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

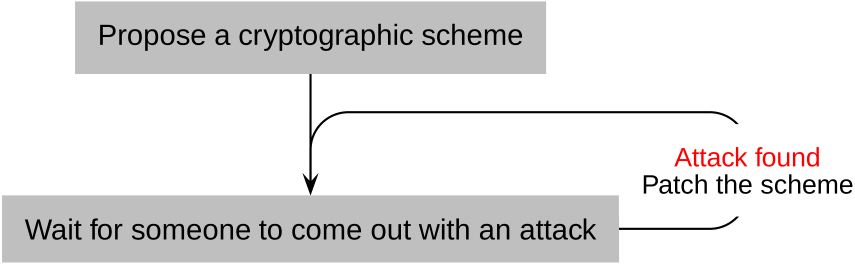
Cryptanalysis-driven Security

Propose a cryptographic scheme

↓

Wait for someone to come out with an attack

Cryptanalysis-driven Security



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

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

How much time is *enough*?

Cryptanalysis-driven Security

Propose a cryptographic scheme

Wait for someone to come out with an attack

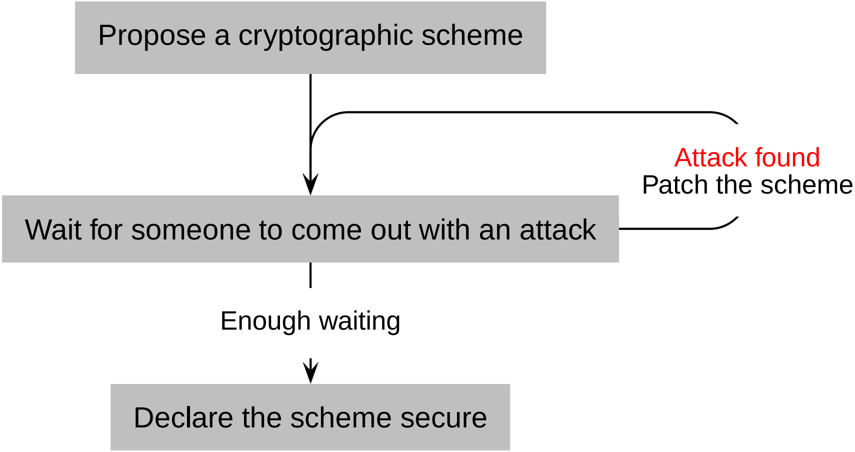
Attack found
Patch the scheme

Enough waiting

Declare the scheme secure

6 months, 1 year, 2 years?

Cryptanalysis-driven Security



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

Attack found
Patch the scheme

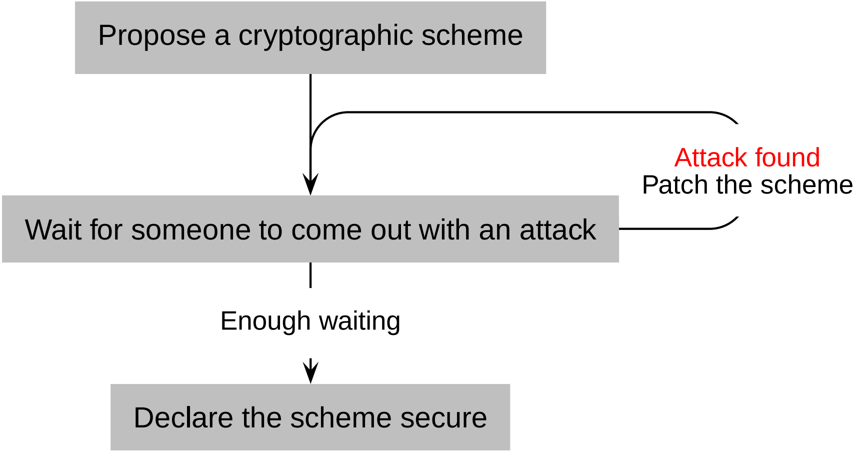
Enough waiting

↓

Declare the scheme secure

Ok, let's say 7 years to be on the safe side

Cryptanalysis-driven Security



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

Attack found
Patch the scheme

Enough waiting

Declare the scheme secure

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 $\mathcal{A}$ 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 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

