

Crypto Engineering - verifying security protocols

Cristian Ene

thanks Jannik Dreier for some slides!

Grenoble Alpes University, Verimag

Master 2: 1st Semester 2023

Administrative information and contents

Administrative information:

- E-mail (me): Cristian.Ene@univ-grenoble-alpes.fr
- E-mail (labs): Thomas.Vigouroux@univ-grenoble-alpes.fr
- Web page: www-verimag.imag.fr/~ene/m2p/
- Tamarin related small exam on Tuesday, January 9, 2024

Schedule

- 1 Security Protocols - introduction and a simple protocol
- 2 Security Protocols - a few attacks and the need for proofs
- 3 A formal language for security protocols
- 4 An introduction to Tamarin
- 5 Formalizing security properties in Tamarin
- 6 Indistinguishability and Security Notions
- 7 Link Between Computational and Symbolic

From crypto primitives to secure distributed applications

- RSA, DSA, ElGamal, AES, ChaCha20, SHA3 . . . provide provably correct cryptographic primitives.
- How can we construct secure distributed applications with these primitives?
 - E-commerce
 - E-banking
 - E-voting
 - Mobile communication
 - Digital contract signing
- Even if cryptography is hard to break, this is not a trivial task

Example: Securing an e-banking application

$A \rightarrow B$: "Send 10000 dollars to account C"

$B \rightarrow A$: "I just did it!"

- How does B know the message originated from A?
- How does B know A just said it?
- Confidentiality, integrity, accountability, non-repudiation, ...?
- We need security protocols

- **What are communication protocols?**

- A set of rules that governs the interaction and transmission of data exchanged between agents or principals. In short, a distributed algorithm whose goal is communication.

- **Examples**

- TCP and UDP govern computer interactions over an IP network
- HTTP governs the exchange of data in the hypertext format.
- SMTP governs the exchange of e-mail.

- **What are security protocols?**

- Communication protocols that operate in an untrusted network or among (some) untrusted agents (principals), and that are used to achieve security goals against a threat model
- Ingredients:
 - **Principal**: a protocol participant, typically human or computer
 - **Security Goal**: the confidentiality or integrity of data, the authentication of a principal
 - **Threat model**: the capabilities of the attacker
- They use cryptographic primitives as basic primitives
- Examples: TLS, IPsec, SSH, WPA, EMV, 5G AKA, Kerberos, Needham-Schroeder,...

Example 1 - Online Banking

- Cryptographic protocol: TLS (HTTPS)
 - **Principals:** Web Browser, Bank Website
 - **Security Goal:** the confidentiality and integrity of data, server authentication
 - **Threat model:** network attacker, phishing website
- Password-based authentication protocol
 - **Principals:** Bank Client, Bank Website
 - **Security Goal:** client authentication
 - **Threat model:** dishonest client

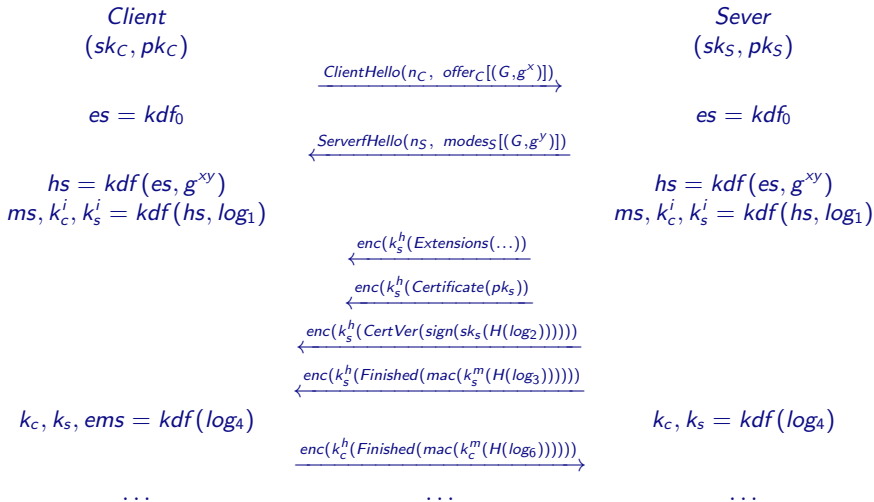
Example 2 - Credit card payment

- Online Transaction Authorization
 - **Principals:** Credit Card, Terminal, Bank Website
 - **Security Goal:** transaction data integrity, card authentication
 - **Threat model:** fake card, tampered terminal
- Cardholder verification (PIN Entry)
 - **Principals:** Client, Credit Card, Terminal
 - **Security Goal:** client authentication
 - **Threat model:** stolen credit card

Modelling security protocols

- Security protocols are small (but **critical**) components embedded within large distributed applications: for example, TLS within a web browser
- the security of the system depends on their correctness
- small recipes, but non trivial to design and understand: a long history of attacks on academic or real word protocols (many attacks, several years later after deployment!)
- why is it so difficult to correctly design security protocols?
 - *cryptografic guarantees* provided by primitives are often misunderstood
 - *rich threat models* are difficult to reason about and to test

TLS 1.3 (a small part of)



Building a secret establishment protocol

- an attempt to design a simple good protocol
- **principals** (they will play **the roles** in a concrete execution of the protocol, i.e. in a **session**) :
 - a set of users, denoted by a, b, c, \dots (not all are necessarily honest!)
 - an **honest** server denoted by s , it never cheats and never gives out secrets
- **security goal**: to establish a **new secret** value k
 - **secrecy**: At the end of the protocol, k is **known** only to a and b , and possibly s , but to no other parties
 - **freshness**: a and b know that k is freshly generated
- k can be used to generate new session keys (further communication protocols between the same principals may be based on these keys)

A secret establishment protocol - notations

- we consider a protocol with 3 **roles**: initiator role A , responder role B (intended to be played by users) and server role S (intended to be played by a server)
- a role A describes how an user (e.g. a, b, \dots) playing this role should behave during a session of the protocol
- we suppose that there is an intruder user i (but nobody knows that i is not honest!)
- A protocol is **informally** specified as a sequence of messages m exchanged between roles (principals) A and B

$A \longrightarrow B : m$ (also called Alice-Bob notation)

A secret establishment protocol - first attempt

- first attempt: a protocol that consists of 3 messages

1. $A \rightarrow S : A, B$	1. A contacts S by sending the identities of the 2 parties who are going to share the secret
2. $S \rightarrow A : K_{AB}$	2. S sends the secret K_{AB} to A
3. $A \rightarrow B : K_{AB}, A$	3. A passes K_{AB} together with its identity to B

- K_{AB} does not contain information about A and B , it is simply a symbolic name for the bit-string representing the secret intended to be shared by the principals playing the roles A and B
- notice the significantly incomplete protocol specification
- we use both M, N and $\langle M, N \rangle$ to denote concatenation

A secret establishment protocol - a naive attempt

- Is the protocol correct? Recall that only the users that play the roles A and B may learn the secret
- In an ideal world, maybe... but a realistic assumption on typical communication networks is:

Security Assumption 1: the intruder is able to steal all messages sent over the network.

⇒ Use cryptography (symmetric/asymmetric encryption, signatures, hashes ...).

A secret establishment protocol - using crypto

- assume that the server playing the role S initially shares a secret key $sk(U, S)$ with each user in the system that can play the role U

$$1. A \rightarrow S : A, B$$

$$2. S \rightarrow A : \{ K \}_{sk(A,S)}, \{ K \}_{sk(B,S)}$$

$$3. A \rightarrow B : \{ K \}_{sk(B,S)}, A$$

- where $\{ m \}_{sk}$ denotes symmetric encryption of message m using the key sk

- Problems? No in case of a **passive adversary**, since:

Perfect Cryptography Assumption : an encrypted message can be decrypted only with the appropriate key.

- Is really B , sharing K (only) with A ?

Security Assumption 2: the intruder is able to intercept and send messages to anybody, under any sender name (that is, he can impersonate any other participant in the protocol.

A secret establishment protocol - a simple attack

- so basically, we assume two things (such an adversary is called a Dolev-Yao adversary):
 - the adversary has complete control over the network
 - **interception** $A \rightarrow O(B) : m$
 - **injection** $O(A) \rightarrow B : m$
 - the adversary cannot break cryptography
- we denote by $m.n$ the n th step of the m th session

$$1.1 \quad a \rightarrow s : a, b$$

$$1.2 \quad s \rightarrow a : \{k\}_{sk(a,s)}, \{k\}_{sk(b,s)}$$

$$1.3 \quad a \rightarrow i(b) : \{k\}_{sk(b,s)}, a$$

$$2.3 \quad i(x) \rightarrow b : \{k\}_{sk(b,s)}, x$$

- The intruder i intercepts the message from a to b and replaces x for a 's identity (x is any agent name).
- Problem: b believes that he is sharing the key with x , whereas he is sharing it with a .

A secret establishment protocol - a simple attack (II)

- 1.1 $a \longrightarrow s : a, b$
- 1.2 $s \longrightarrow a : \{ k \}_{sk(a,s)}, \{ k \}_{sk(b,s)}$
- 1.3 $a \longrightarrow i(b) : \{ k \}_{sk(b,s)}, a$
- 2.3 $i(x) \longrightarrow b : \{ k \}_{sk(b,s)}, x$

- Consequences: it depends on the context in which the protocol is used, for example, b can give away to a information which should have been shared only with x
- Hence, even if i does not get k , the protocol is broken since it does not satisfy the requirement that the users should know who else knows the session key.

A secret establishment protocol - a clever attack

- 1.1 $a \longrightarrow i(s) : a, b$
- 2.1 $i \longrightarrow s : i, a$
- 2.2 $s \longrightarrow i : \{ k \}_{sk(i,s)}, \{ k \}_{sk(a,s)}$
- 1.2 $i(s) \longrightarrow a : \{ k \}_{sk(a,s)}, \{ k \}_{sk(i,s)}$
- 1.3 $a \longrightarrow i(b) : \{ k \}_{sk(i,s)}, a$

- i intercepts the message intended to be for s , and replaces b 's identity with its own identity, so that s generates a key (a bit-string) k , and encrypts it with the keys of the two supposed users i and a
- Since a cannot distinguish between encrypted messages (if she does not have the good key!), she will not detect the alteration

A secret establishment protocol - a clever attack (II)

- 1.1 $a \longrightarrow i(s) : a, b$
- 2.1 $i \longrightarrow s : i, a$
- 2.2 $s \longrightarrow i : \{ k \}_{sk(i,s)}, \{ k \}_{sk(a,s)}$
- 1.2 $i(s) \longrightarrow a : \{ k \}_{sk(a,s)}, \{ k \}_{sk(i,s)}$
- 1.3 $a \longrightarrow i(b) : \{ k \}_{sk(i,s)}, a$

- Hence, a will believe that the session has been successfully completed with b , and that she is sharing the key k with b , whereas i knows this key; so i can further masquerade as b and learn all information that a intends to send to b .
- We suppose that i is a legitimate user registered to s .

Security Assumption 3: the intruder can be a legitimate protocol participant, or is able to corrupt legitimate participants (an insider), an external party (an outsider) or a combination of both.

A secret establishment protocol - a clever attempt

- to prevent this kind of attacks, a good principle is to bound cryptographically the names of the participants A and B to the key k intended to be shared by them.

1. $A \rightarrow S : A, B$

2. $S \rightarrow A : \{ K, B \}_{sk(A,S)}, \{ K, A \}_{sk(B,S)}$

3. $A \rightarrow B : \{ K, A \}_{sk(B,S)}$

- Neither of the previous two attacks does work. Can you justify that the intruder is unable to attack it by eavesdropping, altering and injecting his own messages?
- Still a problem: What about old keys...?

The intruder has memory

- The problem is that the quality of long-term encrypting keys is much better than that of the session keys generated during each execution of the protocol.
- Consequences: communications in different sessions should be separated; in particular, it should be impossible to replay messages from old sessions.

Security Assumption 4: the intruder is able to obtain the value of any “sufficiently old” session key K_s generated in a previous run of the protocol.

The intruder has memory(II)

1.1 $a \rightarrow s :$ a, b
1.2 $s \rightarrow a :$ $\{ k_{ab}, b \}_{sk(a,s)}, \{ k_{ab}, a \}_{sk(b,s)}$
1.3 $a \rightarrow b :$ $\{ k_{ab}, a \}_{sk(b,s)}$

... *extinction of dinosaurs* ...

1000.1 $a \rightarrow i(s) :$ a, b
1000.2 $i(s) \rightarrow a :$ $\{ k_{ab}, b \}_{sk(a,s)}, \{ k_{ab}, a \}_{sk(b,s)}$
1000.3 $a \rightarrow b :$ $\{ k_{ab}, a \}_{sk(b,s)}$

- i masquerades as s , and replays a “very old” key k_{ab}
- Consequences:
 - By **Assumption 1**, i can be expected to know the older encrypted messages $\{ k_{ab}, b \}_{sk(a,s)}$ and $\{ k_{ab}, a \}_{sk(b,s)}$.
 - By **Assumption 4**, i can be expected to know the value of k_{ab} (by cryptanalysis ... or blackmail).
 - Thus i is able to decrypt subsequent messages encrypted with k_{ab} or even to inject messages whose integrity must be protected by k_{ab} .

The intruder has memory(III)

- Even if i does not obtain k_{ab} , the previous attack can be regarded as successful
 - i has succeeded in making a and b to accept an old session key!
 - i can now replay messages protected by k_{ab} in the previous session.
- Imagine a further step in the context of the protocol:

$$4.A \longrightarrow B : \{ request \}_{K_{AB}}$$

The intruder has memory(IV)

1.1 $a \rightarrow s : a, b$
1.2 $s \rightarrow a : \{ k_{ab}, b \}_{sk(a,s)}, \{ a, k_{ab} \}_{sk(b,s)}$
1.3 $a \rightarrow b : \{ k_{ab}, a \}_{sk(b,s)}$
1.4 $a \rightarrow b : \{ \text{transfer } 10000 \text{ euro to } i \}_{k_{ab}}$
...
1000.3 $i(a) \rightarrow b : \{ k_{ab}, a \}_{sk(b,s)}$
1000.4 $i(a) \rightarrow b : \{ \text{transfer } 10000 \text{ euro to } i \}_{k_{ab}}$

- You could say: is b memoryless (he should remember k_{ab})?

Security Assumption 5: “The Principals don’t think”, they only follow the protocol.

A secret establishment protocol - using timestamps

- To prevent replays, we could add a **timestamp** T to each message:

1. $A \rightarrow S : A, B$

2. $S \rightarrow A : \{ K_{AB}, B, T \}_{sk(A,S)}, \{ K_{AB}, A, T \}_{sk(B,S)}$

3. $A \rightarrow B : \{ K_{AB}, A, T \}_{sk(B,S)}$

- B should reject messages that are older than some threshold Δ
- This requires the clocks at A and B to be synchronized
- It still leaves a window of opportunity for replay attacks (within Δ)
- Another solution: use nonces
- **Nonce** : fresh randomly-generated value generated by a principal, intended to be used only once as a challenge that should be returned to that principal to prove that a message is newly generated

A secret establishment protocol - using nonces

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : \{ K_{AB}, B, N_A, \{ K_{AB}, A \}_{sk(B,S)} \}_{sk(A,S)}$
3. $A \rightarrow B : \{ K_{AB}, A \}_{sk(B,S)}$
4. $B \rightarrow A : \{ N_B \}_{K_{AB}}$
5. $A \rightarrow B : \{ N_B - 1 \}_{K_{AB}}$

- A sends her nonce N_A to S in order to get a new key
- B , once he get the new key, challenges A with a fresh nonce N_B
- N_A and N_B do not identify who created them, are just variable names!
- This is a famous protocol proposed by Needham and Schroeder. Is it secure?

A secret establishment protocol - still attacks

1. $a \longrightarrow s :$ a, b, n_a
2. $s \longrightarrow a :$ $\{ k_{ab}, b, n_a, \{ k_{ab}, a \}_{sk(b,s)} \}_{sk(a,s)}$
3. $a \longrightarrow b :$ $\{ k_{ab}, a \}_{sk(b,s)}$
4. $b \longrightarrow a :$ $\{ n_b \}_{k_{ab}}$
5. $a \longrightarrow b :$ $\{ n_b - 1 \}_{k_{ab}}$
- ...
- 1000.4 $i(a) \longrightarrow b :$ $\{ k_{ab}, a \}_{sk(b,s)}$
- 1000.4. $b \longrightarrow i(a) :$ $\{ n'_b \}_{k_{ab}}$
- 1000.5. $i(a) \longrightarrow b :$ $\{ n'_b - 1 \}_{k_{ab}}$

- the same attack as before: i masquerades as a and can convince b to use an old key

A secret establishment protocol - a good protocol?

1. $B \rightarrow A : B, N_B$
2. $A \rightarrow S : A, B, N_A, N_B$
3. $S \rightarrow A : \{ K_{AB}, B, N_A \}_{sk(A,S)}, \{ K_{AB}, A, N_B \}_{sk(B,S)}$
4. $A \rightarrow B : \{ K_{AB}, A, N_B \}_{sk(B,S)}$

- Is “the good” protocol?
- A sends both nonces N_A and N_B to S , who now sends K_{AB} in separate messages for A and B , which can be verified as fresh by the respective recipients
- The protocol avoids all previous attack, as long as the encryption provides confidentiality and integrity...
- It is not a proof... it is just intuition.
- But neither A nor B can deduce at the end of a session that the other actually received the key K_{AB}

A secret establishment protocol - a perfect protocol?

1. $B \rightarrow A : B, N_B$
2. $A \rightarrow S : A, B, N_A, N_B$
3. $S \rightarrow A : \{ K_{AB}, B, N_A \}_{sk(A,S)}, \{ K_{AB}, A, N_B \}_{sk(B,S)}$
4. $A \rightarrow B : \{ K_{AB}, A, N_B \}_{sk(B,S)}, \{ A, N_B \}_{K_{AB}}$
5. $B \rightarrow A : \{ N_B - 1 \}_{K_{AB}}$

- We added a final communication exchange that allows to both A and B to check that the other received the key K_{AB}
- Is this protocol “perfect”?

Other attacks

- All previous attacks were logical or symbolic attacks: they supposed that cryptography is perfect!
- Suppose now that K_{AB} is a password of 8 characters, and that the encryption algorithm is deterministic.
The protocol (recall).

...

1. $b \rightarrow a : b, n_b$

...

4. $a \rightarrow b : \dots, \{ a, n_b \}_{k_{ab}}$

...

- i generates all strings of 8 characters pwd , and since he knows both a, n_b and the encryption algorithm, he can check for which value of pwd , the message $\{ a, n_b \}_{k_{ab}}$ sent during the execution of the protocol and the “computed” message $\{ a, n_b \}_{pwd}$ are equal. This is called a “dictionary attack”.

Needham-Schroeder protocol

- assume that each user U has a pair of public/private keys $(pk(U), sk(U))$ such that everybody knows $pk(U)$, but U is the only one to know $sk(U)$. Moreover, we denote by $\llbracket m \rrbracket_{pk}$ the asymmetric encryption of m using the public key pk .
- NS protocol (1978)

1. $A \longrightarrow B : \llbracket A, N_A \rrbracket_{pk(B)}$
2. $B \longrightarrow A : \llbracket N_A, N_B \rrbracket_{pk(A)}$
3. $A \longrightarrow B : \llbracket N_B \rrbracket_{pk(B)}$

- Is really B , sharing N_A, N_B (only) with A ?
- Gawin Lowe found an attack in 1995 (17 years later!). Can you see it?

Needham-Schroeder protocol - an attack

- man-in-the-middle attack (1995)

- 1.1. $a \longrightarrow i :$ $\llbracket a, n_a \rrbracket_{pk(i)}$
- 2.1. $i(a) \longrightarrow b :$ $\llbracket a, n_a \rrbracket_{pk(b)}$
- 2.2. $b \longrightarrow i(a) :$ $\llbracket n_a, n_b \rrbracket_{pk(a)}$
- 1.2. $i \longrightarrow a :$ $\llbracket n_a, n_b \rrbracket_{pk(a)}$
- 1.3. $a \longrightarrow i :$ $\llbracket n_b \rrbracket_{pk(i)}$
- 2.3. $i(a) \longrightarrow b :$ $\llbracket n_b \rrbracket_{pk(b)}$

- b “thinks” he is talking to a , while he is talking to i , and moreover, n_b (which maybe is intended to be the shared session secret) was disclosed to i

Needham-Schroeder-Lowe protocol

- Fixing the Lowe attack

1. $A \longrightarrow B$: $\llbracket A, N_A \rrbracket_{pk(B)}$
2. $B \longrightarrow A$: $\llbracket N_A, N_B, B \rrbracket_{pk(A)}$
3. $A \longrightarrow B$: $\llbracket N_B \rrbracket_{pk(B)}$

- ... and a type-flaw attack

- 1.1. $i(a) \longrightarrow b$: $\llbracket a, i \rrbracket_{pk(b)}$
- 1.2. $b \longrightarrow i(a)$: $\llbracket i, n_b, b \rrbracket_{pk(a)}$
- 2.1. $i \longrightarrow a$: $\llbracket i, \langle n_b, b \rangle \rrbracket_{pk(a)}$
- 2.2. $a \longrightarrow i$: $\llbracket \langle n_b, b \rangle, n_a, a \rrbracket_{pk(i)}$
- 1.3. $i(a) \longrightarrow b$: $\llbracket n_b \rrbracket_{pk(b)}$

- To fix: change $\llbracket N_A, N_B, B \rrbracket_{pk(A)}$ to $\llbracket B, N_A, N_B \rrbracket_{pk(A)}$ or encode messages correctly (tag each field with its type)

The perfect cryptography and the real world

- And if cryptography is not perfect?
- Let us take a simple El-Gamal encryption scheme:
 - a cyclic group G of order q and generator g
 - $pk(U) = g^{sk(u)}$, and messages are elements of G : $m = g^{m'}$ for some m'
 - $\llbracket m \rrbracket_{pk} = (pk^r \times g^{m'}, g^r)$ for some randomly sampled r (where $g^{m'}$ is the encoding of m).
 - let us assume that $\langle m_1, m_2 \rangle$ is obtained by concatenating the bit-strings, i.e. is mapped to $g^{m'_1 + 2^{|m'_1|} \times m'_2}$
 - then encryption is malleable: if one knows $|m|$ and $n = g^{n'}$, then one can transform $\llbracket m, n \rrbracket_{pk}$ into $\llbracket m, p \rrbracket_{pk}$ for any known $p = g^{p'}$: if $\llbracket m, n \rrbracket_{pk} = (bs_1, bs_2)$, then
$$\llbracket m, p \rrbracket_{pk} = (bs_1 \times (g^{n'})^{-2^{|m'_1|}} \times (g^{p'})^{2^{|m'_1|}}, bs_2).$$

A computational attack

- ... and we get a computational attack

1.1. $a \longrightarrow i$: $\llbracket a, n_a \rrbracket_{pk(i)}$
2.1. $i(a) \longrightarrow b$: $\llbracket a, n_a \rrbracket_{pk(b)}$
2.2. $b \longrightarrow a$: $\llbracket n_a, n_b, b \rrbracket_{pk(a)}$
computation step : i computes $\llbracket n_a, n_b, i \rrbracket_{pk(a)}$
1.2. $i \longrightarrow a$: $\llbracket n_a, n_b, i \rrbracket_{pk(a)}$
1.3. $a \longrightarrow i$: $\llbracket n_b \rrbracket_{pk(i)}$
2.3. $i(a) \longrightarrow b$: $\llbracket n_b \rrbracket_{pk(b)}$

- b is convinced that he is sharing a secret value n_b and that is talking with a
- Symbolic model vs. Computational model

Types of protocol attacks

- Man-in-the-middle attack: $a \leftrightarrow i \leftrightarrow b$
- Replay (or freshness) attack: reuse parts 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”
- Binding attack: using messages for a different purpose than originally intended
- Type flaw attack: substitute a different type of message field

These attack types are not formally defined and there may be overlaps between them.

Security protocols - general goals

- **Confidentiality:** Can the intruder i learn a secret that is meant to be shared only by a and b ?
- **Integrity:** Can the intruder i modify a message from a and get it accepted by b ?
- **Authentication:** Can the intruder i convince b that he is talking to a ?
- **Anonymity:** If a wishes to remain anonymous, can the intruder i disclose its identity?
- **Non-repudiation:** If a sends a message, can she later denies that she sent the message? Or, if b receives a message, can he later denies that he got the message?
- **Fairness:** Can one of a or b obtain an unfair advantage before the transaction is completed?

