# Automatic Approach of Provable Security and its Application for OAEP+

GU Chun-Xiang, Guang Yan, ZHU Yue-Fei

Department of Network Engineering, Information Engineering College, Information Engineering University, Zhengzhou 450002, China

**Abstract**: Probable security is an important criteria for analyzing the security of cryptographic protocols. However, writing and verifying proofs by hand are prone to errors. This paper introduces the *game-based* approach of writing security proofs and its automatic technique. It advocates the automatic security proof approach based on process calculus, and presents the initial game and observational equivalences of OAEP+.

**Key words:** cryptographic protocols; probable security; automatic security proof; process calculus

## 1. Introduction

Information security is nowadays an important issue. Its essential ingredient is cryptography. To be accepted, a cryptographic scheme must come with a proof that it satisfies some standard security properties. However, a problem we have to face is that as a community, we generate more proofs than we carefully verify, and as a consequence some of our published proofs are incorrect. In fact, many of our proofs are truly complex, error-prone and difficult to check.

In 2004, Shop[1] discussed techniques for structuring security proofs as sequences games. Game-playing is an approach to write security proofs that are easy to verify. In this approach, security definitions and intractable problems are written as programs called games, and reduction security proofs are sequences of game transformations [1–3]. Barthe, Cerderquist, and Tarento [4,5] have formalized the generic model and the random oracle model in the interactive theorem prover Coq[6], and proved signature schemes in this framework. Later, Nowak[7] and Affeldt et.al.[8] show how to formalize the game-playing framework in the proof assistant Coq. However, proofs in generic interactive theorem provers require a lot of human effort, in order to build a detailed enough proof for the theorem prover to check it.

Halevi[9] explains that implementing an automatic tool based on compiler techniques to build security proofs with sequences of games, and suggests ideas in this direction, but does not actually implement one. In 2006, B. Blanchet and D. Pointcheval[10] presented a pioneer implementation of game-playing that has been applied to several standard cryptographic schemes taken from the literature [11, 12]. Further works can be found in [13,14]. However, this is still a new research area, and until now there are only a few number of cryptographic schemes can be proved with their automatic tool.

This paper introduces the *game-based* approach of writing security proofs and its automatic technique. It advocates the automatic security proof approach based on process calculus, and presents the initial game and observational equivalences of OAEP+[15]. The rest of this paper is

organized as follows. In Sect. 2, we explain the *game-based* approach of writing security proofs. In Sect. 3, we introduce an implementation with process calculus. In Sect. 4, we present the initial game and observational equivalences of OAEP+. Finally, we conclude in Sect. 7.

## 2. Provable Security and Gamed Based Proofs

Security for cryptograptic primitives is typically defined as an attack game involves a stateful adversary that interacts with some interfaces of whatever scheme that we are dealing with. We usually call these interfaces the challenger. Both adversary and challenger are probabilstic processes that communicate with each other, and so we can model the game as a probability space. Typically, the definition of security is tied to some particular event *S*. Security means that for every "efficient" adversary, the probability that event *S* occurs is "very close to" some specified "target probabilty": typically, either 0 or 1/2.

A canonical game consists of a main loop, where each iteration calls an *adversary routine*, supplying it with the results of the last iteration and getting back the query to be asked in the current iteration. Then the appropriate interface it invoked for the current query, and the result is again fed to the adversary routine in the next iteration. The game is parametrized by some upper bound on the number of iterations (and maybe also upper bounds on other resources of the adversary). It is usually convenient to assume that the adversary fully consumes its resources, and in particular that the number of iterations of the main loop is always exactly equal to the upper bound. Once the main loop is finished, the game may have some output, and this output is typically just the last thing that the adversary has output (more often than not a single bit).

To make it clearer, we show an example of the game for *indistinguishability against adaptive chosen cipher-text attack* (IND-CCA2) of public key encryption.

IND-CCA2 Game:

$$(pk, sk) \leftarrow KGen(1^k);$$
$$(m_0, m_1, s) \leftarrow A_1^{O(.),H(.)}(pk);$$
$$y \leftarrow Enc(m_b, pk);$$
$$b' \leftarrow A_2^{O(.),H(.)}(m_0, m_1, s, y);$$

**KGen**: The challenger takes a security parameter k and runs the *KGen* algorithm to generate the key pair (*pk*,*sk*).

