Introduction à la Cybersécurité (Inter-Semestre 1A)

Mounira Msahli et Pascal Lafourcade

Telecom Paris, Institut Polytechnique de Paris

7 février 2024

References

Pr.Stéphanie Delaune.
Verification of security protocols : from confidentiality to privacy.
LSV, CNRS ENS Cachan, France.

Pr. Bill Young.
Introduction to computer security : Cryptographic protocols.
Department of Computer Sciences University of Texas at Austin, Foundations of Computer Security.

Pr.David Basin.
Symbolic verification of cryptographic protocols using tamarin.
ETH Zurich.

Pr.Pascal Lafourcade.
Security models.
Université Clermont Auvergne.

Bruno Blanchet.
The automatic security protocol verifier proverif.
CNRS, Ecole Normale Supérieure, INRIA, Paris.

Comon-Lundh H. Delaune S. Fournet C. Kremer S. Pointcheval D. Blanchet, B.
Cryptographic protocols formal and computational proofs.
MPRI Lecture Notes.

Plan

- Cryptographic Protocols
- Protocol Attacks
- Formal analysis of security protocols
- Dolev Yao's Model
- Model of protocols : ProVerif

Learning Outcomes

- Understanding of principles of secure design, security goals and attacker capabilities
- Gaining the ability to implement in reliable and effective way simple security protocols. This course is also oriented at getting the students slightly familiar with techniques like formal modelling and verification of security protocols
- It should be mentioned that our objective is not a mere transfer of knowledge but we aim at developing a cognitive learning. In part, the goal is to show how the behaviour of attacker could be predicted and controlled

Cryptographic protocols everywhere !

Cryptographic Protocols

- Definition : a protocol is a structured dialogue among two or more parties in a distributed context controlling the syntax, semantics and synchronization of communication, and designed to accomplish a communication-related function.
- Definition : a cryptographic protocol is a protocol using cryptographic mechanisms to accomplish some security-related function.

Cryptographic Protocols

An analysis of any protocol attempts to answer the following types of questions :

- What are the goals of the protocol ?
- What does the protocol actually achieve ?
- Does it achieve its stated objective ?
- Does it include unnecessary steps or messages ?
- What assumptions are made ?
- Does it encrypt items that could be sent in the clear ?
- Is it susceptible to attack ? What would an attack look like ?

Cryptographic Protocols

Among the goals of cryptographic protocols are the following :

- Unicity : secret shared by exactly two parties
- Integrity : message arrived unmodified
- Authenticity : message claim of origin is true
- Confidentiality : message contents are inaccessible to an eaves-dropper
- Non-repudiation of origin : sender can't deny sending
- Non-repudiation of receipt : receiver can't deny receiving

Cryptographic Protocols

A protocol involves a sequence of message exchanges of the form :

$$A \rightarrow B : M$$

meaning that principal A sends to principal B the message M. Because of the distributed nature of the system and the possibility of malicious actors, there is typically no guarantee that B receives the message, or is even expecting the message.

Cryptographic Protocols : Example

Assumptions

- Names : A, B or Alice, Bob, ...
- Nonces : N_a. Fresh data
- Keys : K and inverse keys K^{-1}
- Asymmetric Encryption : $\{M\}_{K_a}$
- Symmetric Encryption : $\{M\}_{K_{ab}}$
- Message concatenation : $\langle M_1, M_2 \rangle$

Cryptographic Protocols : Example

Consider the following simple protocol :

- A $\rightarrow$ B : $\{\{K\}_{K_a^{-1}}\}_{K_b}$
- B $\rightarrow$ A : $\{\{K\}_{K_b^{-1}}\}_{K_a}$

Informal goal : A shares with B a secret key K, and each part is authenticated to the other.

- What are the assumptions ? Precisely what are the goals ? Are they satisfied ? How can you be sure ?
- However, this protocol is vulnerable. Can you see how ?

Cryptographic Protocol Attacks : Examples

Recall the following protocol, introduced previously.

