

Formal Certification of Game-Based Cryptographic Proofs

Santiago Zanella Béguelin

IMDEA Software, Madrid, Spain



INRIA Sophia Antipolis - Méditerranée, France



Microsoft Research - INRIA Joint Centre, France



2010.12.09

ENS Paris

A tale of two worlds

Formal World

$$\{\{\{(0, K_1)\}_{K_2}, 1\}_{K_3}$$

- Values as symbols
- Primitives as symbolic expressions
- Adversaries as inference engines
- Asymptotic security
- Indirect (computation soundness)
- ProVerif, PCL, AVISPA
- Better suited for protocols?

Computational World

$$f(G(r) \oplus (m \parallel 0^k) \parallel H(G(r) \oplus (m \parallel 0^k)) \oplus r)$$

- Values as bitstrings
- Primitives as functions on bitstrings
- Probabilistic Polynomial-time Adversaries
- Exact security bounds
- Direct
- CryptoVerif, CPCL, CIL
- Better suited for primitives?

This work is in the computational world

A tale of two worlds

Formal World

$\{\{\{(0, K_1)\}_{K_2}, 1\}\}_{K_3}$

- Values as symbols
- Primitives as symbolic expressions
- Adversaries as inference engines
- Asymptotic security
- Indirect (computation soundness)
- ProVerif, PCL, AVISPA
- Better suited for protocols?

Computational World

$f(G(r) \oplus (m \parallel 0^k) \parallel H(G(r) \oplus (m \parallel 0^k)) \oplus r)$

- Values as bitstrings
- Primitives as functions on bitstrings
- Probabilistic Polynomial-time Adversaries
- Exact security bounds
- Direct
- CryptoVerif, CPCL, CIL
- Better suited for primitives?

This work is in the **computational** world

Cryptanalysis-driven design

Propose a cryptographic scheme

```
graph TD; A[Propose a cryptographic scheme] --> B[Wait for someone to come out with an attack]
```

Wait for someone to come out with an attack

Cryptanalysis-driven design

Propose a cryptographic scheme

```
graph TD; A[Propose a cryptographic scheme] --> B[Wait for someone to come out with an attack]
```

Wait for someone to come out with an attack

Cryptanalysis-driven design

Propose a cryptographic scheme

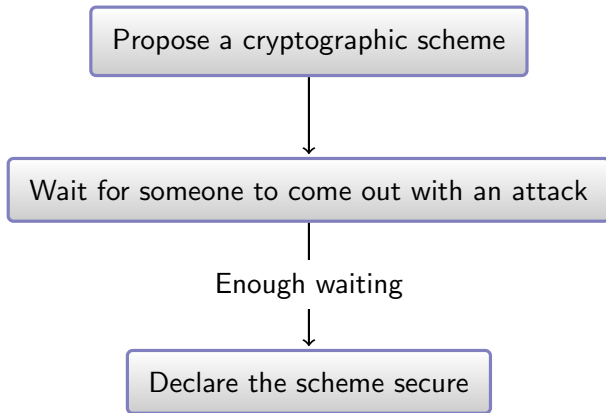
Wait for someone to attack

Attack found!

Cryptanalysis-driven design

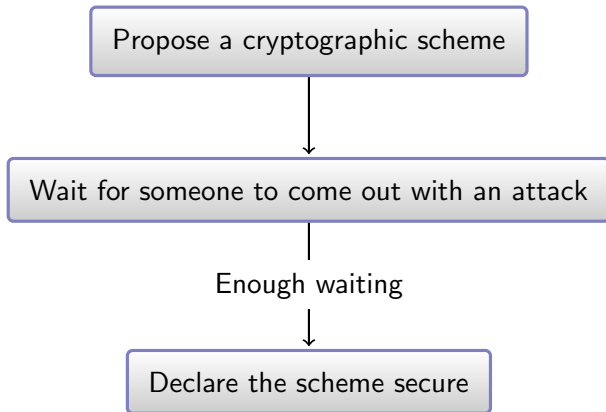


Cryptanalysis-driven design



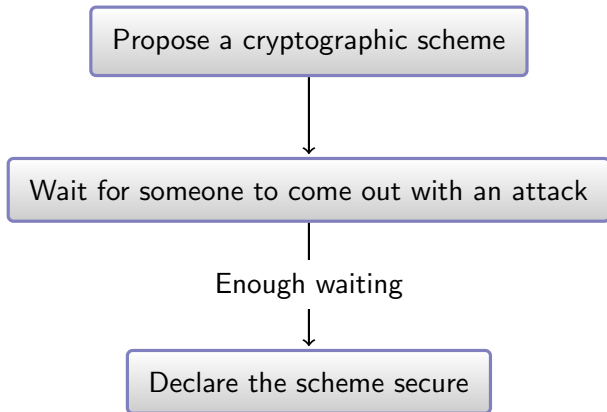
How much time is enough?

Cryptanalysis-driven design



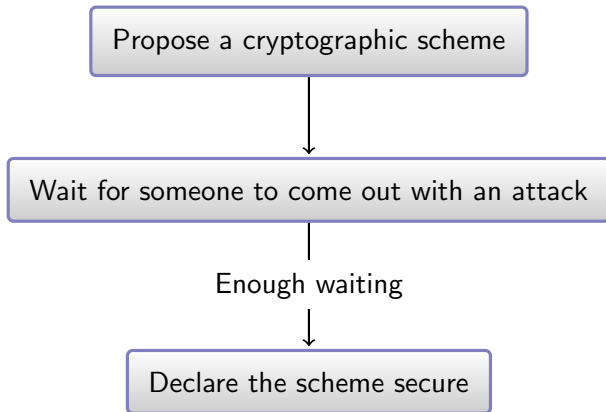
It took **5 years** to break the Merkle-Hellman cryptosystem

Cryptanalysis-driven design



It took **10 years** to break the Chor-Rivest cryptosystem

Cryptanalysis-driven design



Can't we do better?

