Workshop on Formal and Computational Cryptography
FCC 2009

Program and Abstracts

July 11–12, 2009
Port Jefferson, New York, USA

Workshop Organization

Programme Chair

Ralf Küsters (University of Trier, Germany)

Programme Committee

Michael Backes (MPI and Saarland University, Germany)
Gilles Barth (IMDEA Software, Spain)
Bruno Blanchet (CNRS, ENS, INRIA, France)
Ran Canetti (Tel Aviv University, Israel)
Hubert Comon-Lundh (AIST Tokyo, Japan)
Anupam Datta (Carnegie Mellon University, USA)
Cédric Fournet (Microsoft Research, Cambridge, UK)

Local Organization

Scott Stoller (Stony Brook University, USA), General Chair of CSF 2009

Program

July 11, 2009

12:20–14:00 Lunch

14:00–14:05 Welcome

14:05–15:35 Session I

15:35–16:00 Break

16:00–18:00 Session II

July 12, 2009

9:00–10:30 Session III

10:30–11:00 Break

11:00–12:00 Session IV

12:00 Concluding remarks

Computational and Symbolic Anonymity in an Unbounded Network

Hubert Comon-Lundh
ENS Cachan and
AIST, Tokyo, Japan

Masami Hagiya
University of Tokyo, Japan

Yusuke Kawamoto
JSPS Research Fellow and
University of Tokyo, Japan

Hideki Sakurada
NTT Communication Science Laboratories,
Atsugi, Japan

There are two main approaches to the analysis of protocol security. The first considers an attacker modeled as a probabilistic polynomial-time interactive Turing machine (PPT) and a protocol is an unbounded number of copies of PPTs. The attacker is assumed to control the network and can schedule the communications and send fake messages. The security property is defined as an indistinguishability game: the protocol is secure if, for any attacker A, the probability that A gets an advantage in this game is negligible. A typical example is the *anonymity* property, by which an attacker should not be able to distinguish between two networks in one of which identities have been switched. The difficulty in such computational security notions lies in the problem of obtaining detailed proofs: they are in general unmanageable, and cannot be verified by automatic tools.

The second approach relies on a formal model: bitstrings are abstracted by formal expressions (terms), the attacker is any formal process, and security properties, such as anonymity, can be expressed by the observational equivalence of processes. This model is much simpler: there is no coin tossing, no complexity bounds, and the attacker is given only a fixed set of primitive operations (the function symbols in the term algebra). Therefore it is not surprising that security proofs become much simpler and can sometimes be automatized. However, the drawback is that we might miss some attack because the model might be too rough.

Starting with the seminal work of Abadi and Rogaway [2], there have been several results showing the *computational soundness* of the formal models: we do not miss any attacks when considering the abstract model, provided that the security primitives satisfy certain properties; for instance IND-CPA or IND-CCA in the case of encryption. Such results avoid the weaknesses of both approaches to security.

In their original work, Abadi and Rogaway only considered symmetric encryption and a passive attacker with some minor other restrictions. This has been extended in a number of directions. For instance, Backes et al. consider active attackers and several cryptographic primitives and show simulatability theorems, which imply the computational soundness of some formal model [5, 6]. There are also several other soundness results, typically for some trace properties [10, 8] and, more recently, for equivalence properties [7].

In the present work, we show the computational soundness of the observational equivalence in a large fragment of the applied pi-calculus for *public-key encryption* and *ring signatures*. We extend the previous work in the following respects:

1. In addition to security properties that can be checked on each trace, i.e., each individual sequence of events, we consider equivalence properties. The only previous results concerning equivalence properties in the presence of active attackers are [7], in which only symmetric encryption and a particular class of processes is considered, and [9], in which only a fixed number of protocol instances is considered. We do not assume here any bound on the number of protocol instances.

2. We consider an additional security primitive for which no soundness result was previously known for an unbounded number of sessions: *ring signatures*. This primitive is interesting as its main purpose is to provide with the anonymity of a signer. This is well-suited here since, at the protocol level, anonymity is expressed by an equivalence property.

3. Our soundness proof does not use a computable parsing function from bitstrings to terms. This is a major difference with all existing studies, in particular the studies on simulatability. With the primitives that are considered here, it would not be a big additional hypothesis to assume such a parsing function. However, conceptually, this yields different proofs, that open the way for other soundness results, for instance for `XOR`, while such proofs could not be conducted in a simulatability framework [4].

4. We do not restrict ourselves to simple processes, as in [7], but we consider a large fragment of the applied pi-calculus of [1]: we allow negative tests, arbitrary replications, private channels, and so on. The only important restriction with respect to the whole calculus is a determinacy assumption: every copy of a process has first to generate an id and disclose it. In this way, the attacker can schedule to which copy of a process a message is sent. Compared to [3], we have this additional restriction, but our interpretation gives more power to the computational attacker, who may schedule the communications. Hence this strengthens the soundness result. In [3], the authors also provide with a general framework for establishing computational soundness. We depart however from this line as, for instance, we wish to avoid a computable parsing function.

5. Finally, we observe that the ability of the computational attacker to observe the length of bitstrings cannot be soundly represented in a symbolic model. Up to our knowledge, this problem has never been considered in the papers on computational soundness. We propose here a solution, relying on a lazy leaking function in the symbolic model, assuming that honest agents check the length of plaintexts before encrypting, in the computational model.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, January 2001.

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

[3] M. Backes, D. Hofheinz, and D. Unruh. A general framework for computational soundness proofs - or - the computational soundness of the applied pi-calculus. Cryptology ePrint Archive, Report 2009/080, 2009. `http://eprint.iacr.org/`.

[4] M. Backes and B. Pfitzmann. Limits of the cryptographic realization of Dolev-Yao style XOR. In *Proc. 10th European Symposium on Research in Computer Security (ESORICS)*, 2005.

[5] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proc. 10th ACM Concerence on Computer and Communications Security (CCS'03)*, pages 220–230, 2003.

[6] M. Backes, B. Pfitzmann, and M. Waidner. The reactive simulatability (rsim) framework for asynchronous systems. *Information and Computation*, 205(12), 2007.

[7] H. Comon-Lundh and V. Cortier. Computational indistinguishability and observational equivalence. In *Proc. 15th ACM Conference on Computer and Communications Security (CCS'08)*, pages 109–118, 2008.

[8] V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proc. 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 157–171, 2005.

[9] Y. Kawamoto, H. Sakurada, and M. Hagiya. Computationally sound symbolic anonymity of a ring signature. In *Proc. FCS-ARSPA-WITS'08*, pages 161–175, 2008.

[10] D. Micciancio and B. Warinschi. Soundness of formal encryption in presence of an active attacker. In *Proc. Theory of Cryptography Conference (TCC'04)*, volume 2951 of *LNCS*, 2004.

Formal Indistinguishability extended to the Random Oracle Model

Cristian Ene, Yassine Lakhnech and Van Chan Ngo *

Université Grenoble 1, CNRS,Verimag

There are two main frameworks for analyzing cryptographic systems; the *symbolic framework*, originating from the work of Dolev and Yao, and the *computational approach*, growing out of the work of Goldwasser and Micali. A significant amount of effort has been made in order to link both approaches and profit from the advantages of each of them. Indeed, while the symbolic approach is more amenable to automated proof methods, the computation approach can be more realistic. Abadi and Rogaway were the first to prove a formal link between these two models. More precisely, they introduced an equivalence relation on terms and prove that equivalent terms correspond to indistinguishable distributions ensembles, when interpreted in the computational model. The work of Abadi and Rogaway has been extended to active adversaries and various cryptographic primitives.