E-vote protocols - specific goals

'It's not who votes that counts, it's who counts the votes "

(Joseph Stalin)

"Indeed, you won the elections, but I won the count"

(Anastasio Somoza)

- **Eligibility:** Only legitimate voter can vote, and only once.
- **Vote privacy:** the vote of any honest voter is not revealed
- **Receipt-freeness:** a voter cannot prove she vote in a certain way
- **Coercion-resistance:** a voter cannot prove she vote in a certain way, even if the intruder ineracts with the voter during the voting process
- **Individual verifiability:** a voter can verify that her vote was counted
- **Universal verifiability:** the result of the vote was correctly computed
- **Fairness:** no partial results can be obtained before the end of the process vote

An e-vote protocol : Belenios - Participants

The registrar:

- generates and sends privately a signing key to each voter
- sends the corresponding verification keys to the voting server

The voters:

- select their vote, use their voting device to encrypt and sign their vote
- the resulting ballot is sent on an authenticated channel to the voting server (thanks to a login and password mechanism)
- may check at any time that their ballot is present on the bulletin board (BB)
- may revoke, in which case only the last ballot is retained

The voting server:

- maintains the bulletin board (BB), that is, the list of accepted ballots
- upon receiving a ballot from a voter, it checks that the ballot is valid (e.g. the signature is valid) and adds it to the bulleting board

Decryption trustees:

a set of m decryption trustees are selected, out of which $t + 1$ are needed to decrypt the result of the election

Belenios protocol - Phase 1 : Setup

Setup

- for each voter id , the registrar generates a signing key $sk_{id} \xleftarrow{R} \mathbb{Z}_q$ and sends it privately (in practice, by email) to the voter
- the registrar transmits the corresponding list of verification keys $vk_{id1} \dots vk_{idn}$ to the voting server, in some random order, where $vk_{id} = g^{sk_{id}}$
- the voting server publishes the list of verification keys
- the voting server generates a password pwd_{id} for each voter id and sends it privately (in practice, again by email)
- the m trustees run a “Pedersen Distributed Key Generation protocol” in order to generate the public encryption key pk of the election and pairs of private/public keys $(dk_i, pk_i^{dk_i})$ for each trustee i
- pk is published and each trustee i sends pk_i to the server.

Belenios protocol - Phase 2 : Voting

Voting

- the list of accepted ballots BB is public and is initially empty
- to vote, a voter id simply encrypts her vote v yielding a ciphertext $c = enc(v, pk, r)$, produces a proof $\pi = proof_v(v, r, enc(v, pk, r), pk, vk_{id})$ that the vote belongs to the set of valid votes, and signs c , yielding a signature $s = sign(c, sk_{id})$. The ballot $b = ((c, \pi, s), vk_{id})$ is sent to the voting server over an authenticated channel thanks to the password pwd_{id}
- upon receiving a ballot from voter id , the server checks its consistency with respect to 1) the one-to-one association of the verification key vk wrt to id (using a *log* mechanism), 2) the signature, 3) the proof inside the ballot.
Then:
 - possibly adds (id, vk) to *log*
 - if there is a ballot of the form $(b_0, vk) \in BB$, then this ballot is removed
 - finally, the new (b, vk) ballot is added to BB
- at any time, voters may check that their last submitted ballot appears in the public board BB

Belenios protocol - Phase 3 : Tally

Tally

- once the voting phase is over, the list BB of accepted ballots is of the form $((c_1, \pi_1, s_1), vk_1), \dots, ((c_p, \pi_p, s_p), vk_p)$, where the vk_j are all distinct, the proofs π_j and the signatures s_j are valid
- since the encryption used (ElGamal) to produce the ciphertexts c_i is homomorphic, anyone can compute the encrypted result

$$res_e = \prod_{i=1}^P c_i = enc \left(\sum_{i=1}^P v_i, pk, \sum_{i=1}^P r_i \right)$$

- then each trustee i (needed at least $t + 1$ honest trustees) contributes to the decryption by providing $res_e^{dk_i}$ together with a proof pok of correct decryption
- from these contributions, it is possible to compute the decryption of res_e , that is $\sum_{i=1}^P v_i$

Belenios protocol - Properties 1

Ballot privacy

ballot privacy requires that any coalition of at most t trustees together with any coalition of corrupted users cannot infer information from ballots cast by honest voters, even after the election result is announced

Belenios **guarantees vote privacy** provided that both the registrar and the voting server are honest, that the voting device of the voter is honest, and that at most t decryption trustees are corrupted

Belenios **does not satisfy receipt-freeness**, but variants of it (BeleniosRF, BeleniosVS) do

Belenios protocol - Properties 2

Verifiability

A voting scheme is **verifiable** if the result corresponds to the votes of

- all honest voters that have checked that their vote was cast correctly
- at most n valid votes where n is the number of corrupted voters (i.e. the attacker may only use the corrupted voters to cast valid votes);
- a subset of the votes cast by honest voters that did not check their vote was cast correctly

Subnotions:

- Individual Verifiability (cast as intended / recorded as cast)
- Universal Verifiability (tallied as recorded)
- Eligibility Verifiability (avoid ballot stuffing)

Belenios **guarantees verifiability** provided that the registrar or the voting server are honest and the voting device of the voter is honest. The decryption trustees may all be corrupted.

Why Is Security Protocol Analysis Difficult?

- although generally there are less than 10 messages exchanged in a legitimate run (session) of the protocol, there are an infinite number of variations in which the adversary can participate
- **infinite state space:**
 - adversary can create messages of “arbitrary size” ,
 - unbounded number of parallel sessions,
 - unbounded number of nonces (if sessions unbounded)

Real-world protocol standards: ISO/IEC 9798

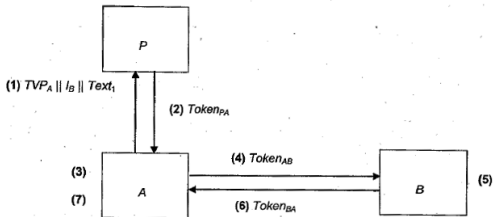


Figure 5 — Mechanism 5 — Four-pass authentication

The form of the token ($Token_{PA}$), sent by P to A, is:

$$Token_{PA} = Text_4 || e_{K_{AP}}(TVP_A || K_{AB} || I_B || Text_3) || e_{K_{BP}}(TN_P || K_{AB} || I_A || Text_2)$$

The form of the token ($Token_{AB}$), sent by A to B, is:

$$Token_{AB} = Text_6 || e_{K_{BP}}(TN_P || K_{AB} || I_A || Text_2) || e_{K_{AB}}(TN_A || I_B || Text_5)$$

The form of the token ($Token_{BA}$), sent by B to A, is:

$$Token_{BA} = Text_8 || e_{K_{AB}}(TN_B || I_A || Text_7)$$

The choice of using either time stamps or sequence numbers in this mechanism depends on the capabilities of the entities involved as well as on the environment.

Real-world protocol specifications: IKE RFC

When using encryption for authentication, Main Mode is defined as follows.

Initiator		Responder
-----		-----
HDR, SA	-->	
	<--	HDR, SA
HDR, KE, [HASH(1),]		
<IDii_b>PubKey_r,		
<Ni_b>PubKey_r	-->	HDR, KE, <IDir_b>PubKey_i,
	<--	<Nr_b>PubKey_i
HDR*, HASH_I	-->	
	<--	HDR*, HASH_R

Aggressive Mode authenticated with encryption is described as follows:

Initiator		Responder
-----		-----
HDR, SA, [HASH(1),] KE,		
<IDii_b>Pubkey_r,		
<Ni_b>Pubkey_r	-->	HDR, SA, KE, <IDir_b>PubKey_i,
	<--	<Nr_b>PubKey_i, HASH_R
HDR, HASH_I	-->	

Real-world protocol specifications: IKE RFC

Harkins & Carrel

Standards Track

[Page 9]

RFC 2409

IKE

November 1998

For signatures: SKEYID = prf(Ni_b | Nr_b, g^{xy})
For public key encryption: SKEYID = prf(hash(Ni_b | Nr_b), CKY-I | CKY-R)
For pre-shared keys: SKEYID = prf(pre-shared-key, Ni_b | Nr_b)

The result of either Main Mode or Aggressive Mode is three groups of authenticated keying material:

SKEYID_d = prf(SKEYID, g^{xy} | CKY-I | CKY-R | 0)
SKEYID_a = prf(SKEYID, SKEYID_d | g^{xy} | CKY-I | CKY-R | 1)
SKEYID_e = prf(SKEYID, SKEYID_a | g^{xy} | CKY-I | CKY-R | 2)

and agreed upon policy to protect further communications. The values of 0, 1, and 2 above are represented by a single octet. The key used for encryption is derived from SKEYID_e in an algorithm-specific manner (see appendix B).

To authenticate either exchange the initiator of the protocol generates HASH_I and the responder generates HASH_R where:

HASH_I = prf(SKEYID, g^{xi} | g^{xr} | CKY-I | CKY-R | SAI_b | IDi_b)
HASH_R = prf(SKEYID, g^{xr} | g^{xi} | CKY-R | CKY-I | SAI_b | IDir_b)

For authentication with digital signatures, HASH_I and HASH_R are signed and verified; for authentication with either public key encryption or pre-shared keys, HASH_I and HASH_R directly authenticate the exchange. The entire ID payload (including ID type, port, and protocol but excluding the generic header) is hashed into both HASH_I and HASH_R.

From attacks to proofs

- Can we be confident that a protocol given in the Alice-Bob notation is correct?
- How can we rigorously prove that the protocol achieves his security goals?
 - What does $A \longrightarrow B$ mean? How A and B parse the received messages and what exactly they do?
 - How we specify formally the security goals?
 - How do we capture the threat model?
- Using our informal notation, we can find and describe attacks.
- To prove mathematically that a given protocol satisfy a security goal, we have to move to a more formal setting

Proofs of protocols : the symbolic model

- The **Dolev-Yao model** was proposed in 1983:
 - messages are terms in a free algebra: $\{ m \}_k, \llbracket m \rrbracket_k, [m]_k, h(m)\dots$
 - no secret value (nonce or key) can be guessed
 - the intruder can only apply functions in the given signature
 - ... but has complete control of the network
- One can add equations between primitives (functions), but anyway, we assume that the only equalities are those given by these equations
- Proofs in this model can be automated

- The **computational model** was proposed in the '80s:
 - the messages are *bitstrings*
 - the cryptographic primitives are *operations on bitstrings*
 - the intruder is any *probabilistic polynomial-time Turing machine*
 - has complete control of the network
- The model is much closer to reality, but proofs in this model are mostly manual

- The **computational model** is still an abstraction, which does not exactly match the reality. It ignores:
 - timing, power consumption, noise
 - tampering attacks
 - errors in implementation: what to answer if a message does not have the expected form?
- In this course we will ignore such kind of attacks

Complexity of proofs in the symbolic model

- If everything is unbounded, then proving secrecy is undecidable
- Even if the number of nonces or the message size is bounded, then proving secrecy is still undecidable
- If everything is bounded, then we get a finite state transition system, and we can apply model checking!
- Bounding only the number of sessions : insecurity is typically **NP-complete**

Solutions to undecidability in the general case

- Uses approximations (generally preserving soundness):
 - abstract interpretation
 - typing
- Allow non-termination

Relevance of the symbolic model

- Advantages:
 - Numerous attacks have been found
 - An attack in the symbolic model can be easily translated in a practical attack in the computational model
 - Proofs are simpler and can be automated
- Drawback: in general, a proof in the symbolic model, does not imply a proof in the computational model

Link between the two models

- Computational soundness theorems:

Proof in the \Rightarrow Proof in the
symbolic model computational model

modulo additional assumptions (initiated by Abadi and Rogaway[2000]).

What are formal models?

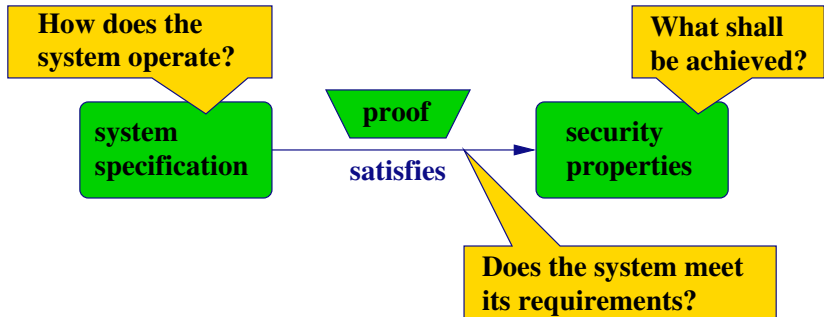
- A **language** is **formal** when it has a well-defined syntax and semantics. Additionally there is often a deductive system for determining the truth of statements.
- **Examples:** propositional logic, first-order logic.
- A **model** (or **construction**) is **formal** when it is specified in a formal 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 and provide a mathematically sound means for reasoning about these models.

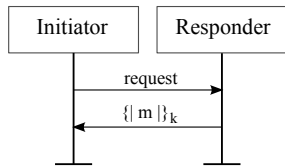
Basis: suitable abstraction of protocols.

Analysis: with formal methods based on mathematics and logic.



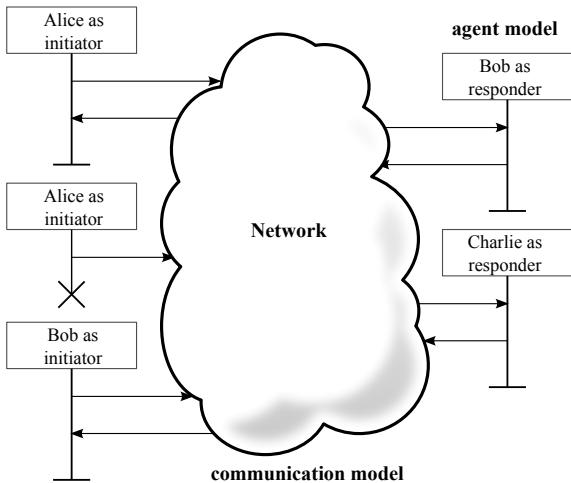
From protocol sequence charts to protocol execution

Protocol specification



cryptographic primitives

Protocol execution



Tamarin - a Term Rewriting-based tool

Modeling protocol and adversary with multiset rewriting

- Specifies a labelled transition system
- Induces a set of traces

Property verification using a guarded fragment of first-order logic

- Security properties are defined as sets of “good” traces
- Tamarin tries to prove that all traces are good, or to find a counterexample trace (attack)

Verification algorithm is sound and complete; termination is not guaranteed

Security protocol as a Labelled transition system

State:

- Adversary's knowledge
- Messages on the network
- Information about freshly generated values
- Protocol's state

The adversary and the protocol interact by updating network messages and freshness information

Tamarin - a Term Rewriting-based tool

Tamarin - a Term Rewriting-based tool

- Protocols and adversaries are specified as multiset rewriting rules
- Security properties are defined as trace properties, checked against the traces of the transition system

Rewriting theory

A labelled rewriting theory is a tuple $R = (\Sigma, E, \mathcal{R}, \mathcal{L})$, with:

- (Σ, E) an equational theory with Σ a signature, E a set of equations
- \mathcal{R} a set of labeled rewriting rules of the general form

$$t - [lset] \rightarrow t'$$

where $lset$ is a set of labels in \mathcal{L} and t, t' are Σ -terms

Definition (Signature)

An unsorted **signature** Σ is a set of function symbols, each having an arity $n \geq 0$. We call function symbols of arity 0 **constants**.

Example (Peano notation for natural numbers)

$\Sigma = \{0, s, +\}$, where 0 is a constant, s has arity 1 and represents the successor function, and $+$ has arity 2 and represents addition. Note that for binary operators we sometimes will use infix notation.

Term Algebra

Definition (Term Algebra)

Let Σ be a signature, \mathcal{X} a set of variables, and $\Sigma \cap \mathcal{X} = \emptyset$. We call the set $\mathcal{T}_\Sigma(\mathcal{X})$ the **term algebra** over Σ . It is the least set such that:

- $\mathcal{X} \subseteq \mathcal{T}_\Sigma(\mathcal{X})$.
- If $t_1, \dots, t_n \in \mathcal{T}_\Sigma(\mathcal{X})$ and $f \in \Sigma$ with arity n , then $f(t_1, \dots, t_n) \in \mathcal{T}_\Sigma(\mathcal{X})$.

The set of **ground terms** \mathcal{T}_Σ consists of terms built without variables, i.e., $\mathcal{T}_\Sigma := \mathcal{T}_\Sigma(\emptyset)$.

Remark: constants are included in \mathcal{T}_Σ and $\mathcal{T}_\Sigma(\mathcal{X})$.

Example (Peano notation for natural numbers (ctd.))

$s(0) \in \mathcal{T}_\Sigma$ $s(s(0)) + s(X) \in \mathcal{T}_\Sigma(\mathcal{X})$ $+s(0)+ \notin \mathcal{T}_\Sigma(\mathcal{X})$

Cryptographic Messages

We generally denote variables with upper case names X, Y, \dots , and function symbols (including constants) with lower case names a, b, f, g, \dots

Definition (Messages)

A message is a term in $\mathcal{T}_\Sigma(\mathcal{X})$, where $\Sigma = \mathcal{A} \cup \mathcal{F} \cup \text{Func} \cup \{\text{pair}, \text{pk}, \text{aenc}, \text{senc}\}$. We call

\mathcal{X}	the set of variables A, B, X, Y, Z, \dots ,
\mathcal{A}	the set of agents a, b, c, \dots ,
\mathcal{F}	the set of fresh values na, nb, k (nonces, keys, ...),
Func	the set of user-defined functions (hash, exp, ...),
$\text{pair}(t_1, t_2)$	pairing, also denoted by $\langle t_1, t_2 \rangle$,
$\text{pk}(t)$	public key,
$\text{aenc}(t_1, t_2)$	asymmetric encryption, also denoted by $\{t_2\}_{t_1}$,
$\text{senc}(t_1, t_2)$	symmetric encryption, also denoted by $\{\{t_1\}\}_{t_2}$.

Definition (Free Algebra)

In the **free algebra** every term is interpreted by itself (syntactically).

- $t_1 =_{\text{free}} t_2$ iff $t_1 =_{\text{syntactic}} t_2$
- $a \neq_{\text{free}} b$ for any different constants a and b

Example (But symmetric cryptography enjoys the following equation E)

$\Sigma = \mathcal{A} \cup \mathcal{F} \cup \{\text{senc}, \text{sdec}\}$, with senc and sdec of arity 2.

(E : $\text{sdec}(\text{senc}(M, K), K) = M$)

- For above example: $\text{sdec}(\text{senc}(X, Y), Y) \neq_{\text{free}} X$.

This is too coarse, as we obviously want to identify those two terms, which means we will need to reason modulo equations.

Equational Theory

Definition (Equation)

An **equation** is a pair of terms, written: $t = t'$, and a set of equations E is called an **equational theory** (Σ, E) . An equation can be oriented as $t \rightarrow t' \in \vec{E}$ or as $t \leftarrow t' \in \overleftarrow{E}$.

Equations are usually oriented from left to right for use in simplification.

Example (Peano natural numbers (ctd.))

The equations E defining the Peano natural numbers are:

$$X + 0 = X$$

$$X + s(Y) = s(X + Y)$$

Using \vec{E} on $s(s(0)) + s(0)$ yields the equational derivation:
 $s(s(0)) + s(0) = s(s(s(0)) + 0) = s(s(s(0)))$.

Algebraic Properties

Example (Equations E_0)

$$\begin{array}{ll} \{\{M\}_K\}_{(K)^{-1}} = M & ((K)^{-1})^{-1} = K \\ \{\{\{M\}_K\}_K = M & \exp(\exp(B, X), Y) = \exp(\exp(B, Y), X) \end{array}$$

Definition (Congruence, Equivalence, Quotient)

A set of equations E induces a congruence relation $=_E$ on terms and thus the equivalence class $[t]_E$ of a term modulo E . The quotient algebra $\mathcal{T}_\Sigma(\mathcal{X})/_E$ interprets each term by its equivalence class.

- Two terms are semantically equal iff $t_1 =_E t_2$.
- For the above example equations (E_0):
 - $a \neq_{E_0} b$ for any distinct constants a and b
 - If $m_1 \neq_{E_0} m_2$ then $h(m_1) \neq_{E_0} h(m_2)$ (since h does not have any equation)
 - $\{\{M\}_{(K)^{-1}}\}_K =_{E_0} M$ (since $K =_{E_0} ((K)^{-1})^{-1}$)
 - $\{\{\{M\}_{\exp(\exp(g, Y), X)}\}_{\exp(\exp(g, X), Y)} =_{E_0} M$

Substitution

Definition (Substitution)

A **substitution** σ is a function $\sigma : \mathcal{X} \rightarrow \mathcal{T}_{\Sigma}(\mathcal{X})$ where $\sigma(x) \neq x$ for finitely many $x \in \mathcal{X}$.

We write substitutions in postfix notation and homomorphically extend them to a mapping $\sigma : \mathcal{T}_{\Sigma}(\mathcal{X}) \rightarrow \mathcal{T}_{\Sigma}(\mathcal{X})$ on terms:

$$f(t_1, \dots, t_n)\sigma = f(t_1\sigma, \dots, t_n\sigma)$$

Example (Applying a substitution)

Given substitution $\sigma = \{X \mapsto \text{senc}(M, K)\}$ and the term $t = \text{sdec}(X, K)$ we can apply the substitution and get $t\sigma = \text{sdec}(\text{senc}(M, K), K)$.

Substitution (ctd.)

Definition (Substitution composition)

We denote with $\sigma\tau$ the composition of substitutions σ and τ , i.e., $\tau \circ \sigma$.

Example (Substitution composition)

For substitutions $\sigma = [x \mapsto f(y), y \mapsto z]$ and $\tau = [y \mapsto a, z \mapsto g(b)]$ we have $\sigma\tau = [x \mapsto f(a), y \mapsto g(b), z \mapsto g(b)]$.

Position

Definition (Position)

A **position** p is a sequence of positive integers. The subterm $t|_p$ of a term t at position p is obtained as follows.

- If $p = []$ is the empty sequence, then $t|_p = t$.
- If $p = [i] \cdot p'$ for a positive integer i and a sequence p' , and $t = f(t_1, \dots, t_n)$ for $f \in \Sigma$ and $1 \leq i \leq n$ then $t|_p = t_i|_{p'}$, else $t|_p$ does not exist.

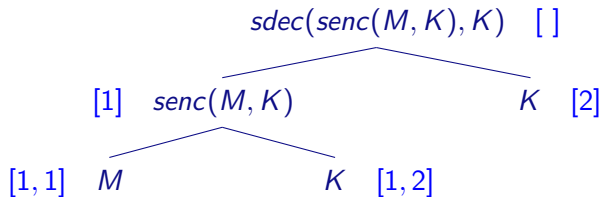
Example (Position in a term)

For the term $t = sdec(senc(M, K), K)$ we have five subterms:

$$\begin{array}{lll} t|_{[]} = t & t|_{[1]} = senc(M, K) & t|_{[1,1]} = M \\ t|_{[1,2]} = K & t|_{[2]} = K & \end{array}$$

Graphical representation of positions in a term

Tree of subterms of $sdec(senc(M, K))$ and their positions.



Definition (Matching)

A term t **matches** another term l if there is a subterm of t , i.e., $t|_p$, such that there is a substitution σ so that $t|_p = l\sigma$. We call σ the **matching substitution**.

Example (Matching)

The term $t = A(B(a, b), c)$ matches with $l = B(x, y)$ since there is position $p = [1]$ and substitution $\sigma = [x \mapsto a, y \mapsto b]$ such that $t|_p = l\sigma$.

Matching and Application

Definition (Application of a rule)

A rule (oriented equation) $l \rightarrow r$ is **applicable** on a term t , if t **matches** l .

Given a matching substitution σ , the result of such a rule application to t is the term obtained from t by replacing the subterm $l\sigma$ at position p , by the right-hand side of the rule instantiated with the matching substitution $r\sigma$, term denoted by $t[r\sigma]_p$.

