Automatically Verified Mechanized Proof of One-Encryption Key Exchange

Bruno Blanchet
blanchet@di.ens.fr

INRIA, École Normale Supérieure, CNRS, Paris

June 2012

Motivation

- OEKE (One-Encryption Key Exchange) [Bresson, Chevassut, Pointcheval, CCS'03]:
    - Variant of EKE (Encrypted Key Exchange)
    - A password-based key exchange protocol.
    - A non-trivial protocol.
    - It took some time before getting a computational proof of this protocol.
- Our goal:
    - Mechanize, and automate as far as possible, its proof using CryptoVerif.
    - This is an opportunity for several interesting extensions of CryptoVerif.

CryptoVerif

CryptoVerif is an automatic prover:

- in the computational model.
- proves secrecy and correspondence (authentication) properties.
- provides a generic method for specifying properties of cryptographic primitives.
- works for N sessions (polynomial in the security parameter), with an active adversary.
- gives a bound on the probability of an attack (exact security).
- possibility to guide the prover (manual mode).

OEKE

$$
\begin{array}{rcl}
\text{Client } U & & \text{Server } S \\
\hline
& \text{shared } pw & \\
x \stackrel{R}{\leftarrow} [1, q-1] & & \\
X \leftarrow g^x & \xrightarrow{U,X} & y \stackrel{R}{\leftarrow} [1, q-1] \\
& & Y \leftarrow g^y \\
Y \leftarrow \mathcal{D}_{pw}(Y^*) & \xleftarrow{S,Y^*} & Y^* \leftarrow \mathcal{E}_{pw}(Y) \\
K_U \leftarrow Y^x & & \\
Auth \leftarrow \mathcal{H}_1(U||S||X||Y||K_U) & & \\
sk_U \leftarrow \mathcal{H}_0(U||S||X||Y||K_U) & \xrightarrow{Auth} & K_S \leftarrow X^y \\
& & \text{if } Auth = \mathcal{H}_1(U||S||X||Y||K_S) \text{ then} \\
& & sk_S \leftarrow \mathcal{H}_0(U||S||X||Y||K_S)
\end{array}
$$

Result (1)

Using CryptoVerif with user guidance, we prove that OEKE is secure.

Assumptions:

- $\mathcal{H}_0$ and $\mathcal{H}_1$ are random oracles,
- $\mathcal{E}$, $\mathcal{D}$ is an ideal cipher,
- g is a generator of a group G of order q that satisfies the computational Diffie-Hellman assumption.

Static corruptions; no dynamic corruptions.

Result (2)

Theorem

The client is authenticated to the server up to probability

$$\frac{N_S + N_U}{N} + (2q_{h0} + 3q_{h1}) \times \mathrm{Succ}_G^{\mathrm{cdh}}(t') + \textit{collision terms}$$

The session key is secret up to probability

$$\frac{N_S + N_U}{N} + (4q_{h0} + 6q_{h1}) \times \mathrm{Succ}_G^{\mathrm{cdh}}(t') + \textit{collision terms}$$

- *dictionary size* N
- N_U *client instances,* N_S *server instances under active attack*
- q_{h0} *hash queries to* $\mathcal{H}_0$, q_{h1} *hash queries to* $\mathcal{H}_1$

Result (3)

- The first term
$$\frac{N_S + N_U}{N}$$
is optimal: the adversary can test one password per session of the client or of the server.
- Improved probability bounds with respect to [Bresson, Chevassut, Pointcheval, CCS'03]

Contributions

- Improved model of random oracle, ideal cipher
- Model of the computational Diffie-Hellman assumption
- Shoup's lemma:
  - Insert an event and later prove that the probability of this event is negligible.
  - Improved computation of probabilities
- New game transformations:
  - Delay random choices
  - Merge array variables
  - Merge branches of tests
- Improved computation of probabilities of collisions, especially useful with passwords.

Proofs by sequences of games

Proofs in the computational model are typically proofs by sequences of games [Shoup, Bellare&Rogaway]:

- The first game is the real protocol.
- One goes from one game to the next by syntactic transformations or by applying the definition of security of a cryptographic primitive. The difference of probability between consecutive games is negligible.
- The last game is "ideal": the security property is obvious from the form of the game. (The advantage of the adversary is 0 for this game.)