The Provable Security paradigm

- 1 Define a security goal and a model for adversaries
- 2 Propose a cryptographic scheme
- 3 Reduce security of the scheme to a cryptographic assumption

IF an adversary \mathcal{A} can break the security of the scheme
THEN the assumption can be broken with little extra effort

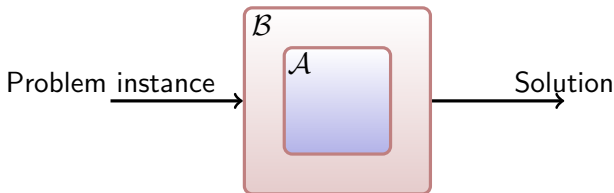
Conversely,

IF the security assumption holds **THEN** the scheme is secure

Proof by reduction

- Assume a polynomial adversary \mathcal{A} breaks the security of a scheme
- Build a polynomial algorithm \mathcal{B} that uses \mathcal{A} to solve a computational hard problem

IF the problem is intractable
THEN the cryptographic scheme is asymptotically secure



Exact security

- Assume an adversary \mathcal{A} breaks the security of a scheme within time t with probability ϵ
- Build an algorithm \mathcal{B} that uses \mathcal{A} to solve a computational hard problem with probability $\epsilon' \geq f(\epsilon)$ within time $t' \leq g(t)$

Bounds matter: the greater $f(\epsilon)$ and the smaller $g(t)$ are, the closer the security of the scheme is related to the problem.

Choosing scheme parameters

- What is the best known method to solve the problem?
- Choose parameters so that the reduction yields a better one

Exact security

- Assume an adversary \mathcal{A} breaks the security of a scheme within time t with probability ϵ
- Build an algorithm \mathcal{B} that uses \mathcal{A} to solve a computational hard problem with probability $\epsilon' \geq f(\epsilon)$ within time $t' \leq g(t)$

Bounds matter: the greater $f(\epsilon)$ and the smaller $g(t)$ are, the closer the security of the scheme is related to the problem.

Choosing scheme parameters

- What is the best known method to solve the problem?
- Choose parameters so that the reduction yields a better one

Game-based proofs

Security proofs in cryptography may be organized as sequences of games [...] this can be a useful tool in taming the complexity of security proofs that might otherwise become so messy, complicated, and subtle as to be nearly impossible to verify

V. Shoup

Game G_0 :

...
... $\leftarrow \mathcal{A}(\dots)$;
...

Game G_1 :

...
...
...

...

Game G_n :

...
... $\leftarrow \mathcal{B}(\dots)$
...

$$\Pr[G_0 : A_0] \leq h_1(\Pr[G_1 : A_1]) \leq \dots \leq h_n(\Pr[G_n : A_n])$$

Start from an initial game encoding the security goal

Game-based proofs

Security proofs in cryptography may be organized as sequences of games [...] this can be a useful tool in taming the complexity of security proofs that might otherwise become so messy, complicated, and subtle as to be nearly impossible to verify
V. Shoup

Game G_0 :

...
... $\leftarrow \mathcal{A}(\dots)$;
...

$\Pr[G_0 : A_0]$

Game G_1 :

...
...
...

$\leq h_1(\Pr[G_1 : A_1])$

...

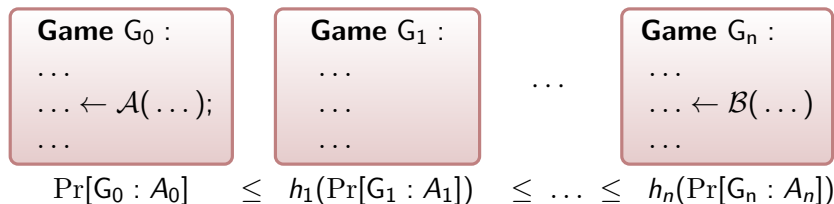
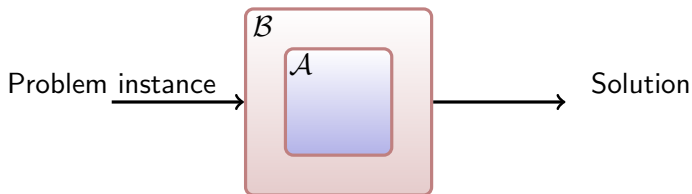
Game G_n :

...
... $\leftarrow \mathcal{B}(\dots)$
...

$\leq \dots \leq h_n(\Pr[G_n : A_n])$

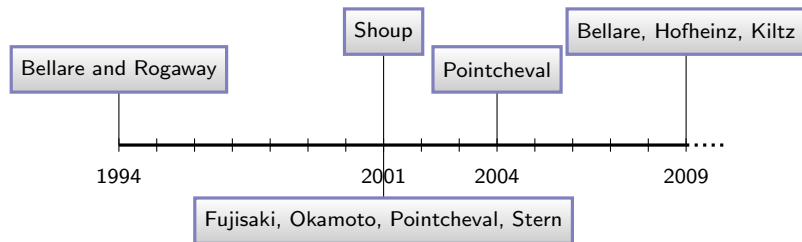
Stepwise transform the game keeping track of probabilities

Game-based proofs



Reach a final game encoding a computational assumption

Things can still go wrong (e.g. RSA-OAEP)



1994 Purported proof of chosen-ciphertext security

2001 Proof is flawed, but can be patched

- 1 ...for a weaker security notion, or
- 2 ...for a modified scheme, or
- 3 ...under stronger assumptions

