

Advanced Cryptography
Link between Computational and Symbolic, Introduction to Cryptoverif
and others

Note lecture12

November 24, 2009

Alboloushi Yousuf, Alnuaimi Mohamad, Alshamsi Eisa, Mohamad
Khoulod, Raeisi Hamad

1. Link between Computational and Symbolic world	2
1.1. Abadi and Rogaway 2000 Paper	2
1.2. Patterns and Expression	2
1.3. Key Recovery Function $F_{Kr}(E,K)$	3
1.4. Inductive definition of key recovery (rec).....	3
1.5. The pattern of an expression.....	4
1.6. Pattern Equivalence	4
1.7. Encryption Cycles	4
1.8. Type-0 Scheme	5
2. Crypto Verif.....	6
2.1. MACs: security definition.....	6
2.2. MACs: intuitive implementation	7
2.3. Formal implementation	7
3. Automated Cryptographic Proofs for Asymmetric Encryption.....	8
3.1. Generic Encryption Scheme.....	8
3.2. The assertion Language	10
3.3. Standard rules	11
3.4. Soundness of the analysis	11
3.4.1. Proposition.....	11
3.4.2. Theorem	11

Link between Computational and Symbolic world

The analysis of security protocols is being carried out mainly by means of two different techniques:

- logical perspective (Symbolic)
 - messages are seen as algebraic objects
 - Abstract way to see the computational world.
 - Example Dolev-Yao Model.
 - Intruder modeled by deduction “Has limited power”
 - Based on constraint solving, First-order Logic, tree automata
 - Easy automated and you can find a lot of tools that compile the protocol for you.
- complexity-theory perspective (Computational)
 - messages are seen as bit strings “0s, 1s”.
 - Adversary is represented by a PPTT “probabilistic polynomial time Turing” machine:
 - An attacker here is a resource bounded probabilistic algorithm, limited by running time and/or memory, but capable of breaking cryptographic operations, if that is computationally feasible.
 - More general and more realistic, but also more complex.

What is the connection between Computational and Symbolic worlds?

Such a relation takes the form of a function mapping algebraic messages m to (distributions over) bit strings $[m]$. Such a map allows one to use algebraic methods, possibly even automated, to reason about security properties of protocols and have those reasoning beveled also in the computational world.

1.1. Abadi and Rogaway 2000 Paper

The idea is if you can prove that your protocol is safe in symbolic world then it will be secure in computational world too.

1.2. Patterns and Expression

Pattern define by following grammar:

Keys= $\{k_1, k_2, \dots\}$
Bool= $\{0, 1\}$
$P ::= k \in \text{Keys} \mid b \in \text{Bool} \mid (P_1, P_2) \mid \{P\}_k$ \square “The box means that the intruder can’t access it or know what inside it”

Expression is a pattern with NO occurrence of box (□) which represents a ciphertext that an attacker cannot decrypt.

For a formal expression E , define the set keys K that occur in E but that the adversary cannot find then replace sub expressions $\{...\}K$ of E , where $k \in K$, by box (□) This gives the pattern of E and it denote it by $P(E)$. We say that $E_1 = E_2$ if $P(E_1) = P(E_2)$.

For a formal expression E , define the set keys K that occur in E but that the adversary cannot find.

Replace sub expressions $\{...\}K$ of E , where $k \in K$, by box (□) This gives the pattern of E and it denote it by $P(E)$. We say that $E_1 = E_2$ if $P(E_1) = P(E_2)$.

1.3. Key Recovery Function $F_{Kr}(E, K)$

The approach of (Micciancio , Warinschi), for an expression E and a set K of Keys “set of known keys”, we define Key Recovery Function $F_{Kr}(E, K)$ as follows:

$F_{Kr}(E, K) = \phi \rightarrow$ zero because you can't deduce any key.

$F_{Kr}(k, K) = \{k\} \cup K$

$F_{Kr}(E_1, E_2, K) = F_{Kr}(E_1, K) \cup F_{Kr}(E_2, K)$

$$F_{Kr}(E_k, K) = \begin{cases} K & \text{if } k \in K \\ F_{Kr}(E, K) & \text{otherwise} \end{cases}$$

Rec (E) consists of the keys that can be recovered from E using information available in E . on other words, The result of $\text{rec}(E)$ is the set of the keys that you can apply to the given protocol.