Example (Application of a rule)

The rule $B(x, y) \rightarrow C(y, x)$ can be applied to term $t = A(B(a, b), c)$ using the matching substitution $\sigma = [x \mapsto a, y \mapsto b]$ and it produces the term $A(C(b, a), c)$, obtained by replacing $B(a, b) = (B(x, y))\sigma$ with $C(b, a) = (C(y, x))\sigma$ in the term t .

Definition (Unification)

We say that $t \stackrel{?}{=} t'$ is **unifiable** in (Σ, E) for $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$, if there is a substitution σ such that $t\sigma =_E t'\sigma$ and we call σ a **unifier**.

For syntactic unification ($E = \emptyset$) there is a **most general unifier** for two unifiable terms, and it is decidable whether they are unifiable.

Unification modulo theories ($E \neq \emptyset$) is much more complicated: undecidable in general, or potentially (infinitely) many unifiers.

This is no good for automated analysis: we need to restrict ourselves.

Unification modulo theories

- When considering other algebras, unifiability is in general undecidable, e.g., associativity and distributivity.
- Even when decidable, there is in general no unique most general unifier, e.g., $\{\exp(X, Y), \exp(X', c)\}$...
- Some unification problems are decidable but **infinitary**: in general, there is an infinite set of most general unifiers, e.g., associativity.

Definition (Equality Relation)

Given (Σ, E) , an E -equality step for $u, v \in \mathcal{T}_\Sigma(\mathcal{X})$ is defined as $u \rightarrow_{(\vec{E} \cup \bar{E})} v$ and denoted as $u \leftrightarrow_E v$.

The transitive-reflexive closure of \leftrightarrow_E is the E -equality relation $=_E$.

Definition (Equality Proof)

A sequence of steps $t_0 \leftrightarrow_E t_1 \leftrightarrow_E \dots \leftrightarrow_E t_n$, witnessing n -step equality of $t_0 \leftrightarrow_E^+ t_n$ is an equality proof.

Equality for Peano natural numbers

Example (Peano natural numbers - remind)

The equations E defining the Peano natural numbers are:

$$X + 0 = X$$

$$X + s(Y) = s(X + Y)$$

Example (Equality reasoning for Peano natural numbers)

Consider how to prove $s(s(0)) + s(0) = s(0) + s(s(0))$:

$$\underline{s(s(0)) + s(0)} = \underline{s(s(s(0)) + 0)} = \underline{s(s(s(0)))}$$

$$= \underline{s(s(s(0) + 0))} = \underline{s(s(0) + s(0))} = s(0) + s(s(0))$$

Complicated! Using termination and confluence, we could have instead computed the normal form of both sides, and simply compared them! (See next slides.)

Termination of \vec{E}

Definition (Termination)

(Σ, \vec{E}) has infinite computations, if there is a function $a : \mathbb{N} \rightarrow \mathcal{T}_\Sigma(\mathcal{X})$ such that

$$a(0) \rightarrow_{\vec{E}} a(1) \rightarrow_{\vec{E}} a(2) \rightarrow_{\vec{E}} \dots \rightarrow_{\vec{E}} a(n) \rightarrow_{\vec{E}} a(n+1) \dots$$

We say it is terminating, when it does not have infinite computations.

Example (Termination)

For $E = \{a = b\}$, \vec{E} is terminating.

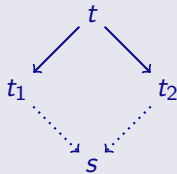
For $E = \{a = b, b = a\}$, \vec{E} is not terminating.

Confluence of \vec{E}

Definition (Confluence)

Confluence is the property that guarantees the order of applying equalities is immaterial, formally:

$$\forall t, t_1, t_2. t \rightarrow^* t_1 \wedge t \rightarrow^* t_2 \Rightarrow \exists s. t_1 \rightarrow^* s \wedge t_2 \rightarrow^* s$$



Example (Confluence)

For $E = \{a = b, a = c\}$, we have that \vec{E} is not confluent, as b and c are reachable from a , but not joinable.

For $E = \{a = b, a = c, b = c\}$, then \vec{E} is confluent.

Subterm-Convergent Rewriting

Termination:

it is always the case that after finitely many rule applications no more rules can be applied, i.e., each term has a normal form (or, is reduced).

Confluence:

if a given term t can be rewritten (in an arbitrary number of steps) to t_1 and t_2 , then there is t' such that both t_1 and t_2 can be rewritten to t' .

Subterm-convergent theory

- A **convergent theory** is a theory that is confluent and terminating
- A **subterm-convergent theory** is convergent and, for each rule $l \rightarrow r$, r is a proper subterm of l , or r is ground and in normal form

Tamarin - syntax overview

Example (Tamarin Syntax)

```
theory Equations
begin
functions: h/1, senc/2, sdec/2
equations: sdec(senc(m,k),k) = m
builtins: diffie-hellman, bilinear-pairing, multiset

/* Other builtins: hashing, symmetric-encryption,
   asymmetric-encryption, signing, revealing-signing */

/* rules, restrictions, lemmas ... */
end
```

Tamarin supports

- any user defined equational theory that is subterm-convergent convergent
- built-in theories: DH exponentiation, bilinear pairing, multisets, XOR

Multiset rewriting in Tamarin

A protocol defines a *setup* and the behavior of a set of *roles*. Every role has a name R and consists of a set of rules, specifying the sending and receiving of messages, branching and looping conditions, and the generation of fresh constants.

In Tamarin, protocols are modeled using *rewrite rules* operating on *multisets of facts*:

$$l \xrightarrow{a} r$$

where l , a , and r are multisets of facts, l is called the *left hand side*, r the *right hand side*, and a the *actions* of the rule.

The left and right hand side of the rule specify which facts are *consumed* or *produced* when executing the rule, the actions are recorded as *event labels* on the trace and are used to specify properties.

Multiset rewriting in Tamarin: example

Example

- rule 1: $\xrightarrow{\text{Init}()} A('5'), C('3')$ ('x' is a constant)
- rule 2: $A(x) \xrightarrow{\text{Step}(x)} B(x)$

or in Tamarin syntax:

```
rule 1: [ ] --[ Init() ]-> [ A('5'), C('3') ]
```

```
rule 2: [ A(x) ] --[ Step(x) ]-> [ B(x) ]
```

```
// A rule without action:
```

```
rule 3: [ C(x) ] --> [ D(x) ]
```

Terms

Fresh terms

Agents generate **fresh terms** using **fresh facts**, denoted by Fr . These fresh terms represent randomness being used, are assumed unguessable and unique, i.e., can represent nonces.

There is a countable supply of fresh terms, each as argument of a fresh fact, usable in rules.

In Tamarin, fresh variables are prefixed with a \sim , e.g., $\sim r$.

Public terms

We define **public terms** to be terms known to all participants of a protocol. These include all agent names and all constants.

In Tamarin, public variables are prefixed with a $\$$, e.g., $\$X$.

Facts

Facts

A fact is represented by $F(t_1, \dots, t_k)$, where F is a fact symbol and t_1, \dots, t_k are terms.

- facts can be **linear** or **persistent**: linear facts can only be consumed **once**, persistent facts **infinitely often**
- by default, facts are linear, persistent facts are marked with a **!**
- there are **special facts**:
 - *Fr* - for fresh data
 - *In* and *Out* - for protocol inputs and outputs
 - *K* - for attacker knowledge

Example (Key revealing and encryption)

```
rule key-reveal:
```

```
  [ !Ltk(~k) ] --[ Reveal(~k) ]-> [ Out(~k) ]
```

```
rule sym-enc: [ C(x), In(y) ] --> [ Out( senc(y,x) ) ]
```

Definition (Fresh rule)

All fresh names are created with the following **built-in rule**, which has no precondition:

$$[] \rightarrow [\text{Fr}(\sim r)]$$

- This is the only rule that produces facts of the form $\text{Fr}(\dots)$,
- The semantics ensures that each created nonce r is fresh, and thus unique.



Dolev-Yao adversary



- Intruder controls the network and can:
 - intercept messages
 - modify messages
 - block messages
 - generate new messages
 - insert new messages
- Perfect cryptography:
 - Abstraction with terms algebra
 - Decryption only if inverse key is known
- Protocol has
 - Arbitrary number of principals
 - Arbitrary number of parallel sessions
 - Messages with arbitrary size



Dolev-Yao Deduction

Definition (Adversary Knowledge)

We represent the adversary knowing a term t by a fact $K(t)$. The adversary's knowledge \mathcal{K} contains all facts of the form $K(t)$, all of which are persistent.

Definition (Adversary Knowledge Derivation)

The adversary can use the following inference rules on the state:

$$\frac{\text{Fr}(x)}{K(x)} \quad \left| \quad \frac{\text{Out}(x)}{K(x)} \quad \left| \quad \frac{K(x)}{\text{In}(x)} \quad \left| \quad \frac{K(t_1) \dots K(t_k)}{K(f(t_1, \dots, t_k))} \right. \right. \right. \forall f \in \Sigma(\text{k-ary})$$

Terms are used modulo the equational theory

Definition (Adversary Knowledge Derivation as **built-in** rewrite rules)

$$[\text{Fr}(x)] \rightarrow [K(x)]$$

$$[\text{Out}(x)] \rightarrow [K(x)]$$

$$[K(x)] \xrightarrow{K(x)} [\text{In}(x)]$$

$$[K(t_1), \dots, K(t_k)] \rightarrow [K(f(t_1, \dots, t_k))] \quad \forall f \in \Sigma(\text{k-ary})$$

Example

Given $K(x), K(\{b, n\}_k), K(k), K(m) \in \mathcal{K}$. Use the equational theory E (containing decryption and pairing) to derive $K(\{m\}_{prf(n,x)})$

$$\begin{array}{c}
 \frac{\frac{\frac{K(\{b, n\}_k) \quad K(k)}{K(dec(k, \{b, n\}_k))}{} E}{K(b, n)} \\
 \frac{K(snd(b, n))}{K(n)} E \\
 \frac{K(m) \quad \frac{K(n) \quad K(x)}{K(prf(n, x))}}{K(\{m\}_{prf(n, x)})}
 \end{array}$$

Well-formedness

A protocol is a finite set of protocol rules. Protocol rules have to be well-formed.

Definition (Well-formedness)

For a protocol rule $l \xrightarrow{a} r$ to be well-formed, the following conditions must be satisfied:

- 1 There is no occurrence of $K(\dots)$ anywhere in the rule.
- 2 Only In, Fr, and other protocol facts occur in the premise l .
- 3 Only Out and protocol facts occur in the conclusion r .
- 4 Every variable in r or a that is not public must occur in l .
- 5 All occurrences of the same fact have the same arity, and the same persistence.

We will define a **trace semantics** for protocols in terms of **labeled transition systems**.

Labeled Multiset Rewriting

Definition (Multiset)

A **multiset** is a set of elements, each imbued with a multiplicity. Instead of stating an explicit multiplicity, we may also simply write elements multiple times.

We use $\setminus^\#$ for the multiset difference, and $\cup^\#$ for the union.

Definition (Labeled multiset rewriting)

A **labeled multiset rewriting rule** is a triple, l, a, r , each of which is a multiset of facts, and written as:

$$l \xrightarrow{a} r$$

Definition (State)

A *state* is a multiset of facts.

Example (State)

$\text{St_R_1}(A, id, k_1, k_2), \text{Out}(k_1), \text{Out}(k_2), \text{Out}(k_2)$

Ground substitution

Definition (Ground substitution)

A substitution is called **ground** when each variable is mapped to a ground term.

Definition (Ground instances)

We call the **ground instances** of a term t all those terms $t\sigma$ that are ground for some (ground) substitution σ .

A fact F is ground if all its terms are ground. The set of all multisets of ground facts is $\mathcal{G}^\#$.

For a rule, its ground instances are those where all facts are ground, and we use

$$ginsts(R)$$

for the set of all ground instances of the set of rules R .

Definition (Steps)

For a multiset rewrite system R we define the labeled transition relation step, $steps(R) \subseteq \mathcal{G}^\# \times ginsts(R) \times \mathcal{G}^\#$, as follows:

$$\frac{I \xrightarrow{a} r \in ginsts(R), \quad I \subseteq^\# S, \quad S' = (S \setminus^\# I) \cup^\# r}{(S, I \xrightarrow{a} r, S') \in steps(R)}$$

A **step** corresponds to a ground instance $I \xrightarrow{a} r$ and is executed as follows:

- 1 let S be the current state
- 2 assume that the needed premises belong to the current state $I \subseteq^\# S$
- 3 then, compute the new state $S' = (S \setminus^\# I) \cup^\# r$
- 4 append a to the end of the current trace

Definition (Execution)

An execution of R is an alternating sequence

$$S_0, (l_1 \xrightarrow{a_1} r_1), S_1, \dots, S_{k-1} (l_k \xrightarrow{a_k} r_k), S_k$$

of states and multiset rewrite rule instances with

- ① $S_0 = \emptyset$
- ② $\forall i : S_{i-1}, (l_i \xrightarrow{a_i} r_i), S_i \in \text{steps}(R)$
- ③ Fresh names are unique, i.e. no ground instance of Fresh rule is used twice: if $i \neq j$ and $(l_i \xrightarrow{a_i} r_i) = ([\] \rightarrow [\text{Fr}(n_i)])$ and $(l_j \xrightarrow{a_j} r_j) = ([\] \rightarrow [\text{Fr}(n_j)])$ then necessarily $n_i \neq n_j$.

Definition (Trace)

The **trace** of an execution

$$S_0, (l_1 \xrightarrow{a_1} r_1), S_1, \dots, S_{k-1}(l_k \xrightarrow{a_k} r_k), S_k$$

is defined by the sequence of the multisets of its action labels, i.e.:

$$a_1; a_2; \dots; a_k$$

Semantics of a rule

Two parts:

- State transition
- Trace event

Example (Transition example)

$$[\text{St_I_2}(A, 17, k), \text{In}(m)] \xrightarrow{\text{Recv}(A, m)} [\text{St_I_3}(A, 17, k, m)]$$

Effects:

- Agent state changes: $\text{St_I_2}(A, 17, k)$ is removed and $\text{St_I_3}(A, 17, k, m)$ is added to the set of facts representing the **global state** of the system
- $\text{In}(m)$ fact is consumed
- $\text{Recv}(A, m)$ action is added to the **trace**

Multiset rewriting in Tamarin: example

Example (Rules in Tamarin syntax:)

```
rule 1: [ ] --[ Init() ]-> [ A('3'), A('5'), C('3') ]
rule 2: [ A(x) ] --[ StepA(x) ]-> [ !B(x) ]
rule 3: [ B(x), B(x) ] --> [ D(x) ] /* silent rule */
rule 4: [ A(x), C(x) ] --[ StepA2(x), StepC2(x) ]-> [ D(x) ]
```

Example (A possible execution:)

```
[ ] --[ Init() ]->
[ A('3'), A('5'), C('3') ] --[ StepA('3') ]->
[ A('5'), C('3'), !B('3') ] --> /* silent step */
[ A('5'), C('3'), !B('3'), D('3') ] --[ Init() ]->
[ A('3'), A('5'), C('3'), A('5'), C('3'), !B('3'), D('3') ]
  --[ StepA2('3'), StepC2('3') ]->
[ A('5'), A('5'), C('3'), !B('3'), D('3'), D('3') ]
```

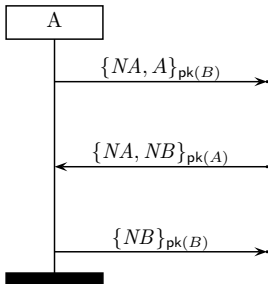
Example (and its associated trace:)

```
[ Init() ] ; [ StepA('3') ] ; [ Init() ] ; [ StepA2('3'), StepC2('3') ]
```

Example protocol: NSPK

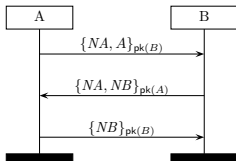
Graphical:

msc NSPK A



PKIs and longterm data

msec NSPK

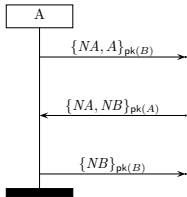


Generate longterm keys and public keys.

$$[Fr(\sim skR)] \rightarrow [!Ltk(R, \sim skR), Out(pk(\sim skR))]$$

Initialization of protocol roles

msec NSPK A

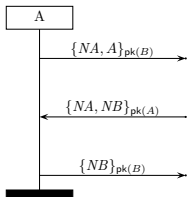


For each role R there must be an initialization rule which is instantiated with a name A and a thread identifier id :

$$[\text{Fr}(\sim id), !\text{Ltk}(A, skA), !\text{Ltk}(B, skB)] \xrightarrow{\text{Create_I}(A, \sim id)} [\text{St_A_1}(A, \sim id, skA, pk(skB))]$$

Protocol rules

msc NSPK A


$$[\text{St_A_1}(A, tid, skA, pk(skB)), \text{Fr}(\sim NA)] \rightarrow$$
$$[\text{St_A_2}(A, tid, skA, pk(skB), \sim NA), \text{Out}(\{\sim NA, A\}_{pk(skB)})]$$
$$[\text{St_A_2}(A, tid, skA, pk(skB), NA), \text{In}(\{NA, NB\}_{pk(skA)})] \rightarrow$$
$$[\text{St_A_3}(A, tid, skA, pk(skB), NA, NB)]$$
$$[\text{St_A_3}(A, tid, skA, pk(skB), NA, NB)] \rightarrow$$
$$[\text{St_A_4}(A, tid, skA, pk(skB), NA, NB), \text{Out}(\{NB\}_{pk(skB)})]$$

Be careful: pattern matching!

Protocol Goals

Goals

What the protocol should achieve, e.g.,

- **Authenticate** messages, binding them to their originator
- Ensure **timeliness** of messages (recent, fresh, ...)
- Guarantee **secrecy** of certain items (e.g., generated keys)

Most common goals

- secrecy
- authentication (many different forms)

Other goals

- anonymity, non-repudiation (of receipt, submission, delivery), fairness, availability, sender invariance, ...

Protocol Properties and Correctness

What does it mean?

Properties

- Semantics of a security protocol P is a set of traces $\|P\| = \text{traces}(P)$. (Traces may be finite or infinite, state- or event-based.)
- Security goal / property ϕ also denotes a set of traces $\|\phi\|$.

Correctness

has an exact meaning

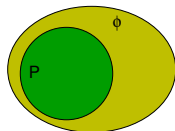
- Protocol P **satisfies** property ϕ , written $P \models \phi$, iff

$$\|P\| \subseteq \|\phi\|$$

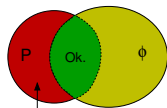
- **Attack traces** are those in

$$\|P\| - \|\phi\|$$

- Every correctness statement is either true or false.
- Later: **how do we find attacks or prove correctness?**



Ok, no attacks.



Attacks.

Specifying properties in Tamarin - Overview

- Many-sorted first-order logic is used to specify security properties
- The logic supports quantification over both messages and time points
- Formulas are interpreted over traces (the temporal domain is \mathbb{Q})
- Trace atoms:
 - F (false)
 - Term equality $t_1 = t_2$
 - Time point ordering and equality: $\#t_i = \#t_j$ or $\#t_i < \#t_j$
 - Actions at time points: $f@i$, for a fact f and a time point $\#i$
- Trace formula: a first-order formula over trace atoms

Specifying properties in Tamarin - Language

Tamarin's *property specification language* is a guarded fragment of a many-sorted first-order logic with a sort for timepoints (prefixed with #). This logic supports quantification over both messages and timepoints:

- **All** for universal quantification (temporal variables are prefixed with #)
- **Ex** for existential quantification (temporal variables are prefixed with #)
- **==>** for implication, **not** for negation
- **|** for disjunction (“or”), **&** for conjunction (“and”)
- **f@i** for action constraints (the sort prefix for the temporal variable 'i' is optional)
- **i < j** for temporal ordering (the sort prefix for the temporal variables 'i' and 'j' is optional)
- **#i = #j** for an equality between temporal variables 'i' and 'j'
- **x = y** for an equality between message variables 'x' and 'y'

Semantics of Trace Formulas

For a trace $T = a_1; a_2; \dots; a_n$ and sort-respecting valuation Θ :

Semantics

$(T, \Theta) \models f@i$	iff	$1 \leq \Theta(i) \leq n$ and $\Theta(f) \in T[\Theta(i)]$
$(T, \Theta) \models i < j$	iff	$\Theta(i) < \Theta(j)$
$(T, \Theta) \models \#i = \#j$	iff	$\Theta(i) = \Theta(j)$
$(T, \Theta) \models t_i = t_j$	iff	$\Theta(t_1) = \Theta(t_2)$
$(T, \Theta) \models \neg\phi$	iff	it is not the case that $(T, \Theta) \models \phi$
$(T, \Theta) \models \phi \wedge \psi$	iff	$(T, \Theta) \models \phi$ and $(T, \Theta) \models \psi$
$(T, \Theta) \models \exists x : s.\phi$	iff	exists $v \in \mathcal{D}_s$ such that $(T, \Theta[x \mapsto v]) \models \phi$

Example

Example

The property that the argument n is distinct in all applications of a fictitious rule generating an action fact $\text{Act1}(n)$:

```
lemma distinct_nonces:
```

```
"All n #i #j. Act1(n)@i & Act1(n)@j ==> #i=#j"
```

or equivalently

```
lemma distinct_nonces:
```

```
all-traces
```

```
"All n #i #j. Act1(n)@i & Act1(n)@j ==> #i=#j"
```

These lemmas require that the property holds for **all** traces, we can also express that there **exists** a trace for which the property does not hold:

```
lemma not_distinct_nonces:
```

```
exists-trace
```

```
"not All n #i #j. Act1(n)@i & Act1(n)@j ==> #i=#j"
```

Guardedness

- All action fact symbols may be used in formulas.
- All variables must be guarded.

Guardedness

For universally quantified variables:

- all variables have to occur in an action constraint right after the quantifier and
- the outermost logical operator inside the quantifier has to be an implication

For existentially quantified variables:

- all variables have to occur in an action constraint right after the quantifier and
- the outermost logical operator inside the quantifier has to be a conjunction

Tamarin overview

- Use multiset rewriting to represent protocol
- Adversary message deduction rules given as multiset rewriting rules
- Properties specified in first-order logic
 - Allows quantification over messages and timepoints
- Verification algorithm is proven sound and complete

- Backwards reachability analysis – searching for insecure states
 - Negate security property, search for *solutions*
- Constraint solving
 - Uses dependency graphs
 - Normal dependency graphs for state-space reduction – efficiency despite undecidability

Issues with Multiset Rewriting

- Incrementally constructing attacks is difficult with (action-)traces :
 - No history of past states
 - No causal dependencies between steps
- Symbolic reasoning modulo an equational theory is difficult, because of cancellation equations
 - If the adversary knows $t = n * x$ for some nonce n , we cannot conclude that n has been used to construct t , as x could be n^{-1}
- Message deduction rules may be applied redundantly
 - Encrypt some plaintext m then decrypt m and send it, instead of just sending m

Dependency Graphs

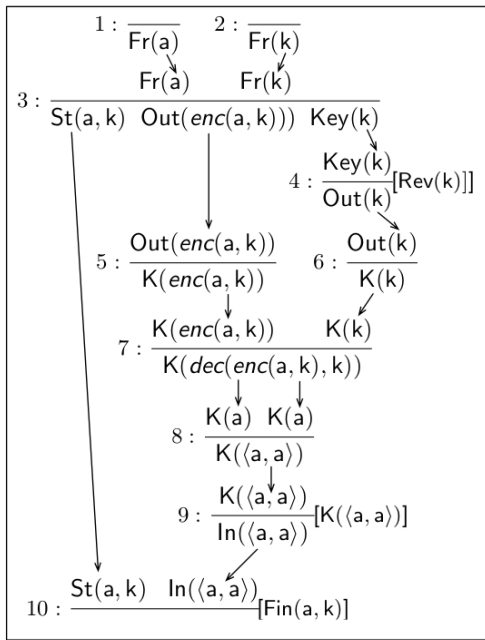
Dependency graph consists of

- nodes labelled with rule instances
- edges represent the dependencies between nodes

Dependency graphs are used to represent protocol executions together with their causal dependencies

- Node indexes denote rule application order
- The trace of the graph is the trace of the execution
- An edge $(i, F) \rightarrow (j, G)$ denotes that $F =_E G$ and F is generated by i and G is consumed by j

Dependency graph example



Message Deduction

$$MD_{\Sigma} = \left\{ \begin{array}{l} \frac{\text{Out}(x)}{K(x)} \quad \frac{K(x)}{\text{In}(x)} [K(x)] \quad \frac{\text{Fr}(x : fr)}{K(x : fr)} \quad \overline{K(x : pub)} \\ \frac{K(x_1) \quad \dots \quad K(x_k)}{K(f(x_1, \dots, x_k))} \text{ for all } f \in \Sigma \end{array} \right\}$$

Nice in theory, but not in practice...

