Vérification des Protocoles Cryptographiques et de leurs Implémentations

Cédric Fournet

http://research.microsoft.com/~fournet

+ Martin Abadi, Pedro Adao, Karthik Bhargavan, Bruno Blanchet, Ricardo Corin, Andrew D. Gordon, Tamara Rezk, Stephen Tse
Vérification des Protocoles Cryptographiques et de leurs Implémentations

1. Modélisation des protocoles en pi calcul
2. Comment vérifier l’usage des protocoles? applications aux services Web (outils, demo)
3. Comment vérifier leurs implémentations? (demo)
4. Cryptographie formelle/concrete
modelling cryptographic primitives
Models for cryptography

- Abstractions are needed to design and analyze protocols; abstractions may hide important flaws of the real system.

- Two main approaches have been successfully applied to protocols and programs that use cryptography

Formal, or symbolic approach (Needham Schroeder, Dolev Yao, ... late 70’s)
  - Structural view of protocols, using simple formal languages, and methods from logic, programming languages, concurrency
  - Compositional, good tool support for verification, automation
  - Too abstract?

Computational approach (Yao, Goldwasser, Micali, Rivest, ... early 80’s)
  - Messages are probability distributions over concrete bitstrings
  - Adversaries range over probabilistic Turing Machines
  - Delicate manual proofs, with scalability issues
  - Seems more accurate, hence more widely accepted
The formal view

- Cryptographic primitives are constructors within an algebra of terms, subject to equational rewriting

- For encryption, $\mathcal{D}(\mathcal{E}(x, k), y) = x$, $M, N ::= x, y, \ldots$
  
  $\mid n, k, \ldots$
  
  $\mid \mathcal{D}(M, N)$
  
  $\mid \mathcal{E}(M, N)$
  
  $\mid \ldots$

- The equations entirely define cryptography: no extra properties
The formal view

- Cryptographic primitives are constructors within an algebra of terms, subject to equational rewriting.

- For encryption, $\mathcal{D}(\mathcal{E}(x, k), y) = x$

- For secrets, keys, nonces... we rely on scopes and renaming.
  - For example, in applied pi:
    \[
    \nu k.\text{send}(\mathcal{E}(M, k)) \approx \nu n.\text{send}(n)
    \]

\[
\nu s.\text{send}(\mathcal{E}(M, \mathcal{D}(M, s)) \approx \nu n.\text{send}(n)
\]
The formal view: strengths

- Often an appropriate level of abstraction
  - Simple, effective intuitions (black boxes)
  - “Enough” interesting details (can adjust equations)
  - Many serious bugs found

- Plenty of methods and foundations
  - process calculi
  - modal logics
  - rewriting systems
  - type systems

- Effective techniques and tools for automated reasoning even for large systems and sophisticated properties
The formal view: limits

- An optimistic view of cryptography
  - The instantiation of black boxes is not obvious: Which algorithm to choose for encryption? modes, IVs?
  - Blind spots on special values and properties

- Observational equivalence properties are especially tricky
  - Absence of information leaks, privacy

- A borderline case: weak secrecy
  - Secrecy for PINs and human passwords, not strong keys
  - Specific design techniques and protocols
  - Pretty good formal modelling and tools
The computational view

- Keys and messages range over concrete bitstrings with finite lengths, not formal terms

- An encryption scheme is a triple of algorithms

\[ \Pi \overset{\text{def}}{=} (\mathcal{K}, \mathcal{E}, \mathcal{D}) \]

- Its properties are stated for a specific usage of encryption
  - Functional correctness (minimal positive properties)
    \[ k := \mathcal{K}; \ldots; \mathcal{D}(\mathcal{E}(x, k), k) = x \text{ when } \ldots \]
  - Cryptographic assumptions (minimal negative properties)
    \[ k := \mathcal{K}; \ldots; y := \mathcal{E}(M_1, k) \]
    \[ \approx k := \mathcal{K}; \ldots; y := \mathcal{E}(M_2, k) \text{ when } \ldots \]
The computational view

- **Probabilistic properties:** We bound the probability that an adversary breaks a protocol.