Game 0		Game 1				Game n
Protocol to prove	$\longleftrightarrow$ p_1 negligible		$\longleftrightarrow$ p_2 negligible	$\cdots$	$\longleftrightarrow$ p_n negligible	Property obvious

Events, probabilities

- $C[G]$ = game G interacting with an adversary (evaluation context) C.
- $\Pr[C[G] : e]$ is the probability that $C[G]$ executes event e.
- More generally, a distinguisher D takes as input a sequence of events $\mathcal{E}$ and returns **true** or **false**.

Definition

$\Pr[C[G] : D]$ is the probability that $C[G]$ executes a sequence of events $\mathcal{E}$ such that $D(\mathcal{E}) = \textbf{true}$.

Example

$\Pr[C[G] : e_1 \vee e_2]$ is the probability that $C[G]$ executes e_1 or e_2.

Indistinguishability

C is acceptable for G with public variables V when C is allowed to read directly the variables of G that are in V.

Definition (Indistinguishability)

We write $G \approx_p^V G'$ when, for all evaluation contexts C acceptable for G and G' with public variables V and all distinguishers D,

$$|\Pr[C[G] : D] - \Pr[C[G'] : D]| \leq p(C, t_D).$$

Intuition: the probability that an adversary distinguishes G from G' is at most p.

Computational Diffie-Hellman assumption

Consider a multiplicative cyclic group G of order q, with generator g.
A probabilistic polynomial-time adversary has a negligible probability of computing g^{ab} from g, g^a, g^b, for random $a, b \in \mathbb{Z}_q$.

Computational Diffie-Hellman assumption in CryptoVerif

Consider a multiplicative cyclic group G of order q, with generator g.
A probabilistic polynomial-time adversary has a negligible probability of computing g^{ab} from g, g^a, g^b, for random $a, b \in \mathbb{Z}_q$.

In CryptoVerif, this can be written

$$\begin{aligned}
&\textbf{foreach } i \le n \textbf{ do } a \xleftarrow{R} Z; b \xleftarrow{R} Z; (OA() := g^a, OB() := g^b,\\
&\qquad \textbf{foreach } i' \le n' \textbf{ do } OCDH(z : G) := z = g^{ab})\\
&\approx\\
&\textbf{foreach } i \le n \textbf{ do } a \xleftarrow{R} Z; b \xleftarrow{R} Z; (OA() := g^a, OB() := g^b,\\
&\qquad \textbf{foreach } i' \le n' \textbf{ do } OCDH(z : G) := \mathit{false})
\end{aligned}$$

g^a actually written $exp(g, a)$; g^{ab} written $exp(g, mult(a, b))$

Computational Diffie-Hellman assumption in CryptoVerif

The previous model is sufficient for proving basic schemes.

- Example: semantic security of hashed El Gamal in the random oracle model (A. Chaudhuri).

This model is not sufficient for OEKE and other practical protocols.

- It assumes that a and b are chosen one after the other under the same **foreach**.
- In practice, one participant chooses a, another chooses b, so these choices are made under different **foreach**s in processes in parallel.

Extending the formalization of CDH in CryptoVerif

foreach $ia \leq na$ **do** $a \xleftarrow{R} Z; (OA() := g^a, Oa() := a,$

foreach $iaCDH \leq naCDH$ **do** $OCDHa(m : G, j \leq nb) := m = g^{b[j].a}),$

foreach $ib \leq nb$ **do** $b \xleftarrow{R} Z; (OB() := g^b, Ob() := b,$

foreach $ibCDH \leq nbCDH$ **do** $OCDHb(m : G, j \leq na) := m = g^{a[j].b})$

$\approx$

foreach $ia \leq na$ **do** $a \xleftarrow{R} Z; (OA() := g^a, Oa() := a,$

foreach $iaCDH \leq naCDH$ **do** $OCDHa(m : G, j \leq nb) :=$

if $Ob[j]$ or Oa has been called **then** $m = g^{b[j].a}$ **else** $false),$

foreach $ib \leq nb$ **do** $b \xleftarrow{R} Z; (OB() := g^b, Ob() := b,$

foreach $ibCDH \leq nbCDH$ **do** $OCDHb$ (symmetric of $OCDHa$))