**Phase 1**: The adversary is given the public key *pk*. It can adaptively issue queries to the decryption oracle *O*(.) and the random oracle *H*(.). Once the adversary decides that Phase 1 is over it outputs two equal length plaintexts $(m_0, m_1)$ and some information *s* that it want to save.

**Challenge**: The challenger picks a random bit $b \in \{0,1\}$, and encrypt $m_b$ with *pk*. It sends the cipher-text *y* as the challenge to the adversary.

**Phase 2**: The adversary can adaptively issue more queries to the decryption oracle *O*(.) and the random oracle *H*(.). The only restriction is that it can not query *O*(.) with the challenge cipher-text *y*. Finally, the adversary outputs a guess *b'*. The adversary wins the game if *b=b'*.

If $|\Pr[b = b'] - 1/2|$ is negligible, then the scheme is secure.

Now, to prove security using the sequence-of-games approach, one prodceeds as follows. One constructs a sequence of games, Game 0, Game 1, . . . , Game n, where Game 0 is the original

attack game with respect to a given adversary and cryptographic primitive. Let $S_0$ be the event $S$, and for $i = 0,..., n$, the construction defines an event $S_i$ in Game *i*, usually in a way naturally related to the definition of *S*. The proof shows that $\Pr[S_i]$ is negligibly close to $\Pr[S_{i+1}]$ for $i = 0,..., n-1$, and that $\Pr[S_n]$ is equal (or negligibly close) to the "target probability." From this, it follows that Pr[*S*] is negligibly close to the "target probability," and security is proved.

# 3. An Implementation with Process Calculus

Attack game is a system that involves probabilstic processes communicate with each other. Process calculus[16] can represent the parallel or serial interacts of the adversary and the challenger, and the probabilistic semantics provide a description of the probabilistic reduction of the sequence of games.

### 3.1 Process Calculus

Blanchet[10,17] provides a process calculus to represent games. The calculus is inspired by the pi-calculus. It introduces arrays for accessing to the whole memory state of the system and replacing lists often used in proofs. The syntax is as following.

- Term

$M ::=$

| | |
|---|---|
| $i$ | *replication index* |
| $x[M_1,..., M_m]$ | var*iable access* |
| $f(M_1,..., M_m)$ | *function application* |

- Input process

$Q ::=$

| | |
|---|---|
| $0$ | *nil* |
| $Q \mid Q'$ | *parallel composition* |
| $!^{i \le n} Q$ | *replication n times* |
| $newChannel\ c; Q$ | *creat channel* |
| $in(c[M_1,..., M_l], (x_1[\tilde{i}]:T_1,..., x_k[\tilde{i}]:T_k)); P$ | *input* |

$\tilde{i}$ and $T_j$ represent the index and type of $x_j$ $(j = 1,..., k)$ respectively.

- Output process

$P ::=$

| | |
|---|---|
| $out(c[M_1,..., M_l], (N_1,..., N_k)); Q$ | *output* |
| $new\ x[i_1,...i_m]:T; P$ | *random number* |
| $let\ x[i_1,...i_m]:T = M\ in\ P$ | *assignment* |
| $if\ defined(M_1,..., M_l) \wedge M\ then\ P\ else\ P'$ | *conditional* |
| $find(\oplus_{j=1}^{m} u_{j1}[\tilde{i}] \le n_{j1},..., u_{jm_j}[\tilde{i}] \le n_{jm_j}\ such$ $that\ defined(M_{j1},..., M_{jl_j}) \wedge M_j\ then\ P_j)\ else\ P$ | *array lookup* |

### 3.2 Game Transitions

There are two main kinds of game transformations that allow us to get the sequence of games:

syntactic transformations and transformations from the definitions of security of primitives.

Syntactic transformations transform games with syntactic properties. For examples, when *x* is defined by an assignment $let\ x[i_1,...i_m]:T = M\ in\ P$, we can replace *x* with its value *M*. Simplify is the most important syntactic transformation, which combines some equations and probabilistic properties to simplify terms and processes.

The security of cryptographic primitives is defined using observational equivalences given as axioms. We denote by $\Pr[Q \sim\!\!\!\rightarrow a]$ the probability that the process *Q* return with *a*, and denote by $\Pr[Q \sim\!\!\!\rightarrow \Re]$ the probability that the process *Q* executes exactly the sequence of events $\Re$, in the order of $\Re$.

