Cryptography

Corso di Laurea Magistrale in Informatica

The Symbolic Model

Ugo Dal Lago

Academic Year 2023-2024

Part I

From Strings to Expressions

The Computational Model

- So far, we have been working with the so-called **computational model**.

The Computational Model

- So far, we have been working with the so-called **computational model**.
- In the computational model:
  1. Cryptographic primitives and protocols, but also adversaries, are modelled using PPT **algorithms**;
  2. The probability of success of the adversaries must be a **negligible** function.

The Computational Model

- So far, we have been working with the so-called **computational model**.
- In the computational model:
  1. Cryptographic primitives and protocols, but also adversaries, are modelled using PPT **algorithms**;
  2. The probability of success of the adversaries must be a **negligible** function.
- This model is the reference model in a variety of frameworks, in particular those in which communication takes place following *relatively simple* protocols.
  - For example: public and private-key ciphers, MAC, digital signature, etc.

The Computational Model

- So far, we have been working with the so-called **computational model**.
- In the computational model:
  1. Cryptographic primitives and protocols, but also adversaries, are modelled using PPT **algorithms**;
  2. The probability of success of the adversaries must be a **negligible** function.
- This model is the reference model in a variety of frameworks, in particular those in which communication takes place following *relatively simple* protocols.
  - For example: public and private-key ciphers, MAC, digital signature, etc.
- The computational model is **sensitive** to the amount of resources used and to the probability of certain events occurring.

Computational Model's Limitations

- The limitations of the computational model are essentially twofold:
  1. Probabilistic reasoning becomes **difficult** as soon as the framework become even slightly more complex than those we are used to working with.
  2. When the number of parties involved increases, it is difficult to even understand **how to define** the concept of efficiency, which is central to the computational approach.

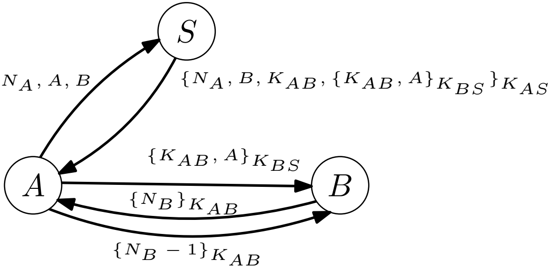
Computational Model's Limitations

- The limitations of the computational model are essentially twofold:
  1. Probabilistic reasoning becomes **difficult** as soon as the framework become even slightly more complex than those we are used to working with.
  2. When the number of parties involved increases, it is difficult to even understand **how to define** the concept of efficiency, which is central to the computational approach.
- When we consider more complex protocols, computational cryptography shows its limitations.
  - Think for example of: *electronic voting protocols*, *cryptocurrencies*, *electronic commerce*, etc.

Computational Model's Limitations

- The limitations of the computational model are essentially twofold:
  1. Probabilistic reasoning becomes **difficult** as soon as the framework become even slightly more complex than those we are used to working with.
  2. When the number of parties involved increases, it is difficult to even understand **how to define** the concept of efficiency, which is central to the computational approach.
- When we consider more complex protocols, computational cryptography shows its limitations.
  - Think for example of: *electronic voting protocols*, *cryptocurrencies*, *electronic commerce*, etc.
- Often the protocols we wish to prove to be secure have the following characteristics:
  1. **Multiple parties** involved.
  2. **Multiple rounds** of interaction.
  3. Cryptographic primitives (encryption, authentication) are used as **subroutines**.

An Example: the Needham-Schroeder Protocol



- The expression $\{M\}_K$ denotes the (symmetric-key) encryption of the message M using the key K.
- N_A and N_B are the so-called “nonces”, i.e. the random values generated by A and B respectively.

Two Attacks Against NS

1. If an attacker had an “old” session key K_{AB} at his disposal, he could replay message 3.
    - B would reply by generating a nonce N_B, but the attacker could decrypt the message and produce $N_B - 1$.
    - At this point, the attacker **impersonates** A.
2. Without B in the message 2, an attacker C could intercept message 1 and modify its second component, turning it into C.
    - From then on, he could **impersonate** B without A noticing.

Two Attacks Against NS

1. If an attacker had an “old” session key K_{AB} at his disposal, he could replay message 3.
    - B would reply by generating a nonce N_B, but the attacker could decrypt the message and produce $N_B - 1$.
    - At this point, the attacker **impersonates** A.
2. Without B in the message 2, an attacker C could intercept message 1 and modify its second component, turning it into C.
    - From then on, he could **impersonate** B without A noticing.