Extending the formalization of CDH in CryptoVerif

$$\textbf{foreach } ia \leq na \textbf{ do } a \xleftarrow{R} Z; (OA() := g^a, Oa() := a,$$
$$\quad \textbf{foreach } iaCDH \leq naCDH \textbf{ do } OCDHa(m : G, j \leq nb) := m = g^{b[j].a}),$$
$$\textbf{foreach } ib \leq nb \textbf{ do } b \xleftarrow{R} Z; (OB() := g^b, Ob() := b,$$
$$\quad \textbf{foreach } ibCDH \leq nbCDH \textbf{ do } OCDHb(m : G, j \leq na) := m = g^{a[j].b})$$

$\approx$

$$\textbf{foreach } ia \leq na \textbf{ do } a \xleftarrow{R} Z; (OA() := g^a, Oa() := ka \leftarrow mark; a,$$
$$\quad \textbf{foreach } iaCDH \leq naCDH \textbf{ do } OCDHa(m : G, j \leq nb) :=$$
$$\qquad \textbf{find } u \leq nb \textbf{ suchthat defined}(kb[u], b[u]) \wedge b[j] = b[u] \textbf{ then}$$
$$\qquad\quad m = g^{b[j].a}$$
$$\qquad \textbf{else if defined}(ka) \textbf{ then } m = g^{b[j].a} \textbf{ else } false),$$
$$\textbf{foreach } ib \leq nb \textbf{ do } b \xleftarrow{R} Z; (OB() := g^b, Ob() := kb \leftarrow mark; b,$$
$$\quad \textbf{foreach } ibCDH \leq nbCDH \textbf{ do } OCDHb \quad \text{(symmetric of } OCDHa\text{))}$$

Extending the formalization of CDH in CryptoVerif

foreach $ia \leq na$ **do** $a \xleftarrow{R} Z; (OA() := g^a, Oa() := a,$

  foreach $iaCDH \leq naCDH$ **do** $OCDHa(m : G, j \leq nb) := m = g^{b[j].a}),$

foreach $ib \leq nb$ **do** $b \xleftarrow{R} Z; (OB() := g^b, Ob() := b,$

  foreach $ibCDH \leq nbCDH$ **do** $OCDHb(m : G, j \leq na) := m = g^{a[j].b})$

$\approx_{(\#OCDHa+\#OCDHb)\times\max(1,e^2\#Oa)\times\max(1,e^2\#Ob)\times pCDH(time+(na+nb+\#OCDHa+\#OCDHb)\times time(exp))}$

foreach $ia \leq na$ **do** $a \xleftarrow{R} Z; (OA() := g^a, Oa() := ka \leftarrow mark; a,$

  foreach $iaCDH \leq naCDH$ **do** $OCDHa(m : G, j \leq nb) :=$

   find $u \leq nb$ **suchthat defined**$(kb[u], b[u]) \wedge b[j] = b[u]$ **then**

    $m = g^{b[j].a}$

   else if defined(ka) **then** $m = g^{b[j].a}$ **else** $false),$

foreach $ib \leq nb$ **do** ...

Extensions for CDH

The implementation of the support for CDH required two extensions of CryptoVerif:

- An array index j occurs as argument of a function.
  - extend the language of equivalences used for specifying assumptions on primitives.
- The equality test $m = g^{ab}$ typically occurs inside the condition of a **find**.
  - This **find** comes from the transformation of a hash function in the Random Oracle Model.

$h(g^{ab})$

becomes

$\mathbf{find}\ u \leq n\ \mathbf{suchthat}\ \mathbf{defined}(x[u], r[u]) \wedge x[u] = g^{ab}\ \mathbf{then}\ r[u]\ \mathbf{else}\ \ldots$

After transformation, we obtain a **find** inside the condition of a **find**.

Shoup's lemma

Goal: bound $\Pr[C[G_o] : e_0]$.

G_0

$\updownarrow$ probability p

G_n

$\updownarrow$ $\Pr[C[G_{n+1}] : e]$

G_{n+1} event e

$\updownarrow$ probability p'

$G_{n'}$ events e_0 and e never executed

$$\begin{aligned}\Pr[C[G_0] : e_0] &\leq p + \Pr[C[G_{n+1}] : e] + p' \\ &\leq p + p' + p' \\ &\leq p + 2p'\end{aligned}$$

Improved version of Shoup's lemma

Goal: bound $\Pr[C[G_0] : e_0]$.

