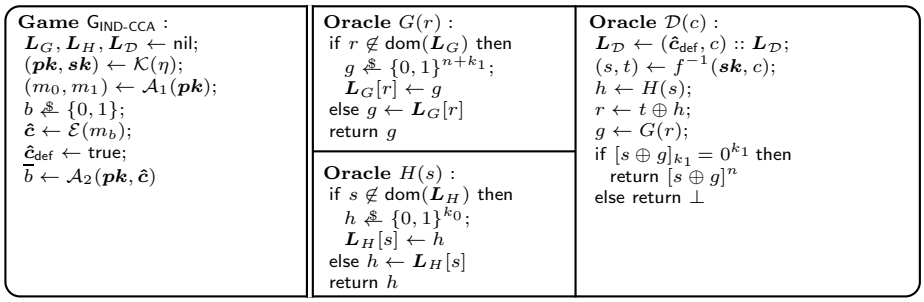
As with any other mathematical activity, formal proofs strive for elegance and conciseness. In our experience, they also provide a natural setting for improving proofs—in the case of cryptography, improvement can be measured by comparing exact security bounds. Yet, what matters most about a formal proof is that it provides a nearly absolute degree of assurance, without requiring expensive human verification.

3 The Statement

The formal statement of the exact IND-CCA security of OAEP is displayed in Figure 1; it comprises the definition of the IND-CCA game and the simulation that reduces security to the partial-domain one-wayness of the trapdoor permutation. The security result is expressed as a lower bound on the success probability of the reduction in terms of the success probability of an IND-CCA adversary. Both probabilities are captured formally by expressions of the form $\Pr[\mathsf{G} : E]$, where G is a game and E an event. The definition of probabilities is taken from Audebaud and Paulin's library [2], whereas the definition of games and events is taken from the CertiCrypt framework [6]. In essence, games are probabilistic programs with calls to adversaries; formally, a game is given by a main command and an environment that provides the code of algorithms and oracles—in contrast, adversaries are formalized as procedures with unknown code. Games have a probabilistic semantics: given an interpretation of adversaries as probabilistic programs, a game G is interpreted as a function $[\![\mathsf{G}]\!]$ from initial states to distributions of final states. The semantics of games is taken from [6]. Events are merely predicates over final states, and $\Pr[\mathsf{G} : E]$ is simply the probability of E in the distribution induced by $[\![\mathsf{G}]\!]$ starting from an empty initial state.

The IND-CCA game involves an adversary $\mathcal{A}$ (modeled by procedures $\mathcal{A}_1$ and $\mathcal{A}_2$), defines algorithms $\mathcal{K}$ for key generation and $\mathcal{E}$ for encryption, and gives the adversary access to a decryption oracle $\mathcal{D}$ and to random oracles G and H. We follow the convention of typesetting global variables in boldface. The first line of the main command initializes oracle memories; the lists $\boldsymbol{L}_G$ and $\boldsymbol{L}_H$ are used to simulate the random oracles G and H, whereas the list $\boldsymbol{L}_{\mathcal{D}}$ is a ghost variable used to track decryption queries and exclude invalid adversaries that query the decryption oracle with the challenge ciphertext during the second phase of the game. The remainder of the game is standard; note that we set a flag $\hat{\mathsf{c}}_{\mathsf{def}}$ just before giving the challenge ciphertext to the adversary in order to distinguish decryption queries made in the second phase of the game from those made in the first phase. The code of the decryption oracle and the encryption and key generation algorithms is a direct transcription of the informal definitions and is omitted.

The code of the game is complemented by a variable policy that declares which variables are accessible to adversaries: $\mathcal{A}$ cannot read nor modify the



$$\mathsf{WF}(\mathcal{A}) \wedge \Pr[\mathsf{G}_{\text{IND-CCA}} : |\boldsymbol{L}_G| \leq q_G + q_{\mathcal{D}} + 1 \wedge |\boldsymbol{L}_{\mathcal{D}}| \leq q_{\mathcal{D}} \wedge (\mathsf{true}, \hat{\boldsymbol{c}}) \notin \boldsymbol{L}_{\mathcal{D}}] = 1 \implies$$
$$\Pr\big[\mathsf{G}_{\text{IND-CCA}} : \overline{b} = b\big] - \frac{1}{2} \leq \Pr[\mathsf{G}_{\text{set-PD-OW}} : s \in S] + \frac{3q_{\mathcal{D}}q_G + q_{\mathcal{D}}^2 + 4q_{\mathcal{D}} + q_G}{2^{k_0}} + \frac{2q_{\mathcal{D}}}{2^{k_1}}$$

Fig. 1. Formal statement of IND-CCA security of OAEP.

values of $\boldsymbol{sk}$, $\boldsymbol{L}_{\mathcal{D}}$, $\boldsymbol{L}_G$, $\boldsymbol{L}_H$, $\hat{\boldsymbol{c}}_{\mathsf{def}}$, and $\hat{\boldsymbol{c}}$, and cannot modify the value of $\boldsymbol{pk}$; on the other hand, the procedures representing the two phases of the adversary can communicate through shared variables. An adversary $\mathcal{A}$ respecting the variable policy is said to be well-formed; this is noted as $\mathsf{WF}(\mathcal{A})$.

