Activity report

SCALP ANR Project

Security of Cryptographic ALgorithms with Probabilities

FinalReport

Chapitre 1

Introduction

Our day-to-day lives increasingly depend upon information and our ability to manipulate it securely. That is, in a way that prevents malicious elements to subvert the available information for their own benefits. This requires solutions based on cryptographic systems (primitives and protocols). Shannon's early work in the 1940's already showed that perfect (unconditional) security is difficult to achieve in practice. Three decades later (1976), Diffie and Hellman invented public-key cryptography, coined the notion of one-way functions and discussed the relationship between cryptography and complexity theory. Shortly after (Rabin 1979), the first cryptosystem with a reductionist security proof appeared. The next breakthrough towards formal proofs of security is the adoption (notably by Goldwasser, Micali, Goldreich and Rivest) of computational theory for the purpose of rigorously defining the security of cryptographic schemes. In this framework, a system is *provably secure* if there is a poly-time reduction proof from a hard problem to an attack against the security of the system. A user of this framework must specify how an arbitrary, probabilistic, poly-time adversary attacker can interact with legitimate users and must specify what the attacker should achieve in order to break the cryptosystem. The provable security framework has been later refined into *the exact (also called concrete) security framework* where better estimates of the computational complexity of attacks is achieved. Beyond the verification of cryptographic systems and constructions, research in the area of provable and exact security investigates the relationship between different security notions and security models.

Provable cryptography has become a very active field of research well represented in top cryptographic conferences, and cryptographic schemes, including standards, are increasingly supported by correctness proofs. Yet, there is a problem with cryptographic proofs, as expressed by Shai Helevi in [68] : *Do we have a problem with cryptographic proofs ? Yes, we do. The problem is that as a community, we generate more proofs than we carefully verify (and as a consequence some of our published proofs are incorrect).* Alexander Dent [64] has even stronger words : *... cryptographers are only just beginning to develop the mathematical rigor that they need in order to be able to produce algorithms and protocols in which one can have true confidence.* J. Stern et al. write in [81] : *...the use of provable security is more subtle than it appears, and flaws in security proofs themselves might have devastating effect on the trustworthiness of cryptography.* Unfortunately, flaws in security proofs are not that seldom (cf.[81, 54]).

Our aim is to develop general computer-aided tools for verifying cryptographic systems with strong correctness guarantees. Our tools should have the following characteristics :

1. Support the provable and exact security frameworks.
2. Be reasonably automated.
3. Be applicable to realistic systems. We target three application areas : password-based group key exchange protocols, computationally sound type systems for information flow and watermarking algorithms.
4. Put emphasis on proof checkability and certification. This is in order to increase the trustworthiness in developed proofs.

Challenges to be addressed

The success of our project will depend on mastering the following key challenges.

Probabilistic language and semantics for cryptographic proofs We need to develop a programming language that allows to describe the cryptographic systems to be verified. As cryptographic systems usually involve randomness and probabilistic behavior the language has to be probabilistic. But the language has to be more complex than a simple probabilistic language as *it must allow the description of games that specify the interaction between adversaries and the systems.* Therefore we shall address the following issues :

1. An adversary can be seen as a black-box that has access to some oracles. The latter may depend on secret keys. Hence, the language has to include variables that refer to unspecified code that has access to oracles. The language has also to include means for specifying access policies to the system variables.
2. Cryptosystems may involve concurrency as many agents and even many instances of the same agent may be running the system. Clearly, interleaving-based semantics is not appropriate for security proofs. There are essentially two possible views : the system is a set of communicating processes and the adversary controls the scheduling of both processes and messages. Or, the system consists of one sequential process, namely the adversary, who has access to some state-full oracles. Obviously, the choice will largely affect the shape of the proofs we will get as well as the degree of automation we can reach.
3. Since we need to support reductionist proofs that involve the construction of (possibly) poly-time probabilistic programs, the question of how such resource bounded computations can be characterized affects the reachable degree of proof automation. Indeed, reductionist proofs involve the proof obligation that consists in showing that the reduction is bounded under the assumption that adversary is bounded.

Formalization of security properties A key issue toward building and popularizing automated tools for the verification of cryptosystems is the development of canonical definitions of widely used security properties. In particular, we should draw a clear picture of the landscape of existing computational non-interference definitions. The same holds for properties we expect a group key exchange protocol or the properties a watermarking system should satisfy. More generally, we need to define and study observational equivalences of probabilistic systems that include adversaries.

Proof theory Verification of cryptosystems needs the development of a specific verification theory that has to include the following proof activities :

- **Invariants.** We need to develop abstract high-level reasoning about distributions defined by probabilistic programs. Typically, we need to prove invariants that state the independence of some random variables. For instance, we want to be able to compositionally show that a given probabilistic program preserves indistinguishability of distributions.
- **Semantics preserving program transformations.** As stated by Shai Halevi [68] many steps in a cryptographic proof boil down to "code motion" or more generally to program optimization. We should, however, not forget how many heads has the beast we are dealing with : probabilistic behavior, unspecified code, oracles, concurrency etc... Thus, we need to specify and prove correctness of code optimization for cryptoprograms.
- **Cryptography-based program transformations.** A different kind of step in a cryptographic proof are reduction proofs. In such steps, we want to establish a property by reducing from a cryptographic or number theoretic assumption such as prime number factoring, RSA, CDH or DDH. At the heart of such proof step, we have the construction of a probabilistic program that uses a set of programs as black-boxes. One of the proof obligations that need to be discharged is that the constructed program runs in poly-time. Consequently, complexity definitions and reasoning need to be captured by our tools and formalizations.

– **Asymptotic reasoning.** An other feature that distinguishes cryptoproofs is that the reasoning about probabilities may be asymptotic. A typical security property states that for any adversary the probability of an attack is ultimately negligible. This is an asymptotic notion. An alternative to asymptotic is to work within the concrete (also called exact) security framework. The aim is here to give more precise bounds on the probability of attacks. This is indeed preferable. But asymptotic reasoning might be useful first to get a simple proof of correctness that can be refined into a concrete proof. We need to formalize both types of reasoning and we need to develop proof strategies that allow to refine an asymptotic proof into an exact one without repeating all proof steps.

Chapitre 2

WPs

The Scalp project is organized into five work-packages. We briefly recall their purpose and describe our results so far, and/or research topics for each one :

2.1 WP1

This WP aims at defining both a language and its semantics for representing random computations that will be used for modeling randomized algorithms.

We have chosen to base our framework on the Why software verification framework. This tool provides both a syntax for writing imperative programs together with annotations, and an enriched weakest least conditions transformer to generate conditions verifications aimed to be given to various provers assistants (Coq, PVS, HOL*,..) or automatic provers as well (Simplify, CVC Lite, etc.).

This tool has already been successfully extended to C programs (Caduceus) and Java programs (Krakatoa).

Yet, several issues had to be adressed toward our end :

- extend the frontend to deal with randomized programs ;
- allow for new atomic predicates specific to such programs ;
- extend the internal representation (monads with effects) for the enriched syntax ;
- generate condition verifications accordingly ;
- provide accordingly a pure fonctional intepretation for the new programs.

All these issues have been adressed during the Scalp project in the following papers [44, 47, 48, 43] and the intermediate Sclap reports.

In the following, we present some of the main features of our language.

