Formally Certifying the Security of Digital Signature Schemes

Santiago Zanella$^{1,2}$
Benjamin Grégoire$^{1,2}$
Gilles Barthe$^3$
Federico Olmedo$^3$

$^1$Microsoft Research - INRIA Joint Centre, France
$^2$INRIA Sophia Antipolis - Méditerranée, France
$^3$IMDEA Software, Madrid, Spain

30th IEEE Symposium on Security & Privacy
2009.05.19
Cryptanalysis-driven Security

Propose a cryptographic scheme

Wait for someone to come out with an attack
Cryptanalysis-driven Security

Propose a cryptographic scheme

Wait for someone to come out with an attack

Attack found
Patch the scheme
Cryptanalysis-driven Security

1. Propose a cryptographic scheme
2. Wait for someone to come out with an attack
3. Enough waiting
4. Declare the scheme secure

- Attack found
  - Patch the scheme
Cryptanalysis-driven Security

1. Propose a cryptographic scheme
2. Wait for someone to come out with an attack
3. Enough waiting
4. Declare the scheme secure

How much time is enough?

Attack found
Patch the scheme
Cryptanalysis-driven Security

Propose a cryptographic scheme

Wait for someone to come out with an attack

Enough waiting

Declare the scheme secure

Attack found
Patch the scheme

6 months, 1 year, 2 years?
Cryptanalysis-driven Security

Propose a cryptographic scheme

Wait for someone to come out with an attack

Attack found
Patch the scheme

Enough waiting

Declare the scheme secure

It took 5 years to break the Merkle-Hellman cryptosystem
Cryptanalysis-driven Security

Propose a cryptographic scheme

Wait for someone to come out with an attack

Enough waiting

Declare the scheme secure

Attack found
Patch the scheme

Ok, let’s say 7 years to be on the safe side
Cryptanalysis-driven Security

Propose a cryptographic scheme

Wait for someone to come out with an attack

Attack found
Patch the scheme

Enough waiting

Declare the scheme secure

It took 10 years to break the Chor-Rivest cryptosystem
Cryptanalysis-driven Security

Propose a cryptographic scheme

Wait for someone to come out with an attack

Enough waiting

Declare the scheme secure

Attack found
Patch the scheme

Can’t we do better?
Reductionist Cryptographic Proofs

1. Define a security goal and an adversarial model
2. Propose a cryptographic scheme
3. Reduce security of the scheme to a cryptographic assumption

IF an adversary $A$ 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 an efficient adversary $\mathcal{A}$ breaks the security of a scheme within time $t$
- Build an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to solve a computational hard problem within time $t + p(t)$
- We are interested in efficient reductions, were $p$ is a polynomial, so that

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

### Asymptotic Security
As long as $p(t)$ is polynomial, attacking the scheme is intractable provided the problem is intractable.

The smaller $p(t)$, the tighter the reduction

$p(t)$ matters

### Exact Security
What is the best known method to solve the problem?
If the best method solves the problem in time $t'$, choose scheme parameters so that the reduction yields a better method,

$t + p(t) < t'$
The Game-playing methodology

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
The Game-playing methodology

\textbf{Game } G_0^n : \\
\ldots \\
\ldots \leftarrow A(\ldots); \\
\ldots \\

Pr_{G_0^n}[A_0]
The Game-playing methodology

\[ \text{Game } G_0^n : \]
\[ \ldots \]
\[ \ldots \leftarrow A(\ldots); \]
\[ \ldots \]

\[ \Pr_{G_0^n}[A_0] \leq h_1(\Pr_{G_1^n}[A_1]) \]

\[ \text{Game } G_1^n : \]
\[ \ldots \]
\[ \ldots \]
The Game-playing methodology

\[
\begin{align*}
\text{Game } G_0^n : & \quad \ldots \\
& \quad \ldots \leftarrow A(\ldots); \\
& \quad \ldots \\
\text{Game } G_1^n : & \quad \ldots \\
& \quad \ldots \\
\text{Game } G_n^n : & \quad \ldots \\
& \quad \ldots \leftarrow B(\ldots); \\
\end{align*}
\]

\[
\Pr_{G_0^n}[A_0] \leq h_1(\Pr_{G_1^n}[A_1]) \leq \ldots \leq h_n(\Pr_{G_n^n}[A_n])
\]
The Game-playing methodology

Game $G_0^n$:

... $\leftarrow A(\ldots)$;
...