The security statement itself takes the form of an implication, whose premise fixes the class of adversaries considered. The statement considers well-formed adversaries that make at most $q_{\mathcal{D}}$ and q_G queries to the decryption and G oracles respectively[4], and that do not query the decryption oracle with the challenge ciphertext in the second phase of the game. Given an IND-CCA adversary $\mathcal{A}$, we show how to construct an inverter $\mathcal{I}$ that uses $\mathcal{A}$ as a subroutine to partially invert the underlying trapdoor permutation. The success probability of the inverter is given by $\Pr[\mathsf{G}_{\text{set-PD-OW}} : s \in S]$, and is lower bounded by:

$$\frac{1}{2}\,\mathsf{Adv}^{\text{IND-CCA}}_{\mathcal{A}} - \frac{3q_{\mathcal{D}}q_G + q_{\mathcal{D}}^2 + 4q_{\mathcal{D}} + q_G}{2^{k_0}} - \frac{2q_{\mathcal{D}}}{2^{k_1}}$$

[4] The formal statement slightly relaxes this condition; it requires the length of $\boldsymbol{L}_G$ be at most $q_G + q_{\mathcal{D}} + 1$ (the 1 accounting for the call to G needed to compute the challenge ciphertext), so that the adversary could trade calls to $\mathcal{D}$ for calls to G.

where the IND-CCA advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A}}$ of $\mathcal{A}$ is defined as usual as

$$2\,\Pr\left[\mathsf{G}_{\mathsf{IND\text{-}CCA}} : \overline{b} = b\right] - 1$$

One additional remark is needed to relate the formal statement to the statement of Theorem 1. Strictly, the formal statement reduces the security of OAEP not to the partial-domain one-wayness of the permutation, but to its *set* partial-domain one-wayness. Both notions are closely related (cf. [15]). We could have formally proven the reduction to the former problem using basically the same argument, but making the inverter return a value uniformly chosen from the domain of $\boldsymbol{L}_H$ instead; this accounts for the multiplicative factor q_H^{-1} in Theorem 1. The reduction from partial-domain one-wayness to set partial-domain one-wayness is inessential to the presentation and can be proven independently and generically for any inverter $\mathcal{I}$.

4 The Proof

One claimed virtue of verifiable security is that there is no need to understand its proof (only its statement) to trust the correctness of a result. Obviously, it remains of interest to understand the thrust of the proof, and if one intends to reproduce the proof—perhaps in a slightly different setting, or for a different scheme, or with a different framework—its ultimate details. This section provides an overview of the techniques used to conduct the proof and delves into the details of one significant proof step, namely eliminating fresh oracle calls to G in the decryption oracle. The code of the proof and all the infrastructure needed to independently verify may be obtained from the authors upon simple request.

Tools The proof makes an extensive use of the techniques provided by the CertiCrypt framework, as reported in [6], and the additional techniques described in [7]. The unifying formalism used by CertiCrypt to justify transitions between games is a Relational Hoare Logic, whose judgments are of the form $\vdash \mathsf{G}_1 \sim \mathsf{G}_2 : \Psi \Rightarrow \Phi$, relating two games G_1 and G_2 w.r.t. two relations Ψ and Φ on states. Such a judgment means that for any initial memories m_1 and m_2 satisfying the precondition $m_1\ \Psi\ m_2$, the distributions $[\![\mathsf{G}_1]\!]\ m_1$ and $[\![\mathsf{G}_2]\!]\ m_2$ are related by the lifting of Φ to distributions[5]. Relational Hoare Logic subsumes observational equivalence $\vdash \mathsf{G}_1 \sim^X_Y \mathsf{G}_2$, which is obtained by setting Ψ and Φ to $=_X$ and $=_Y$, where X (resp. Y) is a set of variables and $=_X$ (resp. $=_Y$) relates memories that coincide on all variables in X (resp. Y).

Both Relational Hoare Logic and observational equivalence statements allow to express that two games perfectly simulate each other. Proofs can be conducted

[5] In the general case, we adopt the definition of lifting from probabilistic process algebra, which is formulated in terms of a max-flow min-cut problem and involves an existential quantification over distributions. For partial equivalence relations, the definition coincides with the usual approach that requires the probability of equivalence classes be the same.

using proof rules à la Hoare Logic—i.e., there is a rule for each construction of the programming language and structural rules—or certified tactics that automate program transformations such as dead code elimination, constant folding and propagation, or procedure call inlining.

We use the logic of *swapping statements* of [7] to prove independence of values from adversary's view. We say that a value is independent from adversary's view at some point in a game if it can be resampled without modifying the meaning of the game. The logic for swapping statements deals with Relational Hoare judgments of the form $\vdash S; \mathsf{G}_1 \sim_Y^X \mathsf{G}_2; S$, where the games $S; \mathsf{G}_1$ and $\mathsf{G}_2; S$ are respectively obtained from games G_1 and G_2 by prefixing and postfixing some code fragment S. Typically, S just resamples part of the state of the game; moreover, the code of oracles in G_1 and G_2 may also differ in the random samplings they perform. In general, the logic of swapping statements can be used to justify eager and lazy sampling transformations—overcoming limitations in [6]. An example of its application is given below.

In addition to Relational Hoare Logic, CertiCrypt formalizes the Fundamental Lemma of Game-Playing [10,20,31], which is used to justify "lossy" steps where two consecutive games in a proof structured as a sequence of games only diverge when a failure event occurs. The Failure Event Lemma of [7] complements the Fundamental Lemma of Game-Playing and allows to bound the probability of a failure event triggered inside an oracle by a function of the number of calls made to the oracle. There exist several specialized versions of this lemma; the simplest instance focuses on games in which the failure event F is triggered by an oracle $\mathcal{O}$ with a probability bounded by a constant ϵ, independent from the argument with which it is called and of any previous calls. In this case, the Failure Event Lemma bounds the probability of event F by $q_{\mathcal{O}}\ \epsilon$, where $q_{\mathcal{O}}$ is a bound on the number of calls to $\mathcal{O}$. While this instance of the Failure Event Lemma suffices to justify most lossy transformations in the proof of OAEP, we also needed to resort to the full generality of the lemma on two occasions; one of them is outlined below.

Proof outline Figure 2 outlines the structure of the proof; the first step from $\mathsf{G}_{\mathsf{IND\text{-}CCA}}$ to G_1 and the final step from G_5 to $\mathsf{G}_{\mathsf{set\text{-}PD\text{-}OW}}$ are not displayed. The reduction successively eliminates all situations in which the plaintext extractor used by the inverter to simulate decryption may fail.