**Definition 1 (Observational equivalence)**. We say that *Q* and *Q'* are observationally equivalent up to probability *p*, written $Q \approx_p Q'$, when for all *t*, for all contexts *C* acceptable for *Q* and *Q'* that run in time at most *t*, for all bit-strings *a*, $|\Pr[C[Q] \sim\!\!\!\rightarrow a] - \Pr[C[Q'] \sim\!\!\!\rightarrow a]| \leq p(t)$ and $\sum_{\Re} |\Pr[C[Q] \sim\!\!\!\rightarrow \Re] - \Pr[C[Q'] \sim\!\!\!\rightarrow \Re]| \leq p(t)$.

**Proposition 1**[16]. 1. $\approx_p$ is reflexive and symmetric. 2. If $Q \approx_p Q'$ and $Q' \approx_{p'} Q''$, then $Q \approx_{p+p'} Q''$. 3. If *Q* executes event *e* with probability at most *p* and $Q \approx_{p'} Q'$, then *Q'* executes event *e* with probability at most $p + p'$.

Observational equivalences $Q \approx_p Q'$ can be used by the prover in order to transform a game $G_i$ with $C[Q]$ into another observationally equivalent game $G_{i+1}$ with $C[Q']$. Proposition 1 provides the up bound for the success probability of the attackers. Observational equivalences describe transformations from the definitions of security of primitives. For example, in the random oracle model, Hash function can be replaced with a random oracle. This can be written as

$$
\begin{aligned}
&{}^{!i_h \leq n_h}(in(c_1[i_h], x:T_1); out(c_2[i_h], hash(x)))\\
&\approx_0 {}^{!i_h \leq n_h}(in(c_1[i_h], x:T_1);\\
&\quad find\ u \leq n_h\ such\ that\ (defined(x[u], r[u]) \wedge (x = x[u]))\ then\\
&\quad\ out(c_2[i_h], r[u])\\
&\quad else\ r \xleftarrow{R} T_2; out(c_2[i_h], r)\\
&\quad )
\end{aligned}
,
$$

where $T_1$ is the domain of Hash function and $T_2$ is its value type.

### 3.3 Criteria for Provable Security

After each successful transformation, we should test whether the desired security properties are proved. So we need some criteria for provable security. As to public key encryption, we need to test the secrecy of the bit value *b* in IND-CCA2 game.

**Definition 2 (one-session secrecy)**. The process Q preserves the *one-session secrecy of* x when $Q \mid Q_x \approx_0 Q \mid Q'_x$, where

$$
\begin{aligned}
&Q_x = in(c_1, (u_1:T_1,....,u_m:T_m));\\
&\quad if\ defined(x[u_1,...,u_m])\ then\ out(c_2, x[u_1,....,u_m])
\end{aligned}
, \text{ and}
$$

$$
\begin{aligned}
&Q'_x = in(c_1, (u_1:T_1,....,u_m:T_m));\\
&\quad if\ defined(x[u_1,...,u_m])\ then\ new\ y:T; out(c_2, y)
\end{aligned}
,
$$

$T_i$ is the type of $[1,n_m]$, $c_1, c_2$ are not belong to channels of $Q$, $u_1,...,u_m, y \notin Var(Q)$, and the value type of $x[u_1,...,u_m]$ is $T$.

Intuitively, the adversary cannot distinguish a process that outputs the value of the secret $x$ from one that outputs a random number $y$. For IND-CCA2 game, we need to prove that the bit value $b$ satisfies the one-session secrecy.

**Proposition 2** [16]. If $Q$ and $Q'$ are observationally equivalent, and $Q$ preserves the one-session secrecy of $x$, then $Q'$ preserves the one-session secrecy of $x$.

# 4. Initial Game and Observational Equivalences of OAEP+

### 4.1 The Description of OAEP+ [15]

Let $k$ be the security parameter, $f$ the trapdoor one-way permutations. $g$ is the inverse of $f$. $k_0, k_1$ are parameters satisfying $k_0 + k_1 < k$ and $2^{k_0}, 2^{k_1}$ are negligible, $m \in \{0,1\}^n$ where $n = k - k_0 - k_1$. And $G: \{0,1\}^{k_0} \to \{0,1\}^n$, $H': \{0,1\}^{n+k_0} \to \{0,1\}^{k_1}$, $H: \{0,1\}^{n+k_1} \to \{0,1\}^{k_0}$ are hash functions. OAEP+ is described as follows.