- 1. A $\rightarrow$ B : $\{\{K\}_{K_a^{-1}}\}_{K_b}$
- 2. A $\rightarrow$ B : $\{\{K\}_{K_b^{-1}}\}_{K_a}$

Suppose an attacker C obtains the message (step 1) : $\{\{K\}_{K_a^{-1}}\}_{K_b} = K\prime$. Then, C initiates a new run of the protocol with B :

- 1. C $\rightarrow$ B : $\{\{K\prime\}_{K_C^{-1}}\}_{K_b}$
- 2. B $\rightarrow$ C : $\{\{K\prime\}_{K_b^{-1}}\}_{K_c}$

The message that B sends back is :
$\{\{K\prime\}_{K_b^{-1}}\}_{K_c} = \{\{\{\{K\}_{K_a^{-1}}\}_{K_b}\}_{K_b^{-1}}\}_{K_c} = \{\{K\}_{K_a^{-1}}\}_{K_c}$
allowing C to extract the original K.

Protocol Attacks : Examples

- **Man-in-the-middle attack :** transmettre les messages via une autre session : A $\leftrightarrow$ I $\leftrightarrow$ B.
- **Replay (or freshness) attack :** reuse parts or all of previous messages.
- **Masquerading attack :** pretend to be another principal.
- **Reflection attack :** send transmitted information back to originator
- **Oracle attack :** take advantage of normal protocol responses as encryption and decryption “services”.
- **Type flaw attack :** substitute a different type of message field.

Example of a reflection attack

We consider this challenge-response authentication protocol :

- M1. $A \rightarrow B : \{|N_a|\}_{K_{ab}}$
- M2. $B \rightarrow A : \{|N_a + 1|\}_{K_{ab}}$

admits a reflection attack (with ‘Oracle’)

Example of a reflection attack

This challenge-response authentication protocol :

1. M1. A $\rightarrow B : \{|N_a|\}_{K_{ab}}$
2. M2. B $\rightarrow A : \{|N_a + 1|\}_{K_{ab}}$

admits a reflection attack (with ‘Oracle’) :

1. (M1.1). $A \rightarrow i(B) : \{|N_a|\}_{K_{ab}}$
2. (M2.1). $i(B) \rightarrow A : \{|N_a|\}_{K_{ab}}$
3. (M2.2). $A \rightarrow i(B) : \{|N_a + 1|\}_{K_{ab}}$
4. (M1.2). $i(B) \rightarrow A : \{|N_a + 1|\}_{K_{ab}}$

A works on behalf the adversary : A acts as an ‘Oracle’, since she provides the correct answer to her own question. A believes (at least) that B is operational, whereas B may no longer exist.

Example of a reflection attack

To fix this attack

Add A's name to message M1 or use different keys for each direction (*i.e.*, $K_{ab} \neq K_{ba}$).

Example of a reflection attack

Reflection attack (with 'oracle')

1 M1.1. A $\rightarrow i(B) : \{|N_a|\}_{K_{ab}}$

2 M2.1. i(B) $\rightarrow A : \{|N_a|\}_{K_{ab}}$

3 M2.2. A $\rightarrow i(B) : \{|N_a + 1|\}_{K_{ab}}$

4 M1.2. i(B) $\rightarrow A : \{|N_a + 1|\}_{K_{ab}}$

This attack requires that A executes several protocols that runs in parallel.

Threat Assumption

The adversary may start any number of parallel protocols that run between any principals including different runs involving the same principals and with principals taking the same or different protocol roles.

Example of a reflection attack

Hence, our formal protocol model will allow :

1. for an unbounded number of protocol runs of arbitrary roles
2. with arbitrary participating principals.

Example of Type flaw attacks

A message consists of a sequence of submessages.
Real messages are bit strings without type information.
10110110001011100011011110100000
Type flaw is when $A \rightarrow B$: M
and B accepts M as valid but parses it differently. I.e.,
B interprets the bits differently than A.

Example of type flaw attacks

"HONEY... HAND ME THE HAIR DRYER"...

Cryptographic Protocols : Diffie-Hellman

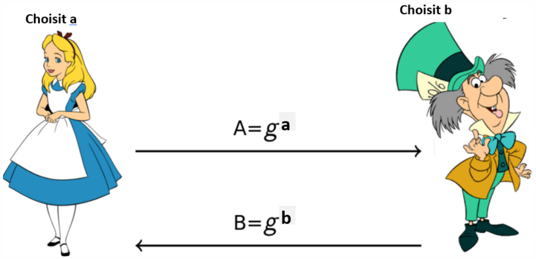


Figure – General model of Diffie-Hellmann Key Agreement

This keys will now match because exponentiation is commutative :

g, p are public parameters.