Starting from game $\mathsf{G}_{\mathsf{IND\text{-}CCA}}$, we use the logic of swapping statements to fix the hash $\hat{\boldsymbol{g}}$ that G gives in response to the random seed in the challenge ciphertext; the computation of the challenge ciphertext unfolds to:

$$\hat{\boldsymbol{r}} \xleftarrow{\$} \{0,1\}^{k_0};\ \hat{\boldsymbol{s}} \leftarrow \hat{\boldsymbol{g}} \oplus (m_b \,\|\, 0^{k_1});\ \hat{\boldsymbol{h}} \leftarrow H(\hat{\boldsymbol{s}});\ \hat{\boldsymbol{t}} \leftarrow \hat{\boldsymbol{h}} \oplus \hat{\boldsymbol{r}};\ \hat{\boldsymbol{c}} \leftarrow f(\boldsymbol{pk}, \hat{\boldsymbol{s}} \,\|\, \hat{\boldsymbol{t}})$$

where $\hat{\boldsymbol{g}}$ is sampled from $\{0,1\}^{k-k_0}$ before the first call to $\mathcal{A}$. We then make G respond to an adversary query $\hat{\boldsymbol{r}}$ with a freshly sampled value instead of $\hat{\boldsymbol{g}}$; this only makes a difference if flag **bad** is set in game G_1. Since at this point $\hat{\boldsymbol{g}}$ is uniformly distributed and independent from the adversary's view, the value $\hat{\boldsymbol{s}}$ computed as $\hat{\boldsymbol{g}} \oplus (m_b \,\|\, 0^{k_1})$ is as well uniformly distributed and independent

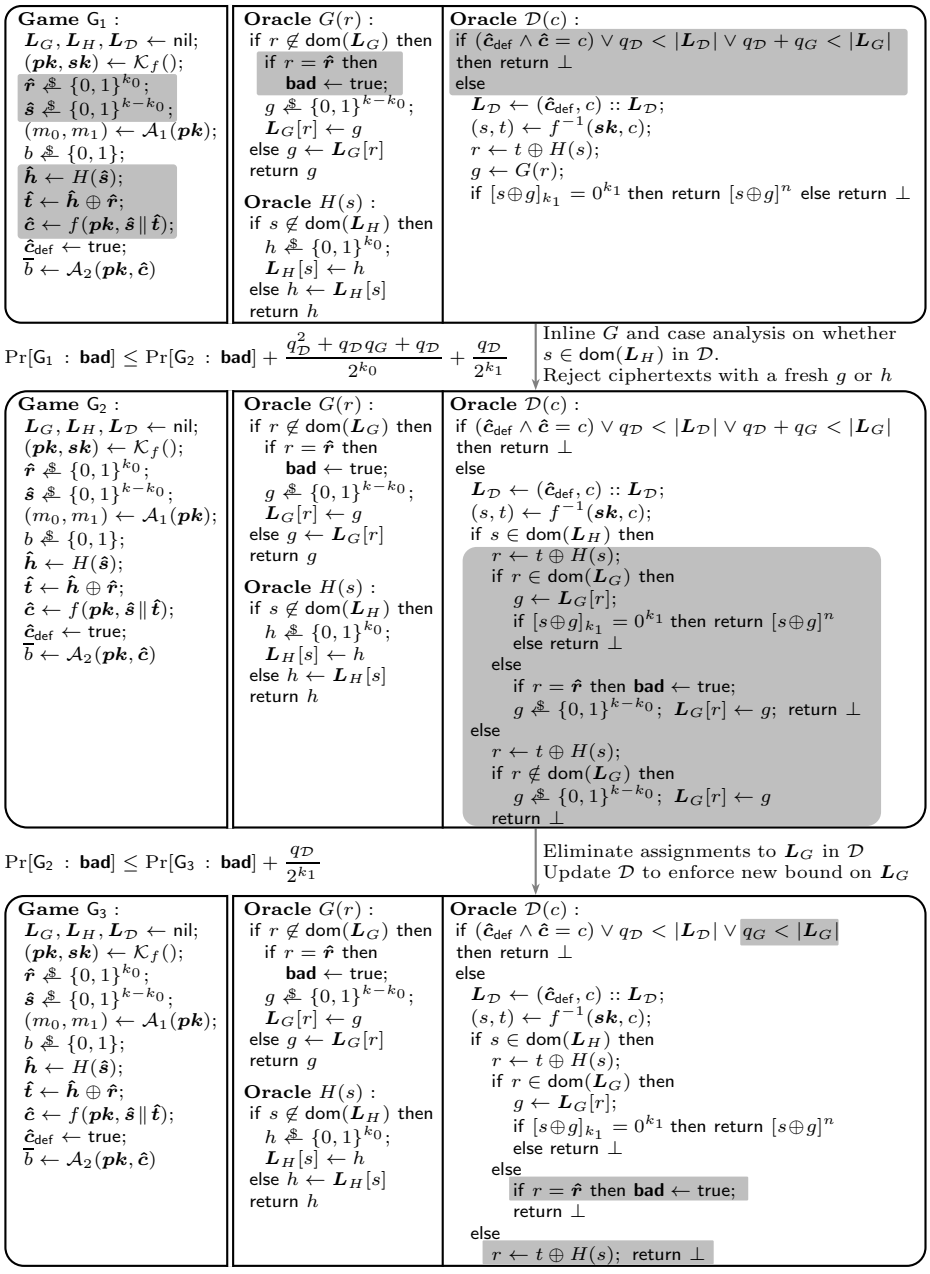


Fig. 2. Outline of the reduction showing the *lossy* transitions. Fragments of code that change between games are highlighted on a gray background.