2.1.1 Games as programs

The essence of code-based cryptographic proofs is to express in a unified semantic framework games, hypotheses, and results. This semanticist perspective allows a precise specification of the interaction between the adversary and the challenger in a game, and to readily answer questions as : Which oracles does the adversary have access to ? Can the adversary read/write this variable ? How many queries the adversary can make to a given oracle ? What is the length of a bitstring returned by the adversary ? Can the adversary repeat a query ? Furthermore, other notions such as probabilistic polynomial-time complexity or termination fit naturally in the same framework and complete the specification of adversaries and games.

The pWhile language

Games are formalized in pWhile, a probabilistic imperative language with procedure calls. Given a set $\mathcal{V}$ of variable identifiers, a set $\mathcal{P}$ of procedure identifiers, a set $\mathcal{E}$ of expressions, and a set $\mathcal{D}$ of *distribution expressions*, the set of commands can be defined inductively by the clauses :

$$\begin{array}{rcll} \mathcal{I} & ::= & \mathcal{V} \leftarrow \mathcal{E} & \text{assignment} \\ & | & \mathcal{V} \xleftarrow{\$} \mathcal{D} & \text{random sampling} \\ & | & \mathsf{if}\ \mathcal{E}\ \mathsf{then}\ \mathcal{C}\ \mathsf{else}\ \mathcal{C} & \text{conditional} \\ & | & \mathsf{while}\ \mathcal{E}\ \mathsf{do}\ \mathcal{C} & \text{while loop} \\ & | & \mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E}) & \text{procedure call} \\ \mathcal{C} & ::= & \mathsf{nil} & \text{nop} \\ & | & \mathcal{I};\ \mathcal{C} & \text{sequence} \end{array}$$

Rather than adopting the above definition, we impose that programs in pWhile are typed. Thus, $x \leftarrow e$ is well-formed only if the types of x and e coincide, and $\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2$ is well-formed only if e is a boolean expression. In practice, we assume that variables and values are typed, and define a dependently typed syntax of programs. An immediate benefit of using dependent types is that the type system of Coq ensures for free the well-typedness of expressions and commands. Although the formalization is carefully designed for being extensible w.r.t. user-defined types and operators (and we do exploit this in practice), it is sufficient for the purpose of this paper to consider an instance in which values are booleans, bitstrings, natural numbers, pairs, lists, and elements of a cyclic group. Similarly, we instantiate $\mathcal{D}$ so that values can be uniformly sampled from the set of booleans, natural intervals of the form $[0..n]$, and bitstrings of a certain length. It is important to note that the formalization of expressions is not restricted to many-sorted algebra : we make a critical use of dependent types to record the length of bitstrings.

Definition 1 (Program)**.** *A program consists of a command and an environment, which maps a procedure identifier to its declaration, consisting of its formal parameters, its body, and a return expression (we use an explicit* return *when writing games, though),*

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

The environment specifies the type of the parameters and the return expression, so that procedure calls are always well-typed.

In a typical formalization, the environment will map procedures to closed commands, with the exception of the adversaries whose code is unknown, and thus modeled by variables of type $\mathcal{C}$. This is a standard trick to deal with uninterpreted functions in a deep embedding. In the remainder of this section we assume an environment E implicitly given.

We let c_i range over $\mathcal{C}$; x_i over $\mathcal{V}$; e_i over $\mathcal{E}$; d_i over $\mathcal{D}$; and G_i over programs. The operator $\oplus$ denotes the bitwise exclusive or on bitstrings of equal length, and $\|$ the concatenation of two bitstrings.

Operational semantics

Programs in pWhile are given a small-step semantics using the measure monad $M(X)$, whose type constructor is defined as

$$M(X) \stackrel{\text{def}}{=} (X \to [0,1]) \to [0,1]$$

and whose operators unit and bind are defined as

$$\begin{array}{l} \mathsf{unit}\ :\ X \to M(X) \stackrel{\text{def}}{=} \lambda x.\ \lambda f.\ f\ x \\ \mathsf{bind}\ :\ M(X) \to (X \to M(Y)) \to M(Y) \\ \qquad \stackrel{\text{def}}{=} \lambda \mu.\ \lambda M.\ \lambda f.\ \mu(\lambda\ x.\ M\ x\ f) \end{array}$$

$$
\begin{array}{rcll}
(\mathsf{nil}, m, [\,]) & \rightsquigarrow & \mathsf{unit}\ (\mathsf{nil}, m, [\,]) & \\
(\mathsf{nil}, m, (x, e, c, l) :: F) & \rightsquigarrow & \mathsf{unit}\ (c, (l, m.\mathsf{glob})\{[\![e]\!]\ m/x\}, F) & \\
(x \leftarrow p(\vec{e});\ c, m, F) & \rightsquigarrow & \mathsf{unit}\ (E(p).\mathsf{body}, (\emptyset\{[\![\vec{e}]\!]\ m/E(p).\mathsf{params}\}, m.\mathsf{glob}), (x, E(p).\mathsf{re}, c, m.\mathsf{loc}) :: F) & \\
(\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2;\ c, m, F) & \rightsquigarrow & \mathsf{unit}\ (c_1;\ c, m, F) & \text{if } [\![e]\!]\ m = \mathsf{true} \\
(\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2;\ c, m, F) & \rightsquigarrow & \mathsf{unit}\ (c_2;\ c, m, F) & \text{if } [\![e]\!]\ m = \mathsf{false} \\
(\mathsf{while}\ e\ \mathsf{do}\ c;\ c', m, F) & \rightsquigarrow & \mathsf{unit}\ (c;\ \mathsf{while}\ e\ \mathsf{do}\ c;\ c', m, F) & \text{if } [\![e]\!]\ m = \mathsf{true} \\
(\mathsf{while}\ e\ \mathsf{do}\ c;\ c', m, F) & \rightsquigarrow & \mathsf{unit}\ (c', m, F) & \text{if } [\![e]\!]\ m = \mathsf{false} \\
 & & & \\
(x \leftarrow e;\ c, m, F) & \rightsquigarrow & \mathsf{unit}\ (c, m\{[\![e]\!]\ m/x\}, F) & \\
(x \xleftarrow{\$} d;\ c, m, F) & \rightsquigarrow & \mathsf{bind}\ ([\![d]\!]\ m)(\lambda v.\ \mathsf{unit}\ (c, m\{v/x\}, F)) &
\end{array}
$$

FIGURE 2.1 – Probabilistic semantics of pWHILE programs

The monad $M(X)$ was proposed by Audebaud and Paulin [38] as a variant of the expectation monad used by Ramsey and Pfeffer [77], and builds on earlier work by Kozen [74]. The formalization of the semantics heavily relies on Paulin's axiomatization in Coq of the $[0, 1]$ real interval—for our purposes, it has been necessary to add division to the library.

The semantics of commands and expressions are defined relative to a given memory, i.e. a mapping from variables to values. We let $\mathcal{M}$ denote the set of memories. Expressions are deterministic ; their semantics is standard and given by a function $[\![\cdot]\!]_{\text{expr}}$, that evaluates an expression in a given memory and returns a value. The semantics of distribution expressions is given by a function $[\![\cdot]\!]_{\text{distr}}$. For a distribution expression d of type T, we have that $[\![d]\!]_{\text{distr}} : \mathcal{M} \to M(X)$, where X is the interpretation of type T. For instance, in the previous subsection we have used $\{0,1\}^\eta$ to denote the uniform distribution on bitstrings of length η (the security parameter), formally, we have $[\![\{0,1\}^\eta]\!]_{\text{distr}} \stackrel{\text{def}}{=} \lambda m\ f.\ \sum_{bs \in \{0,1\}^\eta} \frac{1}{2^\eta} f(bs)$. Thanks to dependent types, the semantics of expressions and distribution expressions is total. In the following, and whenever there is no confusion, we will drop the subscripts in $[\![\cdot]\!]_{\text{expr}}$ and $[\![\cdot]\!]_{\text{distr}}$.