1.4. Inductive definition of key recovery (rec)

Let E be an expression, then we define rec by :

$\text{rec } E = \bigcup_i G_i E = G E E$ where

$G_0(E) = \phi$

$G_i E = F_{Kr}(E, G_{i-1} E)$

Example:

$$E = ((\{ \{ \{ k_2 \}_{k_1} \}_{k_1}, \{ k_3 \}_{k_2} \} , k_1)$$

$$G_0(E) = \phi$$

$$G_1(E) = F_{kr} (E , G_0(E)) = F_{kr} (E , \phi) = \{ k_1 \}$$

$$G_2(E) = F_{kr} (E , \{ k_1 \}) = \{ k_1 , k_2 \}$$

$$Rec(E) = G_3(E) = G_4(E) = F_{kr} (E , \{ k_1 , k_2 \}) = \{ k_1 , k_2 , k_3 \}$$

1.5. The pattern of an expression

We denote by $Keys(E)$ the set of all keys occurring in E , and let $hidden E = Keys E - rec(E)$. Define $Pat(E,K)$ as follows:

$$Pat (b,K) = b$$

$$Pat (k,K) = k$$

$$Pat ((E_1 , E_2) , K) = (Pat(E_1, K) , Pat(E_2, K))$$

$$Pat (\{E\}_k , K) = \begin{cases} Pat(E,K)_k & \text{if } k \in K \\ \square & \text{otherwise} \end{cases}$$

Finally, we define pat by $Pat(E) = Pat(E,rec(E))$. Intuitively, $Pat(E)$ is the pattern corresponding to E , given the knowledge of any keys that may be recovered from E .

<p>Example1: $E = ((\{k_2\}_{k_1})_{k_1}, \{k_3\}_{k_2}), k_1$ $rec(E) = \{k_1, k_2, k_3\}$ $Pat(E) = E$</p>	<p>Example2: $E = ((\{k_2\}_{k_1})_{k_1}, \{k_3\}_{k_2}), k_2$ $rec(E) = \{k_2, k_3\}$ $Pat(E) = \{\square, \{k_3\}_{k_2}\}, k_2$</p>
---	--

1.6. Pattern Equivalence

We write $E = F$ if $Pat(E) = Pat(F)$

We write $E \cong F$ if there is a renaming of keys such that $Pat(E) = Pat(F)$

Example:

$K' \neq K$ without renaming

$K' = K$ with renaming

$\langle k_1, k_2 \rangle \neq \langle k_3, k_3 \rangle \rightarrow$ No bijective

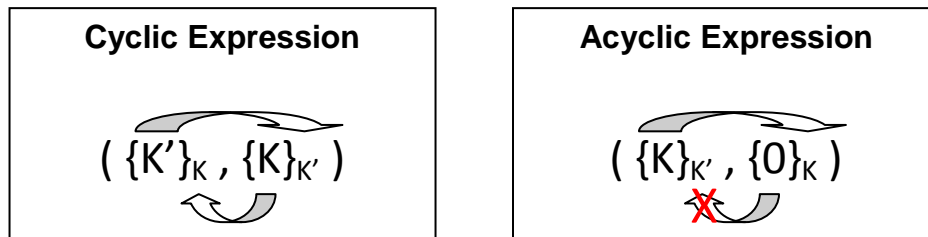
$\langle k_1, k_2 \rangle = \langle k_3, k_4 \rangle$ if k_1 renamed by k_3 and k_2 by k_4

1.7. Encryption Cycles

An encryption cycle is a cycle of the relation encrypts in which there is a key cycle.

Example: $(\{k_2\}_{k_1}; \{k_1\}_{k_2}) \rightarrow$ has a key cycle of size two because you need two operations to go to k_1 .

Encryption cycles are considered to be secure in the symbolic world, however, it is not secure in the computational world. Thus, it is recommended to avoid key cycle in terms in order to have correctness result.