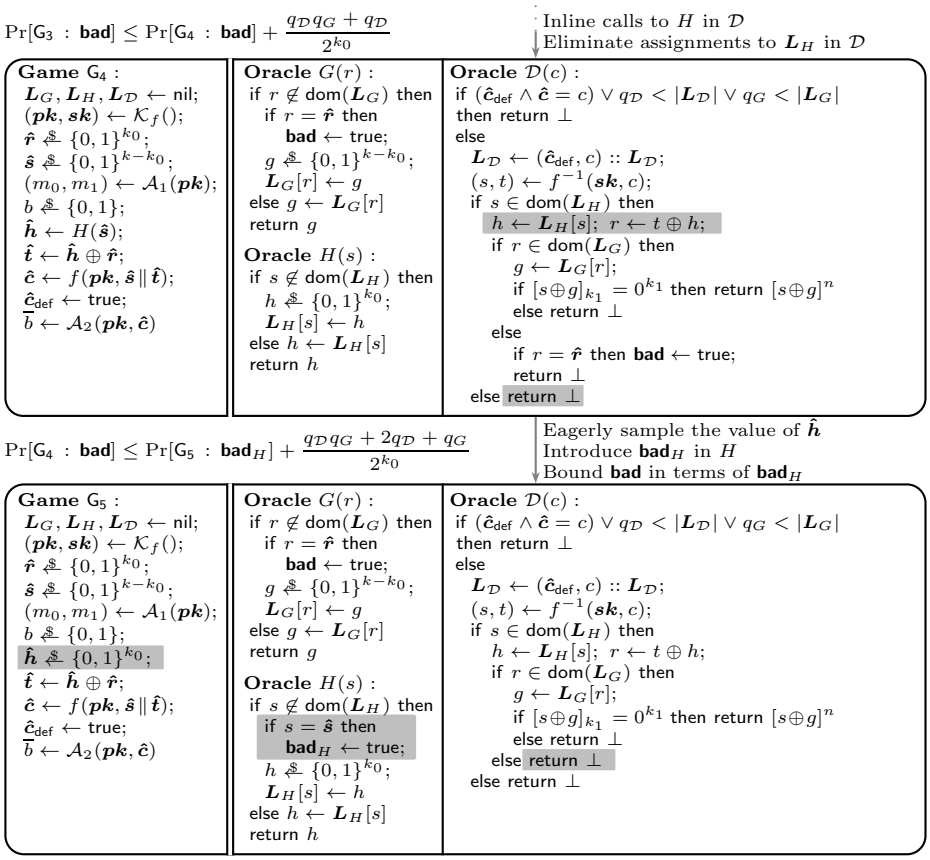


Fig. 2. Outline of the reduction showing the *lossy* transitions. Fragments of code that change between games are highlighted on a gray background.

from the adversary's view. This removes the dependence of the adversary output on the hidden bit b, and thus the probability of a correct guess is exactly $1/2$. Using the Fundamental Lemma we obtain the bound:

$$\Pr\big[\mathsf{G}_{\mathsf{IND\text{-}CCA}} : \overline{b} = b\big] - \Pr\big[\mathsf{G}_1 : \overline{b} = b\big] = \Pr\big[\mathsf{G}_{\mathsf{IND\text{-}CCA}} : \overline{b} = b\big] - \frac{1}{2} \tag{1}$$

$$\leq \Pr[\mathsf{G}_1 : \mathbf{bad}] \tag{2}$$

The transition from G_1 to G_2 modifies the decryption oracle successively by inlining the call to G, and by applying the Fundamental and Failure Event lemmas to reject the ciphertext when there is a small chance it matches the padding. Overall, we prove:

$$\Pr[\mathsf{G}_1 : \mathbf{bad}] \leq \Pr[\mathsf{G}_2 : \mathbf{bad}] + \frac{q_{\mathcal{D}}^2 + q_{\mathcal{D}} q_G + q_{\mathcal{D}}}{2^{k_0}} + \frac{q_{\mathcal{D}}}{2^{k_1}} \tag{3}$$

Next, we eliminate fresh calls to G in the decryption oracle. These calls correspond to the two assignments $\boldsymbol{L}_G[r] \leftarrow g$, since calls to G have been inlined previously. We perform an aggressive elimination and remove both calls. As a result, in game $\mathsf{G_3}$ the length of list $\boldsymbol{L}_G$ (i.e. the number of calls to G) is bounded by q_G rather than $q_{\mathcal{D}} + q_G$. This is the key to improve on the security bound of Pointcheval [27], who only removes the second call. The proof relies on the logic of swapping statements to show that values of discarded calls are "uniformly distributed and independent from the adversary's view". Details appear in next paragraph. Overall, we prove:

$$\Pr[\mathsf{G}_2 : \mathbf{bad}] \leq \Pr[\mathsf{G}_3 : \mathbf{bad}] + \frac{q_{\mathcal{D}}}{2^{k_1}} \tag{4}$$

Likewise, we eliminate calls to H in $\mathcal{D}$, yielding a new game $\mathsf{G_4}$ in which the decryption oracle does not add any new values to the memories of G and H. Using the Fundamental and Failure Event lemmas, we obtain:

$$\Pr[\mathsf{G}_3 : \mathbf{bad}] \leq \Pr[\mathsf{G}_4 : \mathbf{bad}] + \frac{q_{\mathcal{D}} q_G + q_{\mathcal{D}}}{2^{k_0}} \tag{5}$$

We next fix the value $\hat{\boldsymbol{h}}$ that oracle H gives in response to $\hat{\boldsymbol{s}}$, and then make H return a freshly sampled value instead of $\hat{\boldsymbol{h}}$. This allows us to bound the probability of $\mathbf{bad}$ in terms of the probability of a newly introduced event $\mathbf{bad}_H$, that indicates whether the adversary queried the value of $H(\hat{\boldsymbol{s}})$. The proof uses the hypothesis that $\mathcal{A}_2$ cannot query the decryption oracle with the challenge ciphertext, and yields:

$$\Pr[\mathsf{G}_4 : \mathbf{bad}] \leq \Pr[\mathsf{G}_5 : \mathbf{bad}_H] + \frac{q_{\mathcal{D}} q_G + 2q_{\mathcal{D}} + q_G}{2^{k_0}} \tag{6}$$