2004 Filled gaps in Fujisaki et al. 2001 proof

2009 Security definition needs to be clarified

2010 Filled gaps and marginally improved bound in 2004 proof

Beyond Provable Security: Verifiable Security

Goal

Build a framework to formalize game-based cryptographic proofs

- Provide foundations to game-based proofs
- Notation as close as possible to the one used by cryptographers
- Automate common reasoning patterns
- Support exact security
- Provide independently and automatically verifiable proofs

CertiCrypt

Language-based cryptographic proofs

A language-based approach

Security definitions, assumptions and games are formalized using a probabilistic programming language

pWHILE: a probabilistic programming language

\mathcal{C}	::=	skip	nop
		$\mathcal{C}; \mathcal{C}$	sequence
		$\mathcal{V} \leftarrow \mathcal{E}$	assignment
		$\mathcal{V} \stackrel{d}{\leftarrow} \mathcal{DE}$	random sampling
		if \mathcal{E} then \mathcal{C} else \mathcal{C}	conditional
		while \mathcal{E} do \mathcal{C}	while loop
		$\mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \dots, \mathcal{E})$	procedure call

- $x \stackrel{d}{\leftarrow}$: sample the value of x according to distribution d
- The language of expressions (\mathcal{E}) and distribution expressions (\mathcal{DE}) admits user-defined extensions

Some design choices

- CertiCrypt is built on top of the Coq proof assistant
- Deep-embedding formalization
- Strongly-typed language
- Syntax is dependently-typed
(only well-typed programs are admitted)
- Monadic semantics uses Paulin-Mohring's ALEA Coq library

Semantics

Measure Monad

Distributions represented as monotonic, linear and continuous functions of type

$$\mathcal{D}(A) \stackrel{\text{def}}{=} (A \rightarrow [0, 1]) \rightarrow [0, 1]$$

$$\begin{aligned} \text{unit} : A &\rightarrow \mathcal{D}(A) && \stackrel{\text{def}}{=} \lambda x. \lambda f. f \ x \\ \text{bind} : \mathcal{D}(A) &\rightarrow (A \rightarrow \mathcal{D}(B)) \rightarrow \mathcal{D}(B) && \stackrel{\text{def}}{=} \lambda \mu. \lambda F. \lambda f. \mu(\lambda x. F \ x \ f) \end{aligned}$$

Intuition

Given $\mu \in \mathcal{D}(A)$ and $f : A \rightarrow [0, 1]$
 $\mu(f)$ represents the expected value of f w.r.t. μ

Semantics

Programs map an initial memory to a distribution on final memories

$$\llbracket c \in \mathcal{C} \rrbracket : \mathcal{M} \rightarrow \mathcal{D}(\mathcal{M})$$

The probability of an event is the expected value of its characteristic function:

$$\Pr[c, m : A] \stackrel{\text{def}}{=} \llbracket c \rrbracket m \mathbb{1}_A$$

Instrumented and parametrized semantics to characterize PPT:

$$\llbracket c \in \mathcal{C} \rrbracket_{\eta} : \mathcal{M} \rightarrow \mathcal{D}(\mathcal{M} \times \mathbb{N})$$

Semantics

Programs map an initial memory to a distribution on final memories

$$\llbracket c \in \mathcal{C} \rrbracket : \mathcal{M} \rightarrow \mathcal{D}(\mathcal{M})$$

The probability of an event is the expected value of its characteristic function:

$$\Pr[c, m : A] \stackrel{\text{def}}{=} \llbracket c \rrbracket m \mathbb{1}_A$$

Instrumented and **parametrized** semantics to characterize PPT:

$$\llbracket c \in \mathcal{C} \rrbracket_{\eta} : \mathcal{M} \rightarrow \mathcal{D}(\mathcal{M} \times \mathbb{N})$$

Observational equivalence

Definition

$$f =_X g \stackrel{\text{def}}{=} \forall m_1 m_2. m_1 =_X m_2 \implies f m_1 = g m_2$$

$$\models c_1 \simeq_O^I c_2 \stackrel{\text{def}}{=} \forall m_1 m_2 f g.$$

$$m_1 =_I m_2 \wedge f =_O g \implies \llbracket c_1 \rrbracket m_1 f = \llbracket c_2 \rrbracket m_2 g$$

Example

$$\models x \xleftarrow{\$} \{0, 1\}^k; y \leftarrow x \oplus z \simeq_{\{x, y, z\}}^{\{z\}} y \xleftarrow{\$} \{0, 1\}^k; x \leftarrow y \oplus z$$

- Useful to relate probabilities

$$\frac{\text{fv}(A) \subseteq O \quad \models c_1 \simeq_O^I c_2 \quad m_1 =_I m_2}{\text{Pr}[c_1, m_1 : A] = \text{Pr}[c_2, m_2 : A]}$$

- Only a Partial Equivalence Relation

$$\models c \simeq_O^I c \quad \text{not true in general (obviously)}$$

- Generalizes information flow security (take $I = O = \mathcal{V}_{\text{low}}$)

Observational equivalence

Definition

$$f =_X g \stackrel{\text{def}}{=} \forall m_1 m_2. m_1 =_X m_2 \implies f m_1 = g m_2$$

$$\models c_1 \simeq_O^I c_2 \stackrel{\text{def}}{=} \forall m_1 m_2 f g.$$

$$m_1 =_I m_2 \wedge f =_O g \implies \llbracket c_1 \rrbracket m_1 f = \llbracket c_2 \rrbracket m_2 g$$

Example

$$\models x \xleftarrow{\$} \{0, 1\}^k; y \leftarrow x \oplus z \simeq_{\{x, y, z\}}^{\{z\}} y \xleftarrow{\$} \{0, 1\}^k; x \leftarrow y \oplus z$$

- Useful to relate probabilities

$$\frac{fv(A) \subseteq O \quad \models c_1 \simeq_O^I c_2 \quad m_1 =_I m_2}{\Pr[c_1, m_1 : A] = \Pr[c_2, m_2 : A]}$$

- Only a Partial Equivalence Relation

$$\models c \simeq_O^I c \quad \text{not true in general (obviously)}$$

- Generalizes information flow security (take $I = O = \mathcal{V}_{\text{low}}$)

Generalized relational equivalence

Probabilistic extension of Benton's Relational Hoare Logic

Definition

$$\models c_1 \sim c_2 : \Psi \Rightarrow \Phi \stackrel{\text{def}}{=} \forall m_1 m_2. m_1 \Psi m_2 \implies \llbracket c_1 \rrbracket m_1 \simeq_{\Phi} \llbracket c_2 \rrbracket m_2$$

$\mu_1 \simeq_{\Phi} \mu_2$ lifts relation Φ from memories to distributions.