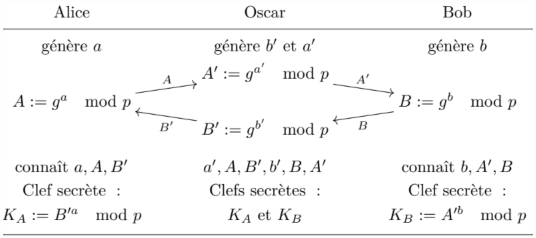
$$(g^b)^a \equiv (g^a)^b \mod [p]$$

$$g^{ba} \equiv^{ab} \mod [p]$$

$$(g^b)^a mod[p] = k = g^{ab} mod[p] = (g^a)^b mod[p]$$

Cryptographic Protocols : Diffie-Hellman

Diffie-Hellman is only secure against *passive* adversaries. An active adversary could simply intercept the transmissions, and switch the values with her self picked exponents.



$$\begin{tikzcd}
\text{Alice} & & \text{Oscar} & & \text{Bob} \\
\text{génère } a & & \text{génère } b' \text{ et } a' & & \text{génère } b \\
& & A' := g^{a'} \mod p \arrow[rrd, "A'"] & & \\
A := g^{a} \mod p \arrow[rru, "A"] & & & & B := g^{b} \mod p \arrow[lld, "B"] \\
& & B' := g^{b'} \mod p \arrow[llu, "B'"] & & \\
\text{connaît } a, A, B' & & a', A, B', b', B, A' & & \text{connaît } b, A', B \\
\text{Clef secrète :} & & \text{Clefs secrètes :} & & \text{Clef secrète :} \\
K_A := B'^{a} \mod p & & K_A \text{ et } K_B & & K_B := A'^{b} \mod p
\end{tikzcd}$$

Figure – Model of Man-in-the-Middle attack on Diffie-Hellmann Key Agreement

For Oscar, to exploit this attack, he has to intercept all transmissions after the key-agreement has been made, and re-encrypt the messages because Alice's and Bob's keys *probably* doesn't match.

Cryptographic Protocols : Diffie-Hellman

To prevent MITM attacks, one can use public-key authentication. This requires that Alice and Bob have each others (certified) public-keys.

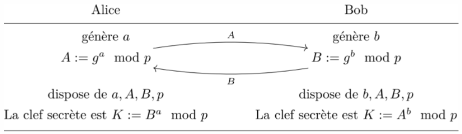


Figure – Model of Authenticated Diffie-Hellman Key Agreement

Needham-Schroeder Protocol 1978

Basic Definition

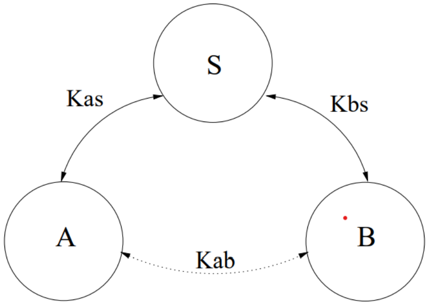
A $\rightarrow$ B : $\{A, N_a\}_{pub(B)}$
B $\rightarrow$ A : $\{N_a, N_b\}_{pub(A)}$
A $\rightarrow$ B : $\{N_b\}_{pub(B)}$
N_a is a nonce generated by A. N_b is a nonce generated by B.

Use of Nonce and Timestamp

- NSP uses nonces (short for "numbers used once"), randomly generated values included in messages.
- If a nonce is generated and sent by A in one step and returned by B in a later step, A knows that B's message is fresh and not a replay from an earlier exchange.
- Note that a nonce is not a timestamp. The only assumption is that it has not been used in any earlier interchange, with high probability.

Needham-Schroeder with three principals

There are three principals : A and B, two principals desiring mutual communication, and ,S, a trusted key server.



It is assumed that A and B already have secure symmetric communication with S using keys K_{as} and K_{bs} , respectively.