Finally, we prove that the probability of $\mathbf{bad}_H$ in $\mathsf{G_5}$ is upper bounded by the probability that the inverter $\mathcal{I}$ in Figure 1 succeeds in partially inverting the permutation f. The proof uses the (standard, non-relational) invariant on $\mathsf{G_5}$:

$$\mathbf{bad}_H \implies \hat{\boldsymbol{s}} \in \mathsf{dom}(\boldsymbol{L}_H)$$

The inverter $\mathcal{I}$ that we build (shown in Fig. 1) gives its own challenge y as the challenge ciphertext to the $\mathsf{IND\text{-}CCA}$ adversary $\mathcal{A}$. The task of the inverter is to return a list of values containing the partial preimage of its challenge which, stated in terms of the variables of game $\mathsf{G_5}$, is $\hat{\boldsymbol{s}}$. Thus:

$$\Pr[\mathsf{G}_5 : \mathbf{bad}_H] \leq \Pr[\mathsf{G}_5 : \hat{\boldsymbol{s}} \in \mathsf{dom}(\boldsymbol{L}_H)] = \Pr[\mathsf{G}_{\mathsf{set\text{-}PD\text{-}OW}} : s \in S] \tag{7}$$

Where the last equality follows from an algebraic equivalence that we prove as a lemma:

$$\hat{\boldsymbol{h}} \xleftarrow{\$} \{0,1\}^{k_0};\ \hat{\boldsymbol{t}} \leftarrow \hat{\boldsymbol{h}} \oplus \hat{\boldsymbol{r}} \sim^{\{\hat{\boldsymbol{r}}\}}_{\{\hat{\boldsymbol{h}},\hat{\boldsymbol{t}},\hat{\boldsymbol{r}}\}} \hat{\boldsymbol{t}} \xleftarrow{\$} \{0,1\}^{k_0};\ \hat{\boldsymbol{h}} \leftarrow \hat{\boldsymbol{t}} \oplus \hat{\boldsymbol{r}}$$

Putting together Equations (1)–(7) concludes the proof of the statement in Figure 1.

Detailed proof of the transition from G_2 *to* G_3 We use the five intermediate games shown in Figure 3. The first transition from G_2 to G_2^1 consists in adding a Boolean flag in the memory of G that will be used to record whether a query originated directly from the adversary or from the decryption oracle. The decryption oracle tests this tag when accessing the memory of G: if the ciphertext queried is valid and its random seed appeared in a previous decryption query, but not yet in a direct query to G, the decryption oracle raises a flag $\mathbf{bad}_1$. We show that this can happen with probability 2^{-k_1} for any single query, since the random seed is uniformly distributed and independent from the adversary's view. In this case, the decryption oracle can safely reject the ciphertext, as done in game G_2^2. The proof proceeds in two steps. We first show that game G_2 is observationally equivalent to game G_2^1 using the relational invariant

$$\boldsymbol{L}_G\langle 1\rangle = \mathsf{map}\ (\lambda(r,(b,g)).(r,g))\ \boldsymbol{L}_G\langle 2\rangle$$

where $e\langle 1\rangle$ (resp. $e\langle 2\rangle$) denotes the value that an expression e takes in the left hand side (resp. right-hand side) program in an equivalence. Therefore,

$$\Pr[\mathsf{G}_2 : \mathbf{bad}] = \Pr\big[\mathsf{G}_2^1 : \mathbf{bad}\big]$$

Game G_2^2 is identical to G_2^1, except that it rejects ciphertexts that raise the $\mathbf{bad}_1$ flag. Applying the Fundamental Lemma, we show that

$$\Pr\big[\mathsf{G}_2^1 : \mathbf{bad}\big] \leq \Pr\big[\mathsf{G}_2^2 : \mathbf{bad}\big] + \Pr\big[\mathsf{G}_2^2 : \mathbf{bad}_1\big]$$

Our next goal is to show that answers to queries tagged as true can be resampled. However, one cannot directly apply the logic of swapping statements at this stage to resample these answers in G because flag $\mathbf{bad}_1$ is set on $\mathcal{D}$ and depends on them. The solution is to introduce a new game G_2^3 that sets another flag $\mathbf{bad}_2$ in the code of G instead of setting $\mathbf{bad}_1$ in the decryption oracle[6]. Flag $\mathbf{bad}_2$ is raised whenever the adversary queries G with the random seed of a valid ciphertext previously submitted to the decryption oracle. We prove that games G_2^2 and G_2^3 satisfy the relational invariant:

$$\mathbf{bad}_1\langle 1\rangle \implies (\mathbf{bad}_2 \vee \phi)\langle 2\rangle$$

where the predicate ϕ is defined as

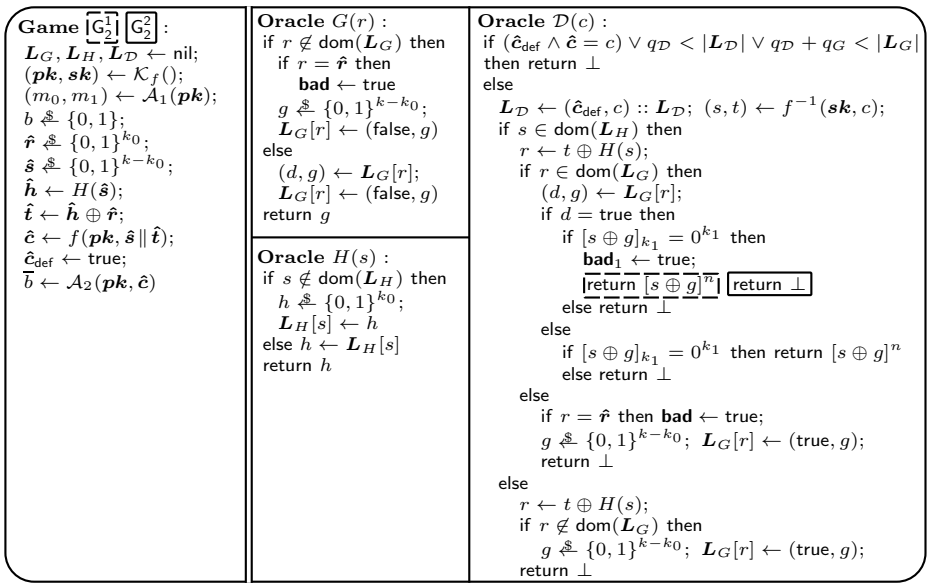
$$\begin{array}{l}\exists (d,c) \in \boldsymbol{L}_{\mathcal{D}}.\ \mathsf{let}\ (s,t) = f^{-1}(\boldsymbol{sk},c),\ r = t \oplus \boldsymbol{L}_H[s]\ \mathsf{in}\\ \quad r \in \mathsf{dom}(\boldsymbol{L}_G) \wedge s \in \mathsf{dom}(\boldsymbol{L}_H) \wedge \mathsf{fst}(\boldsymbol{L}_G[r]) = \mathsf{false} \wedge [s \oplus \mathsf{snd}(\boldsymbol{L}_G[r])]_{k_1} = 0^{k_1}\end{array}$$