- **Asymptotic properties:**
  - Everything, including the complexity of the adversary, is polynomially-bounded using a “security parameter”.
  - We show that an adversary breaks a protocol with negligible probability.

\[ f(\eta) \leq \text{neg}(\eta) \text{ when } \forall c > 0, \exists \eta_c : \eta \geq \eta_c \Rightarrow f(\eta) \leq \eta^{-c}. \]

- Concrete cryptographic semantics is too complicated, especially for large systems.
  - Poor compositionality
  - Overkill?
Bridging the gap

- Can we get the best of both worlds?
- When are these approaches *equivalent*?
  - Can we carry over results from one model to the other?
  - Can we compose their results?

- Soundness property (desired)

  “If a security property can be proven in a formal model, then it holds in a computational model”
computational soundness: a first example

A formal view of encryption

The set of expressions $\text{Exp}$:

$$M, N ::= \text{ expressions}$$

- $i$ bit (for $i \in \text{Bool}$)
- $K$ key (for $K \in \text{Keys}$)
- $(M, N)$ pair
- $\{M\}_K$ encryption (for $K \in \text{Keys}$)

- We treat (non-deterministic, which-key concealing) shared-key encryption
- We consider only passive attackers at the end of the protocol
Formal patterns

We map each expression $M$ to a pattern that the attacker can see with no a priori knowledge:

\[
\begin{align*}
\text{pattern}(i) & = i & \text{(for } i \in \text{Bool}) \\
\text{pattern}(K) & = K & \text{(for } K \in \text{Keys}) \\
\text{pattern}((N_1, N_2)) & = (\text{pattern}(N_1), \text{pattern}(N_2)) \\
\text{pattern}([N]_K) & = \begin{cases} 
\text{pattern}(N) \setminus_K & \text{if } M \vdash K \\
\square & \text{otherwise}
\end{cases}
\end{align*}
\]

where

- $M \vdash M$.
- If $M \vdash (N_1, N_2)$ then $M \vdash N_1$ and $M \vdash N_2$.
- If $M \vdash [N]_K$ and $M \vdash K$ then $M \vdash N$. 
We map each expression $M$ to a pattern that the attacker can see with no a priori knowledge:

\[
\begin{align*}
\text{pattern}(i) &= i \quad \text{(for } i \in \text{Bool}) \\
\text{pattern}(K) &= K \quad \text{(for } K \in \text{Keys}) \\
\text{pattern}((N_1, N_2)) &= (\text{pattern}(N_1), \text{pattern}(N_2)) \\
\text{pattern}({N}K) &= \begin{cases} 
\{\text{pattern}(N)\}K & \text{if } M \vdash K \\
\Box & \text{otherwise}
\end{cases}
\end{align*}
\]

Example: \[\text{pattern}((\{0\}_K)_K, K_2)) = (\{\Box\}_K, K_2)\]
Observational equivalence

Informally, two expressions are equivalent if they look the same to an attacker.

Formally, two expressions are equivalent if they yield the same pattern.

Examples:

\[0 \equiv 0\]

\[0 \not\equiv 1\]

\[\{0\}_K \equiv \{1\}_K\]

\[(K, \{0\}_K) \not\equiv (K, \{1\}_K)\]
Further examples (fine points)

\[ \{0\}_K \cong \{K\}_K \]

Encryption cycles do not leak data.

\[ \{0\}_K \cong \{(1,1),(1,1)\}, ((1,1),(1,1))\} \]

Ciphertexts do not reveal the size of plaintexts.

\[ (\{0\}_K, \{0\}_K) \cong (\{0\}_K, \{1\}_K) \]

Plaintext equalities are concealed.

\[ (\{0\}_K, \{1\}_K) \cong (\{0\}_K, \{1\}_K') \]

Key equalities are concealed.
Computational semantics

We map:

\[ M \in \text{Exp} \quad \eta \in \text{Parameter} \quad \rightarrow \quad \text{a distribution on bitstrings} \quad [M]_\Pi[\eta] \]

and thereby

\[ M \in \text{Exp} \quad \rightarrow \quad \text{an ensemble} \quad [M]_\Pi \]
First, we map each key symbol $K$ that occurs in $M$ to a bitstring $\tau(K)$, using $\mathcal{K}(\eta)$.