Game G_0^η :

$\ldots$

$\ldots \leftarrow \mathcal{A}(\ldots);$

$\ldots$

$\Pr_{G_0^\eta}[A_0]$

The Game-playing methodology

Game G_0^η :

$\ldots$

$\ldots \leftarrow \mathcal{A}(\ldots);$

$\ldots$

Game G_1^η :

$\ldots$

$\ldots$

$\ldots$

$$\Pr_{G_0^\eta}[A_0] \quad \leq \quad h_1(\Pr_{G_1^\eta}[A_1])$$

The Game-playing methodology

Game G_0^η :
$\ldots$
$\ldots \leftarrow \mathcal{A}(\ldots);$
$\ldots$

Game G_1^η :
$\ldots$
$\ldots$
$\ldots$

$\cdots$

Game G_n^η :
$\ldots$
$\ldots \leftarrow \mathcal{B}(\ldots);$
$\ldots$

$$\Pr_{\mathsf{G}_0^\eta}[A_0] \quad \leq \quad h_1(\Pr_{\mathsf{G}_1^\eta}[A_1]) \quad \leq \ \ldots \ \leq \quad h_n(\Pr_{\mathsf{G}_n^\eta}[A_n])$$

The Game-playing methodology

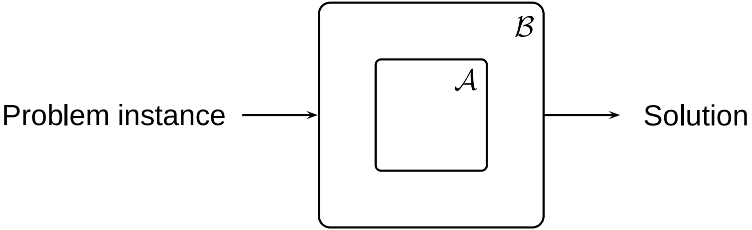
Game G_0^η :
$\ldots$
$\ldots \leftarrow \mathcal{A}(\ldots);$
$\ldots$

Game G_1^η :
$\ldots$
$\ldots$
$\ldots$

$\cdots$

Game G_n^η :
$\ldots$
$\ldots \leftarrow \mathcal{B}(\ldots);$
$\ldots$

$$\Pr_{\mathsf{G}_0^\eta}[A_0] \leq h_1(\Pr_{\mathsf{G}_1^\eta}[A_1]) \leq \ldots \leq h_n(\Pr_{\mathsf{G}_n^\eta}[A_n])$$



CertiCrypt: language-based game-playing proofs

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

pWHILE: a probabilistic programming language

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

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

Computing probabilities

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

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

Example.

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

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

Computing probabilities

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

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

Example.

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

$$\begin{aligned}&\Pr_{\mathsf{G}^\eta,m}[x \neq y] = [\![\mathsf{G}]\!]^\eta\ m\ \mathbb{1}_{x\neq y} = \\ &\tfrac{1}{4}\ \mathbb{1}_{x\neq y}(m[x \mapsto 0, y \mapsto 0]) \quad + \quad \tfrac{1}{4}\ \mathbb{1}_{x\neq y}(m[x \mapsto 0, y \mapsto 1]) \quad + \\ &\tfrac{1}{4}\ \mathbb{1}_{x\neq y}(m[x \mapsto 1, y \mapsto 0]) \quad + \quad \tfrac{1}{4}\ \mathbb{1}_{x\neq y}(m[x \mapsto 1, y \mapsto 1])\end{aligned}$$

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: $\Pr_{G^\eta,m}[A] \stackrel{\text{def}}{=} [\![G]\!]^\eta\ m\ \mathbb{1}_A$

Example.