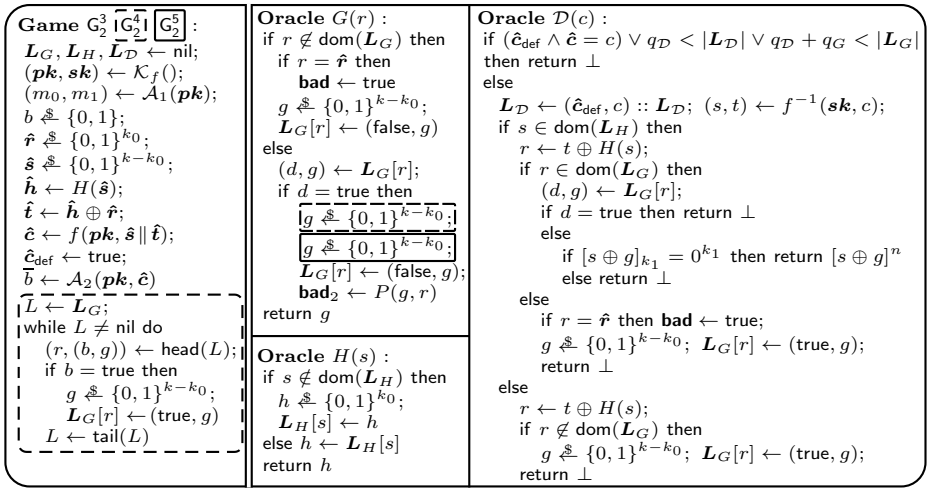
Therefore:

$$\Pr\big[\mathsf{G}_2^2 : \mathbf{bad}\big] + \Pr\big[\mathsf{G}_2^2 : \mathbf{bad}_1\big] \leq \Pr\big[\mathsf{G}_2^3 : \mathbf{bad}\big] + \Pr\big[\mathsf{G}_2^3 : \mathbf{bad}_2 \vee \phi\big]$$

We now consider game G_2^4 where oracle G resamples the answers to queries previously sampled in the decryption oracle. As such answers are uniformly

[6] As $\mathbf{bad}_1$ is not set anymore, we simplify the code of $\mathcal{D}$ by coalescing branches in the innermost conditional.





$P(g, r) \stackrel{\text{def}}{=} \exists (d, c) \in \boldsymbol{L}_{\mathcal{D}}.\ \mathsf{let}\ (s, t) = f^{-1}(\boldsymbol{sk}, c)\ \mathsf{in}\ s \in \mathsf{dom}(\boldsymbol{L}_H)\ \wedge\ r = t \oplus \boldsymbol{L}_H[s]\ \wedge\ [s \oplus g]_{k_1} = 0^{k_1}$

Fig. 3. Games in the transition from G_2 to G_3. Fragments of code inside a box appear only in the game whose name is surrounded by the matching box.

distributed and independent from the adversary's view, the logic for swapping statements can be used to establish that this transformation preserves semantics. Hence:

$$\Pr\left[\mathsf{G}_2^3 : \mathbf{bad}\right] + \Pr\left[\mathsf{G}_2^3 : \mathbf{bad}_2 \vee \phi\right] = \Pr\left[\mathsf{G}_2^4 : \mathbf{bad}\right] + \Pr\left[\mathsf{G}_2^4 : \mathbf{bad}_2 \vee \phi\right]$$

Note that in order to prove semantic equivalence we need to resample the values in $\boldsymbol{L}_G$ associated to queries tagged as true—made by the $\mathcal{D}$—at the end of the game. Using the Failure Event Lemma of [7], we upper bound the probability of $\mathbf{bad}_2 \vee \phi$ in G_2^4:

$$\Pr\left[\mathsf{G}_2^4 : \mathbf{bad}_2 \vee \phi\right] \leq \frac{q_{\mathcal{D}}}{2^{k_1}}$$

We are now only interested in bounding **bad**, so we can remove as dead code the fragment of code at the end of G_2^4 that resamples values in $\boldsymbol{L_G}$, obtaining G_2^5, and prove that

$$\Pr\left[\mathsf{G}_2^4 : \mathbf{bad}\right] = \Pr\left[\mathsf{G}_2^5 : \mathbf{bad}\right]$$

We finally prove that game G_2^5 is observationally equivalent to G_3, in which the code for the oracle G is reverted to its original form and the decryption oracle no longer tampers with the memory of G. Thus,

$$\Pr[\mathsf{G}_2 : \mathbf{bad}] \leq \Pr\left[\mathsf{G}_2^5 : \mathbf{bad}\right] + \frac{q_{\mathcal{D}}}{2^{k_1}} = \Pr[\mathsf{G}_3 : \mathbf{bad}] + \frac{q_{\mathcal{D}}}{2^{k_1}} \qquad \square$$