Needham-Schroeder Protocol

The Needham-Schroeder protocol is :

- 1) A $\rightarrow$ S : A,B,N_a
- 2) S $\rightarrow$ A : $\{N_a, B, K_{ab}, \{K_{ab}, A\}_{K\,bs}\}_{K\,as}$
- 3) A $\rightarrow$ B : $\{K_{ab}, A\}_{K\,bs}$
- 4) B $\rightarrow$ A : $\{N_b\}_{K\,ab}$
- 5) A $\rightarrow$ B : $\{N_b - 1\}_{K\,ab}$

Here N_a and N_b are nonces.

Needham-Schroeder Protocol : Example of attack

- 17 years after the publication of the protocol Gavin Lowe discovered an attack that may occur in the presence of an active adversary. The attack has been coined man-in-the-middle attack.
- Agent A starts a session with a dishonest agent C.
- Agent C uses this message to fake being A to B. B responds to A. As B's message contains the nonce N_a,
- A accepts the message thinking it originates from C. Therefore A sends to C the nonce N_b encrypted with C's public key.
- C can recover the nonce N_b and end the protocol with B who thinks having executed the protocol with B.

Needham-Schroeder Protocol : Example of attack

Denning and Sacco pointed out that the compromise of a session key has bad consequences. An intruder can reuse an old session key and pass it off as a new one as though it were fresh.

- A $\rightarrow$ S : A,B, N_a
- S $\rightarrow$ A :$\{N_a, B, K_{ab}, \{K_{ab}, A\}_{K\,bs}\}_{K_{as}}$
- A $\rightarrow$ B :$\{K_{ab}, A\}_{K_{bs}}$
- B $\rightarrow$ A : $\{N_b\}_{K_{ab}}$
- A $\rightarrow$ B : $\{N_b - 1\}_{K_{ab}}$

Suppose C has cracked K_{ab} from last week's run of the protocol, and has squirreled away message 3 from that session : $\{K_{ab}, A\}_{K_{bs}}$.

Needham-Schroeder Protocol : Example of attack

Suppose C has cracked K_{ab} from last week's run of the protocol, and has squirreled away message 3 from that session : $\{K_{ab}, A\}_{K_{bs}}$.

- C $\rightarrow$ B : $\{K_{ab}, A\}_{K_{bs}}$
- B $\rightarrow$ C : $\{N_b\}_{K_{ab}}$
- C $\rightarrow$ B : $\{N_b - 1\}_{K_{ab}}$

B will believe, he is talking to A.

Needham-Schroeder Protocol : Example of attack

Bauer, et al. pointed out that if key K_{as} were compromised, anyone could impersonate A and establish communication with any other party

- A $\rightarrow$ S : A, B, N_a
- S $\rightarrow$ A : $\{N_a, B, K_{ab}, \{K_{ab}, A\}_{K\,bs}\}_{K_{as}}$
- A $\rightarrow$ B : $\{K_{ab}, A\}_{K_{bs}}$
- B $\rightarrow$ A : $\{N_b\}_{K_{ab}}$
- A $\rightarrow$ B : $\{N_b - 1\}_{K_{ab}}$

The "attacks" discovered by Denning and Sacco and by Bauer, et al. ask what happens if a key is broken.
Is it fair to ask that question ?
Isn't a presumption of any cryptographic protocol that the encryption is strong ?
How might you address these flaws if you were the protocol designer ?

Needham-Schroeder Protocol : Example of attack

- Problem : Message 3 is not protected by nonces. There is no way for B to know if the K_{ab} it receives is current. An intruder has unlimited time to crack an old session key and reuse it as if it was fresh.
- Example Attack : an employee runs the first few steps of the protocol multiple times, gathering up tickets $\{K_{ab}, A\}_{K_{bs}}$ for each different server B in the system. If he's fired, he can still log onto all of the company's servers.

Needham-Schroeder Protocol : Example of attack

- A message consists of a sequence of sub-messages.
- Examples : a principal's name, a nonce, a key, ...
- Messages sent as bit strings. No type information. 1011 0110 0010 1110 0011 0111 1010 0000
- Type flaw is when A $\rightarrow$ B : M and B accepts M as valid but parses it differently. I.e., B interprets the bits differently than A.
- Example : two 16-bit nonces $\{N_a, N_b\}$ could be mistaken as a 32-bit shared key. Let's consider several examples from actual protocols.