$\mu_1 \simeq_{\Phi} \mu_2$ holds if there exists a distribution μ on $\mathcal{M} \times \mathcal{M}$ s.t.

- The 1st projection of μ coincides with μ_1
- The 2nd projection of μ coincides with μ_2
- Pairs with positive measure are in Φ

Proving program equivalence

Goal

$$\models c_1 \simeq'_O c_2$$

A Relational Hoare Logic generalized to arbitrary relations

$$\frac{\models c_1 \sim c_2 : \Phi \Rightarrow \Phi' \quad \models c'_1 \sim c'_2 : \Phi' \Rightarrow \Phi''}{\models c_1; c'_1 \sim c_2; c'_2 : \Phi \Rightarrow \Phi''} [\text{Seq}]$$

$$\frac{\models c_1 \sim c_2 : \Psi \Rightarrow \Phi \quad \models c_2 \sim c_3 : \Psi' \Rightarrow \Phi'}{\models c_1 \sim c_3 : \Psi \circ \Psi' \Rightarrow \Phi \circ \Phi'} [\text{Comp}]$$

...

Proving program equivalence

Goal

$$\vDash c_1 \simeq_O^I c_2$$

Mechanized program transformations

- Transformation: $T(c_1, c_2, I, O) = (c'_1, c'_2, I', O')$
- Soundness theorem

$$\frac{T(c_1, c_2, I, O) = (c'_1, c'_2, I', O') \quad \vDash c'_1 \simeq_{O'}^{I'} c'_2}{\vDash c_1 \simeq_O^I c_2}$$

- Reflection-based Coq tactic
(replace reasoning by computation)

Proving program equivalence

Goal

$$\models c_1 \simeq_O^I c_2$$

Mechanized program transformations

- Dead code elimination (`deadcode`)
- Constant folding and propagation (`ep`)
- Procedure call inlining (`inline`)
- Code movement (`swap`)
- Common suffix/prefix elimination (`eqobs_hd`, `eqobs_tl`)

Proving program equivalence

Goal

$$\models c \simeq_O^I c$$

An –incomplete– tactic for self-equivalence
(eqobs_in)

- Does $\models c \simeq_O^I c$ hold?
- Analyze dependencies to compute I' s.t. $\models c \simeq_O^{I'} c$
- Check that $I' \subseteq I$
- Think about type systems for information flow security

Reasoning about Failure Events

Lemma (Fundamental Lemma of Game-Playing)

Let A, B, F be events and G_1, G_2 be two games such that

$$\Pr[G_1 : A \wedge \neg F] = \Pr[G_2 : B \wedge \neg F]$$

Then, $|\Pr[G_1 : A] - \Pr[G_2 : B]| \leq \max(\Pr[G_1 : F], \Pr[G_2 : F])$

Automation

Syntactic Criterion

When $A = B$ and $F = \text{bad}$. If G_0, G_1 are syntactically identical except after program points setting bad e.g.

Game G_0 :

...
bad \leftarrow true; c_0
...

Game G_1 :

...
bad \leftarrow true; c_1
...

then

- $\Pr[G_0 : A \wedge \neg \text{bad}] = \Pr[G_1 : A \wedge \neg \text{bad}]$
- If game G_i (c_i) terminates with probability 1:
 $\Pr[G_{1-i} : \text{bad}] \leq \Pr[G_i : \text{bad}]$
- If both c_0, c_1 terminate absolutely:
 $\Pr[G_0 : \text{bad}] = \Pr[G_1 : \text{bad}]$

Automation

Syntactic Criterion

When $A = B$ and $F = \text{bad}$. If G_0, G_1 are syntactically identical except after program points setting bad e.g.

Game G_0 :

...
bad \leftarrow true; c_0
...

Game G_1 :

...
bad \leftarrow true; c_1
...

then

- $\Pr[G_0 : A \wedge \neg \text{bad}] = \Pr[G_1 : A \wedge \neg \text{bad}]$
- If game G_i (c_i) terminates with probability 1:
 $\Pr[G_{1-i} : \text{bad}] \leq \Pr[G_i : \text{bad}]$
- If both c_0, c_1 terminate absolutely:
 $\Pr[G_0 : \text{bad}] = \Pr[G_1 : \text{bad}]$

Failure Event lemma

Motivation: the Fundamental Lemma is typically applied in games where only oracles trigger bad.