- In both cases, what makes the attack possible *is not* a weakness in the cryptographic primitives used, but a **logical** error in the construction of the protocol itself.
- Modelling the protocol in the sense of computational cryptography would be an *overkill*.

The Symbolic Model

- Independently of the computational model, an alternative model to the computational one was developed from the 1980s. To some extent, this model solves some of the problems just described.
  - This model is called **symbolic model** or **Dolev-Yao model**, named after the authors of a paper of 1983 in which the model was introduced.

The Symbolic Model

- Independently of the computational model, an alternative model to the computational one was developed from the 1980s. To some extent, this model solves some of the problems just described.
  - This model is called **symbolic model** or **Dolev-Yao model**, named after the authors of a paper of 1983 in which the model was introduced.
- In summary:

	Computational	**Formal**
Messages	Binary Strings	Expressions
Adversaries	Efficient Algorithms	Arbitrary Processes
Attacks	Non-Negligible Probability Event	Possible Event

Expressions

- The expressions of the symbolic model should not be confused with the bit strings of the computational model.
  - They are to be thought of as derivation trees and not as the related binary encoding.
  - The peculiarity with respect to the computational approach is that it is assumed that knowing an expression *does not necessarily* imply knowing its sub-expressions.

Expressions

- The expressions of the symbolic model should not be confused with the bit strings of the computational model.
  - They are to be thought of as derivation trees and not as the related binary encoding.
  - The peculiarity with respect to the computational approach is that it is assumed that knowing an expression *does not necessarily* imply knowing its sub-expressions.
- An expression could for example be $\{K_{AB}, A\}_{K_{BS}}$.
  - An adversary might know the expression.
  - But if he does not know K_{BS}, the expression appears to him as an inscrutable entity.
  - In that case *there is no chance* that the adversary succeeds in to reconstruct, for example, K_{AB}. This is the main difference with the computational model.

The Symbolic Model?

- Actually, **many** symbolic models exist, unlike what happens with the computational ones.
  - For example, it is possible to parameterise which *cryptographic primitives* are available.
  - The *underlying theory* can drastically change (becoming more or less interesting) when expressions change, for example, by adding or removing constraints on the expression' s construction.

The Symbolic Model?

- Actually, **many** symbolic models exist, unlike what happens with the computational ones.
  - For example, it is possible to parameterise which *cryptographic primitives* are available.
  - The *underlying theory* can drastically change (becoming more or less interesting) when expressions change, for example, by adding or removing constraints on the expression' s construction.
- What is common to all symbolic models (differently from computational ones) is their simplicity.
  - Protocols' security can be proved **without assumptions**.
  - Given a protocol, deciding *whether* an adversary exists is a problem that can also be faced with automatic techniques.
  - As a consequence, the symbolic model can form the basis for model-checking, semi-automated theorem proving, logic programming.
  - All this has led to the design of concrete tools (one of which, called `ProVerif`, will be discussed).

Part II

Multiset Rewriting as a Symbolic Model

Sorts, Function Symbols, Predicates

- **Sorts**
  - They represents the type of messages exchanged by the parties, which are now kept distinct from each other.
  - In the computational model, they all become strings!
  - *Examples*: key, message, nonce, cipher.
- **Function Symbols**
  - These are names for functions on the sorts, each of them a certain number of parameters
  - *Examples*:

$$\texttt{enc} : \mathsf{key} \times \mathsf{msg} \to \mathsf{cipher}$$
$$\texttt{dec} : \mathsf{key} \times \mathsf{cipher} \to \mathsf{msg}$$

- **Predicates**
  - These are names of *properties* of tuples of elements, each of a certain sort.
  - *Examples*:

$$\textsc{Knowledge} : \mathsf{cipher}$$
$$\textsc{KeyPair} : \mathsf{pubkey} \times \mathsf{privkey}$$

Terms, Facts

- A sets of sorts together with sets of function symbol and predicates on that sort form a *signature*.
  - Different protocols and situations may require distinct signatures.