$Pr_{G_0^n}[A_0] \leq h_1(Pr_{G_1^n}[A_1]) \leq \ldots \leq h_n(Pr_{G_n^n}[A_n])$

Game $G_1^n$:

...
...
...

Game $G_n^n$:

... $\leftarrow B(\ldots)$;
...

Problem instance $\rightarrow$ Solution
CertiCrypt: language-based game-playing proofs

Formalize security definitions, assumptions and games using a probabilistic programming language.

**PWHILE**: a probabilistic programming language

\[ C ::= \begin{align*}
& \text{skip} \quad \text{nop} \\
& C; \ C \quad \text{sequence} \\
& V \leftarrow E \quad \text{assignment} \\
& V \leftarrow D \quad \text{random sampling} \\
& \text{if } E \text{ then } C \text{ else } C \quad \text{conditional} \\
& \text{while } E \text{ do } C \quad \text{while loop} \\
& V \leftarrow P(E, \ldots, E) \quad \text{procedure call}
\end{align*} \]

- \( x \leftarrow d \): sample the value of \( x \) according to distribution \( d \)
- The language of expressions (\( E \)) and distribution expressions (\( D \)) admits user-defined extensions
Computing probabilities

\[
\left[ G^\eta \right] : \mathcal{M} \rightarrow (\mathcal{M} \rightarrow [0, 1]) \rightarrow [0, 1]
\]

- Interpret \( \left[ G^\eta \right] m \) as the expectation operator of the probability distribution induced by the game

- Probability: \( \text{Pr}_{G^\eta, m}[A] \overset{\text{def}}{=} \left[ G \right]^\eta m \mathbb{1}_A \)

Example.

Let \( G \overset{\text{def}}{=} x \leftarrow \{0, 1\}; y \leftarrow \{0, 1\} \)

\( \text{Pr}_{G^\eta, m}[x \neq y] = \left[ G \right]^\eta m \mathbb{1}_{x \neq y} = \)
Computing probabilities

$$[G^n] : \mathcal{M} \rightarrow (\mathcal{M} \rightarrow [0, 1]) \rightarrow [0, 1]$$

- Interpret $[G^n] m$ as the expectation operator of the probability distribution induced by the game
- Probability: $\Pr_{G^n,m}[A] \overset{\text{def}}{=} [G]^n m \mathbb{1}_A$

Example.

Let $G \overset{\text{def}}{=} x \leftarrow \{0, 1\}; \ y \leftarrow \{0, 1\}$

$$\Pr_{G^n,m}[x \neq y] = [G]^n m \mathbb{1}_{x \neq y} =$$

$$\frac{1}{4} \mathbb{1}_{x \neq y} (m[x \mapsto 0, y \mapsto 0]) + \frac{1}{4} \mathbb{1}_{x \neq y} (m[x \mapsto 0, y \mapsto 1]) + \frac{1}{4} \mathbb{1}_{x \neq y} (m[x \mapsto 1, y \mapsto 0]) + \frac{1}{4} \mathbb{1}_{x \neq y} (m[x \mapsto 1, y \mapsto 1])$$
Computing probabilities

\[ [G^n] : \mathcal{M} \rightarrow (\mathcal{M} \rightarrow [0, 1]) \rightarrow [0, 1] \]

- Interpret \([G^n] m\) as the expectation operator of the probability distribution induced by the game
- Probability: \(\Pr_{G^n, m}[A] \stackrel{\text{def}}{=} [G]^n m \mathbb{1}_A\)