Then we set (roughly):

$$
\begin{align*}
\llbracket 0 \rrbracket_{\eta} & = 0 \\
\llbracket 1 \rrbracket_{\eta} & = 1 \\
\llbracket K \rrbracket_{\eta} & = \tau(K) \\
\llbracket (M, N) \rrbracket_{\eta} & = (\llbracket M \rrbracket_{\eta}, \llbracket N \rrbracket_{\eta}) \\
\llbracket \{ M \} \rrbracket_{\eta} & = \mathcal{E}_{\tau(K)}(\llbracket M \rrbracket_{\eta})
\end{align*}
$$

We tag string representations with their types (that is, “key”, “bool”, “pair”, “ciphertext”).
A soundness theorem

Let $M$ and $N$ be expressions without encryption cycles, and $\Pi$ be a secure encryption scheme (adopting a variant of the standard, computational definition of secure encryption).

If $M \cong N$ then $\llbracket M \rrbracket_\Pi \approx \llbracket N \rrbracket_\Pi$.

That is, if $M$ and $N$ are formally equivalent then their meanings are computationally indistinguishable.
A code-based soundness proof

- We use a “code-based game-playing” proof argument a la Bellare Rogaway

- We use standard (polynomially-bound) commands in a small (probabilistic) imperative while language
  - More convenient and precise than Turing machines
  - Same expressiveness and complexity
Type-0 encryption (hypothesis)

- The Encryption game (G):
  \[ b := \{0, 1\}; \]
  \[ \tau_K := G; \]
  \[ \tau_{K'} := G; \]
  \[ B[if b \ then \mathcal{E}(x, \tau_K) \ else \mathcal{E}(0, \tau_{K'})]; \]
  \[ if \ b = \ g \ then \ 1 \ else \ 0 \]

  The command B has access to the whole memory except b \( \tau_K \tau_{K'} \).

- To win this game, B must guess the value of b and write it to g
  - B can compute on its own
  - B can call the encryption oracle for any value of x, any number of times
  - B can guess at random
    \[ B = g := \{0, 1\} \]
  - Can B do much better?

- Type-0 security hypothesis:
  \[ P[1 \leftarrow G] \leq \frac{1}{2} + \epsilon(\eta) \]
Soundness as a Term game

- A command for sampling terms in single-assignment style

\[
\begin{align*}
[0]x &= x := 0 \\
[1]x &= x := 1 \\
[K]x &= x := \tau_K \\
[(M, N)]x &= [M]x_1; \\
&\quad [N]x_2; \\
&\quad x := (x_1, x_2) \\
[{(M)}_K]x &= [M]x_1; \\
&\quad x := \mathcal{E}(x_1, \tau_K)
\end{align*}
\]
Soundness as a Term game

- The Term game:

  \[ b := \{0, 1\}; \]
  \[ \tau_K := G; \ldots \]
  \[ \text{if } b \text{ then } [M]x \text{ else } [N]x; \]
  \[ A; \]
  \[ \text{if } b = g \text{ then } 1 \text{ else } 0 \]

  The command A has access to the whole memory (with x!) except b \( \tau_K \ldots x_i \)

  \[ P[1 \leftarrow G] \leq \frac{1}{2} + \epsilon(\eta) \]

- A command for sampling terms in single-assignment style

  \[ [0]x = x := 0 \]
  \[ [1]x = x := 1 \]
  \[ [K]x = x := \tau_K \]
  \[ [(M,N)]x = [M]x_1; \]
  \[ [N]x_2; \]
  \[ x := (x_1, x_2) \]
  \[ [\{(M)_K\}]x = [M]x_1; \]
  \[ x := \mathcal{E}(x_1, \tau_K) \]
From Term to Encryption