Terms, Facts

- A sets of sorts together with sets of function symbol and predicates on that sort form a *signature*.
  - Different protocols and situations may require distinct signatures.
- **Terms**
  - They are expressions built from variables and function symbols, respecting the sorts.
  - They can be proved to have a sort in an *environment* assigning sorts to variables.
  - *Example*:

$$k : \mathsf{key}, m : \mathsf{msg} \vdash \mathtt{enc}(k, m) : \mathsf{cipher}$$

Terms, Facts

- A sets of sorts together with sets of function symbol and predicates on that sort form a *signature*.
  - Different protocols and situations may require distinct signatures.
- **Terms**
  - They are expressions built from variables and function symbols, respecting the sorts.
  - They can be proved to have a sort in an *environment* assigning sorts to variables.
  - *Example*:

$$k : \mathsf{key}, m : \mathsf{msg} \vdash \mathtt{enc}(k, m) : \mathsf{cipher}$$

- **Facts**
  - Consist of a predicate applied to expressions, each having the right sort.
  - *Example*:

$$k : \mathsf{key}, m : \mathsf{msg} \vdash \textsc{Knowledge}(\mathtt{enc}(k, m))$$

Rules

- In the multiset rewriting framework, the **state** of a primitive or protocol is a *finite multiset of facts*.
  - A multiset can be seen as a set in which each element can occur more than once.
  - Given n (not necessarily distinct) facts $A_1, \ldots, A_n$, the multiset which contains them is indicated simply as $A_1, \ldots, A_n$.
  - *Example*:

$$\textsc{Knowledge}(\mathtt{enc}(k, m)), \textsc{Knowledge}(k)$$

Rules

- In the multiset rewriting framework, the **state** of a primitive or protocol is a *finite multiset of facts*.
  - A multiset can be seen as a set in which each element can occur more than once.
  - Given n (not necessarily distinct) facts $A_1, \ldots, A_n$, the multiset which contains them is indicated simply as $A_1, \ldots, A_n$.
  - *Example*:

$$\textsc{Knowledge}(\mathtt{enc}(k, m)), \textsc{Knowledge}(k)$$

- The *dynamic evolution* of the underlying state is modelled by rules of the form

$$A_1, \ldots, A_n \rightarrow \exists x_1, \ldots x_m . B_1, \ldots, B_k$$

where $n, m, k \geq 0$, while the A_is and B_js are facts.
  - *Example*:

$$\textsc{Knowledge}(\mathtt{enc}(k, m)), \textsc{Knowledge}(k) \rightarrow \textsc{Knowledge}(m)$$

Rules

- In the multiset rewriting framework, the **state** of a primitive or protocol is a *finite multiset of facts*.
  - A multiset can be seen as a set in which each element can occur more than once.
  - Given n (not necessarily distinct) facts $A_1, \ldots, A_n$, the multiset which contains them is indicated simply as $A_1, \ldots, A_n$.
  - *Example*:

$$\textsc{Knowledge}(\mathtt{enc}(k, m)), \textsc{Knowledge}(k)$$

- The *dynamic evolution* of the underlying state is modelled by rules of the form

$$A_1, \ldots, A_n \rightarrow \exists x_1, \ldots x_m . B_1, \ldots, B_k$$

  where $n, m, k \geq 0$, while the A_is and B_js are facts.
  - *Example*:

$$\textsc{Knowledge}(\mathtt{enc}(k, m)), \textsc{Knowledge}(k) \rightarrow \textsc{Knowledge}(m)$$

- A signature and a set of rules over it forms a **theory**.

Traces

- Given a signature and a set of rules for it, we can look at what happens from an initial state in the form

$$A_1, \ldots, A_n$$

(where the A_is are facts) by repeatedly applying the rules, and forming a *trace* of execution.

Traces

- Given a signature and a set of rules for it, we can look at what happens from an initial state in the form

$$A_1, \ldots, A_n$$

  (where the A_is are facts) by repeatedly applying the rules, and forming a *trace* of execution.
- It is *not* necessary that the left-hand-side of a rule perfectly matches the current state:
  - It is sufficient that it matches is modulo a *substitution*.
  - The left-hand-side must match a sub-multiset of the current state, as there could well be facts which are not involved in the rule.

Traces

- Given a signature and a set of rules for it, we can look at what happens from an initial state in the form

$$A_1, \ldots, A_n$$

  (where the A_is are facts) by repeatedly applying the rules, and forming a *trace* of execution.
- It is *not* necessary that the left-hand-side of a rule perfectly matches the current state:
  - It is sufficient that it matches is modulo a *substitution*.
  - The left-hand-side must match a sub-multiset of the current state, as there could well be facts which are not involved in the rule.
- *Example*:

$$\begin{aligned} & \textsc{Knowledge}(\texttt{enc}(k_1, \texttt{enc}(k_2, m))), \textsc{Knowledge}(k_1), \textsc{Knowledge}(k_2) \\ \to\ & \textsc{Knowledge}(\texttt{enc}(k_2, m)), \textsc{Knowledge}(k_2) \\ \to\ & \textsc{Knowledge}(m). \end{aligned}$$