**Example.**

Let \(G \stackrel{\text{def}}{=} x \leftarrow \{0, 1\}; \ y \leftarrow \{0, 1\}\)

\[
\Pr_{G^n, m}[x \neq y] = [G]^n m \mathbb{1}_{x \neq y} = \\
0 + \frac{1}{4} + \\
\frac{1}{4} + 0
\]
Computing probabilities

\[
[G^\eta] : \mathcal{M} \to (\mathcal{M} \to [0, 1]) \to [0, 1]
\]

- Interpret \([G^\eta] m\) as the expectation operator of the probability distribution induced by the game.
- Probability: \(\Pr_{G^\eta, m}(A) \overset{\text{def}}{=} [G]^\eta m \mathbb{1}_A\)

**Example.**

Let \(G \overset{\text{def}}{=} x \xleftarrow{\$} \{0, 1\}; y \xleftarrow{\$} \{0, 1\}\)

\[
\Pr_{G^\eta, m}(x \neq y) = [G]^\eta m \mathbb{1}_{x \neq y} = \frac{1}{2}
\]
Program equivalence

**Observational equivalence**

\[ f =_X g \quad \text{def} \quad \forall m_1, m_2, m_1 =_X m_2 \implies f \, m_1 = g \, m_2 \]

\[ \models G_1 \models^I_O G_2 \quad \text{def} \quad \forall m_1, m_2, f, g, m_1 =_I m_2 \land f =^O g \implies [G_1] \, m_1 \, f = [G_2] \, m_2 \, g \]

- Only a Partial Equivalence Relation
  \[ \models G \models^I_O G \quad \text{not true in general} \]
- Generalizes information flow security (take \( I = O = \mathcal{V}_{\text{low}} \))

**Example**

\[ \models x \leftarrow^S \{0, 1\}^k; y \leftarrow x \oplus z \sim \{z\}_{\{x, y, z\}}; y \leftarrow^S \{0, 1\}^k; x \leftarrow y \oplus z \]
Program equivalence

### Observational equivalence

\[ f \equiv_X g \quad \text{def} \quad \forall m_1, m_2, \quad m_1 \equiv_X m_2 \implies f \cdot m_1 = g \cdot m_2 \]

\[ \models G_1 \sim^I_O G_2 \quad \text{def} \quad \forall m_1, m_2, \quad f \equiv g \quad \land \quad m_1 \equiv I m_2 \implies \llbracket G_1 \rrbracket m_1 f = \llbracket G_2 \rrbracket m_2 g \]

- Only a Partial Equivalence Relation
  - \[ \models G \sim^I_O G \quad \text{not true in general} \]
- Generalizes information flow security (take \( I = O = \mathcal{V}_{\text{low}} \))

### Example

\[ \models x \leftarrow \{0, 1\}^k; \quad y \leftarrow x \oplus z \sim^{\{z\}}_{\{x,y,z\}} y \leftarrow \$ \{0, 1\}^k; \quad x \leftarrow y \oplus z \]
Using program equivalence to relate probabilities

Let $A$ be an event that depends only on variables in $O$

To prove $\Pr_{G_1,m_1} [A] = \Pr_{G_2,m_2} [A]$ it suffices to find a set of variables $I$ such that

1. $m_1 =_I m_2$
2. $\models G_1 \simeq^I_O G_2$
Goal
\[ \vdash G_1 \sim^I O G_2 \]

A Relational Hoare Logic

\[ \vdash c_1 \sim c_2 : \Phi \Rightarrow \Phi' \quad \vdash c'_1 \sim c'_2 : \Phi' \Rightarrow \Phi'' \]
\[ \vdash c_1 ; c'_1 \sim c_2 ; c'_2 : \Phi \Rightarrow \Phi'' \] [R-Seq]

\[ \ldots \]
Proving program equivalence

Goal
\[ \vdash G_1 \simeq_{O}^I G_2 \]

Mechanized program transformations

- Transformation: \( T(G_1, G_2, I, O) = (G'_1, G'_2, I', O') \)
- Soundness theorem

\[
T(G_1, G_2, I, O) = (G'_1, G'_2, I', O') \implies G'_1 \simeq_{O'}^I G'_2
\]

\[ \vdash G_1 \simeq_{O}^I G_2 \]

- Reflection-based Coq tactic
  (replace reasoning by computation)
Proving program equivalence

Goal
\[ \succeq G_1 \simeq^I_O G_2 \]

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
\[ \vdash G \simeq^I_O G \]

An –incomplete– tactic for self-equivalence

(eqobs_in)

Does \[ \vdash G \simeq^I_O G \] hold?