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

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

$$0 \quad + \quad \frac{1}{4} \quad +$$

$$\frac{1}{4} \quad + \quad 0$$

Computing probabilities

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

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

Example.

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

$$\Pr_{\mathrm{G}^\eta, m}[x \neq y] = [\![\mathrm{G}]\!]^\eta\ m\ \mathbb{1}_{x \neq y} = \frac{1}{2}$$

Program equivalence

Observational equivalence

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

$$\models \mathrm{G}_1 \simeq^I_O \mathrm{G}_2 \stackrel{\text{def}}{=} \forall m_1\, m_2\, f\, g,\ m_1 =_I m_2 \wedge f =_O g \implies [\![\mathrm{G}_1]\!]\ m_1\ f = [\![\mathrm{G}_2]\!]\ m_2\ g$$

- Only a Partial Equivalence Relation

$$\models \mathrm{G} \simeq^I_O \mathrm{G} \qquad \text{not true in general}$$

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

Eample

$$\models x \stackrel{\$}{\leftarrow} \{0,1\}^k; y \leftarrow x \oplus z \simeq^{\{z\}}_{\{x,y,z\}} y \stackrel{\$}{\leftarrow} \{0,1\}^k; x \leftarrow y \oplus z$$

Program equivalence

Observational equivalence

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

$$\models \mathsf{G}_1 \simeq^I_O \mathsf{G}_2 \quad \stackrel{\text{def}}{=} \quad \forall m_1\, m_2\, f\, g,\ m_1 =_I m_2 \ \wedge\ f =_O g \implies [\![\mathsf{G}_1]\!]\ m_1\ f = [\![\mathsf{G}_2]\!]\ m_2\ g$$

- Only a Partial Equivalence Relation

$$\models \mathsf{G} \simeq^I_O \mathsf{G} \qquad \text{not true in general}$$

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

Eample

$$\models x \stackrel{\$}{\leftarrow} \{0,1\}^k; y \leftarrow x \oplus z \simeq^{\{z\}}_{\{x,y,z\}} y \stackrel{\$}{\leftarrow} \{0,1\}^k; x \leftarrow y \oplus z$$

Using program equivalence to relate probabilities

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

To prove $\Pr_{\mathsf{G}_1,m_1}[A] = \Pr_{\mathsf{G}_2,m_2}[A]$ it suffices to find a set of variables I such that

- $m_1 =_I m_2$
- $\models \mathsf{G}_1 \simeq_O^I \mathsf{G}_2$

Proving program equivalence

Goal

$$\models G_1 \simeq^I_O G_2$$

A Relational Hoare Logic

$$\frac{\models c_1 \sim c_2 : \Phi \Rightarrow \Phi' \qquad \models c_1' \sim c_2' : \Phi' \Rightarrow \Phi''}{\models c_1; c_1' \sim c_2; c_2' : \Phi \Rightarrow \Phi''}\text{[R-Seq]}$$

$\ldots$

Proving program equivalence

Goal

$$\models \mathsf{G}_1 \simeq^{I}_{O} \mathsf{G}_2$$

Mechanized program transformations

- Transformation: $T(\mathsf{G}_1, \mathsf{G}_2, I, O) = (\mathsf{G}_1', \mathsf{G}_2', I', O')$
- Soundness theorem

$$\frac{T(\mathsf{G}_1, \mathsf{G}_2, I, O) = (\mathsf{G}_1', \mathsf{G}_2', I', O') \qquad \models \mathsf{G}_1' \simeq^{I'}_{O'} \mathsf{G}_2'}{\models \mathsf{G}_1 \simeq^{I}_{O} \mathsf{G}_2}$$