Related works. A recently emerging branch of relating symbolic and computational models for passive adversaries is based on *static equivalence* from π-calculus, induced by an *equational theory*. Equational theories provide a framework to specify algebraic properties of the underlying signature, and hence, symbolic computations in a similar way as for abstract data types. That is, for a fixed equational theory, a term describes a computation in the symbolic model. Thus, an adversary can distinguish two frames (which roughly speaking are tuples of terms), if he is able to come up with two computations that yield the same result when applied to one frame, but different results when applied to the other frame. Such a pair of terms is called a *test*. Thus, a *static equivalence* relation is fully determined by the underlying equational theory, as two frames are *statically equivalent*, if there is no test that separates them. Baudet, Cortier and Kremer studied soundness and faithfulness of static equivalence for general equational theories and used their framework to prove soundness of exclusive or as well as certain symmetric encryptions. Abadi et al. used static equivalence to analyze guessing attacks.

Recently, Bana, Mohassel and Stegers argued that even though static equivalence works well to obtain soundness results for the equational theories mentioned above, it does not work well in other important cases. Consider for instance the Decisional Diffie Hellman assumption (DDH for short) that states that the tuples (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c), where a, b, c are randomly sampled, are indistinguishable. It does not seem to be obvious to come up with an equational theory for group exponentiation such that the induced static equivalence includes this pair of tuples without including others whose computational indistinguishability is not proved to be a consequence of the DDH assumption. The static equivalence induced by the standard equational theory for group exponentiation includes the pair (g, g^a, g^b, g^{a^2b}) and (g, g^a, g^b, g^c). It is unknown whether the computational indistinguishability of these two distributions can be proved under the DDH assumption. Therefore, Bana et al. propose an alternative approach to build symbolic indistinguishability relations: a *formal indistinguishability relation (FIR)* is defined as a closure of an initial set of equivalent frames with respect to simple operations

* Grenoble, email:name@imag.fr This work has been partially supported by the ANR projects SCALP, AVOTE and SFINCS

which correspond to steps in proofs by reduction. This leads to a flexible symbolic equivalence relation. In order to prove soundness of a FIR it is enough to prove soundness of the initial set of equivalences. Moreover, static equivalence is one instance of a FIR.

Contributions. In this paper, we extend Bana et al.'s approach by introducing a notion of symbolic equivalence that allows us to prove computational security of encryption schemes symbolically. More specifically, we would like to be able to treat generic encryption schemes that transform one-way functions to IND-CPA secure encryption schemes. Therefore, three problems need to be solved. First, we need to cope with one-way functions. This is another example where static equivalence does not seem to be appropriate. It does not seem easy to come with a set of equations that capture the one-wayness of a function. Indeed, consider the term $f(a|b)$, where f is a one-way function and $|$ is bit-string concatenation. We know that we cannot easily compute $a|b$ given $f(a|b)$ for uniformly sampled a and b. However, nothing prevents us from being able to compute a for instance. Introducing equations that allow us to compute a from $f(a|b)$, e.g., $g(f(a|b)) = a$, may exclude some one-way functions and does not solve the problem. Nothing prevents us from computing a prefix of b, a prefix of the prefix, etc .... The second problem that needs to be solved is related to the fact that almost all practical provably secure encryption schemes are analyzed in the random oracle model. The random oracle model is an idealized model in which hash functions are randomly sampled functions. In this model, adversaries have oracle access to these functions. An important property is that if an adversary is unable to compute the value of an expression a and if $H(a)$ has not been leaked behore, then $H(a)$ looks like a uniformly sampled value. Thus, we need to be able to symbolically prove that a value of a given expression a cannot be computed by any adversary. This is sometimes called *weak secrecy* in contrast to indistinguishability based secrecy. To cope with this problem, our notion of symbolic indistinguishability comes along with a *non-derivability* symbolic relation. Thus in our approach, we start from an initial pair of a non-derivability relation and a frame equivalence relation. Then, we provide rules that define a closure of this pair of relations in the spirit of Bana et al.'s work. Also in our case, we prove that computational soundness of the obtained relations can be checked by checking computational soundness of the initial relations. The third problem is related to the fact that security notions for encryption schemes such IND-CPA and real-or-random indistinguishability of cipher-text under chosen plaintext involve *active* adversaries. Indeed, these security definitions correspond to two-phase games, where the adversary first computes a value, then a challenge is produced, then the adversary tries to solve the challenge. Static equivalence and FIR (as defined in by Bana et all.) consider only passive adversaries. To solve this problem we consider frames that include variables that correspond to adversaries. As frames are finite terms, we only have finitely many such variables. This is the reason why we only have a degenerate form of active adversaries which is enough to treat security of encryption schemes and digital signature, for instance. The advantage of defining non-deductibility as we did, and of considering FIR (instead of static equivalence) as the good abstraction of indistinguishability, is that first, we capture "just" what is supposed to be true in the computational setting, and second, if we add more equations to our abstract algebra in a coherent manner with respect to the initial computational assumptions, then our proofs still remain computationally sound. Moreover, the closure rules we propose in our framework are designed with the objective of minimizing the initial relations which depend on the underlying cryptographic primitives and assumptions. We illustrate our framework by proving IND-CPA security of the constructions of Bellare-Rogaway, Hash El Gamal and an encryption scheme proposed by Pointcheval.

Computationally Sound Analysis of Probabilistic Security Protocols with Digitial Signatures

Mihhail Aizatulin Henning Schnoor Thomas Wilke
Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany
{aizatulin|schnoor|wilke}@ti.informatik.uni-kiel.de

June 12, 2009

We propose a framework for the analysis of probabilistic security protocols using digital signatures as main primitive. The key ingredients of this framework are

— a way to specify such protocols,

— a (Dolev-Yao based) symbolic as well as a (Bellare-Rogaway based) computational model for these protocols,

— a version of probabilistic alternating-time temporal logic, denoted pATL*,

— an interpretation of pATL* in both the symbolic and the computational models of the protocols, and, finally,

— a theorem stating that truth of a given formula for is preserved when passing from the symbolic to the computational setting. This theorem explicitely allows to transfer strategies from the symbolic to the computational setting, with obtaining the same success probabilities.

We use our framework to analyze probabilistic contract signing protocols, such as the protocol by Ben-Or, Goldreich, Micali, and Rivest (BGMR protocol, see [BOGMR90]), which was designed to satisfy a probabilistic fairness property, and a new protocol which was designed to satisfy a very strong probabilistic balance property [Aiz08]. We give the first rigorous treatment of the BGMR protocol in a computational setting, see also [NS03].

To illustrate the logic, the formula

$$\langle\langle \mathscr{A}, T, B, X, \mathscr{S}\rangle\rangle \Diamond \left(\langle\langle \mathscr{A}, \mathscr{S}, B\rangle\rangle^{\geq p_a} \Diamond \varphi_{\mathsf{abr}} \wedge \langle\langle \mathscr{A}, \mathscr{S}, B\rangle\rangle^{\geq p_r} \Diamond \varphi_{\mathsf{res}}\right).$$

is the one that we use to formalize our probabilistic notion of unbalance and can be read as follows: there is a reachable state in which the adversary (who also controls scheduling and communication buffers) has an abort strategy with success probability at least p_a and a resolve strategy with probability at least p_r.

References

[Aiz08] Mihhail Aizatulin. A timely and balanced optimistic contract-signing protocol. Diploma thesis, University of Kiel, March 2008.

[BOGMR90] Michael Ben-Or, Oded Goldreich, Silvio Micali, and Ronald L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.

[NS03] Gethin Norman and Vitaly Shmatikov. Analysis of probabilistic contract signing. pages 81–96, 2003.

CoSP: A general framework for computational soundness proofs*

Michael Backes[1,3], Dennis Hofheinz[2], and Dominique Unruh[1]
[1]Saarland University [2]CWI [3]MPI-SWS

Overview. We describe CoSP, a general framework for conducting Computational Soundness Proofs in symbolic models. CoSP enables formulating soundness results in a unified and comparable manner and for applying the computational soundness proofs to various formal calculi. CoSP comprises a general definition of symbolic protocols, their symbolic and computational execution, as well as a definition of computational soundness for trace properties.