- **IF** the probability of triggering bad in an oracle call can be bound as a function of the number of oracle calls so far
- **THEN** the probability of the whole game triggering bad can be bound provided the number of oracle calls is bounded

Failure Event Lemma (simplified)

Assume that $m(\text{bad}) = \text{false}$

- **IF** $\Pr[\mathcal{O}, m : \text{bad}] \leq p$ for every memory m such that $m(\text{bad}) = \text{false}$
- **THEN** $\Pr[\mathcal{G}, m : \text{bad}] \leq p q_{\mathcal{O}}$

Hypothesis holds for oracle

$\mathcal{O}(x) : y \stackrel{\$}{\leftarrow} T$; if $y = y_0$ then $\text{bad} \leftarrow \text{true}$ else ...

with $p = 1/|T|$

Application: PRP/PRF Switching Lemma

Game G_{RP} :

$L \leftarrow \text{nil}; b \leftarrow \mathcal{A}()$

Oracle $\mathcal{O}(x)$:

if $x \notin \text{dom}(L)$ then

$y \xleftarrow{\$} \{0, 1\}^\ell \setminus \text{ran}(L);$

$L \leftarrow (x, y) :: L$

return $L(x)$

Game G_{RF} :

$L \leftarrow \text{nil}; b \leftarrow \mathcal{A}()$

Oracle $\mathcal{O}(x)$:

if $x \notin \text{dom}(L)$ then

$y \xleftarrow{\$} \{0, 1\}^\ell;$

$L \leftarrow (x, y) :: L$

return $L(x)$

Suppose \mathcal{A} makes at most q queries to \mathcal{O} . Then

$$|\Pr[G_{RP} : b] - \Pr[G_{RF} : b]| \leq \frac{q(q-1)}{2^{\ell+1}}$$

- First introduced by Impagliazzo and Rudich in 1989
- Proof fixed by Bellare and Rogaway (2006) and Shoup (2004)

Proof

Game G_{RP} :

$\mathbf{L} \leftarrow \text{nil}; b \leftarrow \mathcal{A}()$

Oracle $\mathcal{O}(x)$:

if $x \notin \text{dom}(\mathbf{L})$ then

$y \xleftarrow{\$} \{0, 1\}^\ell$;

if $y \in \text{ran}(\mathbf{L})$ then ;

bad \leftarrow true;

$y \xleftarrow{\$} \{0, 1\}^\ell \setminus \text{ran}(\mathbf{L})$

$\mathbf{L} \leftarrow (x, y) :: \mathbf{L}$

return $\mathbf{L}(x)$

Game G_{RF} :

$\mathbf{L} \leftarrow \text{nil}; b \leftarrow \mathcal{A}()$

Oracle $\mathcal{O}(x)$:

if $x \notin \text{dom}(\mathbf{L})$ then

$y \xleftarrow{\$} \{0, 1\}^\ell$;

if $y \in \text{ran}(\mathbf{L})$ then ;

bad \leftarrow true

$\mathbf{L} \leftarrow (x, y) :: \mathbf{L}$

return $\mathbf{L}(x)$

$$|\Pr[G_{RP} : b] - \Pr[G_{RF} : b]| \leq \Pr[G_{RF} : \text{bad}]$$

Proof

Failure Event Lemma (less simplified)

Let k be a counter for \mathcal{O} and $m(\text{bad}) = \text{false}$:

- **IF** $\Pr[\mathcal{O}, m : \text{bad}] \leq f(m(k))$ for all memories m such that $m(\text{bad}) = \text{false}$

- **THEN** $\Pr[\mathcal{G}, m : \text{bad}] \leq \sum_{k=0}^{q_{\mathcal{O}}-1} f(k)$

Oracle $\mathcal{O}(x)$:

if $x \notin \text{dom}(\mathbf{L})$ then

$y \xleftarrow{\$} \{0, 1\}^{\ell}$; if $y \in \text{ran}(\mathbf{L})$ then $\text{bad} \leftarrow \text{true}$;

$\mathbf{L} \leftarrow (x, y) :: \mathbf{L}$

return $\mathbf{L}(x)$

- Prove that

$$\Pr[\mathcal{O}, m : \text{bad}] \leq \frac{|m(\mathbf{L})|}{2^{\ell}}$$

Eager/Lazy Sampling

- Interprocedural code motion
- Eager sampling: from an oracle to main game
- Lazy sampling: from main game to an oracle

Motivation

In crypto proofs

- Often need to know that some values are independent and uniformly distributed at some program point
- This holds when values can be resampled preserving semantics!

To prove correctness of eager and lazy sampling, we developed a logic for swapping statements

$$\models E, (c; S) \simeq E', (S; c')$$

Application: PRP/PRF Switching Lemma

Game G_{RF}^{eager} :

$\mathbf{L} \leftarrow \text{nil}; S; b \leftarrow \mathcal{A}()$

Oracle $\mathcal{O}(x)$:

if $x \notin \text{dom}(\mathbf{L})$ then
if $0 < |\mathbf{Y}|$ then
 $y \leftarrow \text{hd}(\mathbf{Y}); \mathbf{Y} \leftarrow \text{tl}(\mathbf{Y})$
else $y \leftarrow_{\$} \{0, 1\}^{\ell}$
 $\mathbf{L} \leftarrow (x, y) :: \mathbf{L}$
return $\mathbf{L}(x)$

where $S \stackrel{\text{def}}{=} \mathbf{Y} \leftarrow []$; while $|\mathbf{Y}| < q$ do $y \leftarrow_{\$} \{0, 1\}^{\ell}$; $\mathbf{Y} \leftarrow \mathbf{Y} ++ [y]$

Prove using the logic:

$$\models E_{RF}, (b \leftarrow \mathcal{A}()); S \equiv E_{RF}^{\text{eager}}, (S; b \leftarrow \mathcal{A}())$$

Prove by induction:

$$\Pr[G_{RF}; S : \text{bad}] = \Pr[G_{RF}^{\text{eager}} : \text{collision}] = \sum_{i=0}^{q-1} \frac{i}{2^{\ell}}$$