Analyze dependencies to compute \( I' \) s.t. \[ \vdash G \simeq^{I'}_O G \]

Check that \( I' \subseteq I \)

Think about information flow security...
The Fundamental Lemma of Game-Playing

Fundamental lemma

If two games $G_1$ and $G_2$ behave identically in an initial memory $m$ unless a failure event “bad” fires, then

$$|\Pr_{G_1,m}[A] - \Pr_{G_2,m}[A]| \leq \Pr_{G_1,2}[\text{bad}]$$
The Fundamental Lemma of Game-Playing

Syntactic criterion

\[
\begin{align*}
\textbf{Game} & \ G_1 : \\
& \ldots \\
& \text{bad} \leftarrow \text{true}; \ c_1 \\
& \ldots \\
\textbf{Game} & \ G_2 : \\
& \ldots \\
& \text{bad} \leftarrow \text{true}; \ c_2 \\
& \ldots \\
\end{align*}
\]

- \( \Pr_{G_1,m}[A \mid \neg \text{bad}] = \Pr_{G_2,m}[A \mid \neg \text{bad}] \)
- \( \Pr_{G_1,m}[\text{bad}] = \Pr_{G_2,m}[\text{bad}] \)

Corollary

\[ |\Pr_{G_1,m}[A] - \Pr_{G_2,m}[A]| \leq \Pr_{G_1,2}[\text{bad}] \]
A digital signature scheme is composed of three algorithms ($KG$, $\text{Sign}$, $\text{Verify}$)

**Key generation** : \((pk, sk) \leftarrow KG(\eta : \mathbb{N})\)
- \(sk\) is the **private** signing key
- \(pk\) is the **public** verification key

**Signing** : \(\sigma \leftarrow \text{Sign}(sk, m)\)

**Verification** : \(0/1 \leftarrow \text{Verify}(pk, m, \sigma)\)

\[\forall m, \text{Verify}(pk, m, \text{Sign}(sk, m)) = 1\]
The Full-Domain Hash Signature Scheme

Consider

- A family of \textit{oneway trapdoor} permutations \((\mathcal{KG}_f, f, f^{-1})\) on a cyclic group \(\mathcal{G}_\eta\) (e.g. RSA)
- A family of hash functions \(H_\eta : \{0, 1\}^* \rightarrow \mathcal{G}_\eta\) (e.g. SHA-1)

The Full-Domain Hash scheme is defined as follows

\[
\eta \quad \mathcal{KG} \quad (pk, sk) = \mathcal{KG}_f(\eta)
\]

\[
sk \quad \text{Sign} \quad \sigma = f_{sk}^{-1}(H(m))
\]

\[
pk \quad \text{Verify} \quad \text{if } f_{pk}(\sigma) = H(m) \text{ then } 1 \text{ else } 0
\]
Existential Unforgeability

We want a signature for a message \( m \) to be hard to forge. Even if...