CoSP permits arbitrary equational theories and computational implementations. It is specifically designed for establishing soundness results in that it abstracts away from many details that are not crucial for proving computational soundness, such as message scheduling, corruption models, and even the internal structure of a protocol. Instead, we treat the whole protocol as a single entity that interacts with an attacker. This allows for a unified treatment of different symbolic models by embedding them into CoSP.

CoSP enables proving computational soundness results in a conceptually modular and generic way: every computational soundness proof phrased in CoSP automatically holds for all embedded calculi, and the process of embedding formal calculi is conceptually decoupled from computational soundness proofs. This is crucial since these two tasks are often tackled using different techniques and pursued by people with different backgrounds in computer science. Asserting computational soundness of x cryptographic primitives within y calculi requires only $x + y$ proofs in CoSP. Without CoSP, proving every combination of calculus and implementation secure would require a total of $x \cdot y$ proofs.

We show how to use CoSP to establish the first computational soundness result for a full-fledged applied π-calculus under active attacks; in particular, we allow arbitrary equational theories. Our result hence yields computational soundness guarantees for the protocol verifier ProVerif.

To lay the foundation, we show computational soundness for public-key encryption and digital signatures in CoSP. This shows the computational soundness of these primitives in the applied π-calculus and all other calculi that can be embedded in CoSP.

The general framework (simplified). In the CoSP framework, we distinguish several main entities. First, one has to specify a *symbolic model* which describes the equational theory and the capabilities of the symbolic adversary. More precisely, the symbolic model consists of a set $\mathbf{C}$ of constructors, a set $\mathbf{N}$ of nonces, a set $\mathbf{D}$ of destructors (partial functions that operate on terms over constructors and nonces), and a deduction relation $\vdash$ (describing the capabilities of the adversary).

Second, one has to specify a *computational implementation* A of the symbolic model. Such a computational implementation A consists of a (partial) function A_f on bitstrings for each constructor or destructor f and of a probability distribution for nonces.

The task of our framework is to give a general definition of what it means that an implementation is a computationally sound implementation of a symbolic model. Loosely speaking, computational soundness means that if a protocol is secure symbolically, it stays secure if we implement the constructors and destructors in the protocol using the implementation A. In order to formalize this, we need a formal notion of a

*A long version of this abstract is available on IACR ePrint 2009/080. Partially funded by the Cluster of Excellence "Multimodal Computing and Interaction"

protocol. This notion should be a general as possible (in order not to exclude any protocols), but structurally as simple as possible (to facilitate proofs).

A *CoSP protocol* is represented as an infinite tree. Each node of the tree represents a possible protocol action. We distinguish nodes for communicating with the adversary, for applying constructors and destructors to terms received or constructed earlier, for using nonces, and for non-deterministic choices. Some nodes may have more than one successor, e.g., since a destructor may fail or succeed, a destructor node has two successors representing these alternatives. This allows us to implement arbitrary branching in the protocol.

In a symbolic execution of a CoSP protocol, we start with the root node of the protocol. In each step, we execute the action specified by the current node and proceed to the appropriate successor (which depends, e.g., on whether the destructor succeeded or not). When an output node is reached, we expect a term from the adversary. The adversary is only allowed to send terms that can be deduced (using $\vdash$) from terms sent earlier by the protocol. The sequence of nodes that are traversed in a protocol run is called a *symbolic trace*.

Similarly, we have a computational execution of a CoSP protocol. Here, instead of constructing, sending, and receiving terms, we operate on bitstrings. For example, in a constructor node with constructor C, instead of applying the constructor C to terms computed earlier, a bitstring is computed by applying the function A_f to bitstrings computed earlier. Besides this, the execution is analogous to the symbolic one. The sequence of nodes that are traversed is called a *computational trace*.

Definition 1 (Computational Soundness) *We say A is a computationally sound implementation of a symbolic model, if the following holds for any trace property $\wp$: If symbolic trace always satisfies $\wp$, then with overwhelming probability, the computational trace satisfies $\wp$.*

Embedding the applied π-calculus. So far, we have defined computational soundness with respect to CoSP protocols. However, for encoding a real-life protocol, infinite trees are cumbersome. Instead, we want that computational soundness as in Definition 1 implies computational soundness in some practically usable calculus. We demonstrate that this is feasible by embedding the applied π-calculus with destructors and events into our framework and showing that computational soundness with respect to some symbolic model implies computational soundness in the applied π-calculus. The proof proceeds in four steps.

1. Since the applied π-calculus only comes with semantics for a symbolic execution, we have to first define a computational semantics of a process P. This is done by specifying a machine Exec_P that runs the process P (using bitstrings instead of terms) and that uses the computational implementation for evaluating constructors and destructors. Exec_P interacts with a computational adversary for getting inputs and outputs, and the adversary also controls the scheduling.
2. We define a machine SExec_P that behaves exactly like Exec_P, but uses terms instead of bitstrings and interacts with a symbolic adversary. In a sense, SExec_P specifies an alternative symbolic semantics of the process P. SExec_P communicates with a symbolic adversary.
3. We show that any trace property that holds in the original semantics of the applied π-calculus also holds in an execution of SExec_P. This argument is purely symbolic and does not involve any computational semantics.
4. Since Exec_P and SExec_P are defined analogously, there is a single CoSP protocol Π_P such that the symbolic execution of Π_P is equivalent to an execution of SExec_P, and the computational execution of Π_P is equivalent to an execution of Exec_P. Since we assume computational soundness for $\mathbf{M}$ and A, this implies that any trace property that holds for an execution of SExec_P also holds for Exec_P.

Summarizing, we have that any trace property that holds for a process P also holds for its computational execution Exec_P. Note that the proof is independent of the actual symbolic model $\mathbf{M}$ and the computational implementation A, i.e., as soon as we have a computational soundness result in CoSP for $\mathbf{M}$ and A, we automatically get the corresponding result in the applied π-calculus.

The Execution Correspondence Theorem for Polynomially Accurate Simulations

Roberto Segala and Andrea Turrini
Dipartimento di Informatica
Università di Verona - Italy

Two important approaches to the verification of security protocols are known under the general names of *symbolic* and *computational*, respectively. In the symbolic approach messages are terms of an algebra and the cryptographic primitives are ideally secure; in the computational approach messages are bit-strings and the cryptographic primitives are secure with overwhelming probability. This means, for example, that in the symbolic approach only who knows the decryption key can decrypt a ciphertext, while in the computational approach the probability to decrypt a ciphertext without knowing the decryption key is negligible. An important question is whether the symbolic approach is sound with respect to the computational approach. The seminal work of Abadi and Rogaway [1] solves the problem in the context of passive adversaries while the work of Warinschi *et al.* [2] deals with active adversaries as well. The core result of Warinschi, which is widely used in the literature, is the *Mapping Lemma*: with overwhelming probability there is a correspondence between computations of the computational model and computations of the symbolic model.

In the context of distributed systems verification is carried out by using typical models of concurrency theory, including notions of (bi)simulation relations that consist of establishing relations between the states of two automata and verifying that an appropriate step condition holds: each transition of the simulated system can be matched by the simulating system up to the given relation. This approach enables modular and hierarchical analysis; this means that we can split a complex system into several subsystems that we can study independently from each other, deriving the properties of the overall system from the properties of each subsystem. Furthermore, the hierarchical verification allows us to build several intermediate refinements between specification and implementation. Often hierarchical and modular verification is simpler and cleaner than direct one-step verification, since each refinement may focus on specific homogeneous aspects of an implementation. One important result of the (bi)simulation relations is the *execution correspondence theorem* that shows close correspondences between computations of automata that are simulation-related. This means that given two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $\mathcal{A}_1$ is simulated by $\mathcal{A}_2$, for each execution ε_1 of $\mathcal{A}_1$ there exists an execution ε_2 of $\mathcal{A}_2$ that is related to ε_1 and each property of $\mathcal{A}_1$ is also a property of $\mathcal{A}_2$. This means, for example, that for each event E that occurs in an execution ε_1 of $\mathcal{A}_1$, E occurs also in an ε_1-related execution ε_2 of $\mathcal{A}_2$. Moreover, the sequence of actions performed by $\mathcal{A}_2$ in ε_2 to reach a state where E occurs is the same of the sequence performed

by $\mathcal{A}_1$.