Problems with Message Deduction

$$\begin{array}{l} 1: \frac{K(\langle a, b \rangle)}{K(a)} \\ \downarrow \\ 2: \frac{K(a) \quad K(c)}{K(\langle a, c \rangle)} \\ \downarrow \\ 3: \frac{K(\langle a, c \rangle)}{K(a)} \\ \downarrow \\ 4: \frac{K(a) \quad K(d)}{K(\langle a, d \rangle)} \end{array}$$

$$\begin{array}{l} 1: \frac{K^\uparrow(a) \quad K^\uparrow(c)}{K^\uparrow(\langle a, c \rangle)} \\ \downarrow \\ 2: \frac{K^\downarrow(\langle a, c \rangle)}{K^\downarrow(a)} \end{array}$$

$$\begin{array}{l} 1: \frac{K^\downarrow(\langle a, b \rangle)}{K^\downarrow(a)} \\ \downarrow \\ 2: \frac{K^\downarrow(a)}{K^\uparrow(a)} \\ \downarrow \\ 3: \frac{K^\uparrow(a) \quad K^\uparrow(d)}{K^\uparrow(\langle a, d \rangle)} \end{array}$$

Idea: Avoid loops using \uparrow and \downarrow by converting only \downarrow to \uparrow , but not the other way round

Normal Message Deduction

$$ND_{\Sigma} = \left\{ \begin{array}{l} \frac{\text{Out}(x)}{K^{\downarrow}(x)} \quad \frac{K^{\uparrow}(x)}{\text{In}(x)} [K(x)] \quad \text{Coerce} : \frac{K^{\downarrow}(x)}{K^{\uparrow}(x)} \\ \\ \frac{\text{Fr}(x : fr)}{K^{\uparrow}(x : fr)} \quad \overline{K^{\uparrow}(x : pub)} \\ \\ \frac{K^{\downarrow}(\langle x, y \rangle)}{K^{\downarrow}(x)} \quad \frac{K^{\downarrow}(\langle x, y \rangle)}{K^{\downarrow}(y)} \\ \\ \text{plus other deconstruction rules} \\ \text{depending on the equations} \\ \\ \frac{K^{\uparrow}(x_1) \quad \dots \quad K^{\uparrow}(x_k)}{K^{\uparrow}(f(x_1, \dots, x_k))} \text{ for all } f \in \Sigma \end{array} \right\}$$

Formalizing Security Properties

Two approaches

Direct formulation

- Formulate property ϕ directly in terms of actions occurring in protocol traces, i.e., as a set of (or predicate on) traces.
- Drawback: standard properties like secrecy and authentication become **highly protocol-dependent**, since they need to refer to the concrete protocol messages.

Protocol instrumentation

- Idea: insert special **claim events** into the protocol roles:

$$\text{Claim_claimtype}(R, t)$$

where R is the executing role, claimtype indicates the type of claim, and t is a message term.

- Serve as interface to **express properties independently of protocol**.
- Example: $\text{Claim_secret}(A, N_A)$ claims that N_A is a secret for role A , i.e., not known to the intruder.

Claim Events

Claim events are part of the protocol rules as actions.

Properties of claim events

- Their only effect is to record facts or claims in the protocol trace.
- They cannot be observed, modified, or generated by the intruder.

Expressing properties using claim events

- Properties of traces tr are expressed in terms of **claim events** and other actions (e.g., adversary knowledge K) occurring in tr .
- Properties are formulated from the **point of view of a given role**, thus yielding security guarantees for that role.
- We concentrate on **secrecy** and various forms of **authentication**, though the approach is not limited to these properties.

Role Instrumentation for Secrecy

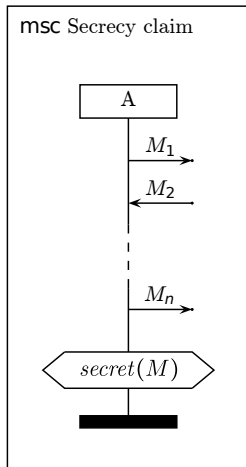
Definition (Secrecy, informally)

The intruder cannot discover the data (e.g., key, nonce, etc.) that is intended to be secret.

Role instrumentation

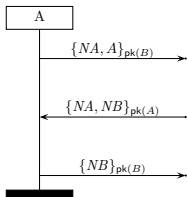
- Insert the claim event $\text{Claim_secret}(A, M)$ into role A to claim that the message M used in the run remains secret.
- Position: At the end of the role.
- For instance, in NSPK, the nonces na and nb should remain secret.

Note: In the graphs, where the executing role is clear from the context, we abbreviate $\text{Claim_claimtype}(A, t)$ to $\text{claimtype}(t)$ inside a hexagon.



Instrumentation for Secrecy of Role A in NSPK

msc NSPK A



$[St_A_1(A, tid, skA, pk(skB)), Fr(NA)] \rightarrow$
 $[St_A_2(A, tid, skA, pk(skB), NA), Out(\{NA, A\}_{pk(skB)})]$

$[St_A_2(A, tid, skA, pk(skB), NA), In(\{NA, NB\}_{pk(skA)})] \rightarrow$
 $[St_A_3(A, tid, skA, pk(skB), NA, NB)]$

$[St_A_3(A, tid, skA, pk(skB), NA, NB)] \xrightarrow{Claim_secret(A, NB)}$
 $[St_A_4(A, tid, skA, pk(skB), NA, NB), Out(\{NB\}_{pk(skB)})]$

Adding `Claim_secret` to role A with respect to `NB`

Formalization of Secrecy

Definition (Secrecy, first attempt)

The secrecy property consists of all traces tr satisfying

$$\forall A, M, i. \text{Claim_secret}(A, M)@i \Rightarrow \neg(\exists j. K(M)@j)$$

- Let $tr = tr_1; tr_2; \dots; tr_k; \dots; tr_n$. We write $x@k$ as a shorthand for $x \in tr_k$.
- Can only require M to remain secret if A runs the protocol with another honest agent, i.e.,
- Trivially broken whenever A or B is instantiated with a compromised agent, since then the adversary rightfully knows M .
- This definition is fine for a **passive adversary**, who only observes network traffic, but does not act as a protocol participant.

Compromised Agent

Definition (Compromised Agent)

A **compromised agent** is under adversary control, i.e., sharing all its information with the adversary and participating in protocols upon its direction. We model this by having the agent give its initial secret information to the adversary, which can then mimic the agent's actions.

We note the fact that an agent A is compromised by a **Rev** event in the trace, attached to the rule that passes its initial secrets (here the private key $Ltk(A, skA)$) to the adversary:

$$[!Ltk(A, skA)] \xrightarrow{\text{Rev}(A)} [\text{Out}(skA)]$$

Exercise: convince yourself that, given the agent's secret, the adversary is capable of performing all of the agent's send and receive steps.

Formalization of Secrecy

Definition (Honesty)

An agent A is **honest** in a trace tr when $\text{Rev}(A) \notin tr$.

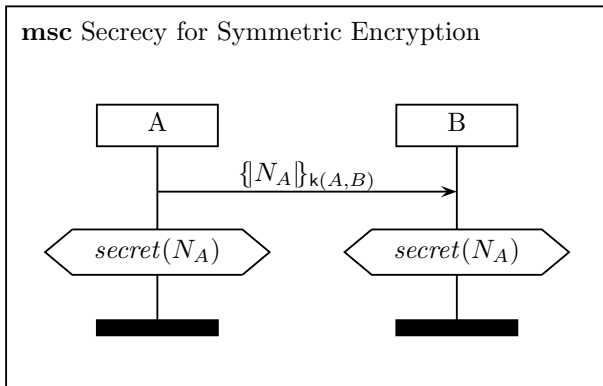
When making a claim in a rule action, all parties B that are expected to be honest need to be listed with a $\text{Honest}(B)$ action in that rule.

Definition (Secrecy)

The secrecy property consists of all traces tr satisfying

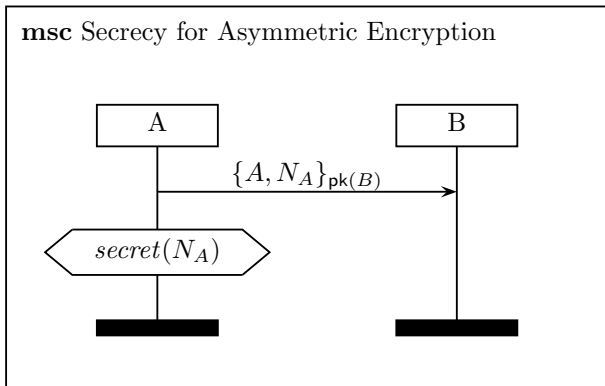
$$\begin{aligned} & \forall A M i. (\text{Claim_secret}(A, M)@i) \\ & \Rightarrow (\neg(\exists j. \text{K}(M)@j) \vee (\exists B j. \text{Rev}(B)@j \wedge \text{Honest}(B)@i)) \end{aligned}$$

Secrecy Example #1



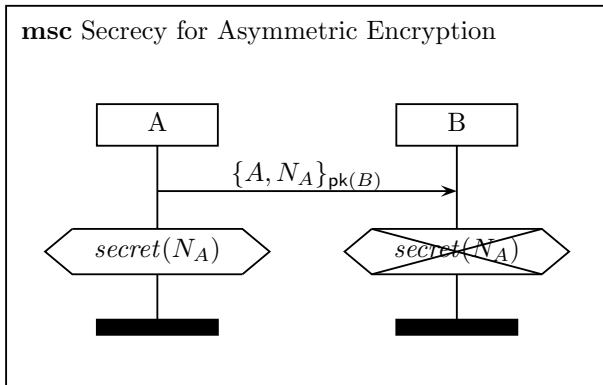
- This is fine: secrecy holds for both A and B .
- We omit the obvious annotations $Honest(A)$, $Honest(B)$ in message sequence charts for 2-party protocols.

Secrecy Example #2



- Secrecy holds for *A*: she knows that only *B* can decrypt message.

Secrecy Example #2



- **Secrecy fails for B**: he does not know who encrypted message!
- ... unless one can **authenticate** the origin of the message

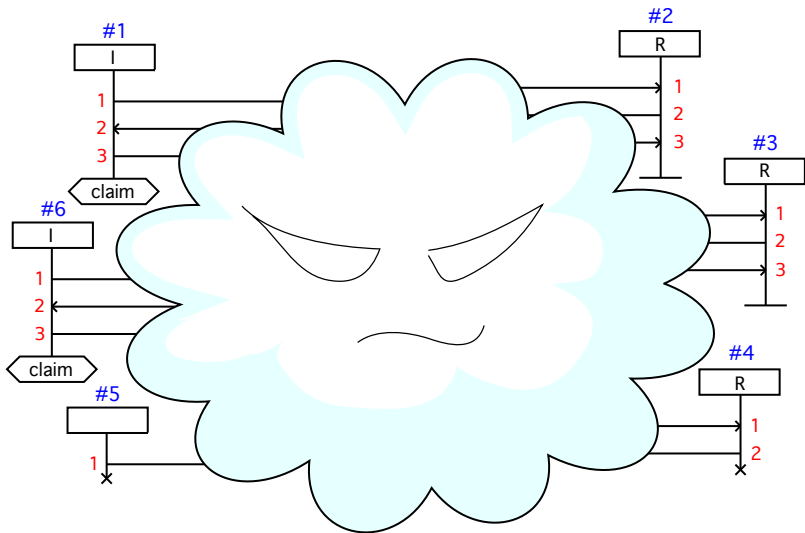
Which authentication are you talking about?

- **No unique definition** of authentication, but a variety of different forms.
- Considerable effort has been devoted to **specifying and classifying, semi-formally or formally**, different forms of authentication (e.g., by Cervesato/Syverson, Clark/Jacob, Gollmann, Lowe, Cremers et al.).

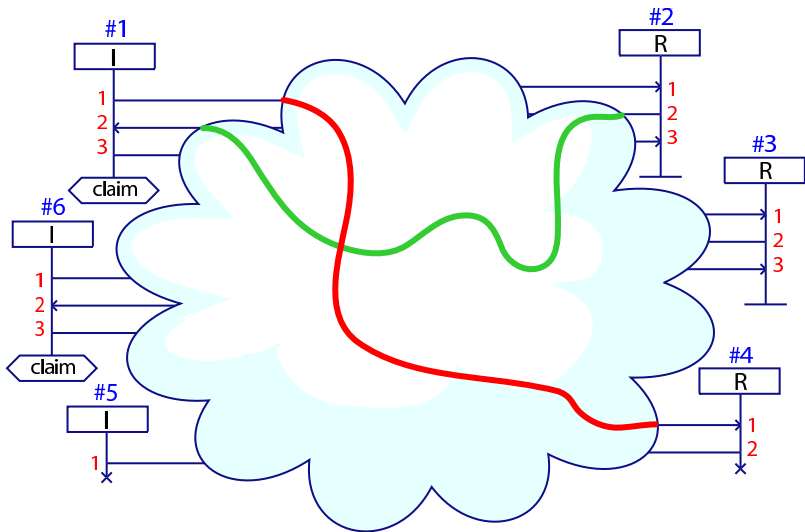
Examples

- ping authentication, aliveness, weak agreement, non-injective agreement, injective agreement, weak and strong authentication, synchronization, and matching histories.

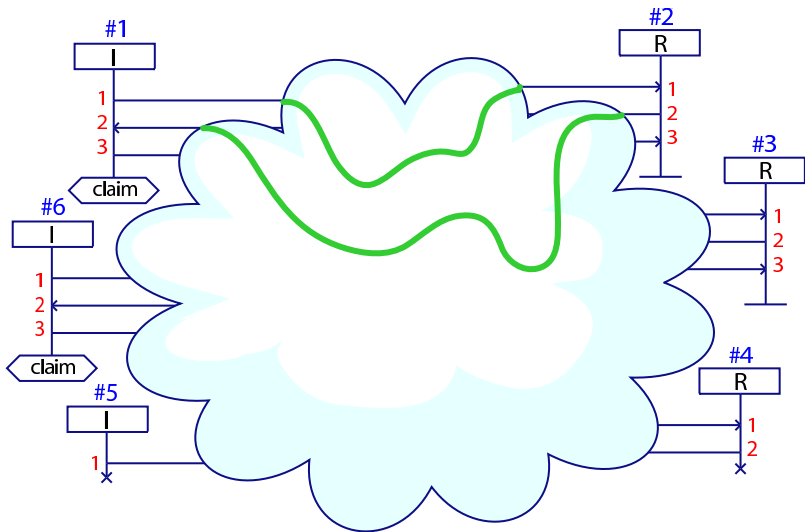
A Picture of the World



Failed Authentication



Successful Authentication



A Hierarchy of Authentication Specifications (1)

[Gavin Lowe, 1997]

Gavin Lowe has defined the following **hierarchy of increasingly stronger authentication properties**¹:

Aliveness A protocol guarantees to an agent a in role A aliveness of another agent b if, whenever a completes a run of the protocol, apparently with b in role B , then b has previously been running the protocol.

Weak agreement A protocol guarantees to an agent a in role A weak agreement with another agent b if, whenever agent a completes a run of the protocol, apparently with b in role B , then b has previously been running the protocol, **apparently with a** .

¹Terminology and notation slightly adapted to our setting.

A Hierarchy of Authentication Specifications (2)

[Gavin Lowe, 1997]

Non-injective agreement A protocol guarantees to an agent a in role A non-injective agreement with an agent b in role B on a message M if, whenever a completes a run of the protocol, apparently with b in role B , then b has previously been running the protocol, apparently with a , and b was acting in role B in his run, and the two principals agreed on the message M .

Injective agreement is non-injective agreement where additionally each run of agent a in role A corresponds to a unique run of agent b .

Also versions including **recentness**: insist that B 's run was recent (e.g., within t time units).

These are quite complex properties. How can we formalize them?

Role Instrumentation for Authentication

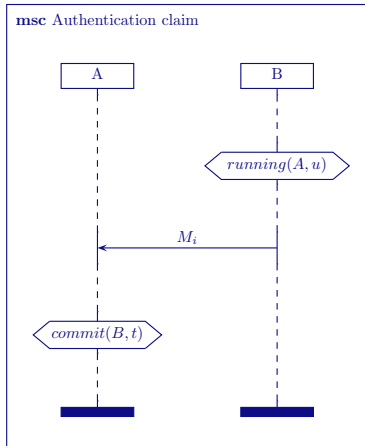
We use two claims to express that role A authenticates role B on t :

In role A :

- Insert a **commit claim** event
 $\text{Claim_commit}(A, B, t)$.
- Position: after A can construct t . Typically, at end of A 's role.

In role B :

- Insert a **running claim** event
 $\text{Claim_running}(B, A, u)$.
- Term u is B 's view of t .
- Position: after B can construct u and **causally preceding**
 $\text{Claim_commit}(A, B, t)$.



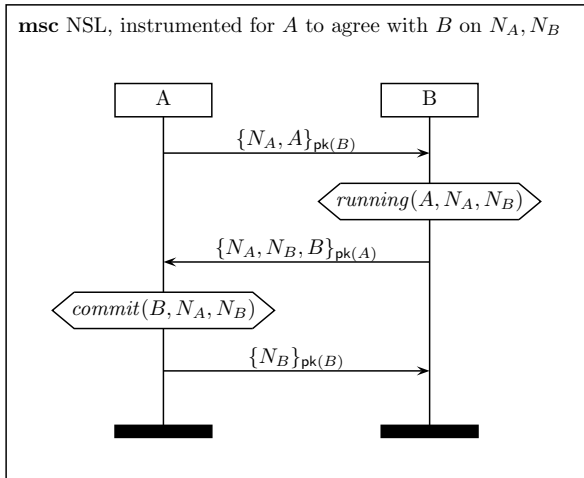
Definition (Non-injective agreement)

The property $Agreement_{NI}(A, B, t)$ consists of all traces satisfying

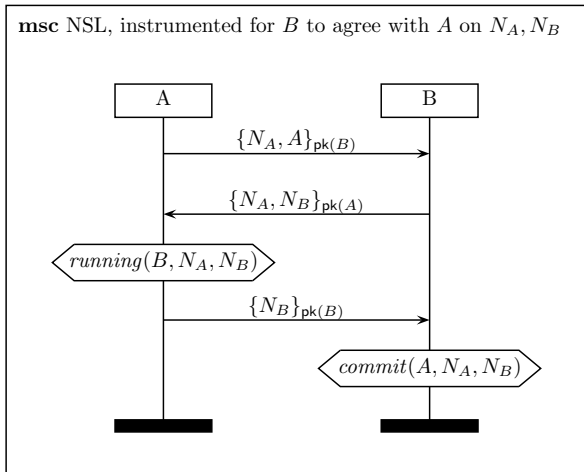
$$\begin{aligned} \forall a \ b \ t \ i. \quad & \text{Claim_commit}(a, b, \langle A, B, t \rangle)@i \\ & \Rightarrow (\exists j. \text{Claim_running}(b, a, \langle A, B, t \rangle)@j \wedge j < i) \\ & \vee (\exists X \ r. \text{Rev}(X)@r \wedge \text{Honest}(X)@i) \end{aligned}$$

- Whenever a commit claim is made with honest agents a and b , then the peer b must be running with the same parameter t , or the adversary has compromised at least one of the two agents.

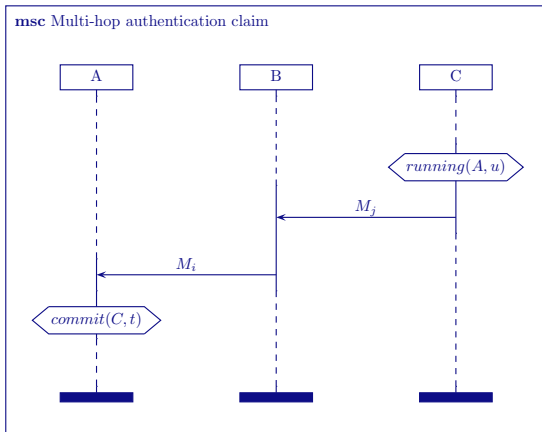
Example: NSL Protocol (1/2)



Example: NSL Protocol (2/2)

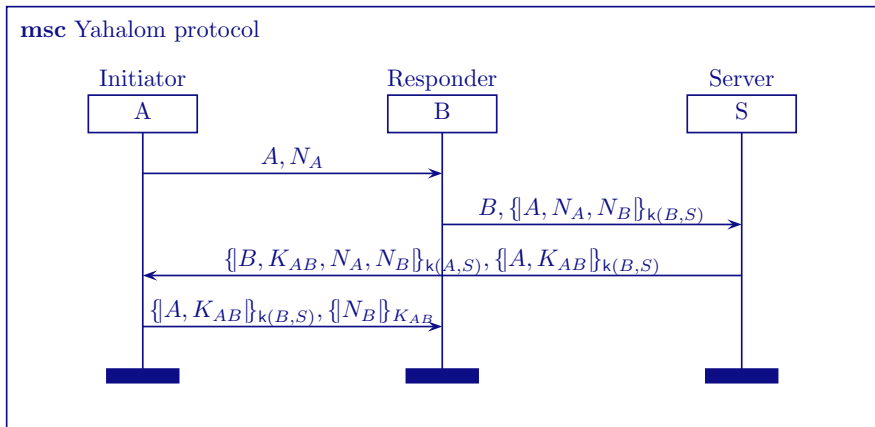


Role Instrumentation for Authentication (cont.)



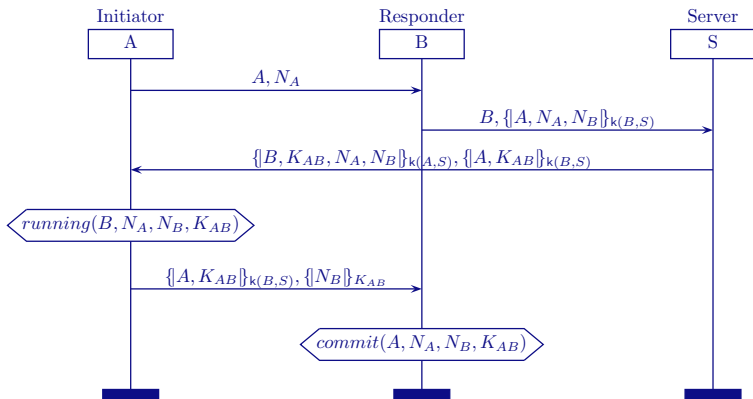
Event causality in multi-hop authentication claims: The *running* event must causally precede the *commit* event and the messages t and u must be known at the position of the claim event in the respective role.

Example: Yahalom Protocol (1/3)

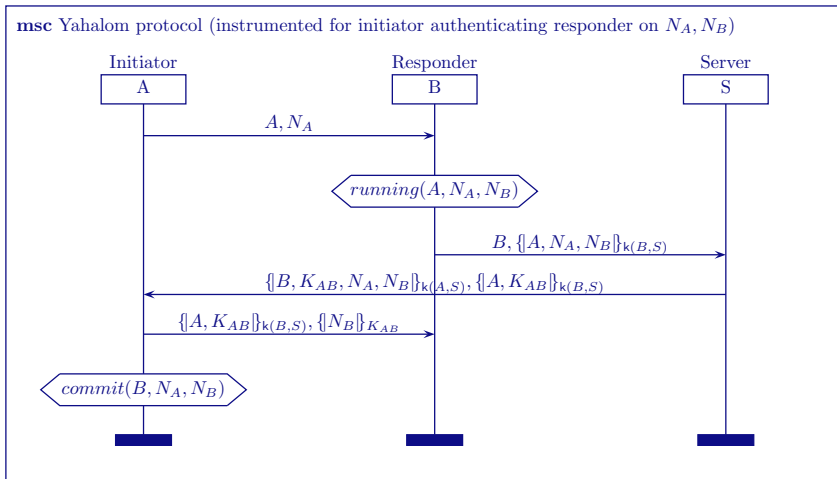


Example: Yahalom Protocol (2/3)

msc Yahalom protocol (instrumented for responder authenticating initiator on N_A, N_B, K_{AB})



Example: Yahalom Protocol (3/3)



Note: agreement for A on K_{AB} is not possible, since B gets K_{AB} after A.