- ...the adversary knows the signatures of \textit{many} messages
- ...the adversary chose those messages
- ...the adversary gets to choose \( m \)

**Definition (Existential unforgeability)**

No efficient adversary \( \mathcal{A} \) with access to a signing oracle \( \text{Sign}(sk, \cdot) \) can forge a fresh signature for a message of its choice.

\[
\Pr \left[ \begin{array}{l}
(pk, sk) \leftarrow \mathcal{K}_G(\eta); \\
(m, \sigma) \leftarrow \mathcal{A}_{\text{Sign}(sk, \cdot)}(pk)
\end{array} \right] \quad \text{Verify}(pk, m, \sigma) = 1 \land m \text{ is fresh} \leq \epsilon(\eta)
\]
Existential Unforgeability as a game

**Game** $G^\eta_{\text{EF}}$:
- $S \leftarrow \text{nil}$;
- $(pk, sk) \leftarrow \mathcal{KG}(\eta)$;
- $(m, \sigma) \leftarrow \mathcal{A}(pk)$;
- $h \leftarrow \text{H}(m)$

**Oracle** $H(m) \overset{\text{def}}{=} \text{return } H^\eta(m)$

**Oracle** $\text{Sign}(m) \overset{\text{def}}{=} $ 
- $S \leftarrow m :: S$;
- return $f^{-1}_{sk}(\text{H}(m))$

$\forall \mathcal{A}, \quad \Pr [G^\eta_{\text{EF}} \mid f_{pk}(\sigma) = h \land m \notin S] \leq \epsilon(\eta)$
**Existential Unforgeability as a game**

<table>
<thead>
<tr>
<th><strong>Game</strong> $G_{\text{EF}}^\eta$</th>
<th><strong>Oracle</strong> $H(m)$ def $H_\eta(m)$ return $H_\eta(m)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \leftarrow \text{nil};$</td>
<td><strong>Oracle</strong> $\text{Sign}(m)$ def $S \leftarrow m :: S;$ return $f_{sk}^{-1}(H(m))$</td>
</tr>
<tr>
<td>$(pk, sk) \leftarrow \mathcal{KG}(\eta);$</td>
<td></td>
</tr>
<tr>
<td>$(m, \sigma) \leftarrow A(pk);$</td>
<td></td>
</tr>
<tr>
<td>$h \leftarrow H(m)$</td>
<td></td>
</tr>
</tbody>
</table>

$$\forall A, \forall pk \; sk, \Pr \left[ G_{\text{EF}}^\eta \mid f_{pk}(\sigma) = h \land m \notin S \right] \leq \epsilon(\eta)$$

For most signature schemes (including FDH) we can exhibit a reduction independent of the way $(pk, sk)$ are generated.
Formalizing assumptions

- \((\mathcal{K}G_f, f, f^{-1})\) is a family of one-way trapdoor permutations

**Game** \(G^n_{OW}\):

\[
\begin{align*}
(pk, sk) & \leftarrow \mathcal{K}G_f(\eta); \\
y & \leftarrow G; \\
x & \leftarrow \mathcal{I}(pk, y)
\end{align*}
\]

\[\forall \mathcal{I}, \Pr[G^n_{OW} \mid x = f^{-1}_{sk}(y)] \text{ is negligible}\]
Formalizing assumptions

- \((\mathcal{KG}_f, f, f^{-1})\) is a family of one-way trapdoor permutations

\[
\text{Game } \mathcal{G}_{\mathcal{OW}}^\eta : \\
(pk, sk) \leftarrow \mathcal{KG}_f(\eta); \\
y \leftarrow \mathcal{G}; \\
x \leftarrow \mathcal{I}(pk, y)
\]

\[\forall \mathcal{I}, \Pr[\mathcal{G}_{\mathcal{OW}}^\eta | x = f_{sk}^{-1}(y)] \text{ is negligible}\]

- Random Oracle Model \((H_\eta \text{ behaves as a random function})\)

\[
\text{Oracle } H(m) \overset{\text{def}}{=} \\
\text{return } H_\eta(m)
\]

\[
\text{Oracle } H(m) \overset{\text{def}}{=} \\
\text{if } m \notin \text{dom}(L) \text{ then} \\
h \leftarrow \mathcal{G}; L \leftarrow (m, h) :: L \\
\text{return } L(m)
\]
Code-based proof of unforgeability of FDH

\[ \text{Game } G_{\text{EF}}^\eta : \]
\[ S \leftarrow \text{nil}; \]
\[ (m, \sigma) \leftarrow A(pk); \]
\[ h \leftarrow H(m) \]

\[ \Pr_{G_{\text{EF}}^\eta} [f_{pk}(\sigma) = h \land m \notin S] \leq \cdots \leq h(\Pr_{G_{\text{OW}}^\eta} [x = f_{sk}^{-1}(x)]) \]