What does it take to trust a proof in CertiCrypt

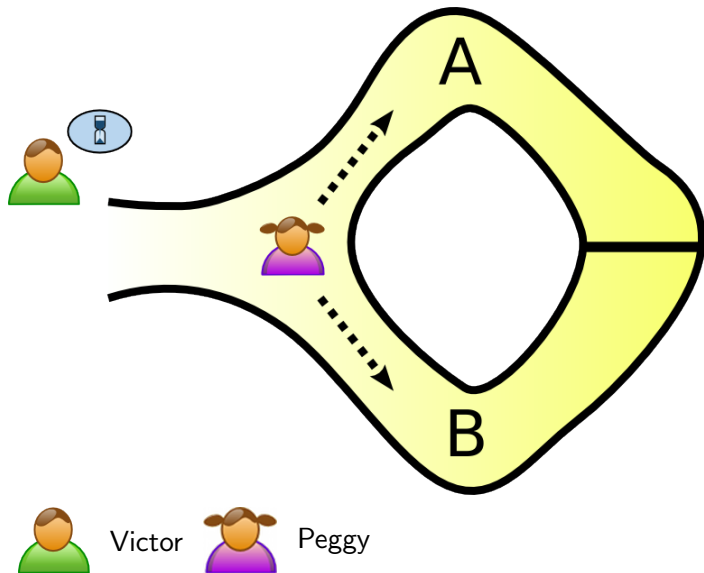
Verification is fully-automated!

(but proof construction is time-consuming)

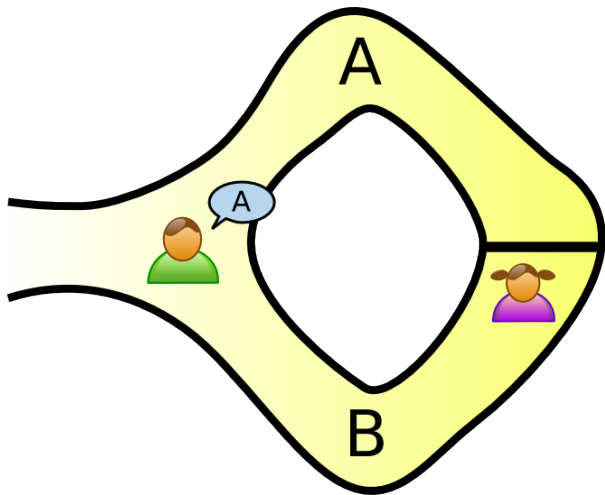
- You need to
 - trust the type checker of Coq
 - trust the language semantics
 - make sure the security statement (a few lines in Coq) is as expected
- You don't need to
 - understand or even read the proof
 - trust tactics, program transformations
 - trust program logics, wp-calculus
 - be an expert in Coq

Zero-Knowledge Proofs

Zero-Knowledge Proofs



Zero-Knowledge Proofs

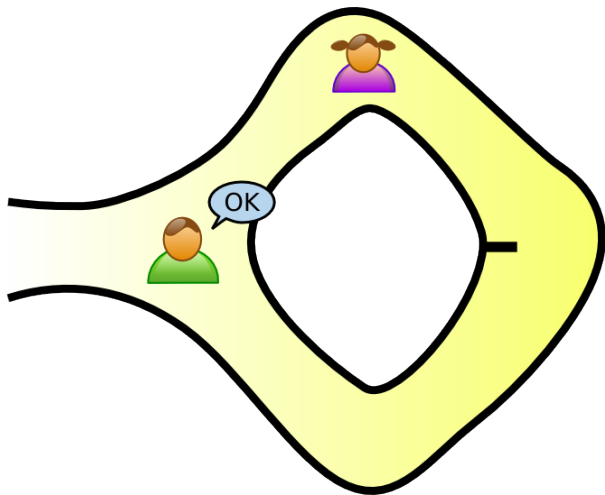


Victor



Peggy

Zero-Knowledge Proofs



Victor



Peggy

If you ever need to explain this to your kids

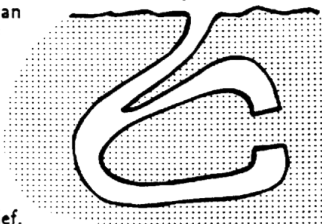
How to Explain Zero-Knowledge Protocols to your Children

Jean-Jacques Quisquater, Louis C. Guillou. CRYPTO'89

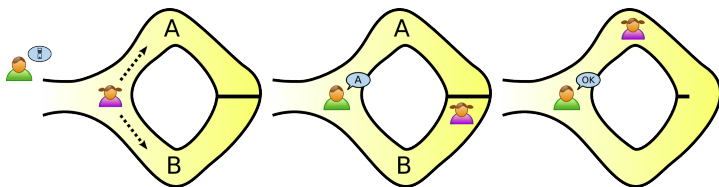
The Strange Cave of Ali Baba

winding passages: one to the left and the other to the right (The Entry of the Cave).

Ali Baba did not see which passage the thief ran into. Ali Baba had to choose which way to go, and he decided to go to the left. The left-hand passage ended in a dead end. Ali Baba searched all the way from the fork to the dead end, but he did not find the thief. Ali Baba said to himself that the thief was perhaps in the other passage. So he searched the right-hand passage, which also came to a dead end. But again he did not find the thief. "This cave is pretty strange," said Ali Baba to himself. "Where has my thief gone?"