Otway-Rees protocol : Example of attack

Another cryptographic protocol is the Otway-Rees protocol. Below is one of several variants :

- A $\rightarrow$ B : M,A,B, $\{N_a, M, A, B\}_{K_{as}}$
- B $\rightarrow$ S : M,A,B, $\{N_a, M, A, B\}_{Kas}, \{N_b, M, A, B\}_{K_{bs}}$
- S $\rightarrow$ B : M, $\{N_a, K_{ab}\}_{Kas}, \{N_b, K_{ab}\}_{K_{bs}}$
- B $\rightarrow$ A : M, $\{N_a, K_{ab}\}_{K_{as}}$

Here M, is a session identifier ; N_a and N_b are nonces.
What are the assumptions ?
What is the goal ?

Otway-Rees protocol : Example of attack

A malicious intruder can arrange for A and B to end up with different keys.

- After step 3, B has received K_{ab}.
- An intruder then intercepts the fourth message.
- The intruder resends message 2, so S generates a new key $K\prime_{ab}$, sent to B.
- The intruder intercepts this message too, but sends to A M,$\{N_a, K\prime_{ab}\}K_{as}$.
- A has $K\prime_{ab}$, while B has K_{ab}.

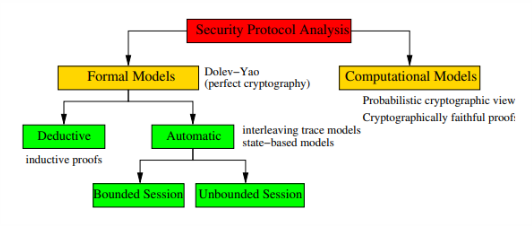
Another problem : although the server tells B that A used a nonce, B doesn't know if this was a replay of an old message.

Formal analysis of security protocols

There are two main ways to “prove” security from an abstract point of view :

- Symbolic approach : cryptographic functions are seen as function on symbolic space. Security properties are formally defined.
- Computational approach : cryptographic functions are seen as functions on bit strings. Security properties are defined in terms of probability and complexity.

Formal analysis of security protocols



Dolev Yao's Intruder

Dolev-Yao 1982

- Intruder controls the network and can :
  - Intercept messages
  - Modify messages
  - Block messages
  - Generate new messages
  - Insert new messages
- Protocol has :
  - Arbitrary number of principals
  - Arbitrary number of parallel sessions
  - Messages with arbitrary size

What are formal models ?

A Language is formal

- When it has a well-defined syntax and semantics. Additionally there is often a deductive system for determining the truth of statements.
- A model (or construction) is formal when it is specified in a formal language
- Standard protocol notation is not formal
- We will see how to formalize such notations

Formal modeling and analysis of protocols

- Goal : formally model protocols and their properties
- Basis : suitable abstraction of protocols.
- Analysis : with formal methods based on mathematics and logic, e.g., theorem proving

Formal modeling and analysis of protocols

Term rewriting is :

- A useful and flexible formalism in general
- Programming languages
- Automated deduction
- Rewriting logic
- Used to represent messages and protocols using formal verification tools

Dolev Yao's Intruder

A sequent is an expression of the form $T \vdash u$.

Definition

- A proof of a sequent $T \vdash u$ is a tree whose nodes are labeled by either sequents or expressions of the form $v \in T$, such that :
- Each leaf is labeled by an expression of the form $v \in T$, and each non-leaf node is labeled by a sequent.
- Each node labeled by a sequent $T \vdash v$ has n children labeled by $v \in T\ s_1, \ldots, v \in T\ s_n$ such that there is an instance of an inference rule with conclusion $v \vdash_E v$ and hypotheses $T \vdash s_1, \ldots, T \vdash s_n$.
- The root of the tree is labeled by $T \vdash u$.

Dolev Yao : Deduction System

Deduction System

On the Security of Public Key Protocols (IEEE Trans. Inf. Th., 1983)

- Size of a proof P of $T \vdash u$ is denoted by |P|, is the number of nodes in the proof.
- A proof P of $T \vdash u$ is minimal if there does not exist a proof $P\prime$ of $T \vdash u$ such that $|P\prime| < |P|$. A sequent is an expression of the form $T \vdash u$.

Dolev-Yao Deduction

Symbolic Manipulation on Terms

May build new messages following deduction rules.