The semantics of commands relates a deterministic state to a (sub-)probability distribution over deterministic states and uses a frame stack to deal with procedure calls. Formally, a deterministic state is a triple consisting of the current command $c : \mathcal{C}$, a memory $m : \mathcal{M}$, and a frame stack $F : \mathsf{list\ frame}$. We let $\mathcal{S}$ denote the set of deterministic states. One step execution $[\![\cdot]\!]^1 : \mathcal{S} \to M(\mathcal{S})$ is defined by the rules of Fig. 2.1. In the figure, we use $a \rightsquigarrow b$ as a notation for $[\![a]\!]^1 = b$ and loc and glob to project memories on local and global variables respectively.

We briefly comment on the transition rules for calling a procedure (3rd rule) and returning from a call (2nd rule). Upon a call, a new frame is appended to the stack, containing the destination variable, the return expression of the called procedure, the continuation to the call, and the local memory of the caller. The state resulting from the call contains the body of the called procedure, the global part of the memory, a local memory initialized to map the formal parameters to the value of the actual parameters just before the call, and the updated stack. When returning from a call with a non-empty stack, the top frame is popped, the return expression is evaluated and the resulting value is assigned to the destination variable after previously restoring the local memory of the caller ; the continuation taken from the frame becomes the current command. If the stack is empty when returning from a call, the execution of the program terminates and the final state is embedded into the monad using the unit operator.

Using the monadic constructions, one can define an n-step execution function $[\![\cdot]\!]_n$:

$$[\![s]\!]_0 \stackrel{\text{def}}{=} \mathsf{unit}\ s \qquad [\![s]\!]_{n+1} \stackrel{\text{def}}{=} \mathsf{bind}\ [\![s]\!]_n\ [\![\cdot]\!]^1$$

Finally, the denotation of a command c in an initial memory m is defined to be the (limit)

distribution of reachable final memories :

$$[\![c]\!]\ m : M(\mathcal{M}) \stackrel{\text{def}}{=} \lambda f.\ \sup\ \{[\![(c, m, [\,])]\!]_n\ f|_{\mathsf{final}} \mid n \in \mathbb{N}\}$$

where $f|_{\mathsf{final}} : \mathcal{S} \to [0, 1]$ is the function that when applied to a state (c, m, F) gives $f(m)$ if it is a final state and 0 otherwise. Since the sequence $[\![(c, m, [\,])]\!]_n\ f|_{\mathsf{final}}$ is increasing and upper bounded by 1, this least upper bound always exists and corresponds to the limit of the sequence.

Computing probabilities The advantage of using this monadic semantics is that, if we use an arbitrary function as a continuation to the denotation of a program, what we get (for free) as a result is its expected value w.r.t. the distribution of final memories. In particular, we can compute the probability of an event A in the distribution obtained after executing a command c in an initial memory m by measuring its characteristic function $\mathbb{1}_A : \Pr c, mA \stackrel{\text{def}}{=} [\![c]\!]\ m\ \mathbb{1}_A$. For instance, one can verify that the denotation of $x \xleftarrow{\$} [0..1];\ y \xleftarrow{\$} [0..1]$ in the memory m is

$$\begin{aligned}\lambda f.\tfrac{1}{4}(&f(m\{0,0/x,y\}) + f(m\{0,1/x,y\})\\ &+f(m\{1,0/x,y\}) + f(m\{1,1/x,y\}))\end{aligned}$$

and conclude that the probability of the event $x \leq y$ after executing the command above is $\frac{3}{4}$.

Probabilistic polynomial-time programs

In general, cryptographic proofs reason about effective adversaries, which can only use a polynomially bounded number of resources. The complexity notion that captures this intuition, and which is pervasive in cryptographic proofs, is that of *strict probabilistic polynomial-time*. Concretely, a program is said to be strict probabilistic polynomial-time (PPT) whenever there exists a polynomial bound (on some security parameter η) on the cost of each possible execution, regardless of the outcome of its coin tosses. Otherwise said, a probabilistic program is PPT whenever the same program, seen as a non-deterministic program, terminates and the cost of each possible run is bounded by a polynomial.

Termination and efficiency are orthogonal. Consider, for instance, the following two programs :

$$b \leftarrow \mathsf{true}; \mathsf{while}\ b\ \mathsf{do}\ b \xleftarrow{\$} \{0, 1\}$$
$$b \xleftarrow{\$} \{0, 1\}; \mathsf{if}\ b\ \mathsf{then}\ \mathsf{while}\ \mathsf{true}\ \mathsf{do}\ \mathsf{nil}$$

The former terminates with probability 1 (it terminates within n iterations with probability $1 - 2^{-n}$), but may take an arbitrarily large number of iterations to terminate. The latter terminates with probability $\frac{1}{2}$, but when it does, it takes only a constant time. We deal with termination and efficiency separately.

Definition 2 (Termination)**.** *The probability that a program c terminates starting from an initial memory m is $[\![c]\!]\ m\ \mathbb{1}_{\mathsf{true}}$. We say that a program c is absolutely terminating, and note it $\mathsf{Lossless}(c)$, iff it terminates with probability 1 in any initial memory.*

To deal with efficiency, we non-intrusively instrument the semantics of our language to compute the cost of running a program. The instrumented semantics ranges over $M(\mathcal{M} \times \mathbb{N})$ instead of simply $M(\mathcal{M})$. We recall that our semantics is implicitly parametrized by a security parameter η, on which we base our notion of complexity.

Definition 3 (Polynomially bounded distribution)**.** *We say that a distribution $\mu : M(\mathcal{M} \times \mathbb{N})$ is (p, q)-bounded, where p and q are polynomials, whenever for every (m, n) occurring with non-zero probability in μ, the size of every value in the memory m is bounded by $p(\eta)$ and the cost n is bounded by $q(\eta)$.*

Definition 4 (Strict probabilistic polynomial-time program)**.** *A program c is strict probabilistic polynomial-time (PPT) iff it terminates absolutely, and there exist polynomial transformers F, G such that for every (p, q)-bounded distribution μ, the distribution $(\mathsf{bind}\ \mu\ [\![c]\!])$ is $(F(p), q + G(p))$-bounded.*

We can recover some intuition by taking $\mu = \mathsf{unit}\ (m, 0)$ in the above definition. In this case, we may paraphrase the condition as follows : if the size of values in m is bounded by some polynomial p, and an execution of the program in m terminates with non-zero probability in memory m', then the size of values in m' is bounded by the polynomial $F(p)$, and the cost of the execution is bounded by the polynomial $G(p)$. It is in this latter polynomial that bounds the cost of executing the program that we are ultimately interested. The increased complexity in the definition is required for proving compositionality results (e.g. the sequential composition of two PPT programs results in a PPT program).