Comparison with the security bound in [27] Pointcheval obtains a slightly different bound:

$$\epsilon' \geq \left(\frac{\epsilon}{2} - \frac{4q_{\mathcal{D}}q_G + 2q_{\mathcal{D}}^2 + 4q_{\mathcal{D}} + 8q_G}{2^{k_0}} - \frac{3q_{\mathcal{D}}}{2^{k_1}}\right)$$

We marginally improve on this bound by reducing the coefficients. As previously mentioned, the improvement stems from the transition from G_2 to G_3, where we eliminate both calls to G, whereas only the second call is eliminated in [27]. In fact, eliminating both calls is not only useful to give a better bound, but also essential for the correctness of the proof. Indeed, the transition from G_3 to G_4 would not be possible if $\mathcal{D}$ modified the memory of G. Concretely, the justification of Equation (27) in [27] contains two minor glitches: firstly, the remark "which just cancels r' from $\boldsymbol{L}_G$" oversees the possibility of this removal having an impact on future queries. Secondly, "the probability for r' to be in $\boldsymbol{L}_G$ is less than $q_G/2^{k_0}$" oversees that the length of $\boldsymbol{L}_G$ is upper bounded by $q_G + q_{\mathcal{D}}$ rather than q_G, as the decryption oracle still adds values to $\boldsymbol{L}_G$; a correct bound for this probability in [27] is $(q_G + q_{\mathcal{D}})/2^{k_0}$.

5 Perspectives

The CertiCrypt framework consists of over 30,000 lines of Coq. Less than 5% of the development is part of the trusted base, covering the definition of the

semantics, of well-formed adversaries, and of probabilistic polynomial-time programs. The remaining 95% consist of proof tools, including the mechanization of common program transformations, of observational equivalence and Relational Hoare Logic, and of the Fundamental Lemma of Game-Playing. The logic of swapping statements, and the Failure Event Lemma, that have been developed specifically for the purpose of this proof, account for about 1,300 and 500 lines of Coq, respectively.

The verifiable proof is over 10,000 lines of Coq scripts, and can be checked fully automatically using Coq version 8.2pl1, the latest, as yet unreleased, version of Audebaud and Paulin's library of probabilities, and the current implementation of the CertiCrypt framework. Most importantly, less than 1% of the verifiable proof needs to be trusted, namely the formal statement of Figure 1.

The structure of the formal proof is more fine grained than the outline of Figure 2, and contains about 30 games. For example, just the transition from $\mathsf{G}_{\mathsf{IND\text{-}CCA}}$ to G_1 overviewed in Section 4 accounts for 10 games. Decomposing transitions into intermediate games is mostly a matter of taste, but common wisdom in formal proofs is to introduce many intermediate lemmas with short proofs rather than a few lemmas with intricate proofs.

The overall proof was completed within about 6 man-months. While substantial, and perhaps even a bit discouraging for a scientist without experience in formal proofs, the effort required to complete the proof is reasonable in comparison with other large-scale formalization projects. Moreover, a significant amount of work was devoted to pinpoint the details of the proof, and to find a means to capture formally "independence from the adversary's view". We expect that formalizing related proofs in the line of [4, 13] would now be significantly faster.

Still, the time and expertise required for developing formal proofs currently make verifiable security an exclusive option that might be considered for proving standards, but that is otherwise too costly for cryptographers to use in their own research. In an attempt to make verifiable security a reasonable (and we believe profitable) alternative for the working cryptographer, we are building dedicated proof engines to which most of the construction of a verifiable proof could be delegated. Preliminary experiments suggest that most formalizations in CertiCrypt, including our proof of OAEP and the proofs in [6,33], rely on relational invariants that fall in a decidable fragment of predicate logic, and that can be established through simple heuristics. We are currently developing a front-end to CertiCrypt that extracts verifiable proofs from a proof sketch submitted by the user consisting of a sequence of games and statements that justify transitions, including relational invariants.

6 Related Work

The motivations behind verifiable security appear in Bellare and Rogaway's seminal article on code-based game-playing proofs [10], and in Halevi's manifesto for computer-aided cryptographic proofs [20]. However, the most extensive realization of verifiable security to date is CertiCrypt [6], which has been used previously

to build verifiable security proofs of the existential unforgeability of FDH signatures (both for the conventional and optimal bounds) and of semantic security of OAEP. CertiCrypt is particularly suitable for formalizing proofs involving algebraic and number-theoretic reasoning, since in addition to automating common techniques used in game-based cryptographic proofs, it gives access to the full expressive power of the logic of Coq and to the many available libraries and theories developed using it. There is a leap in complexity between the proof of IND-CPA security of OAEP and the proof of IND-CCA security presented here. Specifically, tools such as the Failure Event Lemma and the logic of swapping statements were developed to tackle difficulties arising in some transitions in the latter proof. In another attempt to build a system that supports verifiable security, Backes, Berg and Unruh [3] formalize a language for games in the Isabelle proof assistant, and prove the Fundamental Lemma; however, no specific example is reported. Nowak [25], and Affeldt, Marti and Tanaka [1] also report on preliminary experiments with machine-checked proofs.

CryptoVerif [11] is a prover for exact security of cryptographic schemes and protocols in the computational model; it has been used to verify Kerberos [12] and the conventional bound of FDH [11]. CryptoVerif trades off generality for automation, and consequently adopts a non-standard axiomatization of cryptographic primitives based on term rewriting. As a result, sequences of games can sometimes be inferred automatically; yet, at the same time, the connection between CryptoVerif proofs and standard cryptographic proofs is not as strong as one would desire. Finally, CryptoVerif in its current form acts more like a proof engine than a proof checker, and thus does not comply with the objective of verifiable security—see however [17] for preliminary work on certifying successful runs of CryptoVerif. Courant et al. [14] have also developed an automated prover for proving asymptotic security of encryption schemes based on one-way functions. Their prover is able to handle many schemes from the literature, but it cannot handle OAEP. As CryptoVerif, their tool is a proof engine and does not generate verifiable proofs. More recently, Barthe et al. [5] propose a computationally sound logic to reason about cryptographic primitives. Their logic captures many common reasoning steps in cryptographic proofs and has been used to prove the exact security of PSS. There is no tool support for this logic.