Pairing

$$\frac{x \quad y}{<x,y>} \qquad \frac{<x,y>}{x} \qquad \frac{<x,y>}{y}$$

Symmetric encryption

$$\frac{x \quad y}{senc(x,y)} \qquad \frac{senc(x,y) \quad y}{x}$$

Asymmetric encryption

$$\frac{x \quad y}{aenc(x,y)} \qquad \frac{aenc(x,y) \quad sk(y)}{x}$$

Signature

$$\frac{x \quad sk(y)}{sign(x,sk(y))}$$

Deduction relation T and u

Symbolic Manipulation on Terms

We say that u is deducible from T if there exists a proof tree such that :

each leaf is labeled by v with $v \in T$;
for each node labeled by v_0 and having n sons labeled by $v_1, \ldots,$ v_n, there exists a deduction rule R such that

$$\frac{v_1 \ldots v_n}{v_0} \quad \text{is an instance of R}$$

The root is labeled by u.
A subproof of a proof P is a subtree of P.

Exercise

Denning Sacco protocol

Let $T = a, b, c, sk(c), aenc(sign(k, sk(a)), c), senc(s, k)$.
Is s deducible from T ?

Answer : Yep !

$$\frac{senc(s,k) \qquad \dfrac{\dfrac{aenc(sign(k,sk(a)),c) \quad sk(c)}{sign(k,sk(a))}}{k}}{s}$$

Exercise

Denning Sacco protocol

1. A $\rightarrow$ C : aenc(sign(k, priv(A)), pub(C))
2. C(A) $\rightarrow$ B : aenc(sign(k, priv(A)), pub(B))
3. B $\rightarrow$ A : senc(s, k) Attack !

Exercise

Let T_0 = {a, b, c, sk(c), aenc(sign(k,sk(a)), c)}.
Is aenc(sign(k,sk(a)), b) deducible from T_0 ?

Exercise

Answer

Answer : Of course, Yes ! 1. A $\rightarrow$ C : aenc(sign(k, priv(A)), pub(C))
2. C(A) $\rightarrow$ B : aenc(sign(k, priv(A)), pub(B))
3. B $\rightarrow$ A : senc(s, k) Attack !

Answer : Yep !

$$\frac{\frac{aenc(sign(k,sk(a)),c)\;sk(c)}{sign(k,sk(a)) \qquad b}\,k}{aenc(sign(k,sk(a)),b)}$$

The deduction problem

Input : a finite set of terms T (the knowledge of the attacker) and a term u (the secret),
Output : Is u deducible from T ?

How does a cryptographic protocol work (or not) ?

Example : A simplified version of the Denning-Sacco protocol (1981)

$A \rightarrow B$: aenc(sign(k, priv(A)), pub(B))

$B \rightarrow A$: senc(s, k)

What about secrecy of s ?

Consider a scenario where A starts a session with C who is dishonest.

1. $A \rightarrow C$: aenc(sign(k, priv(A)), pub(C)) C knows the key k

How does a cryptographic protocol work (or not) ?

Example : A simplified version of the Denning-Sacco protocol (1981)
A $\rightarrow$ B : aenc(sign(k, priv(A)), pub(B))
B $\rightarrow$ A : senc(s, k)
What about secrecy of s ?
Consider a scenario where A starts a session with C who is dishonest.
1. A $\rightarrow$ C : aenc(sign(k, priv(A)), pub(C)) C knows the key k
2. C(A) $\rightarrow$ B : aenc(sign(k, priv(A)), pub(B))
3. B $\rightarrow$ A : senc(s, k) Attack !

How does a cryptographic protocol work (or not) ?

Exercice

We propose to fix the Denning-Sacco protocol as follows :

Version 1

A $\rightarrow$ B : aenc(hA,B,sign(k, priv(A))i, pub(B))

B $\rightarrow$ A : senc(s, k)

Version 2

A $\rightarrow$ B : aenc(sign(hA,B, ki, priv(A))i, pub(B))

B $\rightarrow$ A : senc(s, k)

Which version would you prefer to use ?

How does a cryptographic protocol work (or not) ?