Although our formalizations of termination and efficiency rely on semantic definitions, it is not necessary for users to reason directly about the semantics of a program to prove it meets those definitions. CertiCrypt implements a certified algorithm showing that every program without loops and recursive calls is lossless. [1] CertiCrypt also provides another algorithm that, together with the first, establishes that a program is PPT provided that, additionally, the program does not contain expressions that might generate values of superpolynomial size or take a superpolynomial time when evaluated in a polynomially bounded memory.

2.2 WP 2

This work packages focuses on the design of a *proof environment for security programs*. There are two approaches to the computer-aided verification of cryptographic systems, more precisely security protocols. The *indirect approach* is based the combination of symbolic verification also called Dolev-Yao) of protocols with so-called correspondence theorems. Such theorems establish soundness of the symbolic framework with respect to the computational. The symbolic framework idealizes the cryptographic primitives and defines attacks as sequences of a fixed set of operations.

On the opposite, the *direct approach* is based on the computational frameworks, where cryptographic primitives and attacks are probabilistic poly-time computations. While in the symbolic approach correctness of a protocol corresponds to the non-existence of an attack, in the computational framework one is interested in the probability of feasible attacks. The book [33] to which we contributed with a chapter [53] is a comprehensive survey.

The partners of the Scalp project developed three complementary approaches to computational security proofs that we briefly discuss in the remaining of this section.

2.2.1 The Computational Indistinguishability Logic - CIL

Computational Indistinguishability Logic (CIL) [36, 62] is a logic that supports concise and intuitive proofs across several models of cryptography. Its starting point is the notion of *oracle system*, an abstract model of interactive games in which adaptive adversaries play against a cryptographic scheme by interacting with oracles. Oracle systems are inspired by probabilistic process algebra, but do not commit to a particular model or syntax. As a result, they provide a unified foundation for cryptographic games, can be formalized neatly in a proof assistant, and capture both the standard model and idealized models such as the random oracle model or ideal ciphertext model. Moreover, oracle systems provide a unifying semantics for the different languages used in practical tools for cryptographic proofs.

CIL features a small set of rules that capture common reasoning patterns, e.g. simulations and reductions steps. The soundness of these rules may be established using (mild variants of) concepts that are well understood by the programming language and concurrency communities, such as contexts and bisimulations. Moreover, CIL features interface rules to connect with external reasoning. The use of external reasoning offers a number of advantages. First of all, it enforces a separation between proof steps which are purely logical or information-theoretic from those which directly involve (reduction-based) security. Secondly, it allows a presentation of the proof system

1. It is of course a weak result in terms of termination of probabilistic programs, but nevertheless sufficient as regards cryptographic applications. Extending our formalization to a certified termination analysis for loops is interesting, but orthogonal to our main goals, and left for future work.

which does not commit to a particular syntactic representation for oracle systems – rules specific to a particular representation may be introduced as "plug-ins" to the rules as presented. Thus, rules for establishing external premises are not considered as part of CIL. Instead, existing proof techniques, including automated techniques, are used to establish external premises.

CIL has been formalized using the theorem prover Coq. In [55, 41], we report on this formalization as well as on an application of CIL to a leakage-resilient key exchange protocol.

In order to enhance the level of automation when using CIL and its formalization, we developed automated verification procedures for asymmetric encryption [57], block cipher modes [65, 66] and MACs.

2.2.2 Certicrypt

CertiCrypt is a framework to construct machine-checked code-based security proofs in the Coq proof assistant [82], supporting :

Faithful and rigorous encoding of games. In order to be readily accessible to cryptographers, we have chosen a formalism that is commonly used to describe games. Concretely, the lowest layer of CertiCrypt is the formalization of pWhile, an imperative programming language with random assignments, structured datatypes, and procedure calls. We provide a deep and dependently-typed embedding of the syntax ; thanks to dependent types, the typability of pWhile programs is obtained for free. We also provide a small-step operational semantics using the measure monad of Audebaud and Paulin [37]. The semantics is instrumented to calculate the cost of running programs ; this offers the means to define complexity classes, and in particular to define formally the notion of effective (probabilistic polynomial-time) adversary. In addition, we also model nonstandard features, such as policies on variable accesses and procedure calls, and use them to capture many assumptions left informal in cryptographic proofs.

Exact security. Many security proofs establish an asymptotic behavior for adversaries and show that the advantage of any effective adversary is negligible w.r.t. a security parameter (which typically determines the length of keys or messages). However, the cryptographic community is increasingly focused on exact security, a much more useful result since it gives hints as to how to choose system parameters in practice to satisfy a security guarantee. The goal of exact security is to provide concrete bounds both for the advantage of the adversary and for its execution time. CertiCrypt supports the former (but for the time being, not the latter).

Full and independently verifiable proofs. CertiCrypt adopts a formal semanticist perspective and goes beyond Halevi's vision in two respects. First, it provides a unified framework to carry out full proofs ; all intermediate steps of reasoning can be justified formally, including complex side conditions that justify the correctness of transformations (about probabilities, groups, polynomials, etc). Second, one notable feature of Coq, and thus CertiCrypt, is to support independent verifiability of proofs, which is an important motivation behind game-based proofs. More concretely, every proof is represented by a proof object, that can be checked automatically by a (small and trustworthy) proof checking engine. In order to trust a cryptographic proof, one only needs to check its statement, and not its details.

Powerful and automated reasoning methods. CertiCrypt formalizes a Relational Hoare Logic and a theory of observational equivalence, and uses them as stepping stones to support the main tools of code-based reasoning through certified, reflective tactics. In particular, CertiCrypt shows that many transformations used in code-based proofs, including common optimizations, are semantics-preserving. One of its specific contributions is to prove formally the correctness of a variant of lazy sampling, which is used ubiquitously in cryptographic proofs. In addition, CertiCrypt supports methods based on failure events (the so-called fundamental lemma of game-playing).

CertiCrypt has been extended in [49, 50] to deal with *differential privacy.*

2.2.3 Easycrypt

EasyCrypt [43] is an automated tool that builds machine-checked proofs from *proof sketches*, which offer a machine-processable representation of the essence of a security proof. We argue that EasyCrypt is significantly easier to use than previous tools, making an important step towards the adoption of computer-aided security proofs by working cryptographers and hence towards fulfilling Halevi's programme. To substantiate our claim, we present computer-aided proofs of security of Hashed ElGamal encryption and the Cramer-Shoup cryptosystem.

EasyCrypt adopts the principled approach mandated by CertiCrypt to conduct game-based proofs and imposes a clear separation between program verification and information-theoretic reasoning. Transitions between games are justified in two steps : first, one proves logical relations between the games using probabilistic Relational Hoare Logic (pRHL) ; second, one applies information-theoretic reasoning to derive claims about the probability of events from pRHL judgments. We provide for each step highly effective mechanisms that build upon a combination of off-the-shelf and purpose-specific tools. Specifically, EasyCrypt implements an automated procedure that computes for any pRHL judgment a set of sufficient conditions for its validity, known as verification conditions. The outstanding feature of this procedure, and the key to the effectiveness of EasyCrypt, is that verification conditions are expressed in the language of first-order logic, without any mention of probability, and can be discharged automatically by state-of-the-art tools such as SMT solvers and theorem provers. The verification condition generator is *proof-producing*, in the sense that it generates Coq files that can be machine-checked using the CertiCrypt framework. Moreover, the connection to CertiCrypt makes it possible to benefit from the expressivity and flexibility of a general-purpose proof assistant for advanced verification goals that fall out of the scope of automated techniques. Additionally, EasyCrypt implements an automated mechanism for proving claims about probability. The mechanism combines some elementary rules to compute (bounds on) probabilities of events—e.g. the probability of a uniformly sampled element to belong to a list—with rules to derive (in)equalities between probabilities of events in games from judgments in pRHL. The combination of these tools with other more mundane features such as decisions procedurs [40] makes EasyCrypt a plausible candidate for adoption by working cryptographers.