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

Proving program equivalence

Goal

$$\models G_1 \simeq^{I}_{O} G_2$$

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

$$\models G \simeq^{I}_{O} G$$

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

- Does $\models G \simeq^{I}_{O} G$ hold?
- Analyze dependencies to compute I' s.t. $\models G \simeq^{I'}_{O} G$
- 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

$$|\Pr_{\mathsf{G}_1,m}[A] - \Pr_{\mathsf{G}_2,m}[A]| \le \Pr_{\mathsf{G}_{1,2}}[\mathrm{bad}]$$

The Fundamental Lemma of Game-Playing

Syntactic criterion

Game G_1 :	**Game** G_2 :
$\ldots$	$\ldots$
bad $\leftarrow$ true; c_1	bad $\leftarrow$ true; c_2
$\ldots$	$\ldots$

- $\Pr_{\mathrm{G}_1,m}[A \mid \neg \mathrm{bad}] = \Pr_{\mathrm{G}_2,m}[A \mid \neg \mathrm{bad}]$
- $\Pr_{\mathrm{G}_1,m}[\mathrm{bad}] = \Pr_{\mathrm{G}_2,m}[\mathrm{bad}]$

Corollary

$$|\Pr_{\mathrm{G}_1,m}[A] - \Pr_{\mathrm{G}_2,m}[A]| \leq \Pr_{\mathrm{G}_{1,2}}[\mathrm{bad}]$$

Digital Signature Schemes

A digital signature scheme is composed of three algorithms $(\mathcal{KG}, \mathsf{Sign}, \mathsf{Verify})$

Key generation : $(pk, sk) \leftarrow \mathcal{KG}(\eta : \mathbb{N})$

- sk is the private signing key
- pk is the public verification key

Signing : $\sigma \leftarrow \mathsf{Sign}(sk, m)$

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

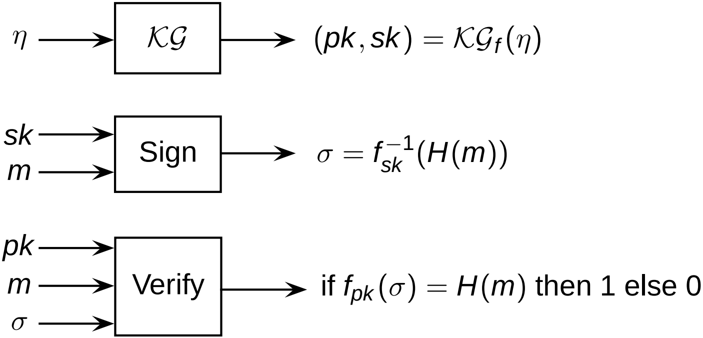
$$\forall m, \mathsf{Verify}(pk, m, \mathsf{Sign}(\mathrm{sk}, \mathrm{m})) = 1$$

The Full-Domain Hash Signature Scheme

Consider

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

The Full-Domain Hash scheme is defined as follows



Existential Unforgeability

We want a signature for a message *m* to be hard to forge. Even if...

- ...the adversary knows the signatures of *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 $\mathsf{Sign}(sk,\cdot)$ can forge a fresh signature for a message of its choice.

$$\Pr\left[\begin{array}{l}(pk,sk)\leftarrow\mathcal{KG}(\eta);\\ (m,\sigma)\leftarrow\mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(pk)\end{array}\middle|\begin{array}{l}\mathsf{Verify}(pk,m,\sigma)=1\wedge\\ m\text{ is fresh}\end{array}\right]\leq\epsilon(\eta)$$

Existential Unforgeability as a game

Game $\mathsf{G}^{\eta}_{\mathsf{EF}}$: $S \leftarrow \mathsf{nil};$ $(pk, sk) \leftarrow \mathcal{KG}(\eta);$ $(m, \sigma) \leftarrow \mathcal{A}(pk);$ $h \leftarrow \mathsf{H}(m)$	**Oracle** $\mathsf{H}(m) \stackrel{\text{def}}{=}$ return $H_\eta(m)$ **Oracle** $\mathsf{Sign}(m) \stackrel{\text{def}}{=}$ $S \leftarrow m :: S;$ return $f_{sk}^{-1}(\mathsf{H}(m))$

$$\forall \mathcal{A}, \qquad \Pr\left[\mathsf{G}^{\eta}_{\mathsf{EF}} \mid f_{pk}(\sigma) = h \wedge m \notin S\right] \leq \epsilon(\eta)$$