In previous work [3] we have extended simulation relations to a model that includes probabilities and computational assumptions, imposing that a simulation exists up to a negligible error. We have also extended the execution correspondence theorem to the new framework and tested the approach on a simple authentication protocol. Technically, for the simulation we impose that given two families of automata $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ and $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$ parameterized on (a security parameter) k and two measures μ_1 and μ_2 that are related up to an error γ, if μ_1 is reachable within a polynomial number of steps in $\mathcal{A}_k^1$ and there is a transition that leaves from μ_1 reaching η_1, then there exists a transition that leaves from μ_2 reaching some η_2 such that η_1 and η_2 are related up to an error $\gamma + \nu(k)$ where ν is a negligible function; for the execution correspondence theorem, we obtain that given two families of automata $\{\mathcal{A}_k^1\}_{k\in\mathbb{N}}$ and $\{\mathcal{A}_k^2\}_{k\in\mathbb{N}}$ parameterized on k and a measure μ_1 reached within a polynomial number of steps in $\mathcal{A}_k^1$, there exists a measure μ_2 that is reached within the same number of steps in $\mathcal{A}_k^2$ such that μ_1 and μ_2 are related up to a negligible error. This result is easily extended to each fixed number of families of automata, so it can be used in a hierarchical analysis to relate the executions of the two ends of the chain of simulations that relate the several levels of refinements.

In this talk we show how the extended execution correspondence theorem can be seen as a generalization of the mapping lemma, thus enabling modular and hierarchical analysis of soundness. More precisely, we define a concrete automaton that models a cryptographic protocol following the computational approach and an ideal automaton that represents the protocol following the symbolic approach. Using the hierarchical and modular approach, we show that there exists a chain of simulations that relates the two automata and thus, by the execution correspondence theorem, it follows that each probability measure that is reachable within a polynomial number of steps in the concrete automaton is related up to a negligible error to a probability measure reachable in the ideal automaton. In other words, this means that with overwhelming probability there is a correspondence between executions of the concrete automaton and executions of the ideal automaton, that is with overwhelming probability there is a correspondence between executions of the computational model and computations of the symbolic model.

References

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

[2] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *TCC'04*, v. 2951 of *LNCS*, pp. 133–151, 2004.

[3] R. Segala and A. Turrini. Approximated computationally bounded simulation relations for probabilistic automata. In *CSF'07*, pp. 140–154, 2007.

From CryptoVerif Specifications to Computationally Secure Implementations of Protocols

(Work in Progress)

David Cadé
École Normale Supérieure, CNRS, INRIA

CryptoVerif [Bla08] is a protocol verifier in the computational model that can automatically prove properties of protocols, such as secrecy and authentication. It generates proofs by sequences of games, like those written manually by cryptographers. The first game describes the specification of the protocol to prove; the next games are deduced by relying on security assumptions on primitives or by syntactic transformations. These transformations are such that consecutive games are indistinguishable and, in the last game, the desired security property is obvious. So the first game also satisfies the property. The games are formalized in a probabilistic polynomial-time process calculus.

We are implementing a compiler that takes a CryptoVerif specification and generates an implementation of the protocol in OCaml. The goal of this work is to obtain implementations of security protocols proved secure in the computational model. We will prove that our compiler is correct, that is, the semantics of the generated code corresponds to the semantics of the specification. Therefore, if CryptoVerif can prove a property on the protocol, then the implementation will also satisfy this property.

The CryptoVerif specification is annotated by the user with hints about the implementation details: the user indicates how the specification is split into modules (*e.g.*, the key generator part, the client part, and the server part), the files that will store data shared between several modules, the name of the OCaml functions corresponding to the cryptographic primitives, and the size of the random numbers. The compiler generates code for each module. For each step of the protocol, inputting a message and outputting the response, the compiler generates a function that takes the received message as argument and returns the response. These functions can then be called by a manually implemented network layer, which can be considered as part of the adversary, so we need not make any security assumptions on it. On the other hand, the CryptoVerif specification makes security assumptions on the cryptographic primitives. We do not verify that the OCaml implementation of these primitives satisfies these assumptions: they still need to be proved manually, but the CryptoVerif speci-

fication states precisely what has to be proved.

A related approach is the work of Bhargavan et al. about the verification of protocols written in ML [BCF07, BCFZ09] and its application to TLS [BCFZ08]. They generate a CryptoVerif specification from a ML implementation and then verify it with CryptoVerif. In contrast to their work, we generate a ML implementation from a CryptoVerif specification. Their use of ML as input language, which is more flexible than the CryptoVerif specification language and also better known, is an advantage of their approach. However they are still limited by the expressive power of CryptoVerif: one sometimes has to tweak the specification in order to prove the desired security properties. Our approach makes it easier to write a specification well-suited for CryptoVerif. Our approach also favors the methodology of first writing the formal specification, proving it and second implementing it. Our compiler facilitates this second step. Finally, we separate clearly what part of the implementation to prove manually (the primitives), what is proved automatically by CryptoVerif (the protocol), and what does not need any proof (the network code). This manual separation simplifies the verification process.

Acknowledgments

We would like to thank Bruno Blanchet for his help on this work. This work was partly supported by the ANR project FormaCrypt and by DGA.

References

[BCF07] Karthikeyan Bhargavan, Ricardo Corin, and Cédric Fournet. Cryptoverifying protocol implementations in ML. In *Workshop on Formal and Computational Cryptography (FCC'07)*, Venice, Italy, July 2007.

[BCFZ08] Karthikeyan Bhargavan, Ricardo Corin, Cédric Fournet, and Eugen Zălinescu. Cryptographically verified implementations for TLS. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*, pages 459–468. ACM, October 2008.

[BCFZ09] Karthikeyan Bhargavan, Ricardo Corin, Cédric Fournet, and Eugen Zălinescu. Automated computational verification for cryptographic protocol implementations. Unpublished draft available at `http://www.msr-inria.inria.fr/projects/sec/fs2cv/`, 2009.

[Bla08] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Transactions on Dependable and Secure Computing*, 5(4):193–207, October–December 2008.

Computationally Sound Proofs of Security for a Key Management API

EXTENDED ABSTRACT

Graham Steel

LSV, INRIA & CNRS & ENS-Cachan

1 Security APIs for Tamper Resistant Devices

Security solutions for information systems are increasingly making use of tamper-resistant cryptographic devices, whether they are smartcards carried by commuters on a mass transit system, or high-throughput Hardware Security Modules in a bank ATM transaction processing processing facility. Over the last few years we have been analysing the key management APIs of such tamper-resistant devices. The design of the API is critical for the security of the system, since it is the API which controls information flow between the data on the tamper resistant device and the hostile outside world. We have shown a number of standard APIs to be susceptible to attacks whereby sensitive keys are revealed in clear [1, 3, 4]. More recently, we have proposed our own generic API and proved security properties for it [2]. We have also proven security properties for a variant of the popular PKCS#11 standard API, albeit with weaker properties than our own design [4]. However, all these proofs are in the symbolic model. Naturally we would like to extend our results to a suitable 'computational' model. We discuss this ongoing work in this extended abstract.