Exercice

We propose to fix the Denning-Sacco protocol as follows :

Version 1

A $\rightarrow$ B : aenc(⟨A,B,sign(k, priv(A))⟩, pub(B))

B $\rightarrow$ A : senc(s, k)

Version 2

A $\rightarrow$ B : aenc(sign(⟨A,B, k⟩, priv(A))⟩, pub(B))

B $\rightarrow$ A : senc(s, k)

Which version would you prefer to use ?   Version 2

$\rightarrow$ Version 1 is still vulnerable to the aforementioned attack.

Dolev Yao Model : actions, roles and protocol

Definition (Action)

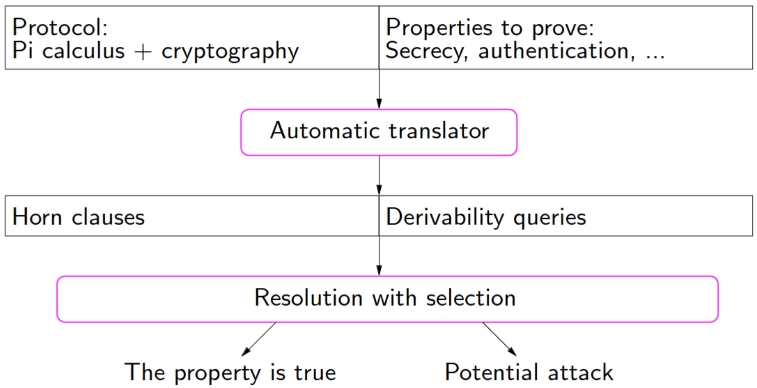
An action is a couple (recv(u), send(v)) such that $u \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \cup \{init\}, v \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \cup \{stop\}$. $Denoted(u \rightarrow v)$.

Example

First and last actions of Needham Schroeder

- $(init, X_b \rightarrow \{N_a, A\}_{pk(X_b)})$
- $(\{N_b\}_{pk(B)} \rightarrow stop)$

Model of protocols : ProVerif



ProVerif : Definition of Cryptographic Primitives

Two kinds of operations :

- Constructors f are used to build terms $f(M_1, ..., M_n)$
  fun $f(T_1, ..., T_n) : T$.
- Destructors g manipulate terms
  Destructors are defined by rewrite rules $g(M_1, ..., M_n) \rightarrow M$.
  reduc forall $x_1 : T_1, ..., x_k : T_k; g(M_1, ..., M_n) = M$.

ProVerif : Examples of constructors and destructors

Shared-key encryption

Two kinds of operations :

- Constructor : Shared-key encryption
  fun encrypt(bitstring, key) : bitstring
- Destructor : Decryption
  reduc forall x : bitstring, y : key ; decrypt(encrypt(x, y), y)= x.

ProVerif : Examples of constructors and destructors

Probabilistic shared-key encryption :

- Constructor : Shared-key encryption
  fun encrypt(bitstring, key, coins) : bitstring.
- Destructor : Decryption
  reduc forall x : bitstring, y : key, z : coins ;
  decrypt(encrypt(x,y,z),y) = x.
  To encrypt m under k, one generates fresh random coins r and computes encrypt(m,k,r)

ProVerif : Examples of constructors and destructors

Constructors(Signature)

- Public key generation :
  fun spk(sskey) : spkey.
- Signature :
  fun sign(bitstring, sskey) : bitstring.

Destructors(Signature verification)

- reduc forall m : bitstring, k : sskey ; checksign(sign(m, k), spk(k)) = m.
- Message extraction :
  reduc forall m : bitstring, k : sskey ; getmess(sign(m, k)) = m.

We represent in the same way public-key encryption and hash functions, . . .

ProVerif : Equations

An alternative way of modeling primitives is using equations :

- More powerful but harder to handle.
- The model used in the applied pi calculus [Abadi, Fournet, POPL'01].
- ProVerif handles equations by translating them into rewrite rules.
  - Advantage : can still use fast, syntactic unification.
  - Limitations : e.g., cannot handle associativity.

ProVerif : Equations

Symmetric encryption in which one cannot detect whether decryption succeeds.