Formalizing Authentication

Definition (Injective agreement)

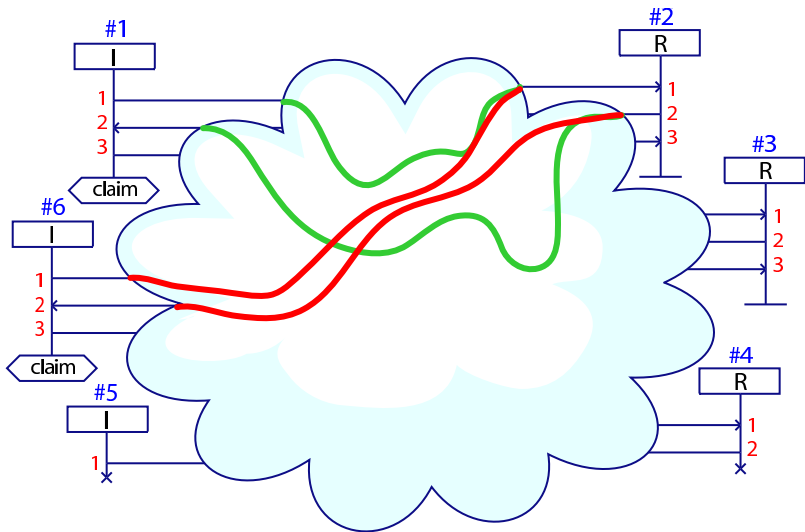
The property $Agreement(A, B, t)$ consists of all traces satisfying:

$$\begin{aligned} \forall a \ b \ t \ i. \quad & \text{Claim_commit}(a, b, \langle A, B, t \rangle)@i \\ & \Rightarrow (\exists j. \text{Claim_running}(b, a, \langle A, B, t \rangle)@j \wedge j < i \\ & \quad \wedge \neg(\exists a_2 \ b_2 \ i_2. \text{Claim_commit}(a_2, b_2, \langle A, B, t \rangle)@i_2 \\ & \quad \quad \wedge \neg(i_2 = i) \\ & \quad \quad \quad) \\ & \quad \quad \quad) \\ & \vee (\exists X \ r. \text{Rev}(X)@r \wedge \text{Honest}(X)@i) \end{aligned}$$

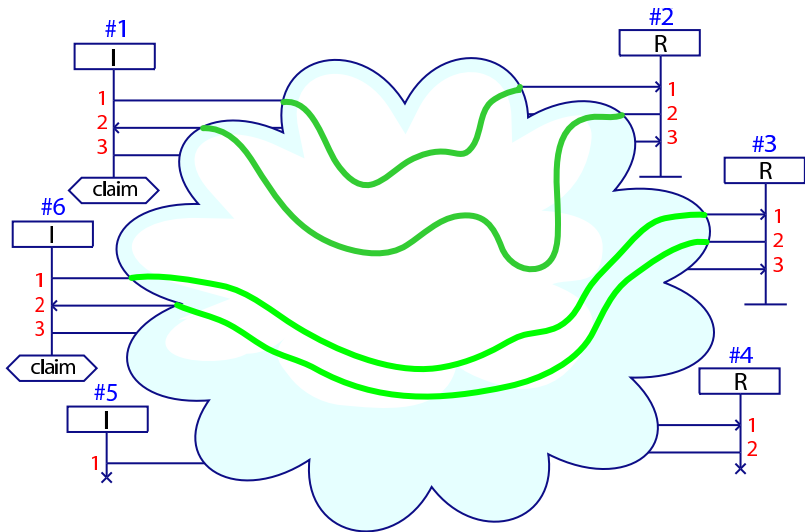
Remarks

- For each commit by a in role A on the trace there is a **unique** matching b executing role B .

Failed Injective Authentication

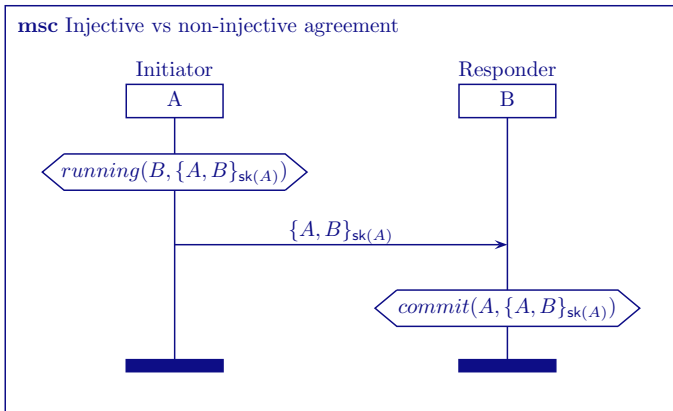


Successful Injective Authentication



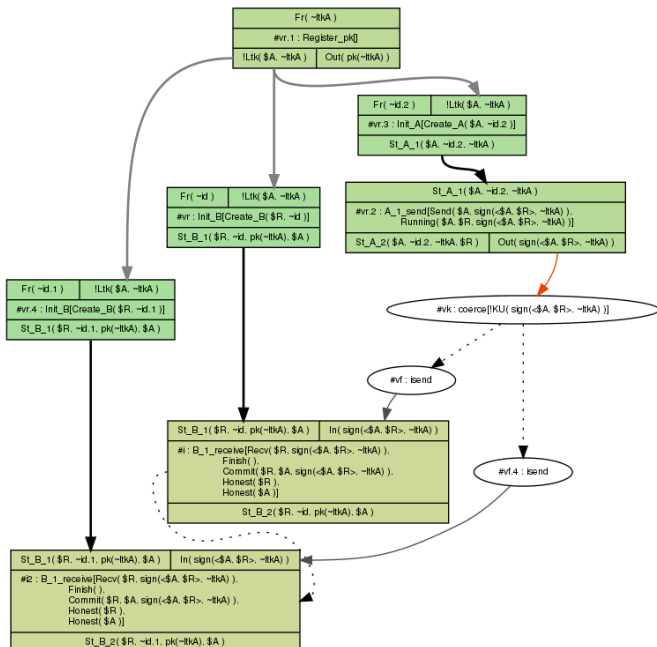
Injective vs Non-injective Agreement

Separating Example



- Non-injective agreement holds.
- Injective agreement fails, since the adversary can replay message to several threads in responder role B .

Injective Agreement counter-example



Formalizing Authentication

Weaker Variants

Definition (Weak agreement)

A trace tr satisfies the property $WeakAgreement(A, B)$ iff

$$\begin{aligned} \forall a b i. \text{Claim_commit}(a, b, \langle \rangle)@i \\ \Rightarrow (\exists j. \text{Claim_running}(b, a, \langle \rangle)@j) \\ \quad \vee (\exists X r. \text{Rev}(X)@r \wedge \text{Honest}(X)@i) \end{aligned}$$

It is sufficient that the agents agree they are communicating, it is not required that they play the right roles. Note also the empty list of data $\langle \rangle$ that is agreed upon, i.e., none.

Formalizing Authentication

Weaker Variants

Definition (Aliveness)

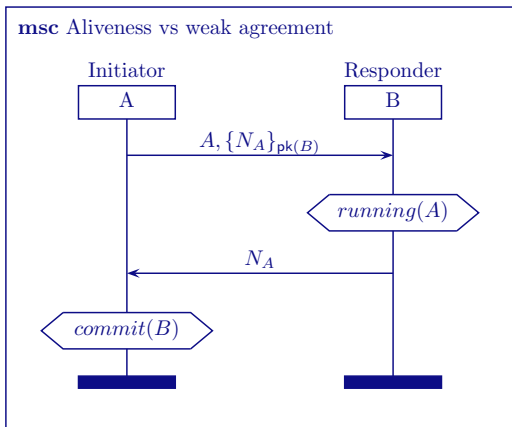
A trace tr satisfies the property $Alive(A, B)$ iff

$$\begin{aligned} \forall a \ b \ i. \quad & \text{Claim_commit}(a, b, \langle \rangle)@i \\ \Rightarrow & (\exists j \ id. \text{Create_B}(b, id)@j \vee \text{Create_A}(b, id)@j) \\ & \vee (\exists X \ r. \text{Rev}(X)@r \wedge \text{Honest}(X)@i) \end{aligned}$$

It is neither required that the agent b , believed to instantiate role B by agent a , really plays role B , nor that he believes to be talking to a .

Aliveness vs Weak Agreement

Separating Example



- Aliveness holds: only B can have decrypted the fresh nonce N_A .
- Weak agreement fails, since adversary may modify unprotected identity A to C in first message so that B thinks he is talking to C .

Weak Agreement counter-example

```
theory ALIVEvsWEAKAGREE
begin
builtins: asymmetric-encryption

/* Public key infrastructure */
rule Register_pk:
  [ Fr(~ltkA) ] -->
  [ !Ltk($A, ~ltkA), Out(pk(~ltkA)) ]

rule Reveal_ltk:
  [ !Ltk(A, ltkA) ] --[ Reveal(A) ]-> [ Out(ltkA) ]

/* We formalize the following protocol
   1. A -> B: A, {na}_pk(B)
   2. A <- B: na
*/
```

Weak Agreement counter-example

```
/* new session thread creation */
```

```
rule Init_A:
```

```
  [ Fr(~id), !Ltk(R, ltkR) ]  
  --[ Create_A($I, ~id) ]->  
  [ St_A_1($I, ~id, pk(ltkR), R) ]
```

```
rule Init_B:
```

```
  [ Fr(~id), !Ltk(R, ltkR) ]  
  --[ Create_B(R, ~id) ]->  
  [ St_B_1(R, ~id, ltkR) ]
```

```
/* 1. A -> B: A, {na}_pk(B) */
```

```
rule A_1_send:
```

```
  [ St_A_1(I, ~id, pkltkR, R), Fr(~ni) ]  
  --[ Send(I, <I, aenc{~ni}pkltkR>), OUT_I_1(aenc{~ni}pkltkR)]->  
  [ St_A_2(I, ~id, pkltkR, R, ~ni), Out(<I, aenc{~ni}pkltkR>) ]
```

```
rule B_1_receive:
```

```
  [ St_B_1(R, ~id, ltkR), In(<I, aenc{ni}pk(ltkR)>) ]  
  --[ Recv(R, <I, aenc{ni}pk(ltkR)>), IN_R_1_ni(ni, aenc{ni}pk(ltkR)) ]->  
  [ St_B_2(R, ~id, ltkR, I, ni) ]
```


Weak Agreement counter-example

```
/* 2. A <- B: na */
rule B_2_send:
  [ St_B_2(R, ~id, ltkR, I, ni) ]
--[ Send(R, ni), Running(R, I) ]->
  [ St_B_3(R, ~id, ltkR, I, ni), Out(ni) ]

rule A_2_receive:
  [ St_A_2(I, ~id, pkltkR, R, ~ni), In(~ni) ]
--[ Recv(I, ~ni), Commit(I, R), Honest(R), Honest(I), Finish()]->
  [ St_A_3(I, ~id, pkltkR, R, ~ni) ]

/* it helps to deal with infinite loops in applying rules */
lemma types [sources]:
"(All ni m1 #i. IN_R_1_ni( ni, m1) @ i ==>
  ((Ex #j. KU(ni) @ j & j < i) | (Ex #j. OUT_I_1( m1 ) @ j))
)"
```

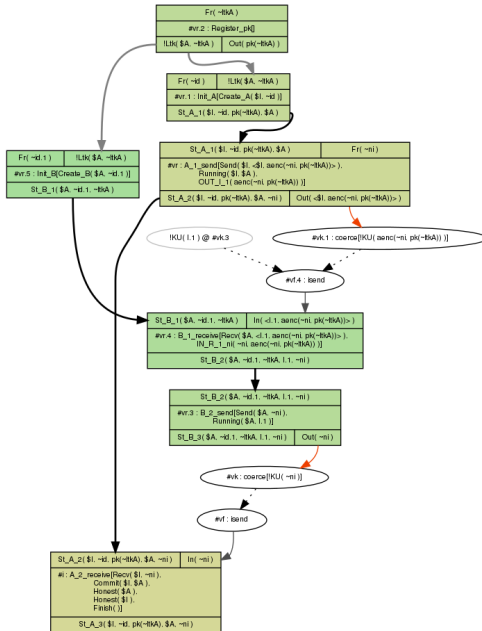
Weak Agreement counter-example

```
/* protocol is executable */
lemma executable:
exists-trace
  "Ex #i. Finish() @i & not (Ex X #j. Reveal(X) @j) "

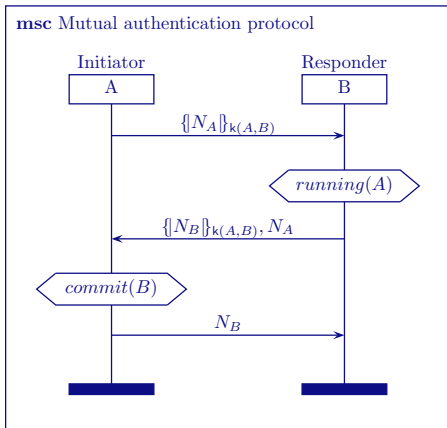
/* succeeds */
lemma aliveness:
all-traces
  "All A B #i. Commit(A,B) @ i ==>
    (Ex #j id. Create_B(B,id) @j) | (Ex #j id. Create_A(B,id)@j)
    | (Ex X #r. Reveal(X) @ r & Honest(X) @i)"

/* fails as expected */
lemma weakagree:
all-traces
  "All A B #i. Commit(A,B) @i ==>
    (Ex #j. Running(B,A) @j) | (Ex X #r. Reveal(X) @ r & Honest(X) @i)"
end
```

Weak Agreement counter-example

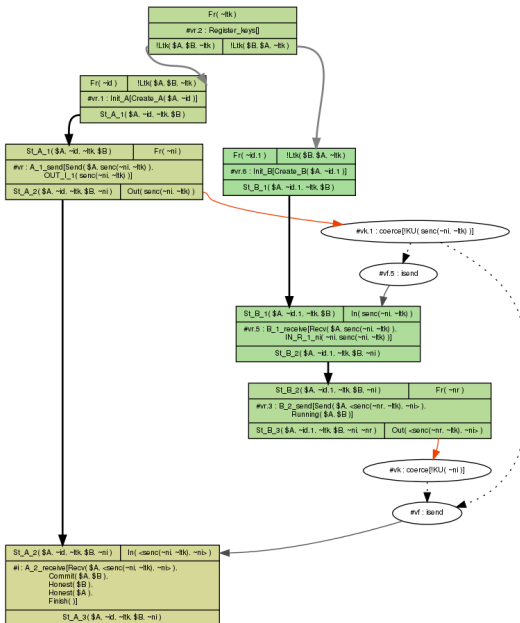


When Even Aliveness Fails ...



- **Reflection attack:** A may complete run without B's participation.
- Hence, aliveness fails.

Attack found by Tamarin



Key-related Properties

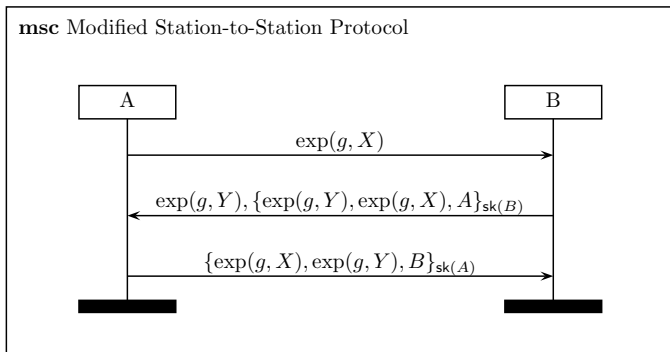
Basic key-oriented goals

- key **freshness**
- (implicit) **key authentication**: a key is only known to the communicating agents A and B and mutually trusted parties
- **key confirmation** of A to B is provided if B has assurance that agent A has possession of key K
- **explicit key authentication** = key authentication + key confirmation
⇒ expressible in terms of secrecy and agreement

Goals concerning compromised keys

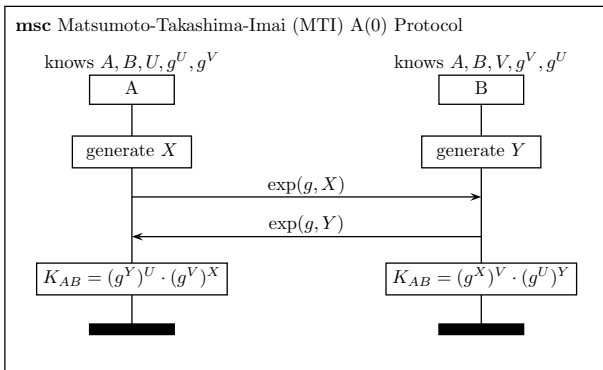
- (**perfect**) **forward secrecy**: compromise of long-term keys of a set of principals does not compromise the session keys established in previous protocol runs involving those principals
- resistance to **key-compromise impersonation**: compromise of long-term key of an agent A does not allow the adversary to masquerade to A as a different principal.

Forward Secrecy: Example 1



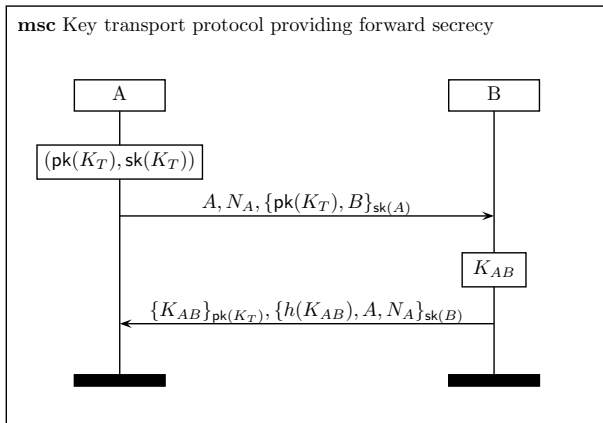
- Signatures are used to authenticate the Diffie-Hellman public keys $\exp(g, X)$ and $\exp(g, Y)$.
- Protocol provides forward secrecy: The adversary cannot derive session key $K_{AB} = \exp(\exp(g, X), Y)$ by compromise of signing keys.

Forward Secrecy: Example 2



- Message exchange as in basic DH; protocol combines long-term and ephemeral DH keys to authenticate exchanged DH public keys.
- Protocol **does not provide forward secrecy**: The adversary can construct the session key $K_{AB} = g^{VX+UY}$ as $(g^X)^V \cdot (g^Y)^U$ from observed messages and long-term private keys U and V .

Forward Secrecy: Example 3



- A generates an ephemeral asymmetric key pair $(pk(K_T), sk(K_T))$.
- Protocol provides **forward secrecy without using Diffie-Hellman** keys: Adversary cannot learn session key by compromise of signing keys.

Built-in message theories 1

hashing

Functions: $h/1$

Equations: No equations

asymmetric-encryption

Functions: $aenc/2$, $adec/2$, $pk/1$

Equations: $adec(aenc(m, pk(sk)), sk) = m$

signing

Functions: $sign/2$, $verify/3$, $pk/1$, $true/0$

Equations: $verify(sign(m,sk),m,pk(sk)) = true$

symmetric-encryption

Functions: $senc/2$, $sdec/2$

Equations: $sdec(senc(m,k),k) = m$

Built-in message theories 2

diffie-hellman

Functions: $\text{inv}/1, 1/0, \wedge/2, */2$

Equations: $(x \wedge y) \wedge z = x \wedge (y \wedge z)$ $x \wedge 1 = x$ $x * y = y * x$
 $(x * y) * z = x * (y * z)$ $x * 1 = x$ $x * \text{inv}(x) = 1$

xor

Functions: $\text{XOR}/2, \text{zero}/0$

Equations:
 $x \text{ XOR } y = y \text{ XOR } x$ $(x \text{ XOR } y) \text{ XOR } z = x \text{ XOR } (y \text{ XOR } z)$
 $x \text{ XOR } \text{zero} = x$ $x \text{ XOR } x = \text{zero}$

multiset

Functions: $+/2$

Equations: $x + y = y + x$ $(x + y) + z = x + (y + z)$

Then you can define a “Smaller” predicate by
predicates: $\text{Smaller}(x,y) \Leftrightarrow \exists x z. x + z = y$

Restrictions

Restrictions can be used to exclude undesired traces: Tamarin will ignore all traces that violate a restriction

- Take care not to exclude attacks!
- Many standard applications:
 - Equality
 - Inequality
 - LessThan
 - GreaterThan
 - OnlyOnce
- Essentially same syntax as lemmas

Restriction Example: Only Once

restriction once:

```
"All #i #j. OnlyOnce()@#i & OnlyOnce()@#j  
    ==> #i=#j"
```

Given the following rules:

```
rule 1: [ ] --[ OnlyOnce() ]-> [ A('5') ]
```

```
rule 2: [ A(x) ] --[ Step(x) ]-> [ B(x) ]
```

The restriction forbids e.g. the following execution (valid without restriction):

```
[ ] --[ OnlyOnce() ]--> [ A('5') ]  
--[ OnlyOnce() ]-> [ A('5'), A('5') ]  
--[ Step('5') ]-> [ A('5'), B('5') ]
```

But the following is still allowed:

```
[ ] --[ OnlyOnce() ]-> [ A('5') ]  
--[ Step('5') ]-> [ B('5') ]
```

Restriction Example: Inequality

restriction InEq:

```
"All x #i. Neq(x,x)@#i ==> F"
```

Given the following rules:

```
rule 1: [ ] --[ A1() ]-> [ A('1') ]
```

```
rule 2: [ ] --[ A2() ]-> [ A('2') ]
```

```
rule 3: [ A(x), A(y) ] --[ Neq(x,y) ]-> [ B(x,y) ]
```

The restriction forbids e.g. the following execution (valid without restriction):

```
[ ] --[ A1() ]-> [ A('1') ]
```

```
--[ A1() ]-> [ A('1'), A('1') ]
```

```
--[ Neq('1','1') ]-> [ B('1','1') ]
```

But the following is still allowed:

```
[ ] --[ A1() ]-> [ A('1') ]
```

```
--[ A2() ]-> [ A('1'), A('2') ]
```

```
--[ Neq('1','2') ]-> [ B('1','2') ]
```

Command line parameters

`tamarin-prover --help` or `manual` to see all parameters.

Most important options:

- `--prove` Attempt to prove the lemmas
- `--stop-on-trace [=DFS|BFS|NONE]` How to search for traces (default DFS)
- `-b --bound [=INT]` Bound the depth of the proofs
- `--heuristic [= (s|S|o|O|p|P|l|c|C|i|I)+]` Sequence of goal rankings to use (default 's')
- `--diff` Turn on observational equivalence mode using diff terms.
- `--quit-on-warning` Strict mode that quits on any warning that is emitted. (Good for debugging!)
- `--parse-only` Just parse the input file and pretty print it as-is (to find syntax errors)
- `-V --version` Print version information (for bug reports)

Exporting and importing proofs and attacks

Running Isabelle 1.2.3

Index Download

Actions

Options

Proof scripts

Multiset rewriting rules and restrictions (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

Lemma Client_session_key_secret:

```
all-traces
"~(∃ S k #i #j.
  ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
  ~(∃ #r. LtkReveal( S ) @ #r)))"
```

simplify

```
solve( Client_1( S, k ) > #i #j )
case Client_1
solve( !KU( ~k ) @ #vk.1 )
case Client_1
solve( !KU( ~ltk ) @ #vk.2 )
by contradiction /* from Formulas */
```

qed

qed

Lemma Client_auth:

```
all-traces
"∀ S k #i.
  ((SessKeyC( S, k ) @ #i) =
  (∃ #a. AnswerRequest( S, k ) @ #a) ∧
  (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
```

by sorry

Lemma Client_auth_injective:

```
all-traces
"∀ S k #i.
  ((SessKeyC( S, k ) @ #i) =
  ((∃ #a.
  (AnswerRequest( S, k ) @ #a) ∧
  (∀ #j. (SessKeyC( S, k ) @ #j) = (#i = #j))) ∧
  (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i)))"
```

by sorry

Lemma Client_session_key_honest_setup:

```
exists-trace
"∃ S k #i.
  (SessKeyC( S, k ) @ #i) ∧ ~(∃ #r. LtkReveal( S ) @ #r)"
```

by sorry

end

Case: Reveal_ltk

Applicable Proof Methods: Goals sorted according to the 'smart' heuristic (loop breakers delayed)

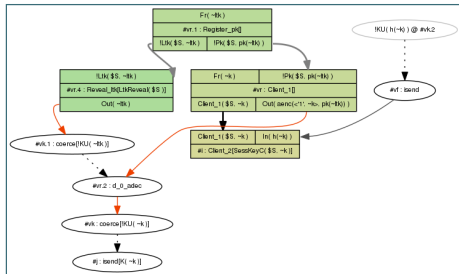
1. contradiction /* from formulas */

2. solve(!KU(h(~k)) @ #vk.2) // nr. 4

a. autoprove (A. for all solutions)

b. autoprove (B. for all solutions) with proof-depth bound 5

Constraint system



last: none

formulas:

```
⊥
∨ #r. (!LtkReveal( S.S ) @ #r) ⇒ !
```

“Download” allows to export proofs. Gives a valid .spthy containing spec and proof, which is again verified on loading.