2 Computational Soundness for APIs

While there is only one Dolev-Yao adversary[1], there are many flavours of 'computational' adversary models. Which is the most suitable for API analysis? It is instructive to look at attacks that have been discovered on existing APIs. For example, in our work on the Eracom variant of PKCS#11, we discovered an attack starting from a scenario in which the intruder learns, by some out-of-model means, a session key (i.e. a key for encrypting and decrypting data). By means of some legitimate API calls with carefully chosen parameters, the attacker is able to turn this session key into a key-encrypting key, and then use it to discover the value of all the keys stored in the device. This kind of attack would seems to suggest a computational model capturing adaptive corruption of keys, i.e. corruption during protocol execution. There is existing work in translating results in a Dolev-Yao model to adaptive corruption models [5]. Panjwani has shown that for a Dolev-Yao style trace containing just symmetric keys encrypted under other symmetric keys, with no key cycles, security against adaptive corruptions can be inferred from the Dolev-Yao result. More precisely, the intruder is given an 'oops' command that models the revelation of a key. He can use this any number of times during his interaction with

[1] more or less..

the API. At the end of his interaction, he will have a certain set of keys in his knowledge set, by the usual Dolev-Yao style rules for symbolic encryption and decryption. Some other keys may remain secret. Panjwani's result shows that these keys remain secret (i.e. random) in an equivalent computational adaptive corruption game, with the usual assumptions (semantic security of the encryption scheme, no key cycles). Note that there are some extra restrictions, however. The intruder may not ask for decryptions under keys he doesn't already know, something which most APIs allow in one form or another. He can't mount other 'active' attacks either, where he e.g. creates fake messages to send to the API.

How close are our symbolic proofs to the domain of this result? We consider first the Eracom-based PKCS#11 variant and its symbolic security proof [4, Fig. 6] (this is the patched version, where the corruption-based attack mentioned above is no longer possible). Most of the output consists of keys encrypted with other keys, but we have also encryption and decryption of arbitrary plaintexts by session keys, and we have hashing used in HMACs to add integrity of key attributes to the key wrapping/unwrapping process. This is important as it prevents an attacker from e.g. turning a session key into a wrapping key, which will lead to as attacks. The Panjwani results cover only symmetric key encryption. Additionally, it is trivial to introduce key cycles in the API as it is currently specified.

We have also proven security properties for a Generic API for implementing key management protocols such as those from the Clark-Jacob corpus. This API effectively describes an encryption scheme that gives us the properties that we want, in that it enforces tagging of elements with their type (session key, nonce, public data) and their intended recipient (which enables us to prove novel properties about secrecy of keys shared between honest users even when the protocol is run between corrupted host machines). This API does not allow long-term keys to be updated, but it does prevent key cycles.

This talk will discuss possible ways to being these results closer together.

References

[1] V. Cortier, G. Keighren, and G. Steel. Automatic analysis of the security of XOR-based key management schemes. In O. Grumberg and M. Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, number 4424 in LNCS, pages 538–552, 2007.

[2] V. Cortier and G. Steel. Synthesising secure APIs. Technical Report RR-6882, INRIA, 2009. 27 pages.

[3] S. Delaune, S. Kremer, and G. Steel. Formal analysis of PKCS#11. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF'08)*, pages 331–344, Pittsburgh, PA, USA, June 2008. IEEE Computer Society Press.

[4] S. Fröschle and G. Steel. Analysing PKCS#11 key management APIs with unbounded fresh data. In *To appear in Proceedings of ARSPA-WITS'09*, 2009.

[5] Saurabh Panjwani. Tackling adaptive corruptions in multicast encryption protocols. In *Theory of Cryptography Conference*, 2007.

Refining Computationally Sound Mechanized Proofs for Kerberos

B. Blanchet* A. D. Jaggard† J. Rao‡ A. Scedrov§ J.-K. Tsay¶

Kerberos is designed to allow a user to repeatedly authenticate herself to multiple servers based on a single login. The PKINIT extension to Kerberos modifies the initial round of the protocol to use a PKI instead of long-term shared keys (*e.g.*, password-derived keys). Especially with PKINIT, Kerberos uses a rich collection of cryptographic operations and constructs, and Kerberos, both with and without the PKINIT extension, is used in real world settings (including Microsoft Windows). Kerberos is thus a great test case for protocol-analysis tools. The CryptoVerif prover works directly in the computational model to prove properties of protocols that are formalized as games.

This talk will both survey some of our earlier work using CryptoVerif to analyze Kerberos, with and without PKINIT, and describe two recent extensions of this work. First, we briefly survey our work [1] to formalize all three rounds of Kerberos (with and without PKINIT) as games that CryptoVerif could analyze. This allowed us to prove, using CryptoVerif, authentication and secrecy properties under certain cryptographic assumptions (*e.g.*, that the public-key encryption scheme satisfies IND-CCA2 security). This work included the definition of a version of key usability that was stronger than that originally given by Datta *et al.* [2]; the stronger version is amenable to being proved using CryptoVerif, and we showed that freshly generated keys in Kerberos are usable in this strong sense for IND-CCA2-secure encryption.

Second, we describe more recent results that extend our initial work on key usability. We suggest the following definition of strong key usability for INT-CTXT-secure encryption; like our strong notion of IND-CCA2 usability, this definition can be captured in the language used by CryptoVerif.

Definition 1 (Strong INT-CTXT Key Usability)**.** *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme ($\mathcal{K}$ is the key generation algorithm, $\mathcal{E}$ the encryption algorithm, and $\mathcal{D}$ the decryption algorithm), Σ be a key exchange protocol, and $\mathcal{A}$ be an adversary. We consider an experiment $\mathbf{Exp}^*_{\mathcal{A},\Sigma,\Pi}(\eta)$ and define the advantage of an adversary $\mathcal{A}$ by $\mathbf{Adv}^{*ke}_{\mathcal{A},\Sigma,\Pi} = \Pr[\mathbf{Exp}^*_{\mathcal{A},\Sigma,\Pi}(\eta) = 1]$. Let S be a set of symmetric encryption schemes. We say that keys exchanged through protocol Σ are* strongly INT-CTXT usable *for schemes in S if, for all $\Pi \in S$ and any probabilistic, polynomial-time adversary $\mathcal{A}$, the advantage $\mathbf{Adv}^{*ke}_{\mathcal{A},\Sigma,\Pi}$ is negligible.*

The experiment $\mathbf{Exp}^*_{\mathcal{A},\Sigma,\Pi}(\eta)$ proceeds as follows:

- *First, $\mathcal{A}$ is given the security parameter η and $\mathcal{A}$ can interact, as an active adversary, with polynomially many protocol sessions of Σ.*
- *At some point, at the request of $\mathcal{A}$, a session identifier* sid *is drawn at random and $\mathcal{A}$ is given access to a encryption oracle $\mathcal{E}_k(.)$ and a decryption oracle $\mathcal{D}_k(.)$, both keyed with a key k locally output in session* sid*.*
- *Adversary $\mathcal{A}$ plays a variant of an INT-CTXT game in which:*
  - *$\mathcal{A}$ may submit messages m to $\mathcal{E}_k(.)$, which returns $\mathcal{E}_k(m)$;*
  - *$\mathcal{A}$ never queries $\mathcal{D}_k(.)$ on a cyphertext output by $\mathcal{E}_k(.)$;*
  - *$\mathcal{A}$ may interact with uncompleted protocol sessions; and*

*CNRS, Ecole Normale Supérieure, INRIA, Bruno.Blanchet@ens.fr. Partially supported by the ANR project FormaCrypt and by DGA.

†DIMACS, Rutgers University, adj@dimacs.rutgers.edu. Partially supported by NSF Grants CNS-0751674 and CNS-0753492 and by ONR Grant N00014-07-1-1039.

‡mail.jesse.rao@gmail.com.