Properties of Zero-Knowledge Proofs



- Completeness
A honest prover always convinces a honest verifier
- Soundness
A dishonest prover (almost) never convinces a verifier
- Zero-Knowledge
A verifier doesn't learn anything from playing the protocol

Formalizing Σ -Protocols

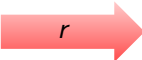
- Prover knows (x, w) s.t. $R(x, w)$ / Verifier knows only x




Prover



Verifier

Computes commitment r 

 Samples challenge c

Computes response s  Accepts/rejects response

Formalizing Σ -Protocols

- Prover knows (x, w) s.t. $R(x, w)$ / Verifier knows only x

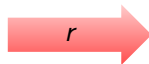


Prover



Verifier

$$(r, state) \leftarrow P_1(x, w)$$



$$c \xleftarrow{\$} C$$

$$s \leftarrow P_2(x, w, state, c)$$



$$b \leftarrow V_2(x, r, c, s)$$

Formalizing Σ -Protocols

A Σ -protocol is given by:

- Types for $x, w, r, s, state$
- A knowledge relation R
- A challenge set C
- Procedures P_1, P_2, V_2

The protocol can be seen as a program

```
protocol( $x, w$ ) :  
  ( $r, state$ )  $\leftarrow$   $P_1(x, w)$ ;  
   $c \xleftarrow{\$}$   $C$ ;  
   $s \leftarrow P_2(x, w, state, c)$ ;  
   $b \leftarrow V_2(x, r, c, s)$ 
```

Formalizing Σ -Protocols

Completeness

$$\forall x, w. R(x, w) \implies \Pr[\text{protocol}(x, w) : b = \text{true}] = 1$$

Soundness

There exists a polynomial time procedure KE s.t.

$$\left. \begin{array}{l} c_1 \neq c_2 \\ (x, r, c_1, s_1) \text{ accepting} \\ (x, r, c_2, s_2) \text{ accepting} \end{array} \right\} \implies \Pr[w \leftarrow \text{KE}(x, r, c_1, c_2, s_1, s_2) : R(x, w)] = 1$$

Honest-Verifier ZK vs. Special Honest-Verifier ZK

protocol(x, w) :

$(r, state) \leftarrow P_1(x, w);$

$c \xleftarrow{\$} C;$

$s \leftarrow P_2(x, w, state, c);$

$b \leftarrow V_2(x, r, c, s)$

protocol(x, w, c) :

$(r, state) \leftarrow P_1(x, w);$

$s \leftarrow P_2(x, w, state, c);$

$b \leftarrow V_2(x, r, c, s)$

Special Honest-Verifier ZK

$$\begin{aligned} \exists S. \forall x, w, c. R(x, w) &\implies \\ \models \text{protocol}(x, w, c) &\simeq_{\{r, c, s\}}^{\{x, c\}} (r, s) \leftarrow S(x, c) \end{aligned}$$

Honest-Verifier ZK

$$\begin{aligned} \exists S. \forall x, w. R(x, w) &\implies \\ \models \text{protocol}(x, w) &\simeq_{\{r, c, s\}}^{\{x\}} (r, c, s) \leftarrow S(x) \end{aligned}$$

Honest-Verifier ZK vs. Special Honest-Verifier ZK

protocol(x, w) :

$(r, state) \leftarrow P_1(x, w);$

$c \xleftarrow{\$} C;$

$s \leftarrow P_2(x, w, state, c);$

$b \leftarrow V_2(x, r, c, s)$

protocol(x, w, c) :

$(r, state) \leftarrow P_1(x, w);$

$s \leftarrow P_2(x, w, state, c);$

$b \leftarrow V_2(x, r, c, s)$

Special Honest-Verifier ZK

$$\begin{aligned} \exists S. \forall x, w, c. R(x, w) &\implies \\ \models \text{protocol}(x, w, c) &\simeq_{\{r, c, s\}}^{\{x, c\}} (r, s) \leftarrow S(x, c) \end{aligned}$$

Honest-Verifier ZK

$$\begin{aligned} \exists S. \forall x, w. R(x, w) &\implies \\ \models \text{protocol}(x, w) &\simeq_{\{r, c, s\}}^{\{x\}} (r, c, s) \leftarrow S(x) \end{aligned}$$

Σ^ϕ -Protocols

Let ϕ be a homomorphism from an additive group (\mathcal{G}, \oplus) to a multiplicative group (\mathcal{H}, \otimes)

$$\phi(a \oplus b) = \phi(a) \otimes \phi(b)$$

Homomorphism ϕ is special if there exists

- 1 a constant $v \in \mathbb{Z}$
- 2 a PPT-computable function $u : \mathcal{H} \rightarrow \mathcal{G}$

such that $\forall x \in \phi[\mathcal{G}]$

$$\phi(u(x)) = x^v$$

Σ^ϕ -Protocols

- A special homomorphism ϕ from an additive group (\mathcal{G}, \oplus) to a multiplicative group (\mathcal{H}, \otimes)
- $c^+ \in \mathbb{N}$ smaller than any prime divisor of special exponent v

This protocol is a ZK proof of knowledge of preimages of ϕ :

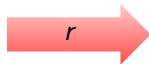
$$R = \{(x, w) \mid x = \phi(w)\}$$



Prover

$$y \xleftarrow{\$} \mathcal{G}; r \leftarrow \phi(y)$$

$$s \leftarrow y \oplus cw$$



Verifier

$$c \xleftarrow{\$} [0..c^+]$$

$$\phi(s) \stackrel{?}{=} r \otimes x^c$$

Formalized Σ^ϕ -Protocols

Protocol	$\mathcal{G} \rightarrow \mathcal{H}$	$\phi(x)$	$u(x)$	v
Schnorr	$\mathbb{Z}_q^+ \rightarrow \mathbb{Z}_p^*$	g^x	0	q
Okamoto	$(\mathbb{Z}_q^+, \mathbb{Z}_q^+) \rightarrow \mathbb{Z}_p^*$	$g_1^{x_1} \otimes g_2^{x_2}$	$(0, 0)$	q
Diffie-Hellman	$\mathbb{Z}_q^+ \rightarrow \mathbb{Z}_p^* \times \mathbb{Z}_p^*$	(g^x, g^{bx})	0	q
Fiat-Shamir	$\mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^*$	x^2	x	2
Guillou-Quisquater	$\mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^*$	x^e	x	e
Feige-Fiat-Shamir	$\{-1, 1\} \times \mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^*$	$s \cdot x^2$	$ x $	2

All these protocols are proved sound, complete and sHVZK in Coq

Combination of Σ^ϕ -protocols

Special homomorphisms are closed under direct product

Proof.