- Encryption and decryption functions :
  fun encrypt(bitstring, key) : bitstring.
  fun decrypt(bitstring, key) : bitstring.
- Equations :
  equation forall x : bitstring, y : key ; decrypt(encrypt(x, y), y) = x.
  equation forall x : bitstring, y : key ; encrypt(decrypt(x, y), y) = x.
- The first equation is standard.
- The second equation is more surprising.
  We can test whether decrypt(x, y) succeeds by testing :
  encrypt(decrypt(x, y), y) = x
- Encrypt and decrypt are two inverse bijections (cf. block ciphers).

ProVerif : Equations

Pi calculus + cryptographic primitives

M, N : :=	terms
x, y , z	variable
a, b, c, k, s	name
f (M1, . . . , Mn)	constructor application
P, Q : :=	processes
out(M, N) ; P	output
in(M, x : T) ; P	input
let x = g ($M_1, ..., M_n$) in P else Q	destructor application
if M = N then P else Q	conditional

ProVerif : Equations

Example : The Denning-Sacco protocol

Message 1. A $\rightarrow B$:$\{k_{sk_A}\}_{pk_B}$ k fresh

Message 2. B $\rightarrow A$:$\{s\}_k$

$$\textbf{new } sk_A : \text{sskey}\,;\ \textbf{let } pk_A = \text{spk}(sk_A) \textbf{ in}$$
$$\textbf{new } \text{skB} : \text{skey}\,;\ \textbf{let } pk_B = \text{pk}(sk_B) \textbf{ in}$$
$$\textbf{out}(\text{c}, pk_A)\,;\ \textbf{out}(\text{c}, pk_B\ )\,;$$

(A) ! **in**(c, x_pkB : pkey) ;
new k : key ; **out**(c, pencrypt(sign(k2b(k), skA), x pkB)) ;
in(c, x : bitstring) ; **let** s = decrypt(x, k) **in** 0

(B) | ! **in**(c, y : bitstring) ; **let** y = pdecrypt(y , skB) **in**
let k2b(k) = checksign(y , pk_A) **in out**(c, encrypt(s, k))

ProVerif : Equations

Security properties : equivalences

- **Process equivalences :**

$$P \approx Q$$

  when the adversary cannot distinguish P from Q.
- Process equivalences can formalize various security properties. Example : P implements an ideal specification Q.
- ProVerif can check :
  - Strong secrecy :

$$P(x) \approx P(y)$$

    for all x, y .
  - Equivalences between processes that differ only by terms they contain

ProVerif : Equations

The Horn clause representation

- The main predicate used by the Horn clause representation of protocols is attacker :
  attacker(M) means “the attacker may have M”.
- We can model actions of the adversary and of the protocol participants thanks to this predicate.

ProVerif : Equations

- Constructors $f(M_1, ..., M_n)$
  $attacker(x_1) \wedge ... \wedge attacker(x_n) \rightarrow attacker(f(x_1, ..., x_n))$

Example : Shared-key encryption encrypt(m, k)

attacker(m) $\wedge attacker(k) \rightarrow attacker(encrypt(m, k))$

- Destructors $g(M_1, ..., M_n) \rightarrow M$
  $attacker(M_1) \wedge ... \wedge attacker(M_n) \rightarrow attacker(M)$

Example : Shared-key decryption decrypt(encrypt(m, k), k) $\rightarrow m$

attacker(encrypt(m, k)) $\wedge attacker(k) \rightarrow attacker(m)$

General coding of a protocol

General coding of a protocol

If a principal A has received the messages $M_1, ..., M_n$ and sends the message M,
$attacker(M_1) \wedge ... \wedge attacker(M_n) \rightarrow attacker(M)$.

Example : Shared-key encryption encrypt(m, k)

Upon receipt of a message of the form pencrypt(sign(y , sk_A), pk_B), B replies with encrypt(s, y) :

attacker(pencrypt(sign(y , sk_A), pk_B)) $\rightarrow$ attacker(encrypt(s, y))

The attacker sends $pencrypt(sign(y, sk_A), pk_B)$ to B, and intercepts his reply encrypt(s, y).

General coding of a protocol

Secrecy

If attacker(M) cannot be derived from the clauses, then M is secret.

The term M cannot be built by an attacker. The resolution algorithm will determine whether a given fact can be derived from the clauses.

Remark

Soundness and completeness are swapped.
The resolution prover is complete
(If attacker(M) is derivable, it finds a derivation.)
The protocol verifier is sound
(If it proves secrecy, then secrecy is true.)

Questions ?