- *KGen*: For a seed $r$, output public key $pk = pkgen(r)$ and secret key $sk = skgen(r)$.
- *Enc*: For input a message $m$, pick a random $x \in \{0,1\}^{k_0}$, compute $s = (G(x) \oplus m) \| H'(x \| m)$, $t = H(s) \oplus x$, $y = f(s \| t, pk)$, output cipher-text $y$.
- *Dec*: For input cipher-text $y$, compute $s \| t = g(y, sk)$, $x = H(s) \oplus t$, $m = G(x) \oplus s[0...n-1]$, $c = s[n....n+k_1-1]$. If $c = H'(x \| m)$ then output plain-text $m$, else output $\perp$, where $s[l_1...l_2]$ denotes the bit-strings of $s$ from the $l_1 - th$ bit to $l_2 - th$ bit.

### 4.2 The Formal Security Description with Process Calculus

We denotes $T_1, T_2, T_3, T_4$ as the random number type of $\{0,1\}^{k_0}, \{0,1\}^n, \{0,1\}^{k_1}, \{0,1\}^k$ respectively, $T_5$ the key seed type, then $G: T_1 \to T_2$, $H': T_1 \times T_2 \to T_3$, $H: T_2 \times T_3 \to T_1$.

In the OAEP＋, the main crypto components include the trapdoor one-way permutation $f$, hash functions $G$, $H'$ and $H$, and the $\oplus$ operation. The observational equivalences about the security properties and assumptions of these components are the foundation for the transformations and the proof. For the description convenience, we define function process as following:

$FP ::=$

| | |
|---|---|
| $M$ | *term* |
| $new\ x[\tilde{i}]: T; FP$ | *ranom number* |
| $let\ x[\tilde{i}]: T = M\ in\ P$ | *assignment* |
| $find(\oplus_{j=1}^{m} u_{j1}[\tilde{i}] \le n_{j1},...,u_{jm_j}[\tilde{i}] \le n_{jm_j})\ such$ $that\ defined(M_{j1},...,M_{jl_j}) \wedge M_j\ then\ P_j)\ else\ P$ | *array lookup* |

In observational equivalences, we use $(x_1: T_1,..., x_k: T_k) \to FP$ to express the process that inputs $(x_1: T_1,..., x_k: T_k)$, and outputs the result of $FP$.

- **Observational Equivalences for One-Way Permutations**

The trapdoor one-way permutation *f* is the kernel component for OAEP＋. We define the success probability of an adversary *A* to invert.

$$Succ^{OW}(A) = \Pr\left[ x = x' \,\middle|\, \begin{array}{c} r \leftarrow T_1, (pk, sk) \leftarrow KGen(r), x \leftarrow T_2, \\ y = f(x, pk), x' = A(y, pk) \end{array} \right]$$

We denote by $Succ^{OW}(t)$ the maximal success probability an adversary can get within time t. Eventually, the following observationally equivalent processes for one-way permutation can be used in the transformations:

$$\begin{array}{l} {}^{i_k \le n_k} new\ r : T_5; \\ ( \\ \quad () \to pkgen(r), \\ \quad {}^{i_f \le n_f} new\ x : T_2; \\ \quad (() \to f(pkgen(r), x) \mid \\ \quad\quad {}^{i_e \le n_e}(x' : T_1) \to (x' = x) \mid \\ \quad () \to x) \mid \\ ) \end{array} \approx_{prob} \begin{array}{l} {}^{i_k \le n_k} new\ r : T_5; \\ ( \\ \quad () \to pkgen(r) \mid \\ \quad {}^{i_f \le n_f} new\ x : T_2; \\ \quad (() \to f(pkgen(r), x) \mid \\ \quad\quad {}^{i_e \le n_e}(x' : T_1) \to find\ such\ that\ defined(k)\ then \\ \quad\quad\quad (x' = x)\ else\ false \mid \\ \quad\quad () \to let\ k : bitstring = mark\ in\ x) \\ ) \end{array}$$

where $prob(t) = n_k \cdot n_f \cdot Succ^{OW}(t + (n_k n_f - 1)t_f + (n_k - 1)t_{pkgen})$, $t_f$ and $t_{pkgen}$ are times for $f(.)$ and $pkgen(.)$ respectively. In the right hand of the observationally equivalence, if *x* has been outputted, then *k* is defined and hence the adversary can determine whether $x' = x$ holds, otherwise, the adversary always output false for $x' = x$.

There are some other algebra properties for trapdoor one-way permutations that can be used for security proof. For example $\forall r : T_1, \forall x : T_2, g(f(x, pkgen(r)), skgen(r)) = x$, and so on. We do not discuss the details further.