G_0

$\updownarrow$ probability p

G_n

$\updownarrow$ differ only when e is executed

G_{n+1} event e

$\updownarrow$ probability p'

$G_{n'}$ events e_0 and e never executed

$$\begin{aligned}
\Pr[C[G_0] : e_0] &\leq p + \Pr[C[G_n] : e_0] \\
&\leq p + \Pr[C[G_{n+1}] : e_0 \vee e] \\
&\leq p + p' + \Pr[C[G_{n'}] : e_0 \vee e] \\
&\leq p + p'
\end{aligned}$$

Improved Shoup's lemma

Lemma

Let C be a context acceptable for G and G' with public variables V.

1. *Improved Shoup's lemma:*
   If G' differs from G only when G' executes event e, then
   $\Pr[C[G] : D] \leq \Pr[C[G'] : D \vee e]$.
2. *If $G \approx_p^V G'$, then* $\Pr[C[G] : D] \leq p(C, t_D) + \Pr[C[G'] : D]$.
3. $\Pr[C[G] : D \vee D'] \leq \Pr[C[G] : D] + \Pr[C[G] : D']$.

We also gain a factor 2 for the probability of events in proofs of secrecy, using a similar technique.

Impact on OEKE

We apply our improved computation of probabilities to the manual proof of [Bresson, Chevassut, Pointcheval, CCS'03].

- dictionary size N
- N_U client instances under active attack
- N_S server instances under active attack
- N_P sessions under passive attack
- q_h hash queries

Impact on OEKE: semantic security

- Standard computation of probabilities:

$$\mathrm{Adv}^{\mathrm{ake}}_{G_0}(C) \leq \frac{4N_S + 2N_U}{N} + 8q_h \times \mathrm{Succ}^{\mathrm{cdh}}_{\mathbb{G}}(t') + \text{collision terms}$$

- Improved computation of probabilities:

$$\mathrm{Adv}^{\mathrm{ake}}_{G_0}(C) \leq \frac{N_S + N_U}{N} + q_h \times \mathrm{Succ}^{\mathrm{cdh}}_{G}(t') + \text{collision terms}$$

- The adversary can test one password per session with the parties.

Impact on OEKE: one-way authentication

- Standard computation of probabilities:

$$\mathrm{Adv}_{G_0}^{\mathrm{c-auth}}(C) \leq \frac{2N_S + N_U}{N} + 3q_h \times \mathrm{Succ}_{\mathbb{G}}^{\mathrm{cdh}}(t') + \text{collision terms}$$

- Improved computation of probabilities:

$$\mathrm{Adv}_{G_0}^{\mathrm{c-auth}}(C) \leq \frac{N_S + N_U}{N} + q_h \times \mathrm{Succ}_{G}^{\mathrm{cdh}}(t') + \text{collision terms}$$

- The adversary can test one password per session with the parties.

This remark is general: it is not specific to OEKE or to CryptoVerif, and can be used in any proof by sequences of games.

CryptoVerif input

CryptoVerif takes as input:

- The assumptions on security primitives: CDH, Ideal Cipher Model, Random Oracle Model.
  - These assumptions are formalized in a library of primitives. The user does not have to redefine them.
- The initial game that represents the protocol OEKE:
  - Code for the client
  - Code for the server
  - Code for sessions in which the adversary listens but does not modify messages (passive eavesdroppings)
  - Encryption, decryption, and hash oracles
- The security properties to prove:
  - Secrecy of the keys sk_U and sk_S
  - Authentication of the client to the server
- Manual proof indications (see next slide)

Manual proof indications (sketch)

1. Insert two events for Shoup's lemma, corresponding to cases in which the adversary breaks the protocol.
   - CryptoVerif cannot guess where events should be inserted.
2. Automatic proof strategy of CryptoVerif.
   - Applies in particular the computational Diffie-Hellman assumption.
3. Reorganize random number generations and merge branches of tests to eliminate uses of the password.

All manual commands are checked by CryptoVerif, so that an incorrect proof cannot be produced.

Conclusion

The case study of OEKE is interesting for itself, but it is even more interesting by the extensions it required in CryptoVerif:

- Treatment of the Computational Diffie-Hellman assumption.
- New manual game transformations, in particular for inserting events and merging branches of tests.
- Optimization of the computation of probabilities for Shoup's lemma.
- Other optimizations of the computation of probabilities in CryptoVerif.

These extensions are of general interest.