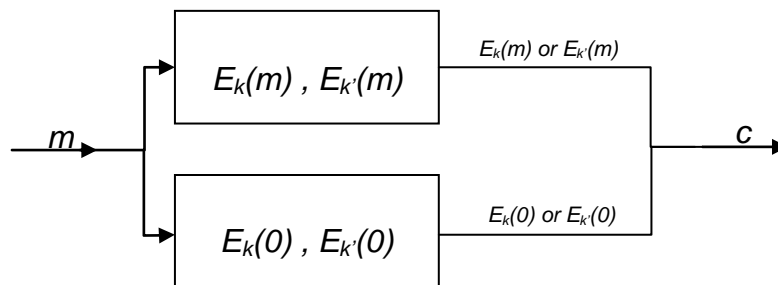


1.8. Type-0 Scheme

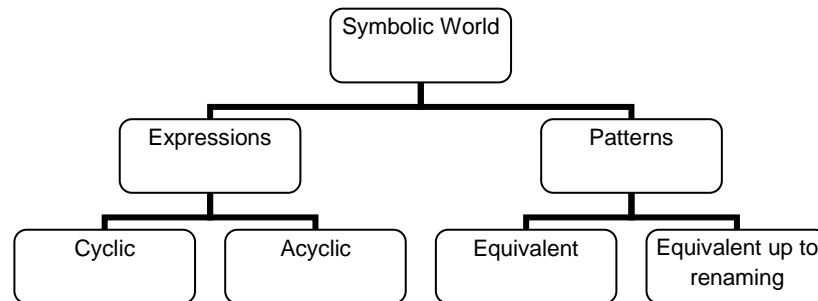
The main goal of this scheme is to prevent an attacker from being able to distinguish between given cipher-texts c and c' . It has three properties as follows:

1. Repetition-hiding: cannot determine if the corresponding plain-texts of c and c' are equivalent.
2. Which-key-hiding: cannot decide if the two cipher-texts are encrypted using the same key.
3. Message-length-hiding: cannot find the length of the corresponding plain-text of a given cipher-text.

The figure below explains how the scheme works. Furthermore, it shows that an adversary cannot determine which encryption box is used only with input/output of encryption.



The figure illustrated below summarizes the symbolic view and its main components.



1. Crypto Verif

CryptoVerif is an automatic protocol prover sound in the computational model. It can prove

- Secrecy.
- Correspondences, which include in particular authentication.

It provides a generic mechanism for specifying the security assumptions on cryptographic primitives, which can handle in particular symmetric encryption, message authentication codes, public-key encryption, signatures, hash functions.

The generated proofs are proofs by sequences of games, as used by cryptographers. These proofs are valid for a number of sessions polynomial in the security parameter, in the presence of an active adversary. CryptoVerif can also evaluate the probability of success of an attack against the protocol as a function of the probability of breaking each cryptographic primitive and of the number of sessions (exact security).

A game is formalized in a process calculus, which is an extension of the pi calculus. The semantic is purely probabilistic.

1.1. MACs: security definition

A MAC takes as input a message and a secret key $\text{mac}(m,k)$. It comes with a checking function check such that $\text{Check}(m,k,\text{mac}(m,k)) = \text{true}$

A MAC guarantees the integrity and authenticity of the message because only someone who knows the secret key can build the mac. More formally, an adversary A that has oracle access to mac and check has a negligible probability to forge a MAC (UF-CMA):

$\text{Max}_A \Pr[\text{check}(m,k,t) \mid k \leftarrow \text{mkgen}; (m,t) \leftarrow A^{\text{mac}(\cdot,k), \text{check}(\cdot,k,\cdot)}]$ is negligible, when the adversary A has not called the mac oracle on message m .

1.2. MACs: intuitive implementation

By the previous definition, the adversary has a negligible probability of forging a correct mac. So when checking a mac with $\text{check}(m; k; t)$ and k is secret, the check can succeed only if m is in the list (array) of messages whose mac has been computed by the protocol. So we can replace a check with an array lookup:

if the call to mac is $\text{mac}(x; k)$, we replace $\text{check}(m; k; t)$ with

$\text{find } j = \leq N \text{ suchthat } \text{defined}(x[j]) \wedge (m = x[j]) \wedge \text{check}(m; k; t) \text{ then true else false}$

Furthermore, we use primed function symbols after the transformation, so that it is not done again.

1.3. Formal implementation

$\text{check}(m; \text{kgen}(r); \text{mac}(m; \text{kgen}(r))) = \text{true}$

$!N$ new $r : \text{keyseed};$ {

$!N(x : \text{bitstring}) \rightarrow \text{mac}(x; \text{kgen}(r));$

$!N' (m : \text{bitstring}; t : \text{macstring}) \rightarrow \text{check}(m; \text{kgen}(r); t)$

\approx

$!N$ new $r : \text{keyseed};$ {

$!N(x : \text{bitstring}) \rightarrow \text{mac}'(x; \text{kgen}'(r));$

$!N' (m : \text{bitstring}; t : \text{macstring}) \rightarrow \text{find } j \leq N \text{ suchthat } \text{defined}(x[j]) \wedge (m = x[j]) \wedge \text{check0}(m; \text{kgen0}(r); t) \text{ then true else false}$

The prover understands such specifications of primitives. The prover applies the previous rule automatically in any (polynomial-time) context, perhaps containing several occurrences of mac and or check :

- Each occurrence of mac is replaced with mac0 .
- Each occurrence of check is replaced with a Find that looks in all arrays of computed MACs (one array for each occurrence of function mac).