Special homomorphisms $\phi_1 : \mathcal{G}_1 \rightarrow \mathcal{H}_1$, $\phi_2 : \mathcal{G}_2 \rightarrow \mathcal{H}_2$

$$\begin{aligned}\phi & : \mathcal{G}_1 \times \mathcal{G}_2 \rightarrow \mathcal{H}_1 \times \mathcal{H}_2 \\ \phi(x_1, x_2) & = (\phi(x_1), \phi(x_2))\end{aligned}$$

- $v \stackrel{\text{def}}{=} \text{lcm}(v_1, v_2)$
- $u(x_1, x_2) \stackrel{\text{def}}{=} (u_1(x_1)^{v/v_1}, u_2(x_2)^{v/v_2})$



A cheap and efficient way of combining Σ^ϕ -protocols to prove knowledge of several preimages!

Combination of Σ^ϕ -protocols

Special homomorphisms are closed under direct product

Proof.

Special homomorphisms $\phi_1 : \mathcal{G}_1 \rightarrow \mathcal{H}_1$, $\phi_2 : \mathcal{G}_2 \rightarrow \mathcal{H}_2$

$$\begin{aligned}\phi & : \mathcal{G}_1 \times \mathcal{G}_2 \rightarrow \mathcal{H}_1 \times \mathcal{H}_2 \\ \phi(x_1, x_2) & = (\phi(x_1), \phi(x_2))\end{aligned}$$

- $v \stackrel{\text{def}}{=} \text{lcm}(v_1, v_2)$
- $u(x_1, x_2) \stackrel{\text{def}}{=} (u_1(x_1)^{v/v_1}, u_2(x_2)^{v/v_2})$



A cheap and efficient way of combining Σ^ϕ -protocols to prove knowledge of several preimages!

...which bring us to combining arbitrary Σ -protocols

Combination of Σ -Protocols

Given

- a Σ -protocol (P^1, V^1) for relation R_1
- a Σ -protocol (P^2, V^2) for relation R_2

Two basic ways of combining them. Given (x_1, x_2)

- AND-combination: prove knowledge of (w_1, w_2) such that

$$R_1(x_1, w_1) \text{ AND } R_2(x_2, w_2)$$

$$R \stackrel{\text{def}}{=} \{((x_1, x_2), (w_1, w_2)) \mid (x_1, w_1) \in R_1 \wedge (x_2, w_2) \in R_2\}$$

- OR-combination: prove knowledge of a w such that

$$R_1(x_1, w) \text{ OR } R_2(x_2, w)$$

without revealing which is the case

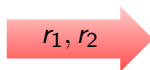
$$R \stackrel{\text{def}}{=} \{((x_1, x_2), w) \mid (x_1, w) \in R_1 \vee (x_2, w) \in R_2\}$$

AND-Combination

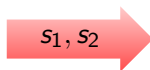


Prover

$$r_1 \leftarrow P_1^1(x_1, w_1)$$
$$r_2 \leftarrow P_1^2(x_2, w_2)$$



$$s_1 \leftarrow P_2^1(x_1, w_1, c)$$
$$s_2 \leftarrow P_2^2(x_2, w_2, c)$$



Verifier

$$c \xleftarrow{\$} \{0, 1\}^\ell$$

$$V_2^1(x_1, r_1, c, s_1) \wedge$$
$$V_2^2(x_2, r_2, c, s_2)$$

- Using the same challenge for both proofs is the key
- Only possible if protocols are Special HVZK

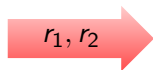
OR-Combination

Suppose w is a witness for x_1 , i.e. $R_1(x_1, w)$

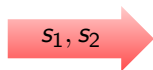


Prover

$$\begin{aligned} r_1 &\leftarrow P_1^1(x_1, w) \\ c_2 &\stackrel{\$}{\leftarrow} \{0, 1\}^\ell \\ (r_2, s_2) &\leftarrow S_2(x_2, c_2) \end{aligned}$$



$$\begin{aligned} c_1 &\leftarrow c_2 \oplus c \\ s_1 &\leftarrow P_2^1(x_1, w, c_1) \end{aligned}$$



Verifier

$$c \stackrel{\$}{\leftarrow} \{0, 1\}^\ell$$

$$\begin{aligned} c &= c_1 \oplus c_2 \wedge \\ &V_2^1(x_1, r_1, c, s_1) \wedge \\ &V_2^2(x_2, r_2, c, s_2) \end{aligned}$$

Sound w.r.t. $\{((x_1, x_2), w) \mid (x_1, w) \in R_1 \vee (x_2, w) \in R_2\}$

Conclusions

Summary of contributions

A language-based approach to computational crypto proofs

- Automated framework to formalize game-based proofs in Coq
- Probabilistic extension of Relational Hoare Logic
- Foundations for techniques used in crypto proofs
- Several case studies
 - PRP/PRF switching lemma
 - Chosen-plaintext security of ElGamal
 - Chosen-plaintext security of Hashed ElGamal in ROM and SM
 - Unforgeability of Full-Domain Hash signatures
 - Adaptive chosen-ciphertext security of OAEP
 - Σ -protocols
 - IBE (F. Olmedo), Golle-Juels (Z. Luo), BLS (M. Christofi)

The road ahead

- We fulfilled our goals, yet we don't believe cryptographers will use proofs assistants (or CertiCrypt) anytime soon
- Started to bridge gap between fully formal machine-checked proofs and pen-and-paper proof sketches
- What if we start building from the other side?

Start from a proof sketch and try to fill in the blanks and justify reasoning steps, building into the tool as much automation as possible. Record and highlight unjustified proof steps and let the user give finer-grained justifications—perhaps interactively, using automated tools.