GUI configuration

Running TAMARIN 1.2.3

Index Download Actions Options

Abbreviate terms
Graph simplification off
Graph simplification L1
Graph simplification L2
Graph simplification L3

Proof scripts

Multiset rewriting rules and restrictions (8)
Raw sources (10 cases, deconstructions complete)
Refined sources (10 cases, deconstructions complete)

```
lemma Client_session_key_secret:
  all_traces
  "~(∃ S k #i #j.
    ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
    (¬(∃ #r. LtkReveal( S ) @ #r)))"

simplify
solve Client_1( ( S, k ) * #i #j )
case Client_1
  solve (KU( -k ) @ #vk.1 )
  case Client_1
    solve (KU( -ltk ) @ #vk.2 )
    case Reveal_ltk
      by contradiction /* from formulas */
qed
qed

lemma Client_auth:
  all_traces
  "∀ S k #i.
    (SessKeyC( S, k ) @ #i) →
    (∃ #a. AnswerRequest( S, k ) @ #a) ∨
    (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i))"

by sorry

lemma Client_auth_injective:
  all_traces
  "∀ S k #i.
    (SessKeyC( S, k ) @ #i) →
    (∃ #a.
      (AnswerRequest( S, k ) @ #a) ∧
      (∀ #j. (SessKeyC( S, k ) @ #j) → (#i = #j))) ∨
    (∃ #r. (LtkReveal( S ) @ #r) ∧ (#r < #i))"

by sorry

lemma Client_session_key_honest_setup:
  exists_trace
  "∃ S k #i.
    (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @ #r))"

by sorry
```

Case: Reveal_ltk

Applicable Proof Methods: Goals sorted according to the 'smart' heuristic (loop breakers disabled)

1. **contradiction** /* from formulas */
2. **solve** (IKU(h(-k)) @ #vk.2) // nr. 4

a. **autoprove** (A. for all solutions)
b. **autoprove** (B. for all solutions) with proof-depth bound 5

Constraint system

The diagram illustrates the constraint system for the 'Reveal_ltk' case. It consists of several nodes and directed edges representing dependencies between constraints. The nodes are:

- FR(-lk)**: A box containing $\#r.1: \text{Register_pk}$ and $(\text{Ltk } \$\$ -lk) \mid \text{PK } \$\$ \text{ pk}(-lk)$.
- Ltk**: A box containing $(\text{Ltk } \$\$ -lk)$ and $\#r.4: \text{Reveal_ltk}(\text{LtkReveal } \$\$)$.
- Client_1**: A box containing $\text{Client}_1(\$\$ -lk)$ and $\text{Out } (-lk)$.
- Client_2**: A box containing $\text{Client}_2(\$\$ -lk)$ and $\text{In } (h(-k))$.
- Client_2_SessKeyC**: A box containing $\#i: \text{Client}_2(\text{SessKeyC } \$\$ -k)$.
- Goal nodes**: $\#vk.1: \text{coreq}[KU(-lk)]$, $\#vk.2: \text{d}_0_adec$, $\#i: \text{band}[K(-k)]$, and $\#i: \text{band}$.

Edges indicate dependencies: $\text{FR}(-lk) \rightarrow \text{Ltk}$, $\text{Ltk} \rightarrow \text{Client}_1$, $\text{Ltk} \rightarrow \text{Client}_2$, $\text{Client}_1 \rightarrow \text{Client}_2$, $\text{Client}_2 \rightarrow \text{Client}_2_SessKeyC$, $\text{Client}_2_SessKeyC \rightarrow \#i: \text{band}[K(-k)]$, $\#i: \text{band}[K(-k)] \rightarrow \#vk.2: \text{d}_0_adec$, $\#vk.2: \text{d}_0_adec \rightarrow \#vk.1: \text{coreq}[KU(-lk)]$, and $\#i: \text{band} \rightarrow \#vk.1: \text{coreq}[KU(-lk)]$. There is also a dashed arrow from $\text{IKU}(h(-k)) @ \#vk.2$ to $\#i: \text{band}$.

last: none
formulas:
⊥
val(\$S) @ #r1 => 1

127.0.0.1:3001/thy/trace/2/overview/proof/Client_session_key_secret/_Client_1/Client_1/Reveal_ltk#

Allows to choose the level of details in graph visualization.

Keyboard shortcuts

Running TAMARIN 1.2.3

Index

Download

Actions »

Options »

Proof scripts

theory **Tutorial** begin

Message theory

Multiset rewriting rules and restrictions (8)

Raw sources (10 cases, deconstructions complete)

Refined sources (10 cases, deconstructions complete)

Lemma Client_session_key_secrety:

```
all-traces
  ~{(∃ S k #i #j.
    ((SessKeyC( S, k ) @ #i) ∧ (K( k ) @ #j)) ∧
    (¬(∃ #r. LtkReveal( S ) @ #r)))}
```

by sorry

Lemma Client_auth:

```
all-traces
  ~∀ S k #i.
    (SessKeyC( S, k ) @ #i) →
    ((∃ #a. AnswerRequest( S, k ) @ #a) ∨
    (∃ #r. LtkReveal( S ) @ #r) ∧ (#r < #i))
```

by sorry

Lemma Client_auth_injective:

```
all-traces
  ~∀ S k #i.
    (SessKeyC( S, k ) @ #i) →
    ((∃ #a.
      (AnswerRequest( S, k ) @ #a) ∧
      (∀ #j. (SessKeyC( S, k ) @ #j) → (#i = #j))) ∨
    (∃ #r. LtkReveal( S ) @ #r) ∧ (#r < #i))
```

by sorry

Lemma Client_session_key_honest_setup:

```
exists-trace
  ~∃ S k #i.
    (SessKeyC( S, k ) @ #i) ∧ (¬(∃ #r. LtkReveal( S ) @ #r))
```

by sorry

end

Visualization display

Theory: Tutorial (Loaded at 21:02:33 from Local "examples/Tutorial.spthy")

Quick introduction

Left pane: Proof scripts display.

- When a theory is initially loaded, there will be a line at the end of each theorem stating "by sorry // not yet proven". Click on *sorry* to inspect the proof state.
- Right-click to show further options, such as *autoprove*.

Right pane: Visualization.

- Visualization and information display relating to the currently selected item.

Keyboard shortcuts

j/k	Jump to the next/previous proof path within the currently focused lemma.
J/K	Jump to the next/previous open goal within the currently focused lemma, or to the next/previous lemma if there are no more <i>sorry</i> steps in the proof of the current lemma.
1-9	Apply the proof method with the given number as shown in the applicable proof method section in the main view.
a/A	Apply the <i>autoprove</i> method to the focused proof step. a stops after finding a solution, and A searches for all solutions. Needs to have a <i>sorry</i> selected to work.
b/B	Apply a bounded-depth version of the <i>autoprove</i> method to the focused proof step. b stops after finding a solution, and B searches for all solutions. Needs to have a <i>sorry</i> selected to work.
?	Display this help message.

Makes proof inspection quicker.

Syntactic sugar: let statements

```
rule Serv_1:
  [ !Ltk($S, ~ltkS)
    , In( request )
  ]
--[ Eq(fst(adec(request, ~ltkS)), '1')
  ]->
  [ Out( h(snd(adec(request, ~ltkS))) ) ]
```

can be written as

```
rule Serv_1:
  let d = adec(request, ~ltkS) in
  [ !Ltk($S, ~ltkS)
    , In( request )
  ]
--[ Eq(fst(d), '1')
  ]->
  [ Out( h(snd(d)) ) ]
```

Syntactic sugar: formal comments

Normal comments are lost when exporting a proof.

```
section{* This comment is kept even  
        when exporting proofs. *}
```

Problem: How to be convinced that your model is correct?

- Check for warnings and errors
- Inspect message theory and case distinctions
- Use sanity lemmas, e.g., to verify that the normal protocol execution is possible
- Change details and check whether the expected happens

Observational Equivalence - Motivation

Two types of properties:

- **Trace properties:**
 - (Weak) secrecy as reachability
 - Authentication as correspondence



- **Observational equivalence**



Why observational equivalence?

Consider classic **Dolev-Yao** intruder for deterministic public-key encryption:

$$\frac{enc(x, pk(k)) \quad k}{x}$$

- Intruder can only decrypt if he knows the secret key

Now consider a simple **voting** system:

- Voter chooses $v = \text{"Yes"}$ or $v = \text{"No"}$
- Encrypt v using server's public key $pk(k)$: $c = enc(v, pk(k))$
- Send c to server

Is the vote **secret**?

- Dolev-Yao: **Yes**, intruder does not know server's secret key
- Reality: **No**, encryption is deterministic and there are only two choices
 - **Attack**: encrypt "Yes", and compare to c

Observational Equivalence vs Reachability

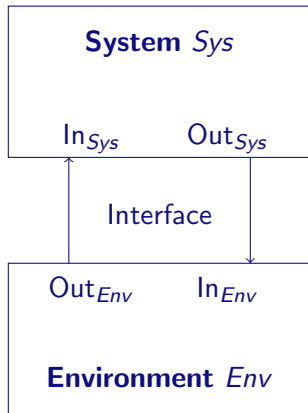
- **Reachability**-based (weak) secrecy is insufficient
- **Stronger** notion: intruder cannot distinguish
 - a system where the voter votes “Yes” from
 - a system where the voter votes “No”
- **Observational equivalence** between two systems
- Can be used to express:
 - Strong secrecy
 - Privacy notions
 - Game-based notions, e.g., ciphertext indistinguishability

Running Example

- **Auction** system
- Property: **strong secrecy** of bids
- Property *violated*: **Shout-out auction**
 - Broadcast bid (e.g., A or B)
 - Send “A” in first system
 - Send “B” in second system
 - Observer knows if he is observing first or second system
- Property **holds**: using **shared symmetric key**
 - Shared symmetric key k between bidder and auctioneer
 - Send “ $\{A\}_k$ ” in first system
 - Send “ $\{B\}_k$ ” in second system
 - Observer has no access to k , does not know which system he is observing

System and environment

- We separate **environment** and **system**
 - System: agents running according to protocol
 - Environment: adversary acting according to its capabilities
- Environment can observe:
 - Output of the system
 - If system reacts at all



Defining observational equivalence

- **Two** system **specifications** given as set of rules
 - One rule per role action (send/receive)
 - Running example shout-out auction:

$$\text{System 1: } \frac{}{\text{Out}_{\text{Sys}}(A)}$$

$$\text{System 2: } \frac{}{\text{Out}_{\text{Sys}}(B)}$$

- Interface and environment/adversary rule(s):

$$\frac{\text{Out}_{\text{Sys}}(X)}{\text{In}_{\text{Env}}(X)}$$

$$\frac{\text{Out}_{\text{Env}}(X)}{\text{In}_{\text{Sys}}(X)}$$

$$\frac{\text{In}_{\text{Env}}(X) \quad K(X)}{\text{Out}_{\text{Env}}(\text{true})}$$

- $K(X)$ represents that environment knows term X
- last rule models comparisons by the adversary
- Each specification yields a labeled transition system
- Observational equivalence is a kind of **bisimulation** accounting for the adversaries' **viewpoint** and **capabilities**
 - Our definition can be instantiated for various adversaries

Diff terms

- General definition **difficult** to verify: requires inventing simulation relation
- Idea: **specialize** for cryptographic protocols
 - Consider strong bid secrecy:
 - both systems differ in **secret bid only**, i.e.
 - both specifications contain **same rule(s)** which differ only in **some terms**
 - Exploit this similarity in description and proof
- Approach: two systems described by one **specification** – using **diff**-terms
 - Running example

$$\overline{\text{Out}_{\text{Sys}}(A)}$$
$$\overline{\text{Out}_{\text{Sys}}(B)}$$

- Is equivalent to one rule with a **diff**-term

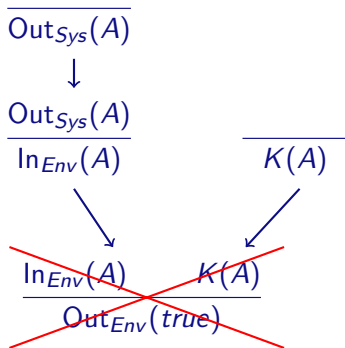
$$\overline{\text{Out}_{\text{Sys}}(\mathbf{diff}(A, B))}$$

Approximating observational equivalence using mirroring

- Both systems contain the same rules modulo diff-terms
- Idea: assume that each rule simulates itself
- **Mirrors** each execution into the other system
- If the mirrors are **valid executions**, we have **observational equivalence** (sound approximation)
- We represent executions using *dependency graphs*
 - Computed via backwards constraint solving

Dependency graphs and mirrors

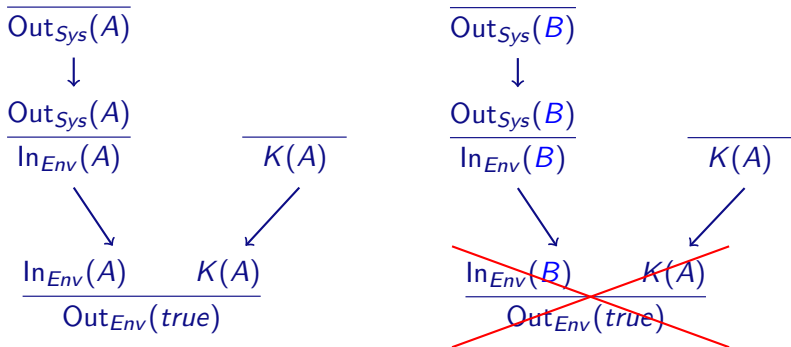
Bidder picks A, observer compares to public value A



- Dependency graph mirror for bidder choice B is **invalid**
 - Adversary choices stay fixed, comparison is with A

Invalid mirrors and attacks

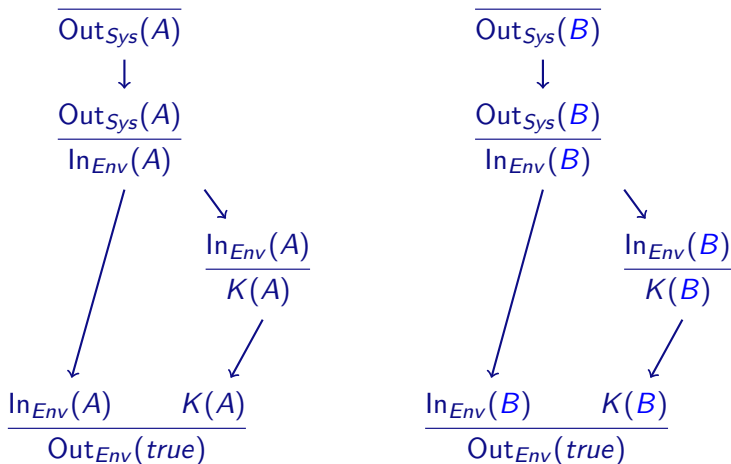
Bidder picks A/B, observer compares to public value A



- **Counter example** to observational equivalence of the given systems

Valid mirror

Observer compares system output to itself



- **All** mirrors need to be valid for observational equivalence

Dependency graph equivalence

A **diff**-system is *dependency graph equivalent* if mirrors of all dependency graphs rooted in any rule on both sides are valid.

- Sound but incomplete approximation
- Efficient and sufficient in practice

Input:

- Protocol specification
- Property: equivalence given two choices for some term(s)
 - Example: random value vs expected value

Output:

- **Yes**, observational equivalent
- **No**, dependency graph with invalid mirror
- Non-termination possible

Probabilistic encryption

Given equational theory for decryption

$$pdec(penc(m, pk(k), r), k) \simeq m.$$

- 1 Agent knows k , published $pk(k)$ previously
- 2 Adversary provides value x , agent selects random r_1, r_2
Agent sends either one of
 - r_1
 - $penc(x, pk(k), r_2)$

Can adversary distinguish random value from encryption?

- No!
- Tamarin verifies this automatically in 0.2 seconds
- Iterates over all rules
- Simple, illustrative toy example to show approach

Weak Secrecy

```
theory probenc
begin
functions: penc/3, pdec/2, pk/1
equations: pdec(penc(m, pk(k), r), k)=m

rule R_pk:
[ Fr(~k) ] --> [ !Ltk($A,~k), !Pk($A, pk(~k)) ]

rule Out_pk:
[ !Pk($A, pubk) ] --> [ Out(pubk) ]

rule Send_:
[ !Ltk($A, k), Fr(~x), Fr(~r2) ]
--[ WSecret(~x) ]->
[ Out(penc(~x , pk(k), ~r2) ) ]

//Weak secrecy
lemma Secret:
"
  All B #i. WSecret(B)@i ==> not( Ex #j. K(B)@j )
"
end
```

Strong Secrecy

```
theory probenc
begin
functions: penc/3, pdec/2, pk/1
equations: pdec(penc(m, pk(k), r), k)=m

rule R_pk:
[ Fr(~k) ] --> [ !Ltk($A,~k), !Pk($A, pk(~k)) ]

rule Out_pk:
[ !Pk($A, pubk) ] --> [ Out(pubk) ]

rule Send_:
[ !Ltk($A, k), In(x), Fr(~r1), Fr(~r2) ]
--[ SSecret(x) ]->
[ Out(diff( ~r1, penc(x , pk(k), ~r2) )) ]

//Strong secrecy implicitly encoded by the equivalence of two systems
```

Probabilistic encryption - reasoning

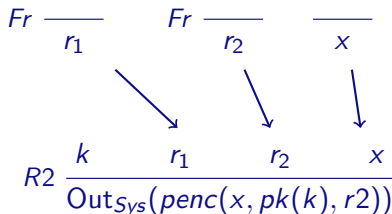
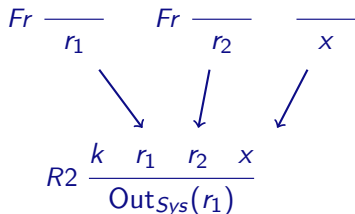
Setup rule (1) that picks long-term key and sends public key:

- No choice in the rule
- Easy to complete dependency graph
- Trivially mirrors itself

$$\begin{array}{c} Fr \text{ ---} \\ | \\ k \\ | \\ \downarrow \\ | \\ k \\ \hline \text{Out}_{\text{Sys}}(pk(k)) \end{array}$$

Probabilistic encryption - reasoning

Rule (2) sending out random or real encryption, taking value x as input and having a stored key k :



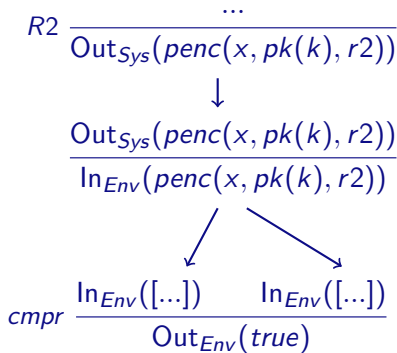
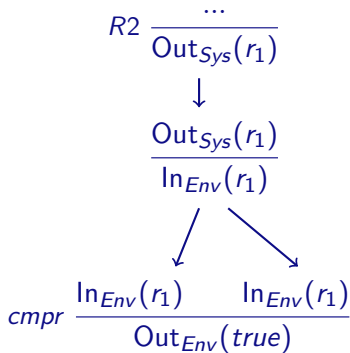
- Same premise in both cases
- Outputs do not matter in root
 - Considered in equality check
- Identical shape of dependency graphs in both cases:
 - Input and key are independent
- Mirror each other

Equality rule: considers two derivations to get some value x

- Both x adversary-generated: trivially mirrors
- x as result of (1)
 - key k is fresh, never known to adversary
 - only derive same x from same source
 - mirrors itself
- x as result of (2)
 - k, r_1, r_2 fresh, not known to adversary
 - only derive same x from same source
 - mirrored - next slide

Probabilistic encryption - reasoning

Truncated graphs shown:



Added new message deduction rule:

$$IEquality : K^\downarrow(x), K^\uparrow(x) \dashv\vdash \square \dashv\vdash \square .$$

Allows to test whether an equality holds.

- If a side can construct the same value twice this rule is applicable
- Ensures that the same equalities hold on both sides

Algorithm

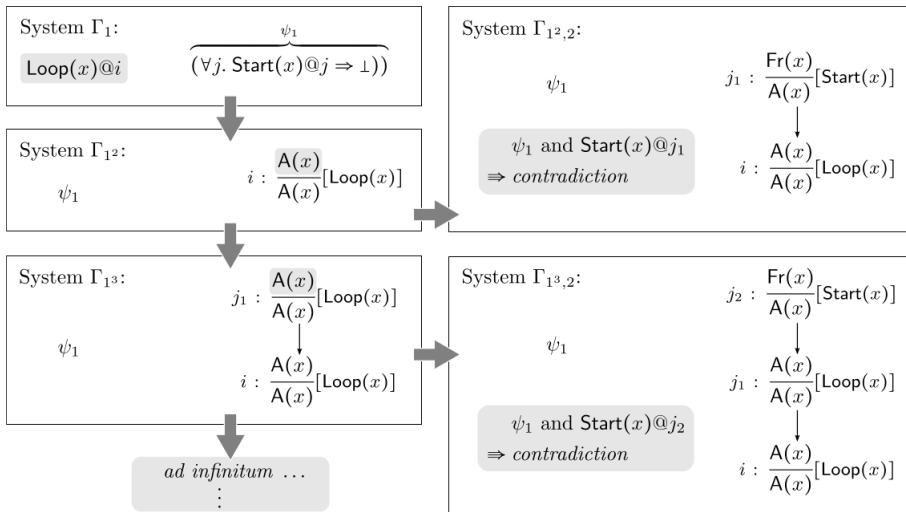
```
1: function VERIFY( $S$ )
2:    $RU \leftarrow L(S) \cup R(S) \cup IF \cup Env$ 
3:   while  $RU \neq \emptyset$  do
4:     choose  $r \in RU$ ,  $RU \leftarrow (RU \setminus \{r\})$ 
5:     compute  $DG \leftarrow dgraphs(r)$  by constraint solving
6:     if  $\exists dg \in DG$  s.t. mirrors( $dg$ ) lacks ground instances
7:       then return "potential attack found: ",  $dg$ 
8:   return "verification successful"
```

Induction Example

$$R_{loop} := \left\{ \frac{Fr(x)}{A(x)}[\text{Start}(x)], \frac{A(x)}{A(x)}[\text{Loop}(x)] \right\}$$

- Proof goal: $\forall x \#i. \text{Loop}(x)@i \Rightarrow \exists \#j. \text{Start}(x)@j$
- Such properties are needed:
 - looping constructs (counters etc.)
 - “Sources” lemmas
- Normal constraint solving does not work

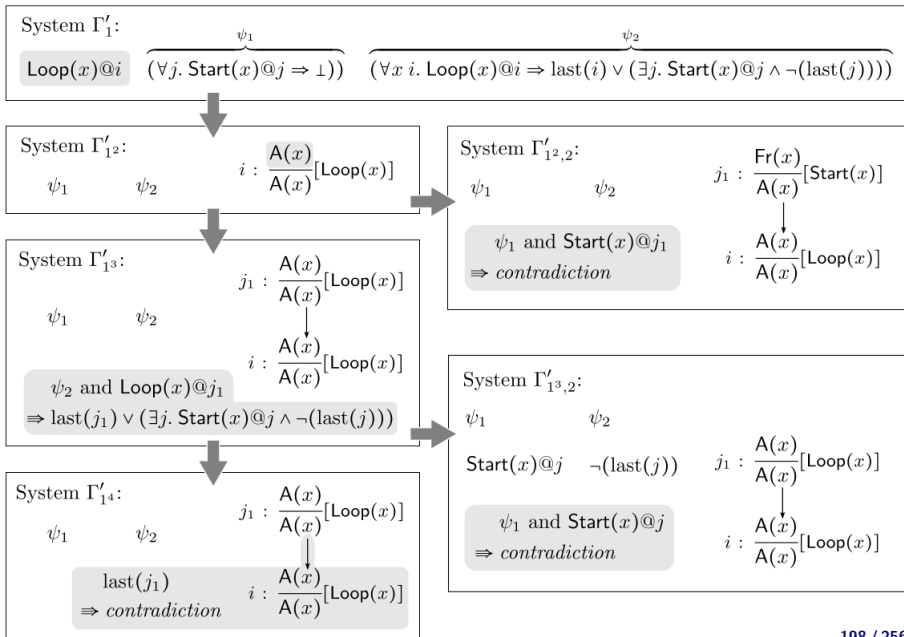
Why standard constraint solving fails



Induction on time points

- Intuitively, induction could help
- Formally, for IH ϕ :
 - Check if ϕ holds for empty trace
 - Consider special last rule index on trace
Assume ϕ holds at all non-last indices, and prove for last
- Added constraint reduction rules for last atoms
- Allows proof of previous example

Inductive proof



Induction on time points

- Required for all "sources" lemmas
- Helps for all looping constructs, used in e.g.:
 - YubiKey
 - TPM
 - PKCS11
 - Group protocols
 - Counters
- Can be activated using lemma annotations or in the interactive mode:
`lemma counter [use_induction]: ...`

Applicable Proof Methods: Goals sorted according to the 'smart' heuristic (loop breakers delayed)

1. **simplify**

2. **induction**

a. **autoprove** (A. **for all solutions**)

b. **autoprove** (B. **for all solutions**) with proof-depth bound 5

Automated Verification and Decidability

We would like to have a program V with ...