- The Term game:
  
  \[ b := \{0, 1\}; \]
  \[ \tau_K := \mathcal{G}; \ldots \]
  \[ \text{if } b \text{ then } [M]x \text{ else } [N]x; \]
  \[ A; \]
  \[ \text{if } b = g \text{ then } 1 \text{ else } 0 \]

  The command A has access to the whole memory (with x!) except b \( \tau_K \ldots x_i \)

- The Encryption game:
  
  \[ b := \{0, 1\}; \]
  \[ \tau_K := \mathcal{G}; \]
  \[ \tau_{K'} := \mathcal{G}; \]
  \[ B[\text{if } b \text{ then } \mathcal{E}(x, \tau_K) \text{ else } \mathcal{E}(0, \tau_{K'})]; \]
  \[ \text{if } b = g \text{ then } 1 \text{ else } 0 \]

  The command B has access to the whole memory except b \( \tau_K \tau_{K'} \)

\[ P[1 \leftarrow G] \leq \frac{1}{2} + \epsilon(\eta) \]

\[ P[1 \leftarrow G] \leq \frac{1}{2} + \epsilon(\eta) \]
From Term to Encryption

- The Term game:

\[ b := \{0, 1\}; \]
\[ \tau_K := \mathcal{G}; \ldots \]
\[ if \ b \ then \ [M]x \ else \ [N]x; \]
\[ A; \]
\[ if \ b = g \ then \ 1 \ else \ 0 \]

The command A has access to the whole memory (with x!) except b \( \tau_K \ldots x_i \)
Eliminate one K at a time

- Intermediate Term game

\[
\begin{align*}
b & := \{0, 1\}; \\
\tau_K & := G; \ldots \\
\tau_{K'} & := G; \\
if b \text{ then } [M]x \text{ else } [M\setminus K]x; \\
A; \\
if b = g \text{ then } 1 \text{ else } 0
\end{align*}
\]

The command A has access to the whole memory (with x!) except b \(\tau_K \ldots \tau_{K'} \ x_i\)

- Starting from an acyclic term, we erase one key at a time, towards a pattern
Eliminate one $K$ at a time

- **Intermediate Term game**

  $b := \{0, 1\};$
  $\tau_K := G; \ldots$
  $\tau_{K'} := G;$
  $if \; b \; then \; [M]x \; else \; [M\backslash K]x;$
  $A;$
  $if \; b = g \; then \; 1 \; else \; 0$

  The command $A$ has access to the whole memory (with $x!$) except $b \; \tau_K \; \ldots \; \tau_{K'} \; x_i$

- **Erasing $K$**

  $0\backslash K = 0$
  $1\backslash K = 1$
  $K\backslash K = \text{error}$
  $K_1\backslash K = K_1$
  $(M, N)\backslash K = (M\backslash K, N\backslash K)$
  $\{M\}_K\backslash K = \{0\}_{K'}$
  $\{M\}_{K_1}\backslash K = \{M\backslash K\}_{K_1}$

- Starting from an acyclic term, we erase one key at a time, towards a pattern
Push the b-test (same distribution)

- Intermediate Term game
  
  \[ b := \{0, 1\}; \]
  \[ \tau_K := \mathcal{G}; \ldots \]
  \[ \tau_{K'} := \mathcal{G}; \]
  \[ [M] x \backslash K; \]
  \[ A; \]
  \[ \text{if } b = g \text{ then } 1 \text{ else } 0 \]

  The command A has access to the whole memory (with \( x! \)) except \( b \tau_K \ldots \tau_{K'} x_i \)