Existential Unforgeability as a game

Game $\mathsf{G}_{\mathsf{EF}}^{\eta}$: $S \leftarrow \mathrm{nil}$; $\cancel{(pk, sk) \leftarrow \mathcal{KG}(\eta)}$; $(m, \sigma) \leftarrow \mathcal{A}(pk)$; $h \leftarrow \mathsf{H}(m)$	**Oracle** $\mathsf{H}(m) \stackrel{\text{def}}{=}$ return $H_\eta(m)$ **Oracle** $\mathsf{Sign}(m) \stackrel{\text{def}}{=}$ $S \leftarrow m :: S$; return $f_{sk}^{-1}(\mathsf{H}(m))$

$$\forall \mathcal{A}, \forall pk\ sk, \Pr\left[\mathsf{G}_{\mathsf{EF}}^{\eta} \mid f_{pk}(\sigma) = h \wedge m \notin S\right] \leq \epsilon(\eta)$$

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

Formalizing assumptions

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

$$\boxed{\begin{array}{l}\textbf{Game } \mathsf{G}_{\mathsf{OW}}^{\eta}: \\ (pk, sk) \leftarrow \mathcal{KG}_f(\eta); \\ y \stackrel{\$}{\leftarrow} \mathcal{G}; \\ x \leftarrow \mathcal{I}(pk, y)\end{array}}$$

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

Formalizing assumptions

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

$$\boxed{\begin{array}{l}\textbf{Game } \mathsf{G}^{\eta}_{\mathsf{OW}} : \\ (pk, sk) \leftarrow \mathcal{KG}_f(\eta); \\ y \xleftarrow{\$} \mathcal{G}; \\ x \leftarrow \mathcal{I}(pk, y)\end{array}} \qquad \forall \mathcal{I}, \Pr[\mathsf{G}^{\eta}_{\mathsf{OW}} \mid x = f_{sk}^{-1}(y)] \text{ is negligible}$$

- Random Oracle Model (H_η behaves as a random function)

$$\boxed{\begin{array}{l}\textbf{Oracle } \mathsf{H}(m) \stackrel{\text{def}}{=} \\ \text{return } H_\eta(m)\end{array}} \equiv \boxed{\begin{array}{l}\textbf{Oracle } \mathsf{H}(m) \stackrel{\text{def}}{=} \\ \text{if } m \notin \mathrm{dom}(L) \text{ then} \\ \quad h \xleftarrow{\$} \mathcal{G}; L \leftarrow (m, h) :: L \\ \text{return } L(m)\end{array}}$$

Code-based proof of unforgeability of FDH

Game $\mathsf{G}_{\mathsf{EF}}^{\eta}$:
$S \leftarrow \mathrm{nil};$
$(m, \sigma) \leftarrow \mathcal{A}(pk);$
$h \leftarrow \mathsf{H}(m)$

$\ldots$? $\ldots$

Game $\mathsf{G}_{\mathsf{OW}}^{\eta}$:
$y \xleftarrow{\$} \mathcal{G};$
$x \leftarrow \mathcal{I}(pk, y)$

$$\Pr_{\mathsf{G}_{\mathsf{EF}}^{\eta}}[f_{pk}(\sigma) = h \wedge m \notin S] \quad \leq \quad \cdots \quad \leq \quad h(\Pr_{\mathsf{G}_{\mathsf{OW}}}[x = f_{sk}^{-1}(x)])$$

- 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

Existential unforgeability of FDH

Consider an adversary $\mathcal{A}$ s.t.