§Department of Mathematics, University of Pennsylvania, scedrov@math.upenn.edu. Partially supported by ONR Grant N00014-07-1-1039, by OSD/AFOSR MURI "Collaborative policies and assured information sharing," and by NSF Grants CNS-0429689, CNS-0524059, and CNS-0830949.

¶Department of Electrical Engineering and Information Sciences, Ruhr-University Bochum, joe-kai.tsay@trust.rub.de.

– *if k is a key that is used at least once for encryption, then $\mathcal{A}$ never queries $\mathcal{D}_k(.)$ on a cyphertext encrypted by any key playing the role of k in any one of the protocol sessions. (*I.e.*, in Kerberos, if k is an authentication, resp. service, key that is used at least once for encryption, then $\mathcal{A}$ never queries $\mathcal{D}_k(.)$ on a cyphertext encrypted by* any *authentication, resp. service, key.)*

- *the experiment outputs* 1 *if the decryption oracle properly decrypts a query by the adversary,* i.e.*, outputs $m \neq \perp$, otherwise the experiment outputs* 0.

We then use CryptoVerif to show that the various fresh keys in Kerberos (with and without PKINIT) are usable in this sense. (As in [1], our cryptographic assumptions here include that the symmetric encryption scheme used in the protocol is INT-CTXT secure; however, INT-CTXT usability was not defined at that time and we only proved usability for IND-CCA2-secure encryption.)

Third, we update the formalization in a number of important technical ways and study the cryptographic assumptions needed by Kerberos. Prompted by and extending comments by Chao Feng [3], we add additional oracles to provide certificates to parties other than just the client. More precisely, we add two oracle processes called CCERT and KCERT, to our model of Public-key Kerberos in CryptoVerif's process calculus: The first oracle process (CCERT) allows the attacker to obtain valid certificates for dishonest clients, whereas the second oracle process (KCERT) allows the attacker to obtain valid certificates for dishonest Kerberos Authentication Servers (KASes). Both oracles are relevant for public-key Kerberos, as in this setting neither the client nor the KAS need to be pre-registered to each other before starting a protocol session (in contrast to basic Kerberos). One may consider KCERT redundant under the assumption that any KAS is a trusted third party; however, the lack of CCERT, which was brought to our attention by Chao Feng [3], prevented some insider attacks, in particular the man-in-the-middle attack on the flawed version PKINIT-25 [4].

Because of these additional oracles, we also need to study some cryptographic requirements on the MAC in a part of the protocol. CryptoVerif is not able to prove some security properties for public-key Kerberos if, as in [1], we assume only UF-CMA security for the MAC used for the `asChecksum` field in PKINIT. CryptoVerif succeeds if we instead make the assumption that the MAC is an HMAC based on a family of collision-resistant hash functions. That is, we are assuming $HMAC_i(k, m) = h_i(k \oplus \mathsf{opad} || h_i(k \oplus \mathsf{ipad} || m))$, where m is a message to be hashed, k is the MAC-key, and i is a non-secret index of the collision-resistant hash function family $\{h_i\}_{i \in I}$. We stress that although CryptoVerif is not able to complete the security proofs for public-key Kerberos if one assumes general UF-CMA security for the MAC for `asChecksum`, we did not discover an attack on PKINIT under this assumption. We believe that CryptoVerif fails to complete the proofs both for model-dependent reasons and because it is presently not capable of temporal reasoning.

In comparison, in the computational analysis of the symmetric encryption scheme of Kerberos in [5], the assumptions on the MAC algorithm are stronger than UF-CMA. There, the MAC is assumed to be a PRF; this holds if the MAC is an HMAC and the underlying compression function is a PRF. This can be seen as another indication that one needs to make strong assumptions on the MAC to prove the correctness of Kerberos.

With the added oracles and the changes we made to the cryptographic assumptions, we are still able to prove authentication, secrecy, and usability results as before. As with our new work on INT-CTXT usability, this illustrates how CryptoVerif is useful for exploring different cryptographic definitions and assumptions.

References

[1] B. Blanchet, A. D. Jaggard, A. Scedrov, and J.-K. Tsay, "Computationally Sound Mechanized Proofs for Basic and Public-Key Kerberos," In *ASIACCS*, pp. 87–99 (2008).

[2] A. Datta, A. Derek, J. C. Mitchell, and B. Warinschi, "Computationally Sound Compositional Logic for Key Exchange Protocols," In *CSFW*, pp. 321–334 (2006).

[3] C. Feng, Personal communication (2009).

[4] I. Cervesato, A. D. Jaggard, A. Scedrov, J.-K. Tsay, and C. Walstad, "Breaking and Fixing Public-Key Kerberos", *Information and Computation* **206**, pp. 402–424 (2008).

[5] A. Boldyreva and V. Kumar, "Provable-Security Analysis of Authenticated Encryption in Kerberos", In *IEEE Symposium on Security and Privacy*, pp. 92–100 (2007).

Computational Indistinguishability Logic

Gilles Barthe Marion Daubignard Bruce Kapron Yassine Lakhnech

Bellare and Rogaway [5] and Shoup [10] have recently emphasized that rigorous proof methods are required to structure and manage the complexity of cryptographic proofs, and suggested the game-based technique as one potential such method. However, despite its popularity as a tool to carry out cryptographic proofs, the proof theory of the game-based technique is not well developed; existing attempts of formalization, e.g. [6, 2], are tied to a particular language, and fail to capture in a compact set of rules the essentials tools of game-based techniques, e.g. negligibility of events and conditional reasoning. In fact, and in contrast to the symbolic model for which proof systems exist, there is a lack of generally applicable proof systems for reasoning about cryptographic systems in the computational model.

The main contribution of this paper is a *Computational Indistinguishability Logic* (CIL) for proving the security of cryptographic schemes in the computational model. To our best knowledge, CIL is the first rigorously justified logic that allows reasoning about security against adaptive adversaries of standard cryptographic primitives, including encryption and signatures, *directly* in the computational model. CIL is versatile, and allows reasoning in the standard model as well as in idealized models such as the random oracle model. Furthermore, CIL captures in a largely language-independent setting many essential aspects of cryptographic proofs; the generality of CIL makes it a good starting point for a systematic exploration of the proof theory of cryptographic proofs.

CIL is designed for reasoning about indistinguishability, which plays a central role in cryptography; informally, two distributions are indistinguishable, if all feasible adversaries have a negligible probability of gaining information from their comparison. Therefore, reasoning about indistinguishability is tightly tied with reasoning about negligibility of events. For this reason, CIL considers two basic forms of assertions: $s \sim t$, stating that the distributions s and t are indistinguishable; and $s : E$ stating that the event E has a negligible probability in s. In these assertions, it is implicitly assumed that s and t are distributions induced by processes[1]: to describe processes, CIL relies on the notion of computational frame, which is inspired from Abadi and Fournet's [1] notion of frame, and which allows consideration of adaptive adversaries by extending an arbitrary language for probabilistic polynomial time computations with random sampling, adversary calls, and oracles. By concentrating on computational frames, one can formulate specific rules for reasoning about oracles, as well as more general rules for reasoning about indistinguishability and negligibility of events. For the latter CIL relies on two external logics: a variant of conditional logic for reasoning about negligible events [8], and a logic of observational equivalence (CEL) to prove that two computational frames yield equal distributions. An essential characteristic of CIL is to support of conditional reasoning, which is often not considered in earlier works such as [9, 7]. Yet conditional reasoning is essential in many applications.

We illustrate the versatility and power of CIL through two emblematic examples: semantic security of OAEP [3], and existential unforgeability of FDH under adaptive chosen message attack [4].

References

[1] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL*, pages 104–115, 2001.

[1] Throughout the paper, we shall often identify processes with the distribution they induce, and use letters s and t both as distributions and processes.