In most cases, the prover succeeds in proving the desired properties when they hold, and obviously it always fails to prove them when they do not hold.

Only cases in which the prover fails although the property hold:

- Needham-Schroeder public-key when the exchanged key is the nonce N_A .
- Needham-Schroeder shared-key: fails to prove that $NB[i] \neq NB[i'] - 1$ with overwhelming probability, where NB is a nonce.
- Showing that the encryption $\epsilon(m, r) = f(r) || H(r) \oplus m || H'(m, r)$ scheme is IND-CCA2.

In conclusion the benefit of automated machine in reduction even if it takes long proof but in short time but the proof is accurate than the proof that done by cryptographers.

2. Automated Cryptographic Proofs for Asymmetric Encryption

The goal of automated cryptographic proofs for asymmetric encryption is to provide a sound automated proof method for IND-CCA security of generic encryption schemes.

Examples of that proofs:

Bellare & Rogaway'93: $f(r) || x \oplus G(r) || H(x || r)$
 Zheng & Seberry'93: $f(r) || G(r) \oplus (x || H(x))$
 OAEP'94 (Bellare & Rogaway): $f(s || r \oplus H(s))$ where $s = x \oplus G(r)$
 OAEP+'02 (Shoup): $f(s || r \oplus H(s))$ where $s = x \oplus G(r) || H_0(r)$
 Fujisaki & Okamoto'99: $E((x || r); H(x || r))$. where E is IND-CPA.

2.1. Generic Encryption Scheme

Here a notation is introduced (a simple programming language) in which the encryption and decryption oracles are specified. The motivation for fixing a notation is obvious: it is mandatory for developing an automatic verification procedure. Let Var be an arbitrary finite non-empty set of variables. Then, the programming language is built according to the following BNF described in Figure 1, where for a bit-string $bs = b_1 \cdot \dots \cdot b_k$ (b_i are bits), $bs[n, m] = b_n \cdot \dots \cdot b_m$, and N is the name of the oracle, c its body and x and y are the input and output variable respectively. Commands are standard, where $x \leftarrow U$ means that the value of x is randomly sampled following the uniform distribution on the appropriate domain, \oplus is the bitwise-xor operation and $||$ is the string concatenation.

<p>Command $c ::= x \leftarrow U \mid x := f(y) \mid x := f^{-1}(y) \mid x := H(y) \mid$ $x := y[n,m]$ $\mid x := y \oplus z \mid x := y \parallel z \mid \text{if } x = y \text{ then } c_1 \text{ else } c_2 \text{ fi} \mid c; c$ Oracle $O ::= N(x,y) : c$</p>

Figure 1: Language grammar.

A generic encryption scheme is a triple $(\mathbf{FF}, E(\text{ine}, \text{oute}) : c, D(\text{ind}, \text{outd}) : c')$:

1. \mathbf{F} is a trapdoor permutation generator that on input η generates an η -bit string
trapdoor permutation (f, f^{-1})
2. $E(\text{ine}, \text{oute}) : c$ and $D(\text{ind}, \text{outd}) : c'$ are oracle declarations

Example on generic encryption scheme is Bellare-Rogaway'93:

$$f(r) \parallel \text{ine} \oplus G(r) \parallel H(\text{ine} \parallel r)$$

<p>Encryption $E(\text{ine}, \text{oute}) =$ $r \xleftarrow{r} U;$ $a = f(r);$ $g := G(r);$ $b := \text{ine} \oplus g;$ $t := \text{ine} \parallel r$ $c := H(t);$ $\text{oute} := a \parallel b \parallel c$</p>	<p>Decryption $D(\text{ind}, \text{outd})$ match ind with $a^* \parallel b^* \parallel c^*;$ $r^* := f^{-1}(a^*);$ $g^* := G(r^*);$ $m^* := b^* \oplus g^*;$ $t^* := m^* \parallel r^*;$ $h^* := H(t^*);$ if $h^* = c^*$ then $\text{outd} := m^*$ else $\text{outd} := \text{error}$</p>
--	---

The aiming for developing a procedure that allows us to prove properties, i.e. invariants, of the encryption oracle. More precisely, the procedure annotates each control point of the encryption command with a set of predicates that hold at that point for any execution except with negligible probability. Given an encryption oracle $E(\text{ine}, \text{oute}) : c$ we want to prove that at the final control point, we have an invariant that tells us that the value of oute is indistinguishable from a random value. As we will show, this implies IND-CPA security.