- The probability loss (given by \( h \)) depends on the sequence of games of the reduction
- For some inverters there exist tighter reductions than for others
- Some inverters have a larger simulation overhead than others

\[ \text{Game } G_{\text{OW}}^\eta : \]
\[ y \leftarrow G; \]
\[ x \leftarrow I(pk, y) \]
Existential unforgeability of FDH

Consider an adversary $A$ s.t.
- $A$ makes at most $q_H(\eta)$ hash queries
- $A$ makes at most $q_S(\eta)$ signature queries

Suppose
- $A$ runs within time $t(\eta)$
- $A$ forges a signature with probability $\epsilon(\eta)$
  i.e. $\epsilon(\eta) = \Pr_{G^{\eta}_{EF}}[f_{pk}(\sigma) = h \land m \notin S]$

We show two different inverters $I$ that use $A$ to invert the trapdoor permutation $f$
- The first admits a simple, suboptimal reduction
- The second admits an optimal reduction, due to Coron
Unforgeability of FDH – suboptimal bound

Theorem

There exists an $I$ that inverts $f$ with probability $\epsilon'(\eta)$ within time $t'(\eta)$, where

$$
\epsilon'(\eta) \geq (q_H(\eta) + q_S(\eta) + 1)^{-1} \epsilon(\eta)
$$

$$
t'(\eta) \leq t(\eta) + (q_H(\eta) + q_S(\eta)) \Theta(T_f)
$$
Unforgeability of FDH – suboptimal bound

**Game** $G_{OW}$:
- $y \leftarrow G$;
- $x \leftarrow I(y)$

$I(y) \overset{\text{def}}{=} y'$
- $j \leftarrow [0..q_H + q_S]$;
- $i \leftarrow 0$;
- $P, L \leftarrow \text{nil}$;
- $(m, \sigma) \leftarrow A()$;
- return $\sigma$

**Oracle** $H(m) \overset{\text{def}}{=} $ if $m \notin \text{dom}(L)$ then
  - if $i = j$ then $h \leftarrow y'$;
  - else $r \leftarrow G$; $h \leftarrow f_{pk}(r)$
  - $P \leftarrow (m, r) :: P$;
  - $L \leftarrow (m, h) :: L$;
  - $i \leftarrow i + 1$
- return $L(m)$

**Oracle** $\text{Sign}(m) \overset{\text{def}}{=} h \leftarrow H(m)$; return $P(m)$

- Inverter succeeds when $m$ is the $j$-th hash query
- That occurs with probability $(q_H(\eta) + q_S(\eta) + 1)^{-1}$
- Overhead is just one extra $f$ computation per hash call
- Signing is simulated without knowing $sk$, $I$ keeps the preimages under $f$ of all but the $j$-th hash value
Theorem

Assume \( f \) is homomorphic w.r.t. the group operation. There exists an \( \mathcal{I} \) that inverts \( f \) with probability \( \epsilon'(\eta) \) within time \( t'(\eta) \), where

\[
\epsilon'(\eta) \geq \frac{1}{q_s(\eta) + 1} \left(1 - \frac{1}{q_s(\eta) + 1}\right)^{q_s(\eta)} \epsilon(\eta) \\
\approx \exp(-1) q_s(\eta)^{-1} \epsilon(\eta) \\
t'(\eta) \leq t(\eta) + (q_H(\eta) + q_S(\eta)) \Theta(T_f)
\]
Unforgeability of FDH – optimal bound