[2] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *Proceedings of the 36th ACM Symposium on Principles of Programming Languages*, pages 90–101. ACM Press, 2009.

[3] M. Bellare and P. Rogaway. Optimal asymmetric encryption – How to encrypt with RSA. In *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer-Verlag, 1995.

[4] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with rsa and rabin. In *EUROCRYPT*, pages 399–416, 1996.

[5] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, 2006.

[6] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *IEEE Symposium on Security and Privacy*, pages 140–154. IEEE Computer Society, 2006.

[7] Judicamël 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 15th ACM Conference on Computer and Communications Security*, pages 371–380. ACM Press, 2008.

[8] Joseph Y. Halpern. From qualitative to quantitative proofs of security properties using first-order conditional logic. In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 454–459. AAAI Press, 2008.

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

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

Automated Proofs for Encryption Modes. ⋆

Martin Gagné[2], Pascal Lafourcade[1], Yassine Lakhnech[1], and Reihaneh Safavi-Naini[2]

[1] Université Grenoble 1, CNRS,Verimag, FRANCE
[2] Department of Computer Science, University of Calgary, Canada

A block cipher (e.g. AES, Blowfish, DES, Serpent and Twofish) is a symmetric key algorithm that takes a fixed size input message block and produces a fixed size output block. A mode of operation is a method for encrypting an arbitrary length message using a block cipher. Important modes of operation are Electronic Code Book ECB, Cipher Block Chaining (CBC), Cipher FeedBack (CFB), Output FeedBack (OFB), and Counter mode (CTR). Modes of operation have different applications and provide different levels of security and efficiency. An important question when a mode of operation is used for encryption is the level of security that the mode provides, assuming the underlying block cipher is "secure". The answer to this question is not straightforward. For example if one uses the naive ECB mode with a "secure" block cipher, then the encryption scheme obtained is not even semantically secure. Other modes, like CBC, will provide confidentiality only if the initial vector is chosen adequately.

Recent years have seen an explosion of new modes of operation for block ciphers. These new modes of operation often offer improved security guarantees, or additional security features. They also tend to be more complex than the traditional modes of operation, and arguments for proving their security can similarly become much more complicated – sometimes so complicated that flaws in the security proofs could go unnoticed for years.

Proofs generated by automated verification tools can provide us with an independent argument for the security of modes of operation, thereby increasing our confidence in the security of the cryptographic protocols. While the rules used by the prover must also be proven by humans, and are therefore also susceptible to error, they tend to be much simpler than the protocols they will be used to check, which ensures that mistakes are far less likely to go unnoticed. In this paper, we take a first step towards building an automated prover for modes of operation, and show how to automatically generate proofs for many traditional block cipher modes of operation.

We propose a compositional Hoare logic for proving semantic security of modes of operation for symmetric key block ciphers. We first notice that most of modes use a small set of operations such as xor, concatenation, and selection of random values. We introduce a simple programming language to specify encryption modes and an assertion language that allows to state invariants and axioms,

⋆ This work was supported by ANR SeSur SCALP, SFINCS, AVOTE and iCORE.

and rules to establish such invariants. The assertion language requires only four predicates: one that allows us to express that the value of a variable is indistinguishable from a random value when given the values of a set of variables, one which states that an expression has not yet been submitted to the block cipher, and two bookkeeping predicates that allow us to keep track of "fresh" random values and counters. Transforming the Hoare logic into an (incomplete) automated verification procedure is quite standard. Indeed, we can interpret the logic as a set of rules that tells us how to propagate the invariants backwards. We were able to automatically verify the semantic security of several encryption modes including CBC, CFB, CTR and OFB. Of course our system does not prove ECB mode, because ECB is not semantically secure.

Our automated verification program proves the security of the block cipher modes of operation by showing that, from the adversary's point of view, the ciphertext is indistinguishable from a random string of the same length. This notion is strictly stronger than the traditional notion of semantic security. Future work in this area includes the study of the exact power of our rule set by attempting to prove the security of other modes of operation that are expressible using our language. This may suggest new rules that are not currently part of our model. We also intend to extend the language to include more operation to enable us to study modes of operation that use more complex operations and a wider array of cryptographic primitives.

On the Computational Soundness of Cryptographic Verification by Typing

Cédric Fournet
Microsoft Research

May 6, 2009

Type systems are effective tools for establishing the security of cryptographic protocols and their implementations. They offer modularity and scalability, and have been successfully applied to the verification of large protocols and applications [e.g. Bhargavan et al., 2009]. However, they traditionally rely on abstract assumptions on cryptographic primitives, expressed in symbolic models.

In this talk, I present recent and ongoing work on designing (and adapting) type systems to rely instead on computational assumptions [Abadi et al., 2006, Fournet and Rezk, 2008]. These results and others [e.g. Laud, 2005, 2008] suggest that typing can be just as effective under more standard and realistic hypotheses.

In particular, I plan to demo the automated verification of protocols written in ML by typing against a cryptographic interface, using the F7 refinement-type checker [Bengtson et al., 2008, Bhargavan et al., 2008]. For each cryptographic library, the computational soundness of typechecking follows from a code-based game-hopping argument between different ML implementations of the same interface, according to standard assumptions such as IND-CCA2 or INT-CMA.

References

M. Abadi, R. Corin, and C. Fournet. Computational secrecy by typing for the pi calculus. In *APLAS'06*, volume 4279 of *LNCS*, pages 253–269, Nov. 2006.

J. Bengtson, K. Bhargavan, C. Fournet, A. D. Gordon, and S. Maffeis. Refinement types for secure implementations. In *21st IEEE Computer Security Foundations Symposium (CSF'08)*, pages 17–32, 2008.

K. Bhargavan, C. Fournet, and A. D. Gordon. F7: refinement types for F#, Sept. 2008. Available from `http://research.microsoft.com/F7/`.

K. Bhargavan, R. Corin, P.-M. Dénielou, C. Fournet, and J. Leifer. Cryptographic protocol synthesis and verification for multiparty sessions. In *22nd IEEE Computer Security Foundations Symposium (CSF'09)*, 2009.

C. Fournet and T. Rezk. Cryptographically sound implementations for typed information-flow security. In *35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'08)*, pages 323–335, 2008.

P. Laud. Secrecy types for a simulatable cryptographic library. In *12th ACM Conference on Computer and Communications Security*, pages 26–35, 2005.

P. Laud. On the computational soundness of cryptographically-masked flows. In *35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'08)*, pages 337–348, 2008.

Computational Soundness of RCF Implementations*

Michael Backes[1,2], Matteo Maffei[1], and Dominique Unruh[1]
[1]Saarland University [2]MPI-SWS

Abstract

Recently, increasing attention has been given to the formal verification of security properties on the source code of cryptographic protocols. One of the most prominent results in this direction is F7 (Bengtson et al., CSF 2008), a security type-checker for the F# programming language. F# is encoded in a typed concurrent lambda calculus named RCF. The underlying type theory is based on refinement types for expressing pre- and post-conditions within first-order logic. Programs are type-checked against a symbolic cryptographic library, which abstracts from the computational implementation of the cryptographic primitives. Any well-typed program is guaranteed to be secure with respect to this abstraction.

We give a computational soundness result for the RCF type system. That is, we show that a well-typed program is not only guaranteed to be secure when executed with the symbolic library, but also when executed with an actual computational implementation of public-key encryption and signatures.

Overview. The Refined Concurrent FPC (RCF) [2] is a simple programming language extending the Fixed Point Calculus [3] with refinement types and primitives for concurrency. Although very simple, this core calculus is expressive enough to encode a large fragment of F#.