- Input:
 - some description of a program P
 - some description of a functional specification S
- Output: *Yes* if P satisfies S , and *No* otherwise.
- Optional extra: in the *No* case, give a counter-example, i.e. an input on which P violates the specification.

Forget it:

Theorem (Rice)

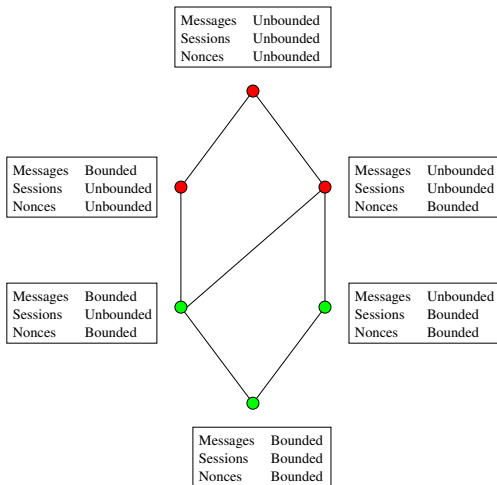
Let S be any non-empty, proper subset of the computable functions. Then the verification problem for S (the set of programs P that compute a function in S) is undecidable.

The Sources of Infinity

For security protocols, the **state space** can be infinite for (at least) the following reasons:

- Messages** The intruder can compose arbitrarily complex messages from his knowledge, e.g., i , $h(i)$, $h(h(i))$, \dots
- Sessions** Any number of sessions (or threads) may be executed.
- Nonces** Unbounded number of fresh nonces generated.

(Un)decidability: Complete picture



Bottom line: need at least two bounded parameters for decidability.

Bibliography

- David Basin, Sebastian Mödersheim, and Luca Viganò. *OFMC: A symbolic model checker for security protocols*. International Journal of Information Security, 4(3), 2005.
- Hubert Comon, Véronique Cortier, John Mitchell. *Tree automata with one memory, set constraints, and ping-pong protocols*. ICALP 2001.
- Shimon Even and Oded Goldreich. *On the security of multi-party ping-pong protocols*, Symposium on Foundations of Computer Science, IEEE Computer Society, 1983.
- N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. *Undecidability of bounded security protocols*. In Workshop on Formal Methods and Security Protocols (FMSP '99), 1999.
- J. Millen and V. Shmatikov. *Constraint Solving for Bounded-Process Cryptographic Protocol Analysis*. CCS 2001.
- Michaël Rusinowitch and Mathieu Turuani. *Protocol Insecurity with Finite Number of Sessions is NP-complete*. CSFW, 2001.

Precomputation

- **Idea:** for all facts in rule premises precompute their *possible sources* (improves efficiency)
- Sources are (combinations of) rules yielding that fact as (part of the) result
- Initial precomputations are called *raw sources*
- Sometimes these precomputations are incomplete, and give *partial deconstructions*
- GUI shows both raw and *refined sources*:

theory sources begin

Message theory

Multiset rewriting rules (5)

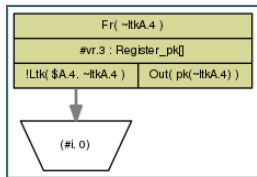
Raw sources (8 cases, 6 partial deconstructions left)

Refined sources (8 cases, deconstructions complete)

Examples of sources

Sources of "!Ltk(t.1, t.2) ►₀ #i" (1 cases)

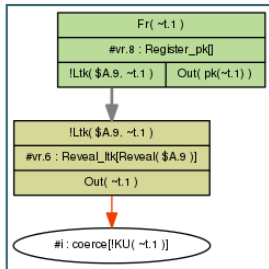
Source 1 of 1 / named "Register_pk"



"!Ltk(t.1, t.2) ►₀ #i"

Sources of "!KU(~t.1) @ #i" (6 cases)

Source 1 of 6 / named "Reveal_ltk"



"!KU(~t.1) @ #i"

Partial deconstructions: example protocol

1: $I \rightarrow R: \{ni, I\}_{pk(R)}$

2: $R \rightarrow I: \{ni\}_{pk(I)}$

rule I_1:

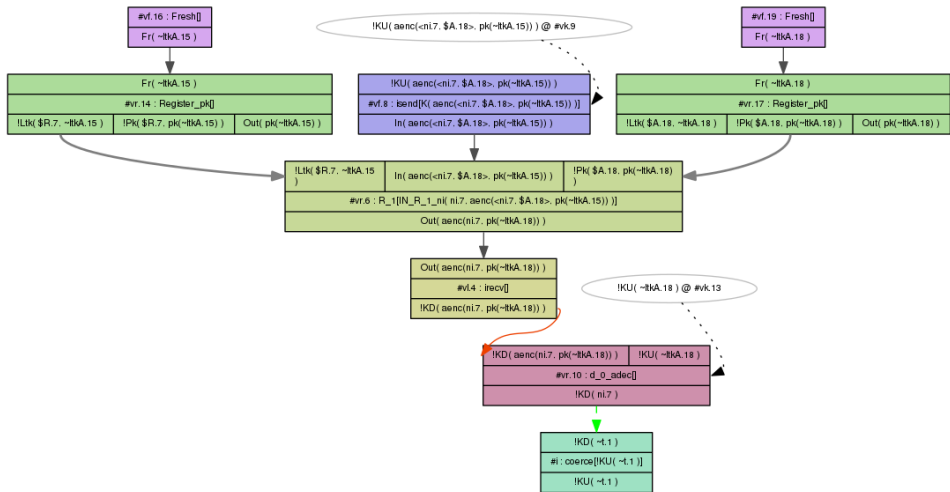
```
let m1 = aenc{~ni, $I}pkR
in
[ Fr(~ni) , !Pk($R, pkR) ]
--[ OUT_I_1(m1) ]->
[ Out( m1 ) ]
```

rule R_1:

```
let m1 = aenc{ni, I}pk(ltkR)
    m2 = aenc{ni}pkI
in
[ !Ltk($R, ltkR) , In( m1 ) , !Pk(I, pkI) ]
--[ IN_R_1_ni( ni, m1 ) ]->
[ Out( m2 ) ]
```

(This looks like a decryption oracle for values ni .)

Partial deconstruction: example



Partial deconstructions - issues

- Source for “everything”
- Complicates proofs and can easily cause non-termination
- Need to resolve such partial deconstructions
- *Realization*: result not “everything”, but only values actually sent by legitimate party or adversary-generated terms
- Claim (and then prove) this using *sources lemma*

Example:

```
lemma types [sources]:
```

```
"(All ni m1 #i.
```

```
  IN_R_1_ni( ni, m1) @ i
```

```
==>
```

```
( (Ex #j. K(ni) @ j & j < i)
```

```
| (Ex #j. OUT_I_1( m1 ) @ j) ) ) "
```


Explain where terms can come from or what their form must be.

How Tamarin proceeds:

1. Determine possible sources (raw)
2. Apply sources lemma to raw sources to get refined sources
3. Prove sources lemma with respect to raw sources
4. Prove other lemmas with respect to refined sources

Problem: What to do if Tamarin does not terminate?

- Use interactive mode to inspect the proof (use bounded autoprover!)
- If you find a loop, try to avoid it:
 - Find an intermediate helping lemma:
`lemma help [reuse]: ...`
 - Tweak your model: add or remove details, ...
- To find a known attack: use BFS



- Getting security protocols is not easy
- Symbolic analysis can help
- Tamarin has an interesting set of features and applications, many of whom I did not talk about (Human interaction security protocols, user-defined heuristics, parallelization, preprocessors, SAPIC, ...)
- ... but is still under development!

(-: Happy Proving :-)

Proofs of protocols : the symbolic model

- The **Dolev-Yao model** was proposed in 1983:
 - messages are terms in a free algebra: (m_1, m_2) , $\{m\}_k$, $[m]_k$, $h(m)$...
 - no secret value (nonce or key) can be guessed
 - the intruder can only apply functions in the given signature
 - ... but has complete control of the network
 - choice in semantics : non - deterministic
- One can add equations between primitives (functions), but anyway, we assume that the only equalities are those given by these equations.
- Proofs in this model are simpler and can be automated.

Proofs of protocols : the computational model

- The **computational model** was proposed in the '80s:
 - the messages are *bitstrings*
 - the cryptographic primitives (encryption, decryption, key-generation, signing..) are *probabilistic polynomial-time algorithms on bitstrings*
 - secrets can be guessed ... with *very small* probability
 - the intruder is any *interactive probabilistic polynomial-time algorithm*
 - has complete control of the network
- The model is much closer to reality, but proofs in this model are mostly manual.

- Computational world:
Done by hand, long and often complex and hard to understand. (Sometime proofs are error-prone). Often based on reduction to cryptographic assumptions.
- Symbolic world:
Based on constraint solving, First-order Logic, Tree automata etc ... these proofs are automatic.

Exemple: asymmetric encryption

In the symbolic world:

$$(AD) \quad \frac{T_0 \vdash \{u\}_{pk(v)} \quad T_0 \vdash sk(v)}{T_0 \vdash u}$$

$$(AC) \quad \frac{T_0 \vdash u \quad T_0 \vdash pk(v)}{T_0 \vdash \{u\}_{pk(v)}}$$

In the real world: ElGamal Encryption:

- a cyclic group G of order q and generator g (g and q depend on some integer η - the security parameter)
- $\mathcal{K}_\eta() = x \leftarrow^R \mathbb{Z}_q; sk \leftarrow x; pk \leftarrow g^x; return(sk, pk)$
- $\{m\}_{pk} = r \leftarrow^R \mathbb{Z}_q; return(pk^r \times g^m, g^r)$.

But what can we assume about this primitive?

IND-CPA encryption

After having chosen two plaintexts m_0 and m_1 , upon receiving the encryption of m_b (for a random bit b), it should be hard to guess which message has been encrypted:

$$\begin{aligned} Adv_{\eta}^{IND-CPA}(\mathcal{A}) &= Pr[b' = 1 | (sk, pk) \leftarrow^R \mathcal{K}_{\eta}(); (m_0, m_1) \leftarrow^R \mathcal{A}(pk); \\ &\quad c = \{m_1\}_{pk}; b' \leftarrow^R \mathcal{A}(c)] \\ &\quad - Pr[b' = 1 | (sk, pk) \leftarrow^R \mathcal{K}_{\eta}(); (m_0, m_1) \leftarrow^R \mathcal{A}(pk); \\ &\quad c = \{m_0\}_{pk}; b' \leftarrow^R \mathcal{A}(c)] \end{aligned}$$

The encryption is IND-CPA, if the advantage $Adv_{\eta}^{IND-CPA}(\mathcal{A})$ is negligible for any ppt-adversary \mathcal{A} .

Let us take it slowly...

Security parameter

In practice, we use keys, nonces, messages... they all have some length. Also, the algorithms we use, take some time to execute. So, the **security parameter**, denoted η , is an integer such that:

- the key/plaintext length in asymmetric encryption and signing can be η
- the key/block length in block ciphers/symmetric encryption can be η
- the key/tag length in MACs or output length in hash functions can be η
- the execution time of honest participants and of the intruder is **polynomial** in η

Computational indistinguishability - definition

- Let $\mathcal{D}^0 = \{\mathcal{D}_\eta^0\}_{\eta \in \mathbb{N}}$ and $\mathcal{D}^1 = \{\mathcal{D}_\eta^1\}_{\eta \in \mathbb{N}}$ be two (families of) distributions over bit-strings. The advantage $Adv^{\mathcal{D}^0, \mathcal{D}^1}(\mathcal{A})$ of an adversary \mathcal{A} is defined by

$$Adv^{\mathcal{D}^0, \mathcal{D}^1}(\mathcal{A}) = Pr[b' = 1 | d \leftarrow^R \mathcal{D}_\eta^0; b' \leftarrow^R \mathcal{A}(\eta, d)] \\ - Pr[b' = 1 | d \leftarrow^R \mathcal{D}_\eta^1; b' \leftarrow^R \mathcal{A}(\eta, d)]$$

The distributions \mathcal{D}^0 and \mathcal{D}^1 are **computationally indistinguishable** (denoted $\mathcal{D}^0 \approx \mathcal{D}^1$), if the advantage $Adv^{\mathcal{D}^0, \mathcal{D}^1}(\mathcal{A})$ is negligible for any ppt-algorithm \mathcal{A} .

- $f : \mathbb{N} \rightarrow \mathbb{R}$ is **negligible** if $\forall n. \exists N_n. \forall \eta \geq N_n. f(\eta) \leq \frac{1}{\eta^n}$
(for all polynomials p , $\lim_{\eta \rightarrow \infty} f(\eta) \times p(\eta) = 0$)

Computational indistinguishability - examples

- statistical indistinguishability implies computational indistinguishability

$$[d \leftarrow^R \{0, 1\}^\eta : d] \approx [d \leftarrow^R \{0, 1\}^\eta; \text{if } d = 0^\eta \text{ then } d = 1^\eta : d]$$

- Decisional Diffie Hellman Problem is (assumed) difficult.
Let G a cyclic group of order q and generator g (g and q depend on η). Then

$$[x, y \leftarrow^R \mathbb{Z}_q : (g^x, g^y, g^{x \times y})] \approx [x, y, z \leftarrow^R \mathbb{Z}_q : (g^x, g^y, g^z)]$$

We can prove that ElGamal encryption is IND-CPA secure, assuming that the Decisional Diffie Hellman Problem is hard.

Computational indistinguishability : transitivity

Theorem

If $\mathcal{D}^0 \approx \mathcal{D}^1$ and $\mathcal{D}^1 \approx \mathcal{D}^2$ then $\mathcal{D}^0 \approx \mathcal{D}^2$.

Proof.

- Suppose that $\mathcal{D}^0 \not\approx \mathcal{D}^2$, and let \mathcal{A} be a ppt-adversary that can distinguish \mathcal{D}^0 and \mathcal{D}^2 with non-negligible advantage.
- For $i \in \{0, 1, 2\}$, let $p_\eta^i = \Pr[b' = 1 | d \leftarrow^R \mathcal{D}_\eta^i; b' \leftarrow^R \mathcal{A}(\eta, d)]$.
- There is a polynomial q , such that for infinitely many η , $p_\eta^0 - p_\eta^2 \geq 1/q(\eta)$.
- Then, either $p_\eta^0 - p_\eta^1 \geq 1/(2 * q(\eta))$ or $p_\eta^1 - p_\eta^2 \geq 1/(2 * q(\eta))$ for infinitely many η .
- Hence \mathcal{A} distinguishes either \mathcal{D}^0 and \mathcal{D}^1 , or \mathcal{D}^1 and \mathcal{D}^2 with non-negligible advantage.

□

Computational indistinguishability : proof by reduction

Definition

A family of distributions \mathcal{E} is **polynomial-time constructible**, if there is a ppt-algorithm $\Psi_{\mathcal{E}}$, such that the output of $\Psi_{\mathcal{E}}(\eta)$ is distributed identically to \mathcal{E}_{η} .

Definition

Given two families of distributions \mathcal{D} and \mathcal{E} , we define $\mathcal{D} \parallel \mathcal{E}$ by

$$(\mathcal{D} \parallel \mathcal{E})_{\eta} = [x \leftarrow^R \mathcal{D}_{\eta}; y \leftarrow^R \mathcal{E}_{\eta} : (x, y)]$$

Theorem

If $\mathcal{D}^0 \approx \mathcal{D}^1$ and \mathcal{E} is polynomial-time constructible, then $(\mathcal{D}^0 \parallel \mathcal{E}) \approx (\mathcal{D}^1 \parallel \mathcal{E})$.

Computational indistinguishability : proof by reduction

Theorem. If $\mathcal{D}^0 \approx \mathcal{D}^1$ and \mathcal{E} is polynomial-time constructible, then $(\mathcal{D}^0 \parallel \mathcal{E}) \approx (\mathcal{D}^1 \parallel \mathcal{E})$.

Proof.

- Suppose that $(\mathcal{D}^0 \parallel \mathcal{E}) \not\approx (\mathcal{D}^1 \parallel \mathcal{E})$, and let \mathcal{A} be a ppt-adversary that can distinguish $(\mathcal{D}^0 \parallel \mathcal{E})$ and $(\mathcal{D}^1 \parallel \mathcal{E})$ with non-negligible advantage.
- Define an adversary B by

$$B(\eta, x) = [y \leftarrow^R \Psi_{\mathcal{E}}(\eta); b' \leftarrow^R \mathcal{A}(\eta, (x, y)) : b']$$

- We can see that if x is distributed according to \mathcal{D}_{η}^i , then the argument of \mathcal{A} is distributed according to $(\mathcal{D}^i \parallel \mathcal{E})_{\eta}$.
- Hence the advantage of B is equal to the advantage of \mathcal{A} .

□

Computational indistinguishability : hybrid argument

Definition

Let \mathcal{D} be a family of distributions, and p be a positive polynomial. We define the family of distributions $\vec{\mathcal{D}}$ by

$$\vec{\mathcal{D}}_\eta = [x_1 \leftarrow^R \mathcal{D}_\eta; \dots; x_{p(\eta)} \leftarrow^R \mathcal{D}_\eta : (x_1, \dots, x_{p(\eta)})]$$

To sample $\vec{\mathcal{D}}_\eta$, sample \mathcal{D}_η $p(\eta)$ times, and construct the tuple of sampled values.

Theorem

If $\vec{\mathcal{D}}^0 \approx \vec{\mathcal{D}}^1$ then $\mathcal{D}^0 \approx \mathcal{D}^1$.

Theorem

If $\mathcal{D}^0 \approx \mathcal{D}^1$ and \mathcal{D}^i are polynomial-time constructible, then $\vec{\mathcal{D}}^0 \approx \vec{\mathcal{D}}^1$.

Computational indistinguishability : hybrid argument 1

If $\vec{\mathcal{D}}^0 \approx \vec{\mathcal{D}}^1$ then $\mathcal{D}^0 \approx \mathcal{D}^1$.

Theorem.

Proof.

- Suppose that $\mathcal{D}^0 \not\approx \mathcal{D}^1$, and let \mathcal{A} be a ppt-adversary that can distinguish \mathcal{D}^0 and \mathcal{D}^1 with non-negligible advantage.
- Define an adversary B by

$$B(\eta, (x_1, \dots, x_{p(\eta)})) = [b' \leftarrow^R \mathcal{A}(\eta, x_1) : b']$$

- We can see that if $(x_1, \dots, x_{p(\eta)})$ is distributed according to $\vec{\mathcal{D}}^i_\eta$, then the argument x_1 of \mathcal{A} is distributed according to \mathcal{D}^i_η .
- Hence the advantage of B is equal to the advantage of \mathcal{A} .



Computational indistinguishability : hybrid argument 2

Theorem. If $\mathcal{D}^0 \approx \mathcal{D}^1$ and \mathcal{D}^i are polynomial-time constructible, then $\overrightarrow{\mathcal{D}^0} \approx \overrightarrow{\mathcal{D}^1}$.

Proof.

- Suppose that $\overrightarrow{\mathcal{D}^0} \not\approx \overrightarrow{\mathcal{D}^1}$, and let \mathcal{A} be a ppt-adversary that can distinguish $\overrightarrow{\mathcal{D}^0}$ and $\overrightarrow{\mathcal{D}^1}$ with non-negligible advantage.
- Hence for some polynomial q , and for infinitely many η ,
 $Pr[\mathcal{A}(\eta, x) = 1 | x \leftarrow^R \overrightarrow{\mathcal{D}^0}_\eta] - Pr[\mathcal{A}(\eta, x) = 1 | x \leftarrow^R \overrightarrow{\mathcal{D}^1}_\eta] \geq 1/q(\eta)$.
- Assume for now that p is a constant.
- For $k \in \{0, \dots, p\}$, define $\overrightarrow{\mathcal{E}^k}_\eta$ by
 $\overrightarrow{\mathcal{E}^k}_\eta = [x_1 \leftarrow^R \mathcal{D}^0_\eta; \dots; x_k \leftarrow^R \mathcal{D}^0_\eta; x_{k+1} \leftarrow^R \mathcal{D}^1_\eta; \dots; x_p \leftarrow^R \mathcal{D}^1_\eta : (x_1, \dots, x_p)]$
- Thus $\overrightarrow{\mathcal{E}^0}_\eta = \overrightarrow{\mathcal{D}^0}$ and $\overrightarrow{\mathcal{E}^p}_\eta = \overrightarrow{\mathcal{D}^1}$. Define $P^k_\eta = Pr[\mathcal{A}(\eta, x) = 1 | x \leftarrow^R \overrightarrow{\mathcal{E}^k}_\eta]$.
- Then for infinitely many η , $1/q(\eta) \leq P^p_\eta - P^0_\eta = \sum_{i=1}^p (P^i_\eta - P^{i-1}_\eta)$.
- Then for some j and infinitely many η , $(P^j_\eta - P^{j-1}_\eta) \geq 1/(p \times q(\eta))$.
- Then for some j , $\overrightarrow{\mathcal{E}^{j-1}}_\eta \not\approx \overrightarrow{\mathcal{E}^j}_\eta$, and \mathcal{A} can distinguish them.

Computational indistinguishability : hybrid argument 3

Proof.

(Continuation 1)

- There is some j , such that $\overrightarrow{\mathcal{E}}^{j-1}_\eta \not\approx \overrightarrow{\mathcal{E}}^j_\eta$, and \mathcal{A} can distinguish them.
- Now if \mathcal{A} can distinguish $\overrightarrow{\mathcal{E}}^{j-1}_\eta$ and $\overrightarrow{\mathcal{E}}^j_\eta$, how can we construct B that can distinguish $\overrightarrow{\mathcal{D}}^0 = \overrightarrow{\mathcal{E}}^p_\eta$ and $\overrightarrow{\mathcal{D}}^1 = \overrightarrow{\mathcal{E}}^0_\eta$?
- Define B as follows. On input (η, x) :
 1. Let $x_1 \leftarrow^R \mathcal{D}_\eta^0; \dots; x_{j-1} \leftarrow^R \mathcal{D}_\eta^0$.
 2. Let $x_j := x$.
 3. Let $x_j \leftarrow^R \mathcal{D}_\eta^1; \dots; x_p \leftarrow^R \mathcal{D}_\eta^1$.
 4. Call $b' \leftarrow^R \mathcal{A}(\eta, (x_1, \dots, x_p))$ and return b' .
- The advantage of this distinguisher is at least $1/(p \times q(\eta))$.
- Remark: the above proof was not constructive.