2.3 WP 3

This work package focuses on *proofs of protocols and construction of cryptographic primitives.*

The applicability of the approaches presented in WP2 have been applied to several constructons and security protocols including signature schemes, hash constructions, asymmetric encryptions schemes, cipher block modes,zero-knowledge proofs and security protols [57, 42, 46, 52, 39, 58, 66, 41, 55, 45, 56, 62, 51].

2.4 WP 4

This work package focuses on *Computational flow of information.*

The notion of *non-interference* has been introduced in [67], with the aim of capturing unwanted information flow in programs. Non-interference assumes a separation between secret (high, private) variables and public (low) variables and requires that executing the program in two initial states that coincide on the public variables leads to final states that coincide on the public variables. In Dennings' seminal paper [63], an expression is classified H if it contains a secret variable ; otherwise, it is classified L. The paper introduces two basic principles to avoid information flow : first, to prevent explicit flow, a H expression may not be assigned to a L variable ; second, to prevent implicit flows, an H guarded conditional or loop may not affect L variables. Later, Volpano, Smith, and Irvine [84] casted these principles as a type system and showed that they suffice to ensure non-interference. Since this early work, information flow analysis has been extended to deal with other issues such as nontermination, concurrency, nondeterminism, and exceptions ; see [78] for a survey. In many applications, however, it is desirable to allow information to flow from

secret to public variables in a controlled way. This is called declassification in the literature. In a useful survey, Sabelfeld & Sands [79] classify declassification techniques according to the following dimensions : "what", "who", "where" and "when".

In [59], we are interested in *cryptography-based declassification*, where encrypted secret data can be published without leaking information about the secrets. The non-interference setting has been extended in [83] to cope with one-way functions and in [75, 76, 80] to cope with probabilistic encryption. We consider length-preserving deterministic encryption, i.e., block ciphers. These are widely used in practice (DES, AES, Idea, etc.). Non-interference type systems developed for probabilistic encryption are not applicable for deterministic encryption. To illustrate some of the subtleties of deterministic encryption, let us consider the following examples where l, l', l'' are public variables and h, h' are secret variables, νl assigns a value sampled from the uniform distribution to the variable l, $+$ is a bijective operator and $\mathsf{Enc}(k, e)$ denotes the encryption of e with the symmetric key k. We assume that the encryption function $\mathsf{Enc}(k, \cdot)$ is a pseudo-random permutation.

1. A simple program is the following : $l := \mathsf{Enc}(k, h); l' := \mathsf{Enc}(k, h')$. The equality $\mathsf{Enc}(k, h) = \mathsf{Enc}(k, h')$ is almost never true in case of probabilistic encryption, independently whether $h = h'$. Hence, this program does not leak information in case of probabilistic encryption. This is not true in the case of deterministic encryption as we have $h = h'$ if and only if $l = l'$ at program termination. Indeed, deterministic encryption is not repetition concealing in contrast to probabilistic encryption. Consider now, the program $\nu l; l' := \mathsf{Enc}(k, l + h)$, where the value of l is randomly sampled. It does not leak information, even if the attacker is given the value of l. Yet, we have to be careful concerning how the value of l is used. Indeed, the execution of the command $l'' := \mathsf{Enc}(k, l + h')$ at the end of this program would leak information. However, the following slightly modified program : $\nu l; l' := \mathsf{Enc}(k, l + h); l'' := \mathsf{Enc}(k, l' + h')$ does not leak information. Notice that this version corresponds to a simplified block encryption, using the CBC mode : (l, l', l'') can be seen as the cipher text obtained by encrypting the secret (h, h').
2. Consider now the program : **if** $\mathsf{Enc}(k, h) = \mathsf{Enc}(k, h')$ **then** $l := 0$; **else** $l := 1$; **fi** . Again this program does not leak information in case of probabilistic encryption. But it does in the deterministic case, for we have : $l = 0$ at the end of the program iff $\mathsf{Enc}(k, h) = \mathsf{Enc}(k, h')$ iff $h = h'$. The equality $\mathsf{Enc}(k, h) = \mathsf{Enc}(k, h')$ is almost never true in case of probabilistic encryption. Thus, while equalities are hidden by probabilistic encryption, they are preserved by deterministic encryption. In other words, deterministic encryption is not repetition concealing in contrast to probabilistic encryption.
3. Consider now the following program : $l := \nu r; \mathsf{Enc}(k, h + r); l' := \mathsf{Enc}(k, h' + r)$. Again we can infer $h = h'$ from $l = l'$ since the same random value r is used in the second encryption.

2.4.1 Contributions

We design a type system for information flow for an imperative language that includes block ciphers and show its soundness under the assumption that the encryption scheme is a pseudo-random permutation. Our soundness proof is carried in the concrete (exact) security framework that aims at providing concrete estimates about the security of the considered system.

This is to our knowledge the first time that a type system for non-interference is proven correct in the concrete security framework. One can distinguish three security proof settings : first, the symbolic setting, also called formal and Dolev-Yao, where cryptographic primitives are operators on formal expressions (terms) and security proofs are reachability or observational equivalence proofs ; second, the computational setting where cryptographic primitives are algorithms and security proofs are asymptotic based on poly-time reductions ; third, the concrete security setting where proofs are also by reduction but no asymptotics are involved and reductions are as efficient as possible.

2.5 WP 5

This work package focuses on *Watermarking algorithms* [60].

Watermarking consists in embedding some information inside a document in a *robust* and usually *imperceptible* way. It is notably used in digital property claims : a content owner may mark a document before its release, in order to be able to recognize copies of it and claim ownership. Of course, this cannot be done in a reliable and convincing way unless properties of the watermarking scheme have been solidly established.

A wide literature is dedicated to techniques for marking sound, still images and videos [61] that are robust against common transformations such as scaling, resampling, cropping, etc. This is achieved by using meaningful notions of distance and performing watermarking in an adequate space, such as frequency spectrums. One may also consider documents as unstructured bit strings, and measure distortions using the Hamming distance, *i*.e., the number of positions where bits differ. In this setting, simple watermarking techniques may be used. The *bit-flipping* scheme involves a secret mask K which is a bitstring of the same length as the original document that is to be marked. Marking a document O (called *support* in this context) is done by changing the value of bits at positions set in K — we say a position is set when the corresponding bit is 1. This can be expressed as a bitwise xor : $\texttt{mark}(K, O) = O \oplus K$. The induced distortion is the number k of bits set in K ; this quantity is typically chosen to be a small fraction of the size of the support, and its value is assumed to be public. Detection of suspect copies O', written $\texttt{detect}(O, K, O')$, is done by counting the number of positions set in K where O and O' differ — in other words, the number of positions set in K where O' coincides with $\texttt{mark}(K, O)$. If that number exceeds a predefined threshold, the suspect document O' is claimed to be a copy of $\texttt{mark}(K, O)$. The idea behind this scheme is that an attacker who does not know K has little chance of changing enough bits for evading detection unless it also distorts the document so much that it renders it worthless. A variant of this technique is *substitution* watermarking where, instead of flipping bits at chosen positions in the original support, a secret message is used as a substitution for those bits. It is important to point out that although working with unstructured bitstrings is idealized, it has practical counterparts. For instance, the Agrawal, Haas and Kiernan's method [35] for watermarking numerical databases hides a mark by applying a substitution to least significant bits, and quality measurement is proportional to the Hamming distance.