- Zero'ing K's encryptions
  
  \[ [0] x \backslash K = x := 0 \]
  \[ [1] x \backslash K = x := 1 \]
  \[ [K_1] x \backslash K = x := \tau_{K_1} \]
  \[ [K] x \backslash K = \text{error} \]
  \[ [(M, N)] x \backslash K = [M] x_1 \backslash K; \]
  \[ [N] x_2 \backslash K; \]
  \[ x := (x_1, x_2) \]
  \[ [{\{M\}}_K] x \backslash K = \text{if } b \text{ then } \]
  \[ [M] x_1 \backslash K; \]
  \[ x := \mathcal{E}(x_1, \tau_K) \]
  \[ \text{else } x := \mathcal{E}(0, \tau_{K'}) \]
  \[ [{\{M\}}_{K_1}] x \backslash K = [M] x_1 \backslash K; \]
  \[ x := \mathcal{E}(x_1, \tau_{K_1}) \]
Move dead code (same distribution)

- Intermediate Term game
  
  \[ b := \{0, 1\}; \]
  
  \[ \tau_K := G; \ldots \]
  
  \[ \tau_{K'} := G; \]
  
  \[ [M]x\backslash K; \]
  
  \[ A; \]
  
  \[ if\ b = g\ then\ 1\ else\ 0 \]

  The command \(A\) has access to the whole memory (with \(x!\)) except \(b\ \tau_K \ldots \tau_{K'}\ x_i\)

- Zero’ing \(K\)’s encryptions

  \[ [0]x\backslash K = x := 0 \]
  
  \[ [1]x\backslash K = x := 1 \]
  
  \[ [K_1]x\backslash K = x := \tau_{K_1} \]
  
  \[ [K]x\backslash K = \text{error} \]
  
  \[ [(M, N)]x\backslash K = [M]x_1\backslash K; \]
  
  \[ [N]x_2\backslash K; \]
  
  \[ x := (x_1, x_2) \]
  
  \[ [\{M\}_K]x\backslash K = [M]x_1\backslash K; \]
  
  \[ if\ b\ then\]
  
  \[ x := \mathcal{E}(x_1, \tau_K) \]
  
  \[ else\ x := \mathcal{E}(0, \tau_{K'}) \]
  
  \[ [\{M\}_{K_1}]x\backslash K = [M]x_1\backslash K; \]
  
  \[ x := \mathcal{E}(x_1, \tau_{K_1}) \]
Apply cryptographic hypothesis

- Intermediate Term game

  \[ b := \{0, 1\}; \]
  \[ \tau_K := G; \ldots \]
  \[ \tau_{K'} := G; \]
  \[ [M]x\backslash K; \]
  \[ A; \]
  \[ if \, b = g \, then \, 1 \, else \, 0 \]

  Let B be the command in red
  
  The command B has access to the whole memory except b \( \tau_K \tau_{K'} b \)

- Zero’ing K’s encryptions

  \[ [0]x\backslash K = x := 0 \]
  \[ [1]x\backslash K = x := 1 \]
  \[ [K_1]x\backslash K = x := \tau_{K_1} \]
  \[ [K]x\backslash K = \text{error} \]

  \[ [\langle M, N \rangle]x\backslash K = [M]x_1\backslash K; \]
  \[ [N]x_2\backslash K; \]
  \[ x := (x_1, x_2) \]

  \[ [\{M\}_K]x\backslash K = [M]x_1\backslash K; \]
  \[ if \, b \, then \]
  \[ x := E(x_1, \tau_K) \]
  \[ else \, x := E(0, \tau_{K'}) \]

  \[ [\{M\}_{K_1}]x\backslash K = [M]x_1\backslash K; \]
  \[ x := E(x_1, \tau_{K_1}) \]
Apply cryptographic hypothesis

- Intermediate Term game

\[ b := \{0, 1\}; \]
\[ \tau_K := \mathcal{G}; \ldots \]
\[ \tau_{K'} := \mathcal{G}; \]
\[ [M] x \backslash K; \]
\[ A; \]
\[ \text{if } b = g \text{ then } 1 \text{ else } 0 \]

Let B be the command in red

This game is also (exactly) an encryption game, so

\[ P[1 \leftarrow G] \leq \frac{1}{2} + \epsilon(\eta) \]