| Game $G_{OW}$: | Oracle $H(m) \overset{\text{def}}{=} \begin{cases} 
  & \text{if } m \notin \text{dom}(L) \text{ then} \\
  & \quad r \gets G; \\
  & \quad \text{if } T(i) \text{ then } h \gets y' \times f(r) \\
  & \quad \text{else } h \gets f(r) \\
  & \quad P \gets (m, r) :: P; \\
  & \quad L \gets (m, h) :: L; \\
  & \quad i \gets i + 1 \\
\end{cases} \\
| \begin{align*} 
  & y \gets G; \\
  & x \gets \mathcal{I}(y) \\
  & \mathcal{I}(y) \overset{\text{def}}{=} y' \gets y; \\
  & T \gets \text{nil}; \text{ Init}_T; \\
  & i \gets 0; \\
  & P, L \gets \text{nil}; \\
  & (m, \sigma) \gets A(); \\
  & h \gets H(m); \\
\end{align*} | return $L(m)$ |

Init$_T \overset{\text{def}}{=} \text{while } |T| \leq q \text{ do } (b \overset{\$}{\gets} \langle \text{true} \leftrightarrow p, \text{false} \leftrightarrow 1 - p \rangle; \ T \gets b :: T)$

- Each entry in $T$ is true with probability $p$
- Inverter succeeds when
  - The $T$-entry for $m$ is true
  - The $T$-entries of messages in sign queries are all false
- That occurs with probability $p \left(1 - p\right)^{q_S(\eta)}$
### Unforgeability of FDH – optimal bound

| Game $G_{OW}$: | Oracle $H(m)$ $\overset{\text{def}}{=}\
| y \leftarrow G; |$ if $m \not\in \text{dom}(L)$ then \\
| x \leftarrow \mathcal{I}(y) | \quad r \leftarrow G; \\
| \mathcal{I}(y) \overset{\text{def}}{=} | \quad \text{if } T(i) \text{ then } h \leftarrow y' \times f(r) \\
| y' \leftarrow y; | \quad \text{else } h \leftarrow f(r) \\
| T \leftarrow \text{nil}; \text{ Init}_T; | P \leftarrow (m, r) :: P; \\
| i \leftarrow 0; | L \leftarrow (m, h) :: L; \\
| P, L \leftarrow \text{nil}; | i \leftarrow i + 1 \\
| (m, \sigma) \leftarrow \mathcal{A}(); | \text{return } L(m) \\
| h \leftarrow H(m); | $\text{Oracle } \text{Sign}(m) \overset{\text{def}}{=}\
| $\text{return } \sigma \times P(m)^{-1}$ | $h \leftarrow H(m); \text{ return } P(m)$ |

Init$_T \overset{\text{def}}{=} \text{ while } |T| \leq q \text{ do } (b \leftarrow \langle \text{true} \leftrightarrow p, \text{false} \leftrightarrow 1 - p \rangle; \ T \leftarrow b :: T)$

- Indeed, thanks to the homomorphic property of $f$,
  
  $$h = f_{pk}(\sigma) \implies y \times P(m) = f_{pk}(\sigma)$$
  
  $$\implies f_{sk}^{-1}(y \times P(m)) = \sigma$$
  
  $$\implies f_{sk}^{-1}(y) = \sigma \times P(m)^{-1}$$
Unforgeability of FDH – optimal bound

| Game $G_{OW}$: | Oracle $H(m)$ \( \overset{\text{def}}{=} \)
|-----------------|-----------------------------
| $y \leftarrow \mathcal{G};$ | if $m \not\in \text{dom}(L)$ then
| $x \leftarrow \mathcal{I}(y)$ | $r \leftarrow \mathcal{G};$
| $\mathcal{I}(y) \overset{\text{def}}{=} $ | if $T(i)$ then $h \leftarrow y' \times f(r)$
| $y' \leftarrow y;$ | else $h \leftarrow f(r)$
| $T \leftarrow \text{nil};$ | $P \leftarrow (m, r) :: P;$
| $\text{Init}_T;$ | $L \leftarrow (m, h) :: L;$
| $i \leftarrow 0;$ | $i \leftarrow i + 1$
| $P, L \leftarrow \text{nil};$ | return $L(m)$
| $(m, \sigma) \leftarrow \mathcal{A}();$ | Oracle $\text{Sign}(m) \overset{\text{def}}{=} $
| $h \leftarrow H(m);$ | $h \leftarrow H(m);$ return $P(m)$
| return $\sigma \times P(m)^{-1}$ | 