Watermarked documents may be subject to attacks by malevolent users. A protocol is said to be *robust* against a particular attack if the mark can still be detected with high probability in copies modified by this attack. As observed above, it is useless to consider attacks that introduce excessive distortion : though such attacks are usually easy, they result in valueless data. Numerous attack scenarios can be envisioned, and it is impossible to test them all one by one. Nevertheless, we expect a protocol to be *provably robust*, *i*.e., robust against any attack. Provable security for watermarking is difficult to achieve, and there is currently little work in that direction. One reason for this is that watermarking protocols may be attacked at various levels, sometimes independently of the nature of documents and the marking algorithms. For example, in the context of digital property, someone may apply his own mark to marked data in order to claim ownership. To resolve such issues, authorities have to be introduced. To address such issues separately, Hopper, Molnar and Wagner [70] have introduced a distinction between strong watermarking schemes and non-removable embeddings. The former refers to the general protocol used to claim ownership, while the latter is the core technique for marking documents, consisting of a marking and a detection algorithms. Using cryptography and trusted third parties, they have formally proved (on paper) that strong watermarking protocols, robust against arbitrary attackers, can be derived from non-removable embeddings. However, they did not address the robustness of particular embedding techniques, which depends on the kind of document that is considered. More generally, there is surprisingly little literature on robustness against arbitrary attackers, even in the idealized setting of bitstring algorithms. The problem has been identified, and several authors have proposed definitions and methodologies [34, 72], but they did not provide proofs of their results even for simple schemes. To the best of our knowledge, the only such proof has been carried out in the context of graph watermarking [73]. This proof relies on complex assumptions, concerning in

particular the limited knowledge of the attacker, which may be hard to formalise. In contrast, our approach is very elementary, relying on a clear modelisation and basic combinatorics. We also note that, although a central piece in that proof of robustness for graphs is the (proved) robustness of a technique for marking integer vectors, this latter technique is not suitable for marking bitstrings under the Hamming distance, since it applies a modification to each component of the vector — which also eases considerably the robustness argument.

In Scalp, we make two important steps towards provably secure watermarking protocols. First, we present a new proof of robustness against arbitrary attackers, which applies to bit-flipping and substitution watermarking for bitstrings. The key idea here is an efficient reduction of robustness to secrecy of the key K, which is used to obtain a tight enough bound on the probability of successful attacks. Second, we have fully formalized these proofs using the Coq proof assistant and the Alea library for reasoning about probabilities and probabilistic programs. Therefore, we provide a solid methodology for further work in the area. The formalization of our work is also interesting in that it involves aspects such as knowledge and probabilistic algorithms which are challenging for the formal methods community. There are indeed few significant formalisations of randomised programs, although this field is receiving increasing attention [38, 69, 71]. Our work is most closely related to the CertiCrypt framework for formalizing proofs of cryptographic protocols. Those proofs proceed by reducing cryptographic games to simpler ones corresponding to mathematical assumptions on cryptographic primitives. As we shall see, our work on watermarking does not necessitate a model as complex as the one used in CertiCrypt. But the main difference lies in the nature of our proofs : while our argument also revolves around a reduction, it does not rely as heavily on program transformations but instead on concrete computations on bitstrings. The end result is also quite different : our proof is self-contained and does not rely on assumptions about complex mathematical primitives. Therefore, our Coq development provides an interesting case study for Alea, showing how elements of computational information theory may be carried out in that framework.

Chapitre 3

Events

3.0.1 Meetings

- 2007/09/18, ProVal-LRI, Orsay, kickoff meeting.
- 2008/01/07, DCS-Verimag, Grenoble, first official meeting. (Full session)
- 2008/04/03, EVEREST-Inria, Sophia-Antipolis. (Full session)
- 2008/06/13, Plume-LIP, Lyon. (Full session)
- 2009/02/13, Cnam.
- 2009/05/28+29, IMDEA, Madrid with participation of Imdea members. (Full session)
- 2009/06/12, ProVal-LRI.
- 2009/06/16, Cnam.
- 2009/09/10, Cnam.
- 2009/10/14, Orsay (Work meeting)
- 2009/11/27, ENS Lyon. (Full session)
- 2010/5/25+26, IMDEA, Madrid with participation of Imdea members. (Full session)
- 2010/12/9, Cnam.
- 2011/11/13, Grenoble.
- Weekly meetings between CNAM and Orsay members (Work meeting).
- Several visits of G. Barthe to Verimag and of Y. Lakhnech to IMDEA, Madrid.

3.0.2 Research dissemination

- Organisation of Computer-Aided Security, december 2011, Grenoble.
- Organisation of VETO, Workshop on Computer Science Security and Electronic Vote
- Organisation of MITACS, French-Canadian Workshop on Foundation and Practice of Security
- Presentation of Certicrypt at EJCP 2009 summer school and Summer School on Provable Security(Barcelona).
- invited paper : Essays in Honour of Willem-Paul de Roever.
- PLPV 2010 invited talk on CertiCrypt
- CoSyProofs course on CertiCrypt
- CoSyProofs presentations 2009 and 2010.
- Sylvain Heraud (Everest) visited AIST (Japan) for two months, working on machine check cryptographic proof.
- Foundations and Practice of Security 2008, 2009, 2010, 2011.

3.0.3 Defended Ph.D. theses

- Santiago Zanella Béguelin, december 10th, 2010.
- Marion Daubognard, december 12th 2011.

– Sylvain Heraud, march 12th 2012.

Chapitre 4

Publications

[1] Philippe Audebaud and Christine Paulin-Mohring. Proofs of randomized algorithms in Coq. *Science of Computer Programming*, 2009.

[2] Gilles Barthe, Marion Daubignard, Bruce M. Kapron, and Yassine Lakhnech. Computational indistinguishability logic. In *ACM Conference on Computer and Communications Security*, pages 375–386, 2010.

[3] Gilles Barthe, Marion Daubignard, Bruce M. Kapron, Yassine Lakhnech, and Vincent Laporte. On the equality of probabilistic terms. In *LPAR (Dakar)*, pages 46–63, 2010.

[4] Gilles Barthe, Mathilde Duclos, and Yassine Lakhnech. A computational indistinguishability logic for the bounded storage model. In *FPS*, pages 102–117, 2011.

[5] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella. Formal certification of elgamal encryption. In *fast08*. springer, 2009.

[6] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago 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. Springer, 2011.

[7] Gilles Barthe, Benjamin Grégoire, Romain Janvier, and Santiago Zanella Béguelin. A framework for language-based cryptographic proofs. In *ACM SIGPLAN Workshop on Mechanizing Metatheory*, 2007.

[8] Gilles Barthe, Benjamin Grégoire, Yassine Lakhnech, and Santiago 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. Springer, 2011.

[9] Gilles Barthe, Benjamin Grégoire, Frederico Olmedo, and Santiago Zanella. Formally certifying the security of digital signature schemes. In *IEEE Symposium on Security and Privacy*. IEEE, May 2009.