- $\mathcal{A}$ makes at most $q_{\mathsf{H}}(\eta)$ hash queries
- $\mathcal{A}$ makes at most $q_{\mathsf{S}}(\eta)$ signature queries

Suppose

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

We show two different inverters $\mathcal{I}$ that use $\mathcal{A}$ 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 $\mathcal{I}$ that inverts f with probability $\epsilon'(\eta)$ within time $t'(\eta)$, where

$$\begin{aligned} \epsilon'(\eta) &\geq (q_{\mathsf{H}}(\eta) + q_{\mathsf{S}}(\eta) + 1)^{-1}\, \epsilon(\eta) \\ t'(\eta) &\leq t(\eta) + (q_{\mathsf{H}}(\eta) + q_{\mathsf{S}}(\eta))\, \Theta(T_f) \end{aligned}$$

Unforgeability of FDH – suboptimal bound

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

$\mathcal{I}(y) \stackrel{\text{def}}{=}$
$y' \leftarrow y$;
$j \xleftarrow{\$} [0..q_{\mathsf{H}} + q_{\mathsf{S}}]$;
$i \leftarrow 0$;
$P, L \leftarrow \mathsf{nil}$;
$(m, \sigma) \leftarrow \mathcal{A}()$;
return σ

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

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

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

Unforgeability of FDH – optimal bound

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

$$\begin{aligned} \epsilon'(\eta) &\geq \frac{1}{q_S(\eta)+1}\left(1-\frac{1}{q_S(\eta)+1}\right)^{q_S(\eta)} \epsilon(\eta) \\ &\approx \exp(-1)\, q_S(\eta)^{-1}\, \epsilon(\eta) \\ t'(\eta) &\leq t(\eta) + (q_H(\eta)+q_S(\eta))\, \Theta(T_f) \end{aligned}$$

Unforgeability of FDH – optimal bound

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

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

- 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 $p\ (1 - p)^{q_S(\eta)}$

Unforgeability of FDH – optimal bound

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

$$\mathsf{Init}_T \stackrel{\text{def}}{=} \text{while } |T| \le q \text{ do } (b \xleftarrow{\$} \langle \mathsf{true} \mapsto p, \mathsf{false} \mapsto 1 - p \rangle;\ T \leftarrow b :: T)$$

- Indeed, thanks to the homomorphic property of f,

$$\begin{aligned} h = f_{pk}(\sigma) \quad &\Longrightarrow \quad y \times P(m) = f_{pk}(\sigma) \\ &\Longrightarrow \quad f_{sk}^{-1}(y \times P(m)) = \sigma \\ &\Longrightarrow \quad f_{sk}^{-1}(y) = \sigma \times P(m)^{-1} \end{aligned}$$

Unforgeability of FDH – optimal bound

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

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

- Overhead is just one extra f computation and one group operation per hash call
- The bound is maximized for $p = (q_{\mathsf{S}}(H) + 1)^{-1}$

Practical Interpretation for RSA-FDH

- Assume a reasonable bound on the number of hash queries, e.g. $q_{\mathsf{H}} \leq 2^{60}$
- Assume a reasonable bound on the number of sign queries, e.g. $q_{\mathsf{S}} \leq 2^{20}$
- 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

Practical Interpretation for RSA-FDH

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

Modulus size	NFS	First reduction	Optimal reduction
512	2^{58}	$2^{60}t + 2^{138}$	$2^{22}t + 2^{100}$
1024	2^{80}	$2^{60}t + 2^{140}$	$2^{22}t + 2^{102}$
2048	2^{111}	$2^{60}t + 2^{142}$	$2^{22}t + 2^{104}$
4096	2^{149}	$2^{60}t + 2^{144}$	$2^{22}t + 2^{106}$

Practical Interpretation for RSA-FDH

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

Modulus size	NFS	First reduction	Optimal reduction
512	2^{58}	2^{140}	2^{102}
1024	2^{80}	2^{141}	2^{103}
2048	2^{111}	2^{142}	2^{104}
4096	2^{149}	2^{144}	2^{106}

- 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