- **Observational Equivalences for Hash Functions**

In the random oracle, we assume that hash functions are random oracles. That is, for an input, if it has been in the access list, then the process returns with the result in the list, otherwise, it picks a random value as the respond. The observation equivalence for *G*(.), *H'*(.) and *H*(.) are as following.

$$\begin{array}{l} {}^{i_G \le n_G}(x : T_1) \to G(x) \end{array} \approx_0 \begin{array}{l} {}^{i_G \le n_G}(x : T_1) \to find\ \ j \le n_G\ \ such\ that \\ \quad defined(x[j], r[j]) \wedge (x = x[j])\ then\ r[j]\ , \\ \quad else\ (new\ r : T_1; r) \end{array}$$

$$\begin{array}{l} {}^{i_{H'} \le n_{H'}}(x : T_1, y : T_2) \to H'(x, y) \end{array} \approx_0 \begin{array}{l} {}^{i_H \le n_{H'}}(x : T_1, y : T_2) \to find\ \ j \le n_{H'}\ \ such\ that \\ \quad defined(x[j], y[j], r[j]) \wedge \\ \quad ((x = x[j]) \wedge (y = y[j]))\ then\ r[j] \\ \quad\quad else\ (new\ r : T_1; r) \end{array} ,$$

and $^{i_H \le n_H}(x:T_2, y:T_3) \to H(x,y) \approx_0$ $^{i_H \le n_H}(x:T_2, y:T_3) \to$ *find* $j \le n_H$ *such that* *defined*$(x[j], y[j], r[j]) \wedge ((x = x[j]) \wedge (y = y[j]))$ *then* $r[j]$ *else* $(new\ r:T_1; r)$ .

- **Observational Equivalences for** $\oplus$

$^{i_X \le n_X} new\ a:T; (x:T) \to a \oplus x \approx_0 {}^{i_X \le n_X} new\ a:T; (x:T) \to a$

That is, for random $a:T$, the probability distribution of $a \oplus x$ is the same as that of $a$.

To proof the security of OAEP+ automatically, we should further give the description of the initial game. According to the definition of IND-CCA2, we first define the following sub-processes.

- **Three Hash Function Processes**

$let\ processG = {}^{!i_G \le n_G}(in(c_1[i_G], x:T_1); out(c_2[i_G], G(x)))$ ;

$let\ processH' = {}^{!i_{H'} \le n_{H'}}(in(c_3[i_{H'}], (x:T_1, y:T_2)); out(c_4[i_{H'}], H'(x,y)))$ ;

$let\ processH = {}^{!i_H \le n_H}(in(c_5[i_H], (x:T_2, y:T_3)); out(c_6[i_H], H(x,y)))$,

where $n_G, n_{H'}, n_H$ are the up bound for the accesses to $G(.), H'(.)$ and $H(.)$ respectively.

- **The Key Generation Process**

$let\ processKGen =$
$in(start, ());$
$new\ r:T_5;$
$let\ pk = pkgen(r)\ in$
$let\ sk = skgen(r)\ in$
$out(c_7, pk);$

- **The Decryption Process**

$let\ processDec =$
$^{!i_D \le q_D}($ $in(c_8[i_D], a:T_4);$
$find\ such\ that\ defined(ch) \wedge (a = ch)\ then\ Yield\ else$
$let\ s \| t = g(a, sk)\ in$
$let\ x = H(s) \oplus t\ in$
$m = G(x) \oplus s[0...n-1]\ in$
$if\ s[n,...n+k_1-1] = H'(x \| m)\ then$
$out(c_9[i_D], m)$
$)$ ,

where $q_D$ is the up bound for the accesses to decryption oracle, *ch* denotes the challenge cipher-text. *Yield* denotes the end process, that is, the adversary can access the decryption oracle with *ch* as input.

- **The Challenge Cipher-text Generation Process**

$let\ processT =$

$in(c_{10}, (m_0 : T_2, m_1 : T_2))$;
$new\ b : bool$;
$let\ menc = test(b, m_0, m_1)\ in$
$new\ x : T_1$;
$let\ s_1 : T_1 = G(x) \oplus menc\ in$
$let\ s_2 : T_1 = H'(x, menc)\ in$
$let\ t : T = H(s_1, s_2) \oplus x\ in$
$let\ ch : T_4 = f(s_1 \| s_2 \| t, pk)\ in$
$out(c_{11}, ch)$;

The function $test(b : bool, x : T_2, y : T_2)$ is defined as $\forall\ x : T_2, y : T_2$, $test(true, x, y) = x$, and $test(false, x, y) = y$.