A few words now concerning how we present the verification procedure. First, we present the invariant properties we are interested in the assertion language. Then, we present a set of rules of the form $\{\phi\}c\{\phi'\}$ meaning that execution of command c in any distribution that satisfies ϕ leads to a distribution that satisfies ϕ' . Using Hoare logic terminology, this means that the triple $\{\phi\}c\{\phi'\}$ is valid.

2.2. The assertion Language

States facts about randomness of the values of the variables, their secrecy and the randomness of their hashed values. Such that:

$$\psi ::= \text{Indis}(vx; V_1; V_2) | \text{WS}(x; V) | H(H, e)$$
$$\varphi; ; = \text{true} | \psi | \varphi \wedge \varphi$$

$\text{Indis}(vx; V_1; V_2)$: x is indistinguishable from a uniformly sampled value, even when the adversary is given the values in V_1 and the f -images of those in V_2 .

$\text{WS}(x; V)$: the value of x is hard to compute given the values in V .

$H(H, e)$: the value of the expression e has not been queried to H .

Example 1 on indistinguishably:

Let f be a one-way permutation.

Distribution D1:

Uniformly sample x in $\{0, 1\}^n$.

Return $x, H(x)$

Distribution D2:

Uniformly sample x in $\{0, 1\}^n$.

Uniformly sample x' in $\{0, 1\}^n$.

Return $x', H(x)$

D1 $\not\equiv$ D2

$x \xleftarrow{r} U$

$y := H(x)$;

$z := x$ Indis(z ; y) does not hold

Example 2 on indistinguishably:

Let f be a one-way permutation.

Distribution D1:

Uniformly sample x in $\{0, 1\}^n$.

Return $f(x), H(x)$

Distribution D2:

Uniformly sample x in $\{0, 1\}^n$.

Uniformly sample x_0 in $\{0, 1\}^n$.

Return $f(x_0), H(x)$

D1 $\not\equiv$ D2

$x \xleftarrow{r} U$

$y := H(x);$
 $z := f(x)$ Indis($z; y$) holds

Because the adversary can query H on x and compare it to y .

2.3. Standard rules

Sequential composition and consequence rule I A set of axioms for each command - this provides a semantic characterization for one-way functions, hash functions in the ROM,

- $\{true\} x \stackrel{r}{\leftarrow} U \{Indis(vx) \wedge H(H, x)\},$
- $\{Indis(vy; V U \{y\}; \emptyset)\} x := f(y) \{WS(y; V U \{x\})\}$ if $y \in V U \{x\}$
 $\{WS(y; V) \wedge H(H, y)\} x := H(y) \{Indis(vx; V U \{x\}; \emptyset)\}$

2.4. Soundness of the analysis

2.4.1. Proposition

Let X be distribution that is computable in polyt-time with oracle access to hash functions and given the function f .

Then, for every rule $\{\varphi\}c\{\varphi'\}$, we have

$X \models \varphi$ implies $[[c]]X \models \varphi'$.

2.4.2. Theorem

Let $GE = (F, E(\text{ine}, \text{oute}) : c, D(\text{ind}, \text{outd}) : c_0)$ be an asymmetric encryption scheme.

If $\{true\}c\{Indis(_oute; \text{ine}, \text{oute}, s; ;)\}$ then E is RR-C.

Example:

Bellare & Rogaway's 1993 generic construction.

$r \stackrel{r}{\leftarrow} \{0,1\}^{n_0}$ $\neg Indis(vr) \wedge H(G, r) \wedge H(H, \text{ine}||r)$
 $a := f(r)$ $\neg Indis(va; Var - r) \wedge H(G, r) \wedge WS(r; Var - r) \wedge H(H, \text{ine}||r)$
 $g := G(r)$ $\neg Indis(va; Var - r) \wedge Indis(vg; Var - r) \wedge WS(r; Var - r) \wedge H(H, \text{ine}||r)$
 $e := \text{ine} _ g$ $\neg Indis(va; Var - r) \wedge Indis(ve; Var - g, r) \wedge WS(r; Var - r) \wedge H(H, \text{ine}||r)$
 $d := \text{ine}||r$ $\neg Indis(va; Var - r, d) \wedge Indis(ve; Var - r, d, g) \wedge WS(d; Var - r, d) \wedge H(H, d)$
 $c := H(d)$ $\neg Indis(va; Var - r, d) \wedge Indis(ve; Var - r, d, g) \wedge Indis(vc, Var - r, d)$
 $\text{oute} := a||e||c$ $\neg Indis(v \text{oute}; \text{ine}, \text{oute}, s)$