$\text{Init}_T \overset{\text{def}}{=} \text{while } |T| \leq q \text{ do } (b \leftarrow \langle \text{true} \leftrightarrow p, \text{false} \leftrightarrow 1 - p \rangle; \; T \leftarrow b :: T)$

- Overhead is just one extra $f$ computation and one group operation per hash call
- The bound is maximized for $p = (q_S(H) + 1)^{-1}$
Assume a reasonable bound on the number of hash queries, e.g. $q_H \leq 2^{60}$

Assume a reasonable bound on the number of sign queries, e.g. $q_S \leq 2^{20}$

Note that the owner of this private key can enforce this limit

You want a reduction to yield a method to invert RSA better than the best known method

The best known method to invert RSA is to factor the modulus

The best known method to factor large integers is the Number Field Sieve
The overhead is the same (up to constant factors) in both reductions: \((q_H + q_S) T_f \approx 2^{60} T_f\), for RSA \(T_f = O(|n|^2)\).

To invert \(f\) with probability close to 1, the first inverter has to be iterated \(q_H + q_S + 1 \approx 2^{60}\) times, the second has to be iterated only \(\exp(1) q_S \approx 2^{22}\) times.

<table>
<thead>
<tr>
<th>Modulus size</th>
<th>NFS</th>
<th>First reduction</th>
<th>Optimal reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>(2^{58})</td>
<td>(2^{60} t + 2^{138})</td>
<td>(2^{22} t + 2^{100})</td>
</tr>
<tr>
<td>1024</td>
<td>(2^{80})</td>
<td>(2^{60} t + 2^{140})</td>
<td>(2^{22} t + 2^{102})</td>
</tr>
<tr>
<td>2048</td>
<td>(2^{111})</td>
<td>(2^{60} t + 2^{142})</td>
<td>(2^{22} t + 2^{104})</td>
</tr>
<tr>
<td>4096</td>
<td>(2^{149})</td>
<td>(2^{60} t + 2^{144})</td>
<td>(2^{22} t + 2^{106})</td>
</tr>
</tbody>
</table>
Practical Interpretation for RSA-FDH

- The overhead is the same (up to constant factors) in both reductions: \((q_H + q_S) T_f \approx 2^{60} T_f\), for RSA \(T_f = O(|n|^2)\).
- To invert \(f\) with probability close to 1, the first inverter has to be iterated \(q_H + q_S + 1 \approx 2^{60}\) times, the second has to be iterated only \(\exp(1) q_S \approx 2^{22}\) times.

<table>
<thead>
<tr>
<th>Modulus size</th>
<th>NFS</th>
<th>First reduction</th>
<th>Optimal reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>(2^{58})</td>
<td>(2^{140})</td>
<td>(2^{102})</td>
</tr>
<tr>
<td>1024</td>
<td>(2^{80})</td>
<td>(2^{141})</td>
<td>(2^{103})</td>
</tr>
<tr>
<td>2048</td>
<td>(2^{111})</td>
<td>(2^{142})</td>
<td>(2^{104})</td>
</tr>
<tr>
<td>4096</td>
<td>(2^{149})</td>
<td>(2^{144})</td>
<td>(2^{106})</td>
</tr>
</tbody>
</table>

- For \(t = 2^{80}\), the optimal reduction allows to use a modulus half as large as the original reduction would suggest.
What does it take to trust a proof in CertiCrypt

Proof verification is fully-automated!
(but proof construction is still time-consuming)

- **You need to**
  - trust the type checker of Coq
  - trust the definition of the language semantics
  - make sure the security statement (a few lines in Coq) is what you expect it to be

- **You don’t need to**
  - understand or even read the proof
  - trust proof tactics, program transformations
  - trust program logics, wp-calculus
  - be an expert in Coq