Programs are type-checked against a symbolic cryptographic library, which is based on so-called sealing functions. A sealing function takes as input a plaintext m, stores m along with a fresh handle a in a hidden list, and returns a; the unsealing function has access to the same hidden list, takes as input the handle a, and returns m. For example, when encoding public-key encryption using sealing functions, an encryption key is represented by a sealing function that can be applied to the plaintext m and returns the handle a as ciphertext. The corresponding unsealing function is used as the decryption key. The main advantage of this encoding is that the type of cryptographic material as well as the type of cryptographic operations is not hard-coded in the type system but is instead expressed as the type of the corresponding functions. This makes it possible to smoothly extend the framework to different cryptographic primitives without proving the soundness of the type system from scratch.

Security-related program points are decorated with assumptions of the form $\mathsf{assume}\ C$ and assertions of the form $\mathsf{assert}\ C$, where C is a formula in first-order logic. Authorization policies are expressed themselves as assumptions. A program is deemed secure if at run-time each asserted formula is entailed by the previously assumed ones. This notion of security allows for expressing standard authentication properties, such as non-injective agreement, as well as more sophisticated authorization policies, such as access control policies.

Computational soundness of RCF. We give a computational soundness result for RCF. More precisely, we achieve the following result: Let $\mathrm{Lib}_{\mathrm{seal}}$ denote a sealing-based symbolic library abstracting public-key encryption and digital signatures. Let $\mathrm{Lib}_{\mathrm{comp}}$ denote a computational implementation of the symbolic library that uses an IND-CCA secure encryption scheme and a strongly existentially unforgeable signature scheme. We say an RCF-program P is computationally safe if for any polynomial-time adversary interacting with the protocol, it holds that all assertions executed by P during the protocol execution are a logical consequence of the assumptions executed earlier by the program. We write $P + \mathrm{Lib}_{\mathrm{seal}}$ for a program P "linked" against the library $\mathrm{Lib}_{\mathrm{seal}}$, and $P + \mathrm{Lib}_{\mathrm{comp}}$ analogously.

Result 1 (Computational Soundness (informal)) *Assume a program P (subject to certain technical conditions) such that $P + \mathrm{Lib}_{\mathrm{seal}}$ typechecks using the type system from [2]. Then $P + \mathrm{Lib}_{\mathrm{comp}}$ is computationally safe.*

Our proof is based on the CoSP framework [1]. This allows us to base the proof on general computational soundness results in the CoSP framework. As a consequence, when computational soundness results for additional cryptographic primitives (such as zero-knowledge) are available in the CoSP framework, these can be added with minimal additional effort to our soundness result.

*Partial funding by the Emmy Noether Program of the DFG and by the Cluster of Excellence "Multimodal Computing and Interaction".

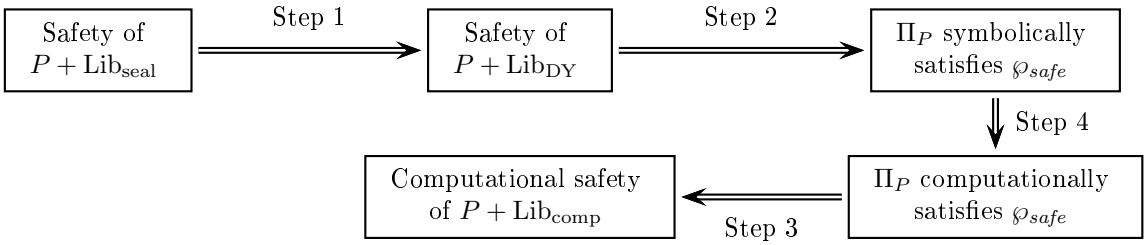

Figure 1: Steps in the proof of Result 1.

Overview of the proof. The proof of Result 1 consists of four steps:

1. The basic idea of $\mathrm{Lib}_{\mathrm{seal}}$ is to use seals, that is pairs of functions that access a shared reference to store secret information. Although this approach turns out to be very viable for develop security type systems, the presence of side effects in the definitions of seals makes it very difficult to argue about seals. For our proof, we introduce a second library $\mathrm{Lib}_{\mathrm{DY}}$. This library represents any cryptographic object (e.g., ciphertexts, keys, signatures) as a term, which both the adversary and the program may access only via explicitly exported functions. That is, $\mathrm{Lib}_{\mathrm{DY}}$ encapsulates a Dolev-Yao model of public-key encryption and signatures. Then, by giving a mapping from executions of $P + \mathrm{Lib}_{\mathrm{DY}}$ to executions of $P + \mathrm{Lib}_{\mathrm{seal}}$, we show that $P + \mathrm{Lib}_{\mathrm{DY}}$ is safe if $P + \mathrm{Lib}_{\mathrm{seal}}$ is.
2. We express $P + \mathrm{Lib}_{\mathrm{DY}}$ in the CoSP framework. In the CoSP framework, a protocol is expressed as an infinite tree whose nodes correspond to actions such as communication with the adversary or application of constructors and destructors. Since each function exported by $\mathrm{Lib}_{\mathrm{DY}}$ corresponds to an application of a constructor or a destructor, the CoSP protocol Π_P corresponding to P only needs to model the semantics of P and can directly encode any call to $\mathrm{Lib}_{\mathrm{DY}}$ as a single constructor or destructor node. We then have that an execution of $P + \mathrm{Lib}_{\mathrm{DY}}$ and a symbolic execution of Π_P essentially have the same traces. Hence if $P + \mathrm{Lib}_{\mathrm{DY}}$ is safe, a symbolic execution of Π_P in the CoSP framework is safe as well (where safety in the CoSP framework is a certain trace property $\wp_{safe}$).
3. $\mathrm{Lib}_{\mathrm{DY}}$ and $\mathrm{Lib}_{\mathrm{comp}}$ have the same interface, and the only difference is that whenever $\mathrm{Lib}_{\mathrm{DY}}$ would apply a constructor or destructor, $\mathrm{Lib}_{\mathrm{comp}}$ would apply the corresponding computational implementation (e.g., run the encryption function). Furthermore, the CoSP framework specifies a computational execution of Π_P which operates on bitstrings and uses the computational implementation instead of the symbolic constructors and destructors. We then have that an execution of $P + \mathrm{Lib}_{\mathrm{comp}}$ and a computational execution of Π_P essentially have the same traces. Hence, if the computational execution of Π_P in the CoSP framework is safe (where safety is expressed using the same trace property $\wp_{safe}$ as in Step 2), then $P + \mathrm{Lib}_{\mathrm{comp}}$ is computationally safe.
4. Computational soundness in the CoSP framework guarantees that for any protocol Π, if the symbolic execution of Π satisfies a trace property, then the computational execution of Π satisfies that trace property. In [1] a computational soundness result for public-key encryption and signatures is given. Thus if the symbolic execution of Π_P satisfies the trace property $\wp_{safe}$, then the computational execution of Π_P satisfies the trace property $\wp_{safe}$.

Thus, if $P + \mathrm{Lib}_{\mathrm{seal}}$ is safe, then $P + \mathrm{Lib}_{\mathrm{comp}}$ is computationally safe. This gives Result 1.

Note that only the proof of Step 1 depends on the cryptographic primitives we use. Hence, given a computational soundness result in the CoSP framework for a more expressive set of cryptographic primitives (including, e.g., zero-knowledge proofs), only Step 1 needs to be adapted to extend Result 1 to these primitives. Since Step 1 is a relatively standard argument and does not involve any cryptography, this makes our proof very easy to extend.

References

[1] M. Backes, D. Hofheinz, and D. Unruh. CoSP: A general framework for computational soundness proofs. Abstract in these proceedings, long version on IACR ePrint 2009/080, 2009.

[2] J. Bengtson, K. Bhargavan, C. Fournet, A. D. Gordon, and S. Maffeis. Refinement types for secure implementations. In *CSF 2008*, pages 17–32, 2008.

[3] A. Gunter. *Semantics of Programming Languages: Structures and Techniques.* MIT Press, 1992.