Somehow surprisingly, Koblitz [23] recently published an article that vehemently dismisses relying on computer-assisted proof building and proof checking. While Koblitz rightfully points to some weaknesses of existing tools—e.g. lack of automation and unduly verbosity—a closer look at the article reveals a fragmentary knowledge of the state-of-the-art in machine-checked proofs, and a profound misconception on the role of formal verification.

7 Conclusion

Verifiable security goes beyond provable security by providing independently verifiable evidence that proofs are correct. We used the Coq proof assistant to build the first verifiable proof of IND-CCA security of OAEP. Our proof is a strong

indicator that proof assistants are mature enough to support the construction of cryptographic proofs, and gives strong empirical evidence that dedicated tactics could improve automation and reduce the length and development time of formal proofs. Making verifiable security an appealing alternative for working cryptographers is the next objective.

References

1. R. Affeldt, M. Tanaka, and N. Marti. Formal proof of provable security by game-playing in a proof assistant. In *1st International Conference on Provable Security, ProvSec 2007*, volume 4784 of *Lecture Notes in Computer Science*, pages 151–168, Berlin, 2007. Springer.
2. P. Audebaud and C. Paulin-Mohring. Proofs of randomized algorithms in Coq. *Sci. Comput. Program.*, 74(8):568–589, 2009.
3. M. Backes, M. Berg, and D. Unruh. A formal language for cryptographic pseudocode. In *15th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2008*, volume 5330 of *Lecture Notes in Computer Science*, pages 353–376. Springer, 2008.
4. M. Backes, M. Dürmuth, and D. Unruh. OAEP is secure under key-dependent messages. In *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 506–523. Springer, 2008.
5. G. Barthe, M. Daubignard, B. Kapron, and Y. Lakhnech. Computational indistinguishability logic. In *17th ACM Conference on Computer and Communications Security, CCS 2010*, New York, 2010. ACM.
6. G. Barthe, B. Grégoire, and S. Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009*, pages 90–101, New York, 2009. ACM.
7. G. Barthe, B. Grégoire, and S. Zanella Béguelin. Programming language techniques for cryptographic proofs. In *1st International Conference on Interactive Theorem Proving, ITP 2010*, volume 6172 of *Lecture Notes in Computer Science*, pages 115–130, Berlin, 2010. Springer.
8. M. Bellare, D. Hofheinz, and E. Kiltz. Subtleties in the definition of IND-CCA: When and how should challenge-decryption be disallowed? Cryptology ePrint Archive, Report 2009/418, 2009.
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, Berlin, 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, Berlin, 2006. Springer.
11. B. Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Trans. Dependable Sec. Comput.*, 5(4):193–207, 2008.
12. B. Blanchet, A. D. Jaggard, A. Scedrov, and J.-K. Tsay. Computationally sound mechanized proofs for basic and public-key Kerberos. In *15th ACM Conference on Computer and Communications Security, CCS 2008*, pages 87–99, New York, 2008. ACM.
13. A. Boldyreva. Strengthening security of RSA-OAEP. In *Topics in Cryptology – CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 399–413. Springer, 2009.

14. 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.
15. E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. *J. Cryptology*, 17(2):81–104, 2004.
16. G. Gonthier. Formal Proof — The Four Colour Theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.
17. J. Goubault-Larrecq. Towards producing formally checkable security proofs, automatically. In *21st IEEE Computer Security Foundations Symposium, CSF 2008*, pages 224–238. IEEE Computer Society, 2008.
18. T. Hales. Formal Proof. *Notices of the AMS*, 55(11):1370–1380, 2008.
19. T. Hales, J. Harrison, S. McLaughlin, T. Nipkow, S. Obua, and R. Zumkeller. A revision of the proof of the Kepler conjecture. *Discrete and Computational Geometry*, 44(1):1–34, 2010.
20. S. Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181, 2005.
21. G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: formal verification of an OS kernel. In *22nd ACM Symposium on Operating Systems Principles, SOSP 2009*, pages 207–220. ACM Press, 2009.
22. G. Klein and T. Nipkow. A machine-checked model for a Java-like language, virtual machine and compiler. *ACM Trans. Program. Lang. Syst.*, 28(4):619–695, 2006.
23. N. Koblitz. Another look at automated theorem-proving. *J. Math. Cryptol.*, 1(4):385–403, 2008.
24. X. Leroy. Formal certification of a compiler back-end, or: programming a compiler with a proof assistant. In *33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2006*, pages 42–54, New York, 2006. ACM.
25. D. Nowak. A framework for game-based security proofs. In *9th International Conference on Information and Communications Security, ICICS 2007*, volume 4861 of *Lecture Notes in Computer Science*, pages 319–333, Berlin, 2007. Springer.
26. L. C. Paulson. The inductive approach to verifying cryptographic protocols. *J. of Comput. Secur.*, 6(1-2):85–128, 1998.
27. D. Pointcheval. Provable security for public key schemes. In *Advanced Courses on Contemporary Cryptology*, chapter D, pages 133–189. Birkhäuser Basel, 2005.
28. M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1979.
29. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
30. V. Shoup. OAEP reconsidered. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 239–259, Berlin, 2001. Springer.
31. V. Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004.
32. The Coq development team. The Coq Proof Assistant Reference Manual Version 8.2. Online – `http://coq.inria.fr`, 2009.
33. S. Zanella Béguelin, B. Grégoire, G. Barthe, and F. Olmedo. Formally certifying the security of digital signature schemes. In *30th IEEE Symposium on Security and Privacy, S&P 2009*, pages 237–250, Los Alamitos, Calif.A, 2009. IEEE Computer Society.