- Zero'ing K’s encryptions

\[ [0] x \backslash K = x := 0 \]
\[ [1] x \backslash K = x := 1 \]
\[ [K_1] x \backslash K = x := \tau_{K_1} \]
\[ [K] x \backslash K = \text{error} \]
\[ [(M, N)] x \backslash K = [M] x_1 \backslash K; \]
\[ [N] x_2 \backslash K; \]
\[ x := (x_1, x_2) \]
\[ [\{M\}_K] x \backslash K = [M] x_1 \backslash K; \]
\[ \text{if } b \text{ then} \]
\[ x := \mathcal{E}(x_1, \tau_K) \]
\[ \text{else } x := \mathcal{E}(0, \tau_{K'}) \]
\[ [\{M\}_{K_1}] x \backslash K = [M] x_1 \backslash K; \]
\[ x := \mathcal{E}(x_1, \tau_{K_1}) \]
Conclude by transitivity

- Back to the original Term game:

\[ b := \{0, 1\}; \]
\[ \tau_K := G; \ldots \]
\[ \tau_{K'} := G; \]
\[ \text{if } b \text{ then } [M]x \text{ else } [N]x; \]
\[ A; \]
\[ \text{if } b = g \text{ then } 1 \text{ else } 0 \]

\[ [M] \approx [M \setminus K] \approx \cdots \approx [M \setminus \tilde{K}] = [N \setminus \tilde{K}] \approx \cdots \approx [N] \]

Using triangular inequalities, we can bound the probability of breaking this game:

\[ P[1 \leftarrow G] \leq \frac{1}{2} + 2|\tilde{K}|\epsilon(\eta) \]
computational soundness of cryptography
Lots of recent progress

- AbadiRogaway00: soundness of sym enc wrt passive attacks;
- MicciancioWarinschi02, HorvitzGligor03: completeness of sym enc wrt passive attacks;
- Bana04, AdãoBanaScedrov05: more general soundness and completeness of sym enc wrt passive attacks;

- BackesPfitzmannWaidner03: simulatability based soundness of pub enc, sigs, sym enc, Macs wrt active attacks;
- MicciancioWarinschi04: soundness of pub enc wrt active attacks;
- Laud04: soundness of sym enc wrt active attacks;
- Herzog04: soundness of enc wrt active attacks (in particular non-malleability properties as addressed in simulatability results);
- HerzogCanetti06: simulatability based soundness for Message Authentication, Key-Exchange.
Recent & ongoing work (sample)

- Using probabilistic polytime process calculi
- Using type systems
  - For some specific primitives, use typing + side analyses to enforce the rules of cryptographic games
- Using logic
  - For some specific primitives, show that a set of axioms for reasoning on protocol traces are computationally sound

- Coq formalization and proofs for concrete cryptography
  - Ongoing work by Barthe Corin Gregoire Janvier Zanella...
Adapting formal types for secrecy

1. “Secrecy types for asymmetric communication” AbadiBlanchet’03
   A formally sound type system for asymmetric channels and public-key encryption, with “double-typing” for authentication

2. “Secrecy and group creation” CardelliGuelliGordon’05
   A formally sound type system for symmetric channels organized into typed channel groups

3. “Secrecy types for a simulatable cryptographic library” Laud’05
   A computationally-sound type system partly adapted from 1, relying on the BPW idealized crypto library

4. “Computational Secrecy by Typing for the Pi Calculus” AbadiCorinFournet ‘06
   A distributed implementation for typed pi processes, adapted from 1 and 2, relying on typed translations of communications into 3
Cryptoverif

- Blanchet’s automated prover for computational cryptography
  - Language: a variant of applied pi with built-in polynomial probabilistic features
  - Games are expressed as contextual equivalences
  - The tool applies a series of games and rewriting to “eliminate” concrete cryptography
  - Secrecy, authenticity properties checked on final games
  - Proves asymptotic as well as exact probabilistic security
Summary on computational crypto

- Formal and computational cryptography provide different, complementary approaches
  - Two strong, well-established traditions (25 years)
  - Lots of recent progress on their connection

- For studying large systems that rely on cryptography, one needs to embed detailed computational models within more abstract, general-purpose models.