According to the definition of IND-CCA2 in section 2.2, first, the system runs the key generation process *processKGen*, follows with the processes *processDec* and *processT*. In the following attacks, the adversary can access the decryption process at most $q_D$ times, and at some time, the adversary can access the challenge cipher-text generation process. Hence, the *processDec* is parallelized with *processT*. In the random oracle, the attack can access the random oracle at any time. Composed with the above processes, we can get the initial game of OAEP+ in the random oracle as following

- **Initial Game**

$processG_0$ =

```
processG | processH'| processH |
(
   processKGen;
   (processDec | processT)
)
```

Up to now, we have described the Initial Game and Observational Equivalences of OAEP+. Next, we explain how we organize the transformations in order to prove the security. At the beginning of the proof, and after each successful cryptographic transformation, we should execute syntactic transformations, and test whether the desired one-session secrecy are proved. If so, we have proved. Otherwise, we try to execute each available cryptographic transformation in turn. When such a cryptographic transformation fails, it returns some syntactic transformations that could make the desired transformation work. Then we try to perform these syntactic transformations. When the syntactic transformations finally succeed, we retry the desired cryptographic transformation, and so on.

# 5. Conclusion

This paper introduces the game-based approach of writing security proofs and its automatic technique. It advocates the automatic security proof approach based on process calculus, and presents the initial game and observational equivalences of OAEP+. Up to now, this is still a new research area, only a small number of cryptography protocols can be disposed with this approach, and we should make further researches on the transformation rules and strategies.

# References


1. V. Shoup. Sequences of games: a tool for taming complexity in security proofs, Cryptology ePrint Archive 2004/332. 2004.
2. M. Bellare and P. Rogaway. Code-Based Game-Playing Proofs and the Security of Triple Encryption. EuroCrypt 2006, LNCS 4004, pp. 409–426. Springer.
3. D. Pointcheval. Provable Security for Public Key Schemes. In Contemporary Cryptology, Advanced Courses in Mathematics CRM Barcelona, pp. 133–189. Birkh¨auser Publishers, 2005.
4. G. Barthe, J. Cederquist, and S. Tarento. A machine-checked formalization of the generic model and the random oracle model. In IJCAR'04, LNCS 3097, pages 385-399. Springer, July 2004.
5. S. Tarento. Machine-checked security proofs of cryptographic signature schemes. In ESORICS'05, LNCS 3679, pages 140-158. Springer, Sept. 2005.
6. The LogiCal Project, INRIA. The Coq proof assistant: http://coq.inria.fr.
7. D. Nowak. A framework for game-based security proofs. In ICICS 2007, LNCS 4861, pp. 319-333. Springer, 2007.
8. R. Affeldt, M. Tanaka, and N. Marti, Formal Proof of Provable Security by Game-playing in a Proof Assistant. Provsec 2007, LNCS 4784,pp.151-168. November, 2007.
9. S. Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181, June 2005.
10. B. Blanchet and D. Pointcheval. Automated Security Proofs with Sequences of Games. CRYPTO 2006, LNCS 4117, pp. 537-554. Springer. 2006.
11. M. Bellare and P. Rogaway. Random Oracle are Practical: A Paradigm for Designing Efficient Protocols. In 1st ACM Conference on Computer and Communications Security (CCS 1993), p. 62–73. ACM Press.
12. M. Bellare and P. Rogaway. The Exact Security of Digital Signatures—How to Sign with RSA and Rabin. EuroCrypt 1996, LNCS 1070, p. 399–416. Springer.
13. B. Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Transactions on Dependable and Secure Computing*, 5(4):193-207, October-December 2008.
14. B. Blanchet and Avik Chaudhuri. Automated Formal Analysis of a Protocol for Secure File Sharing on Untrusted Storage. In IEEE Symposium on Security and Privacy, pages 417-431, Oakland, CA, May 2008.
15. V.Shoup. OAEP reconsidered. CRYPTO'2002, LNCS 2139, Springer-Verlag, Berlin, pp.239-259. 2001.
16. B. Blanchet. A computationally sound mechanized prover for security protocols. In IEEE Symposium on Security and Privacy, pages 140-154, May 2006.
17. J. C. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for the analysis of cryptographic protocols. Theoretical Computer Science, 353(1-3): 118-164, Mar. 2006.