[10] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella. Formal certification of code-based cryptographic proofs. In *36th symposium Principles of Programming Languages*. ACM Press, January 2009.

[11] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella. Programming language techniques for cryptographic proofs. In *Interactive Theorem Proving, international Conference, ITP 2010, Edinburgh, Scotland, July 11-14, 2010, Proceedings*, Lecture Notes in Computer Science. Springer, 2010.

[12] Gilles Barthe and Boris Köpf. Information-theoretic bounds for differentially private mechanisms. In *CSF*, pages 191–204, 2011.

[13] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. In *POPL*, pages 97–110, 2012.

[14] Gilles Barthe, Federico Olmedo, and Santiago Zanella Béguelin. Verifiable security of boneh-franklin identity-based encryption. In *ProvSec*, pages 68–83, 2011.

[15] Santiago Zanella Béguelin, Gilles Barthe, Sylvain Heraud, Benjamin Grégoire, and Daniel Hedin. A machine-checked formalization of sigma-protocols. In *Computer Security Foundations Symposium, CSF 2010, Edinburgh, Scotland, July 17-19, 2010, Proceedings*. IEEE, 2010.

[16] Emmanuel Bresson, Yassine Lakhnech, Laurent Mazaré, and Bogdan Warinschi. *Formal Models and Techniques for Analyzing Security Protocols*, chapter Computationally Sound Analysis of Encrypting with Diffie Hellman Keys. IOS Press, 2011.

[17] Pierre Corbineau, Mathilde Duclos, and Yassine Lakhnech. Certified security proofs of cryptographic protocols in the computational model : An application to intrusion resilience. In *CPP*, pages 378–393, 2011.

[18] J. Courant, M. Daubignard, C. Ene, P. Lafourcade, and Y. Lakhnech. Automated proofs for asymmetric encryption. *Journal of Automated Reasoning*, 46 :261–291, 2011. 10.1007/s10817-010-9186-x.

[19] Judicaël Courant, Marion Daubignard, Cristian Ene, Pascal Lafourcade, and Yassine Lakhnech. Towards automated proofs for asymmetric encryption schemes in the random oracle model. In *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 371–380. ACM, 2008.

[20] Judicaël Courant, Marion Daubignard, Cristian Ene, Pascal Lafourcade, and Yassine Lakhnech. Automated proofs for asymmetric encryption. In *Concurrency, Compositionality, and Correctness*, pages 300–321, 2010.

[21] Judicaël Courant, Cristian Ene, and Yassine Lakhnech. Computationally sound typing for non-interference : The case of deterministic encryption. In *FSTTCS 2007 : Foundations of Software Technology and Theoretical Computer Science, 27th International Conference, New Delhi, India, December 12-14, 2007, Proceedings*, volume 4855 of *Lecture Notes in Computer Science*, pages 364–375, 2007.

[22] P. Courtieu, D. Baelde, C. Paulin-Mohring, X. Urbain, and D. Gross-Amblard. Formal Proofs of Robustness for Watermarking Algorithms. In *TYPES WOKSHOP 2011*, 2011.

[23] Cas J. F. Cremers, Pascal Lafourcade, and Philippe Nadeau. Comparing state spaces in automatic security protocol analysis. In *Formal to Practical Security*, pages 70–94, 2009.

[24] Dennis Dams, Ulrich Hannemann, and Martin Steffen, editors. *Concurrency, Compositionality, and Correctness, Essays in Honor of Willem-Paul de Roever*, volume 5930 of *Lecture Notes in Computer Science*. Springer, 2010.

[25] Marion Daubignard. *Formal Methods for Concrete Security Proofs*. PhD thesis, University of Grenoble, Verimag, France, 2011.

[26] Cristian Ene, Yassine Lakhnech, and Van Chan Ngo. Formal indistinguishability extended to the random oracle model. In *ESORICS*, pages 555–570, 2009.

[27] Laurent Fousse, Pascal Lafourcade, and Mohamed Alnuaimi. Benaloh's dense probabilistic encryption revisited. In *AFRICACRYPT*, pages 348–362, 2011.

[28] Martin Gagné, Pascal Lafourcade, Yassine Lakhnech, and Reihaneh Safavi-Naini. Automated security proof for symmetric encryption modes. In *ASIAN*, pages 39–53, 2009.

[29] Martin Gagné, Pascal Lafourcade, Yassine Lakhnech, and Reihaneh Safavi-Naini. Automated verification of block cipher modes of operation, an improved method. In *FPS*, pages 23–31, 2011.

[30] Pascal Lafourcade, Vanessa Terrade, and Sylvain Vigier. Comparison of cryptographic verification tools dealing with algebraic properties. In *Formal Aspects in Security and Trust*, pages 173–185, 2009.

[31] Christine Paulin-Mohring. *From Semantics and Computer Science : Essays in Honor of Gilles Kahn*, chapter A constructive denotational semantics for Kahn networks in Coq. Cambridge University Press, 2009.

[32] Jérémie Tharaud, Sven Wohlgemuth, Isao Echizen, Noboru Sonehara, Günter Müller, and Pascal Lafourcade. Privacy by data provenance with digital watermarking - a proof-of-concept implementation for medical services with electronic health records. In *IIH-MSP*, 2010.

Chapitre 5

Bibliographie

[33] *Formal Models and Techniques for Analyzing Security Protocols*. IOS Press, 2011.

[34] André Adelsbach, Stefan Katzenbeisser, and Ahmad-Reza Sadeghi. A computational model for watermark robustness. In *Proceedings of the 8th international conference on Information hiding*, IH'06, pages 145–160, Berlin, Heidelberg, 2007. Springer-Verlag.

[35] Rakesh Agrawal, Peter J. Haas, and Jerry Kiernan. Watermarking Relational Data : Framework, Algorithms and Analysis. 12(2) :157–169, 2003.

[36] Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors. *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*. ACM, 2010.

[37] Philippe Audebaud and Christine Paulin-Mohring. Proofs of randomized algorithms in Coq. In Tarmo Uustalu, editor, *Mathematics of Program Construction, MPC'2006*, volume 4014 of *LNCS*, Kuressaare, Estonia, 2006. Springer-Verlag.

[38] Philippe Audebaud and Christine Paulin-Mohring. Proofs of randomized algorithms in Coq. *Science of Computer Programming*, 2009.

[39] Gilles Barthe, Marion Daubignard, Bruce M. Kapron, and Yassine Lakhnech. Computational indistinguishability logic. In *ACM Conference on Computer and Communications Security*, pages 375–386, 2010.

[40] Gilles Barthe, Marion Daubignard, Bruce M. Kapron, Yassine Lakhnech, and Vincent Laporte. On the equality of probabilistic terms. In *LPAR (Dakar)*, pages 46–63, 2010.

[41] Gilles Barthe, Mathilde Duclos, and Yassine Lakhnech. A computational indistinguishability logic for the bounded storage model. In *FPS*, pages 102–117, 2011.

[42] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella. Formal certification of elgamal encryption. In *fast08*. springer, 2009.

[43] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago 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. Springer, 2011.

[44] Gilles Barthe, Benjamin Grégoire, Romain Janvier, and Santiago Zanella Béguelin. A framework for language-based cryptographic proofs. In *ACM SIGPLAN Workshop on Mechanizing Metatheory*, 2007.

[45] Gilles Barthe, Benjamin Grégoire, Yassine Lakhnech, and Santiago 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. Springer, 2011.