Computational indistinguishability : being constructive

Proof.

(Continuation 2)

- Suppose that $\vec{\mathcal{D}}^0 \not\approx \vec{\mathcal{D}}^1$, and let \mathcal{A} be a ppt-adversary that can distinguish $\vec{\mathcal{D}}^0$ and $\vec{\mathcal{D}}^1$ with non-negligible advantage.
- Hence for some polynomial q , and for infinitely many η ,
 $Pr[\mathcal{A}(\eta, x) = 1 | x \leftarrow^R \vec{\mathcal{D}}^0_\eta] - Pr[\mathcal{A}(\eta, x) = 1 | x \leftarrow^R \vec{\mathcal{D}}^1_\eta] \geq 1/q(\eta)$.
- Define B as follows. On input (η, x) :
 0. Let $j \leftarrow^R \{1, \dots, p\}$.
 1. Let $x_1 \leftarrow^R \mathcal{D}^0_\eta; \dots; x_{j-1} \leftarrow^R \mathcal{D}^0_\eta$.
 2. Let $x_j := x$.
 3. Let $x_j \leftarrow^R \mathcal{D}^1_\eta; \dots; x_p \leftarrow^R \mathcal{D}^1_\eta$.
 4. Call $b' \leftarrow^R \mathcal{A}(\eta, (x_1, \dots, x_p))$ and return b' .
- The advantage of this distinguisher is at least $1/(p \times p \times q(\eta))$.

□

Computational indistinguishability : p depends on η

Proof.

(Continuation 3)

- Suppose that $\overrightarrow{\mathcal{D}}^0 \not\approx \overrightarrow{\mathcal{D}}^1$, and let \mathcal{A} be a ppt-adversary that can distinguish $\overrightarrow{\mathcal{D}}^0$ and $\overrightarrow{\mathcal{D}}^1$ with non-negligible advantage.
- Hence for some polynomial q , and for infinitely many η ,
 $Pr[\mathcal{A}(\eta, x) = 1 | x \leftarrow^R \overrightarrow{\mathcal{D}}^0_\eta] - Pr[\mathcal{A}(\eta, x) = 1 | x \leftarrow^R \overrightarrow{\mathcal{D}}^1_\eta] \geq 1/q(\eta)$.
- Define B as follows. On input (η, x) :
 0. Let $j \leftarrow^R \{1, \dots, p(\eta)\}$.
 1. Let $x_1 \leftarrow^R \mathcal{D}^0_\eta; \dots; x_{j-1} \leftarrow^R \mathcal{D}^0_\eta$.
 2. Let $x_j := x$.
 3. Let $x_j \leftarrow^R \mathcal{D}^1_\eta; \dots; x_{p(\eta)} \leftarrow^R \mathcal{D}^1_\eta$.
 4. Call $b' \leftarrow^R \mathcal{A}(\eta, (x_1, \dots, x_{p(\eta)}))$ and return b' .
- The advantage of this distinguisher is at least $1/(p(\eta) \times p(\eta) \times q(\eta))$.

□

Back to the symbolic world

Messages are defined by the following grammar

$$\text{Keys} = \{k, k_1, k_2, \dots, \}$$
$$\text{Coins} = \{r, s, r_1, r_2, \dots, \}$$
$$\text{Bits} = \{0, 1\}$$
$$e ::= k \in \text{Keys}$$
$$| b \in \text{Bits}$$
$$| (e_1, e_2)$$
$$| \{e\}_k^r$$

- If $\{e\}_k^r$ and $\{e'\}_{k'}^{r'}$ both occur in the same context, then $e = e'$ and $k = k'$.

Recall: Dolev-Yao Deduction System

We denote $T_0 \vdash e$ for “message e is deducible from the set of messages T_0 ”.

Deduction System for Dolev Yao theory: $T_0 \vdash^? s$

$$(C0) \quad \frac{}{T_0 \vdash 0}$$

$$(A) \quad \frac{u \in T_0}{T_0 \vdash u}$$

$$(P) \quad \frac{T_0 \vdash u \quad T_0 \vdash v}{T_0 \vdash (u, v)}$$

$$(C) \quad \frac{T_0 \vdash u \quad T_0 \vdash v}{T_0 \vdash \{u\}_v^r}$$

$$(C1) \quad \frac{}{T_0 \vdash 1}$$

$$(UL) \quad \frac{T_0 \vdash (u, v)}{T_0 \vdash u}$$

$$(UR) \quad \frac{T_0 \vdash (u, v)}{T_0 \vdash v}$$

$$(D) \quad \frac{T_0 \vdash \{u\}_v^r \quad T_0 \vdash v}{T_0 \vdash u}$$

Symbolic world: meaning of a message

- Now we want to define an equivalence \sim between messages, such that $e_1 \sim e_2$ when e_1 and e_2 “look the same”. Intuitively, by doing any permitted operation, we cannot distinguish e_1 and e_2 .
- Examples:
 - $\{0\}_v^r \sim \{1\}_v^{r'}$.
 - $(\{0\}_v^r, v) \not\sim (\{1\}_v^{r'}, v)$.
 - $(\{0\}_v^r, (\{1\}_v^s)) \sim (\{1\}_v^{r'}, \{1\}_u^{s'})$.
- How can we define formally \sim ?

Key Recovery **rec** function

- We define **rec**(E) as being the set of keys that can be recovered from E using information available in E only: $\mathbf{rec}(E) = \{k \in \text{Keys} \mid E \vdash k\}$.
- Example:

$$\mathbf{rec}(\left(\left(\left(\left\{\left\{k_2\right\}_{k_1}^r\right\}_{k_1}^{r'}\right), \left\{\left(k_3, 0\right)\right\}_{k_2}^{r''}\right), k_1\right)) = \{k_1, k_2, k_3\}$$

- But we want a constructive definition of **rec**(E)!

Key Recovery Function in one pass $\mathbf{F}_{kr}(E, K)$

- Following the approach of [Micciancio,Warinschi], for an expression E and a set K of keys, we define Key Recovery Function $\mathbf{F}_{kr}(E, K)$ as follows:

$$\mathbf{F}_{kr}(b, K) = K$$

$$\mathbf{F}_{kr}(k, K) = \{k\} \cup K$$

$$\mathbf{F}_{kr}((E_1, E_2), K) = \mathbf{F}_{kr}(E_1, K) \cup \mathbf{F}_{kr}(E_2, K)$$

$$\mathbf{F}_{kr}(\{E\}_k, K) = \begin{cases} K & \text{if } k \notin K \\ \mathbf{F}_{kr}(E, K) & \text{otherwise.} \end{cases}$$

Example

$$E = (((\{k_2\}_{k_1}^r})_{k_1}^{r'}, \{(k_3, 0)\}_{k_2}^{r''}), k_1)$$

- $\mathbf{F}_{kr}(E, \emptyset) = \{k_1\}$
- $\mathbf{F}_{kr}(E, \{k_1\}) = \{k_1, k_2\}$
- $\mathbf{F}_{kr}(E, \{k_1, k_2\}) = \{k_1, k_2, k_3\}$
- $\mathbf{F}_{kr}(E, \{k_1, k_2, k_3\}) = \{k_1, k_2, k_3\}$

Inductive Definition of Key Recovery **rec**

- Let E be an expression, then we define **rec** by $\mathbf{rec}(E) = \bigcup_i G_i(E) = G_{|E|}(E)$ where

$$G_0(E) = \emptyset$$

$$G_i(E) = \mathbf{F}_{kr}(E, G_{i-1}(E)).$$

Example

$$E = ((\{\{k_2\}_{k_1}^r\}_{k_1}^{r'}, \{(k_3, 0)\}_{k_2}^{r''}), k_1)$$

- $G_1(E) = \mathbf{F}_{kr}(E, \emptyset) = \{k_1\}$
- $G_2(E) = \mathbf{F}_{kr}(E, \{k_1\}) = \{k_1, k_2\}$
- $G_3(E) = \mathbf{F}_{kr}(E, \{k_1, k_2\}) = \{k_1, k_2, k_3\}$
- $G_4(E) = \mathbf{F}_{kr}(E, \{k_1, k_2, k_3\}) = \{k_1, k_2, k_3\} = G_3(E) = \mathbf{rec}(E)$

Patterns

- *Patterns* are defined by the following grammar

Keys = $\{k, k_1, k_2, \dots, \}$

Coins = $\{r, s, r_1, r_2, \dots, \}$

Bits = $\{0, 1\}$

$P ::= k \in \text{Keys}$

| $b \in \text{Bits}$

| (P_1, P_2)

| $\{P\}_k^r$

| \square^r

- Remark: a message is a pattern with no occurrence of \square
- e.g., $\{(\{\square^r\}_{k_2}^s, 1)\}_{k_1}^{s'}$ is a pattern

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 $\text{PAT}(E, K)$ as follows:

$$\text{PAT}(b, K) = b$$

$$\text{PAT}(k, K) = k$$

$$\text{PAT}((E_1, E_2), K) = (\text{PAT}(E_1, K), \text{PAT}(E_2, K))$$

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

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

Examples

$$\mathbf{pat}(E) = \mathbf{PAT}(E, \mathbf{rec}(E))$$

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

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

$$\mathbf{pat}(E) = E$$

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

$$\mathbf{rec}(E) = \{k_2, k_3\}$$

$$\mathbf{pat}(E) = (\square^{r'}, \{k_3\}_{k_2}^{r''}, k_2)$$

Example (Exercise)

$$\begin{aligned} \text{pat}((\mathbf{0}, \mathbf{1})) &= ? \\ \text{pat}(\left(\left\{\mathbf{0}\right\}_{K_1}^r, \left(\left\{\mathbf{1}\right\}_{K_2}^{r'}, K_1\right)\right)) &= ? \\ \text{pat}(\left(\left\{\left\{K_1\right\}_{K_2}^{r'}\right\}_{K_3}^{r''}, K_3\right)) &= ? \end{aligned}$$

Pattern Equivalence

- We define E to be **symbolically equivalent** to F (and we write $E \sim F$) if there is a renaming (a bijection) σ_K of keys of F and a renaming σ_R of random coins of F such that $\mathbf{pat}(E) = \mathbf{pat}(F\sigma_K\sigma_R)$

Examples:

- $k' \sim k$
- $(k_1, k_2) \not\sim (k_3, k_3)$ (no bijection !)
- but $(k_1, k_2) \sim (k_4, k_1)$
- $(\{0\}_k^r, \{0\}_k^{r'}) \sim (\{0\}_k^{r'}, \{1\}_k^s)$
- but $(\{0\}_k^r, \{0\}_k^r) \not\sim (\{0\}_k^{r'}, \{1\}_k^s)$
- $\{((1, 1), (1, 1))\}_k^r \sim \{1\}_k^r$
- $(\{0\}_k^{r'}, \{0\}_k^s) \sim (\{0\}_{k'}^{r'}, \{0\}_{k''}^s)$

What is the meaning of a term in the computational world?

- $0, 1 ? \dots$ are constants, hence deterministic
- $((0, 1), 1) \dots$ using some deterministic concatenation
- $k \dots$ keys are randomly sampled
- $r \dots$ coins are randomly sampled
- $\{0\}_k^r \dots$ in general, encryption is probabilistic

Hence to a term, we will associate a **distribution**.

Implementation - primitives

- Let $\langle u, v \rangle : \{0, 1\}^* \times \{0, 1\}^* \mapsto \{0, 1\}^*$ be an easily computable and invertible injective function.
- Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme:
 - $\mathcal{K}(1^\eta)$: generates random key.
 - $\mathcal{E}^r(1^\eta, k, x)$: encrypts x with k using random coins r .
 - $\mathcal{D}(1^\eta, k, y)$: decrypts y with k .
 - Correctness:

$$\forall \eta, \forall x, \forall r$$

$$k := \mathcal{K}(1^\eta); y := \mathcal{E}^r(1^\eta, k, x); x' := \mathcal{D}(1^\eta, k, y) :$$

$$(x = x')?$$

Implementation of formal terms

Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme and $\eta \in \mathbb{N}$ be the security parameter. We associate to the formal term M a distribution $\llbracket M \rrbracket_\eta$ and by consequence a family of distributions $\llbracket M \rrbracket$.

- For each $k \in \text{Keys}(M)$ do $\tau_K(k) \stackrel{R}{\leftarrow} \mathcal{K}_\eta()$.
- For each $r \in \text{Coins}(M)$ do $\tau_R(r) \stackrel{R}{\leftarrow} \{0, 1\}^\omega$.
- $\llbracket k \rrbracket_\eta = \tau_K(k)$.
- For $b \in \{0, 1\}$, $\llbracket b \rrbracket_\eta = b$.
- $\llbracket (e_1, e_2) \rrbracket_\eta = \langle \llbracket e_1 \rrbracket_\eta, \llbracket e_2 \rrbracket_\eta \rangle$.
- $\llbracket \{e\}_k^r \rrbracket_\eta = \mathcal{E}^{\tau_R(r)}(1^\eta, \tau_K(k), \llbracket e \rrbracket_\eta)$.

Computational equivalence of terms

We say that two terms E and F are **computationally equivalent**, and we write $E \approx F$, when the associated distributions are computationally indistinguishable

$$\llbracket E \rrbracket \approx \llbracket F \rrbracket$$

Do we have that $E \sim F$ implies that $E \approx F$?

YES, if we avoid key cycle in terms, and if encryption is *type-0*-secure.

Encryption Cycles - examples:

- $\{k\}_k^r$
- $\{k_2\}_{k_1}^r, \{k_1\}_{k_2}^{r'}$
- $\{k_2\}_{k_1}^r, \{k_3\}_{k_2}^{r'}, \{k_1\}_{k_3}^{r''}$
- etc ...

Definition

Definition (Key Cycles)

Let $k, k' \in \text{Keys}$. We say that k encrypts k' in the term M , denoted by $k \succ_M k'$, if $\{N\}_k^r \in \text{st}(M)$ and $k' \in \text{st}(N)$.

A term M is **acyclic** iff the relation \succ_M is acyclic.

Example

- Let $M = (\{\{k_1\}_{k_2}^{r_1}\}_{k_3}^{r_2}, \mathbf{0})$. \succ_M is defined by $k_3 \succ_M k_2 \succ_M k_1$. M is acyclic.
- Let $M = \{k\}_k^r$. We get $k \succ_M k$. M has a cycle of size 1.
- Let $M = (\{k_1\}_{k_2}^{r_1}, \{k_2\}_{k_1}^{r_2})$. We have that $k_1 \succ_M k_2$ and $k_2 \succ_M k_1$. M has a cycle of size 2.

Type-0 Scheme - intuition

- A *type-0* encryption scheme has the following properties:
 - 1 message-hiding: given two messages m and m' of equal length and a cipher-text c of one of them, one cannot tell which of m or m' corresponds to c (IND-CPA)
 - 2 Which-key-hiding: given cipher-texts c, c' , one cannot tell if they were encrypted under the same key
 - 3 Message-length-hiding: given cipher-text c , one cannot determine the length of the corresponding plain-text

Type-0 Scheme - formal definition

Definition

Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme and \mathcal{A} be an adversary:

$$\begin{aligned} \text{Adv}_{\Pi, \eta}^0(\mathcal{A}) &= \Pr[\mathcal{A}^{\mathcal{O}_1^0(\cdot), \mathcal{O}_2^0(\cdot)}(\eta) = 1 | k \leftarrow^R \mathcal{K}_\eta()] \\ &\quad - \Pr[\mathcal{A}^{\mathcal{O}_1^1(\cdot), \mathcal{O}_2^1(\cdot)}(\eta) = 1 | k_1, k_2 \leftarrow^R \mathcal{K}_\eta()] \end{aligned}$$

Π is **Type-0 secure** if $\text{Adv}_{\Pi, \eta}^0(\mathcal{A})$ is negligible, for any ppt \mathcal{A} .

- $\mathcal{A}^{\mathcal{O}}$ is an adversary having access to a (or several) oracle(s) \mathcal{O} .
- $\mathcal{O}_1^1(\cdot)$ is an oracle that takes a message m and answers $c \leftarrow^R \mathcal{E}_{k_1}(m)$.
- $\mathcal{O}_2^1(\cdot)$ is an oracle that takes a message m and answers $c \leftarrow^R \mathcal{E}_{k_2}(m)$.
- $\mathcal{O}_1^0(\cdot)$ is an oracle that takes a message m and answers $c \leftarrow^R \mathcal{E}_k(0)$.
- $\mathcal{O}_2^0(\cdot)$ is an oracle that takes a message m and answers $c \leftarrow^R \mathcal{E}_k(0)$.

Theorem

Soundness If e_1, e_2 do not contain encryption cycles, and if encryption is implemented using a type-0 secure scheme, then

$$e_1 \sim e_2 \text{ implies } e_1 \approx e_2$$

Proof is by a hybrid reduction-style argument.

From now on we suppose that expressions do not contain encryption cycles, and that encryption is implemented using a type-0 secure scheme.

Implementation of formal patterns

- For each $k \in \text{Keys}(M)$ do $\tau_K(k) \stackrel{R}{\leftarrow} \mathcal{K}_\eta()$.
- $k_\square \stackrel{R}{\leftarrow} \mathcal{K}_\eta()$.
- For each $r \in \text{Coins}(M)$ do $\tau_R(r) \stackrel{R}{\leftarrow} \{0, 1\}^\omega$.
- $\llbracket k \rrbracket_\eta = \tau_K(k)$.
- For $b \in \{0, 1\}$, $\llbracket b \rrbracket_\eta = b$.
- $\llbracket (e_1, e_2) \rrbracket_\eta = \langle \llbracket e_1 \rrbracket_\eta, \llbracket e_2 \rrbracket_\eta \rangle$.
- $\llbracket \{e\}_k^r \rrbracket_\eta = \mathcal{E}^{\tau_R(r)}(1^\eta, \tau_K(k), \llbracket e \rrbracket_\eta)$.
- $\llbracket \square^r \rrbracket_\eta = \mathcal{E}^{\tau_R(r)}(1^\eta, k_\square, 0)$.

Replacing one key

For a key $k_0 \in \text{Keys}$, define

- $\text{replace}(k, k_0) = k$.
- $\text{replace}(b, k_0) = b$.
- $\text{replace}((e_1, e_2), k_0) = (\text{replace}(e_1, k_0), \text{replace}(e_2, k_0))$.
- $\text{replace}(\{e\}_k^r, k_0) = \{\text{replace}(e, k_0)\}_k^r$ if $k \neq k_0$.
- $\text{replace}(\{e\}_{k_0}^r, k_0) = \square^r$.
- $\text{replace}(\square^r, k_0) = \square^r$.

Replacing one key is sound

Theorem

Soundness Let e an expression, and k_0 a key occurring in e only as encryption key. Then $\llbracket e \rrbracket \approx \llbracket \text{replace}(e, k_0) \rrbracket$.

Proof.

Assume that B distinguishes $\llbracket e \rrbracket$ from $\llbracket \text{replace}(e, k_0) \rrbracket$, and let construct an adversary \mathcal{A}^O that wins against the security of the type-0 encryption scheme. $\mathcal{A}^O(1^\eta)$ works as follows:

- For each $k \in \text{Keys}(e) \setminus \{k_0\}$ do $\tau_K(k) \stackrel{R}{\leftarrow} \mathcal{K}_\eta()$.
- For each $r \in \text{Coins}(e)$ do $\tau_R(r) \stackrel{R}{\leftarrow} \{0, 1\}^\omega$.
- Let $L = \{\}$ (the empty mapping).
- Compute the “semantics” v of e by invoking $\text{Sem}^O(e)$.
- Return $B(1^\eta, v)$.



Replacing one key is sound

Theorem Let e an expression, and k_0 a key occurring in e only as encryption key. Then $\llbracket e \rrbracket \approx \llbracket \text{replace}(e, k_0) \rrbracket$.

Proof.

(Continuation)

$\text{Sem}^{\mathcal{O}}(e)$ is: case e of

- k : return $\tau_K(k)$ (note that $k \neq k_0$).
- b : return b .
- (e_1, e_2) : Let $v_i = \text{Sem}^{\mathcal{O}}(e_i)$; return $\langle v_1, v_2 \rangle$.
- \square^r : return $\mathcal{O}_2(0)$.
- $\{e\}_k^r$: Let $w = \text{Sem}^{\mathcal{O}}(e)$;
 - If $k \neq k_0$, then return $\mathcal{E}^{\tau_R(r)}(1^\eta, \tau_K(k), w)$.
 - If $k = k_0$ and $L(r)$ is not defined, then $L(r) = \mathcal{O}_1(w)$; return $L(r)$.

One can see that $\text{Sem}^{\mathcal{O}^1}(e) = \llbracket e \rrbracket$ and $\text{Sem}^{\mathcal{O}^0}(e) = \llbracket \text{replace}(e, k_0) \rrbracket$, hence \mathcal{A} distinguishes \mathcal{O}^0 from \mathcal{O}^1 , with the same advantage that \mathcal{B} distinguishes $\llbracket e \rrbracket$ from $\llbracket \text{replace}(e, k_0) \rrbracket$.

□

Soundness

Theorem If e_1, e_2 do not contain encryption cycles, and if encryption is implemented using a type-0 secure scheme, then

$$e_1 \sim e_2 \text{ implies } e_1 \approx e_2$$

Proof.

- If $\{k_1, \dots, k_n\} = \text{Keys}(e)$, and for all $i, j, i < j \implies k_j \not\prec_e k_i$, then
$$\text{replace}(\dots \text{replace}(\text{replace}(e, k_1), k_2) \dots, k_n) = \mathbf{pat}(e)$$
- Apply the previous lemma to each key in $\mathbf{hidden}(e)$, we get
$$\llbracket e \rrbracket \approx \llbracket \mathbf{pat}(e) \rrbracket$$
- Permuting the formal keys and coins does not change the generated probability distribution over bit-strings. If $e_1 \sim e_2$, then $\mathbf{pat}(e_1) = \mathbf{pat}(e_2 \sigma_K \sigma_R)$ for some permutations σ_K on keys and σ_R on coins, and by transitivity

$$\llbracket e_1 \rrbracket \approx \llbracket \mathbf{pat}(e_1) \rrbracket = \llbracket \mathbf{pat}(e_2 \sigma_K \sigma_R) \rrbracket = \llbracket \mathbf{pat}(e_2) \rrbracket \approx \llbracket e_2 \rrbracket,$$

hence $e_1 \approx e_2$. □

Example

$$\begin{aligned} & \llbracket ((\{k_4, 0\}_{k_3}^{r_1}, \{k_3\}_{k_2}^{r_2}), (\{\{11\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)) \rrbracket \\ & \quad \approx \\ & \llbracket ((\{k_4, 0\}_{k_3}^{r_1}, \square^{r_2}), (\{\{11\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)) \rrbracket \\ & \quad \approx \\ & \llbracket ((\square^{r_1}, \square^{r_2}), (\{\{11\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)) \rrbracket \\ & \quad \approx \\ & \llbracket ((\square^{r_1}, \square^{r_2}), (\{\square^{r_4}\}_{k_1}^{r_3}, k_1)) \rrbracket \\ & \quad \approx \\ & \llbracket ((\{1\}_{k_2}^{r_1}, \square^{r_2}), (\{\{0\}_{k_2}^{r_4}\}_{k_1}^{r_3}, k_1)) \rrbracket \\ & \quad \approx \\ & \llbracket ((\{1\}_{k_2}^{r_1}, \{k_2\}_{k_3}^{r_2}), (\{\{0\}_{k_2}^{r_4}\}_{k_1}^{r_3}, k_1)) \rrbracket \end{aligned}$$

- Introduced in [AR 00] M. Abadi et P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). Journal of Cryptology, 15(2):103–127, 2002.
- Models security of encryption against a *passive* computational adversary
- Adversary observes two *expressions* which have the same *pattern* assuming secure encryption – goal is to distinguish them with non-negligible probability

RESULT:

Symbolic indistinguishability implies computational security

(Some) Extensions

- [Micciancio, Warinschi] give completeness for modified (authenticated) encryption, and also soundness for active adversaries
- [Micciancio, Panjwani] give a soundness theorem for an adversary which can adaptively attack the encryption scheme
- [Backes, Pfitzmann, Waidner] give an cryptographic implementation of Dolev-Yao terms in a general (UC) setting
- [Canneti, Herzog] give a formal (automated) approach to universal composability