[46] Gilles Barthe, Benjamin Grégoire, Frederico Olmedo, and Santiago Zanella. Formally certifying the security of digital signature schemes. In *IEEE Symposium on Security and Privacy*. IEEE, May 2009.

[47] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella. Formal certification of code-based cryptographic proofs. In *36th symposium Principles of Programming Languages*. ACM Press, January 2009.

[48] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella. Programming language techniques for cryptographic proofs. In *Interactive Theorem Proving, international Conference, ITP 2010, Edinburgh, Scotland, July 11-14, 2010, Proceedings*, Lecture Notes in Computer Science. Springer, 2010.

[49] Gilles Barthe and Boris Köpf. Information-theoretic bounds for differentially private mechanisms. In *CSF*, pages 191–204, 2011.

[50] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. In *POPL*, pages 97–110, 2012.

[51] Gilles Barthe, Federico Olmedo, and Santiago Zanella Béguelin. Verifiable security of boneh-franklin identity-based encryption. In *ProvSec*, pages 68–83, 2011.

[52] Santiago Zanella Béguelin, Gilles Barthe, Sylvain Heraud, Benjamin Grégoire, and Daniel Hedin. A machine-checked formalization of sigma-protocols. In *Computer Security Foundations Symposium, CSF 2010, Edinburgh, Scotland, July 17-19, 2010, Proceedings*. IEEE, 2010.

[53] Emmanuel Bresson, Yassine Lakhnech, Laurent Mazaré, and Bogdan Warinschi. *Formal Models and Techniques for Analyzing Security Protocols*, chapter Computationally Sound Analysis of Encrypting with Diffie Hellman Keys. IOS Press, 2011.

[54] Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock. Errors in computational complexity proofs for protocols. In Roy [**?**], pages 624–643.

[55] Pierre Corbineau, Mathilde Duclos, and Yassine Lakhnech. Certified security proofs of cryptographic protocols in the computational model : An application to intrusion resilience. In *CPP*, pages 378–393, 2011.

[56] J. Courant, M. Daubignard, C. Ene, P. Lafourcade, and Y. Lakhnech. Automated proofs for asymmetric encryption. *Journal of Automated Reasoning*, 46 :261–291, 2011. 10.1007/s10817-010-9186-x.

[57] Judicaël Courant, Marion Daubignard, Cristian Ene, Pascal Lafourcade, and Yassine Lakhnech. Towards automated proofs for asymmetric encryption schemes in the random oracle model. In *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 371–380. ACM, 2008.

[58] Judicaël Courant, Marion Daubignard, Cristian Ene, Pascal Lafourcade, and Yassine Lakhnech. Automated proofs for asymmetric encryption. In *Concurrency, Compositionality, and Correctness*, pages 300–321, 2010.

[59] Judicaël Courant, Cristian Ene, and Yassine Lakhnech. Computationally sound typing for non-interference : The case of deterministic encryption. In *FSTTCS 2007 : Foundations of Software Technology and Theoretical Computer Science, 27th International Conference, New Delhi, India, December 12-14, 2007, Proceedings*, volume 4855 of *Lecture Notes in Computer Science*, pages 364–375, 2007.

[60] P. Courtieu, D. Baelde, C. Paulin-Mohring, X. Urbain, and D. Gross-Amblard. Formal Proofs of Robustness for Watermarking Algorithms. In *TYPES WOKSHOP 2011*, 2011.

[61] Ingemar J. Cox, Matthew L. Miller, and Jeffrey A. Bloom. *Digital Watermarking*. Morgan Kaufmann Publishers, Inc., San Francisco, 2001.

[62] Marion Daubignard. *Formal Methods for Concrete Security Proofs*. PhD thesis, University of Grenoble, Verimag, France, 2011.

[63] Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Commun. ACM*, 20(7) :504–513, 1977.

[64] Alexander W. Dent. Fundamental problems in provable security and cryptography. *Phil Trans R Soc A*, 364 :3215–3230, 2006.

[65] Martin Gagné, Pascal Lafourcade, Yassine Lakhnech, and Reihaneh Safavi-Naini. Automated security proof for symmetric encryption modes. In *ASIAN*, pages 39–53, 2009.

[66] Martin Gagné, Pascal Lafourcade, Yassine Lakhnech, and Reihaneh Safavi-Naini. Automated verification of block cipher modes of operation, an improved method. In *FPS*, pages 23–31, 2011.

[67] Joseph A. Goguen and José Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[68] Shai Halevi. A plausible approach to computer-aided cryptographic proofs. `http://theory.lcs.mit.edu/~shaih/pubs.html`, 2005.

[69] Osman Hasan and Sofiene Tahar. *Probabilistic Analysis Using Theorem Proving*. VDM Verlag, 2008.

[70] Nicholas Hopper, David Molnar, and David Wagner. From weak to strong watermarking. In *Proceedings of the 4th conference on Theory of cryptography*, TCC'07, pages 362–382, Berlin, Heidelberg, 2007. Springer-Verlag.

[71] Joe Hurd. *Formal Verification of Probabilistic Algorithms*. PhD thesis, University of Cambridge, 2002.

[72] Stefan Katzenbeisser. A computational model for digital watermarks. in Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2005), 2005.

[73] Sanjeev Khanna and Francis Zane. Watermarking maps : hiding information in structured data. In *SODA*, pages 596–605, 2000.

[74] Dexter Kozen. Semantics of probabilistic programs. *J. of Computer and System Science*, 1981.

[75] Peeter Laud. Handling encryption in an analysis for secure information flow. In *ESOP*, pages 159–173, 2003.

[76] Peeter Laud and Varmo Vene. A type system for computationally secure information flow. In Liskiewicz and Reischuk [**?**], pages 365–377.

[77] Maciej Liskiewicz and Rüdiger Reischuk, editors. *Fundamentals of Computation Theory, 15th International Symposium, FCT 2005, Lübeck, Germany, August 17-20, 2005, Proceedings*, volume 3623 of *Lecture Notes in Computer Science*. Springer, 2005.

[78] John Mitchell, editor. *Symposium on Principles of Programming Languages*, 2002.

[79] Norman Ramsey and Avi Pfeffer. Stochastic lambda calculus and monads of probability distributions. In Mitchell [**?**], pages 154–165.

[80] Bimal K. Roy, editor. *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, volume 3788 of *Lecture Notes in Computer Science*. Springer, 2005.

[81] A. Sabelfeld and A. Myers. Language-Based Information-Flow Security. *IEEE Journal on Selected Areas in Comunications*, 21 :5–19, January 2003.

[82] Andrei Sabelfeld and David Sands. Declassification : Dimensions and principles. *Journal of Computer Security*, 2007.

[83] Geoffrey Smith and Rafael Alpízar. Secure information flow with random assignment and encryption. In *FMSE*, pages 33–44, 2006.

[84] Jacques Stern, David Pointcheval, John Malone-Lee, and Nigel P. Smart. Flaws in applying proof methodologies to signature schemes. In Yung [**?**], pages 93–110.

[85] The Coq Development Team. *The Coq Proof Assistant Reference Manual – Version V8.1*, July 2006. `http://coq.inria.fr`.

[86] Dennis M. Volpano. Secure introduction of one-way functions. In *CSFW*, pages 246–254, 2000.

[87] Dennis M. Volpano, Cynthia E. Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2/3) :167–188, 1996.

[88] Moti Yung, editor. *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*. Springer, 2002.