The Theory of a Finite Automaton

- **Sorts**:

$$\mathsf{state}, \mathsf{symb}, \mathsf{string}$$

- **Function Symbols**:

$$\texttt{cons} : \mathsf{symb} \times \mathsf{string} \to \mathsf{string} \qquad \texttt{q}_1, \ldots, \texttt{q}_n : \mathsf{state}$$

$$\texttt{a}_1, \ldots, \texttt{a}_m : \mathsf{sym} \qquad \texttt{nil} : \mathsf{string}$$

The Theory of a Finite Automaton

- **Sorts**:

$$\mathsf{state}, \mathsf{symb}, \mathsf{string}$$

- **Function Symbols**:

$$\mathtt{cons} : \mathsf{symb} \times \mathsf{string} \to \mathsf{string} \qquad \mathtt{q}_1, \ldots, \mathtt{q}_n : \mathsf{state}$$

$$\mathtt{a}_1, \ldots, \mathtt{a}_m : \mathsf{sym} \qquad \mathtt{nil} : \mathsf{string}$$

- **Predicates**:

$$\textsc{CurrentState} : \mathsf{state} \qquad \textsc{InputLeft} : \mathsf{string}$$

The Theory of a Finite Automaton

- **Sorts**:

$$\mathsf{state}, \mathsf{symb}, \mathsf{string}$$

- **Function Symbols**:

$$\mathtt{cons} : \mathsf{symb} \times \mathsf{string} \to \mathsf{string} \qquad \mathtt{q}_1, \ldots, \mathtt{q}_n : \mathsf{state}$$

$$\mathtt{a}_1, \ldots, \mathtt{a}_m : \mathsf{sym} \qquad \mathtt{nil} : \mathsf{string}$$

- **Predicates**:

$$\textsc{CurrentState} : \mathsf{state} \qquad \textsc{InputLeft} : \mathsf{string}$$

- **Rules**:

$$\begin{aligned} &\textsc{CurrentState}(\mathtt{q}_i), \textsc{InputLeft}(\mathtt{cons}(\mathtt{a}_j, x)) \\ \to\ &\textsc{CurrentState}(\mathtt{q}_k), \textsc{InputLeft}(x) \end{aligned}$$

whenever $\delta(q_i, a_j) = q_k$, and δ is the transition function of the finite automaton.

Turing Machines and Existential Quantification

- Turing Machines can be seen as infinitary variations on finite automata in which the number of states is not necessarily finite, thanks to a possibly infinite tape.

Turing Machines and Existential Quantification

- Turing Machines can be seen as infinitary variations on finite automata in which the number of states is not necessarily finite, thanks to a possibly infinite tape.
- We thus need *fresh values*, and existential quantification becomes essential.

Turing Machines and Existential Quantification

- Turing Machines can be seen as infinitary variations on finite automata in which the number of states is not necessarily finite, thanks to a possibly infinite tape.
- We thus need *fresh values*, and existential quantification becomes essential.
- We need an additional sort called cell modelling cells, and new predicates

$$\textsc{Content} : \mathsf{cell} \times \mathsf{symb} \qquad \textsc{Adjacency} : \mathsf{cell} \times \mathsf{cell}$$

  modeling cell content and position.

Turing Machines and Existential Quantification

- Turing Machines can be seen as infinitary variations on finite automata in which the number of states is not necessarily finite, thanks to a possibly infinite tape.
- We thus need *fresh values*, and existential quantification becomes essential.
- We need an additional sort called cell modelling cells, and new predicates

$$\textsc{Content} : \mathsf{cell} \times \mathsf{symb} \qquad \textsc{Adjacency} : \mathsf{cell} \times \mathsf{cell}$$

  modeling cell content and position.
- There is a rule which extends the tape further of one cell:

$$\begin{aligned} &\textsc{Adjacent}(x, \mathtt{c}_{eot}) \\ \longrightarrow\ &\exists y.\textsc{Adjacent}(x, y), \textsc{Content}(y, \mathtt{blank}), \textsc{Adjacent}(y, \mathtt{c}_{eot}) \end{aligned}$$

Safety Problems

- The **MSR safety problem** consists, given a theory, a set of initial facts X and set of bad facts Y, to determine whether *there exists* a trace leading from a fact in X to a fact in Y, namely a trace in the form

$$\mathbf{S}_1 \longrightarrow \mathbf{S}_2 \longrightarrow \ldots \longrightarrow \mathbf{S}_n$$

  where $\mathbf{S}_1 \in X$ and $\mathbf{S}_n \in Y$.

Safety Problems

- The **MSR safety problem** consists, given a theory, a set of initial facts X and set of bad facts Y, to determine whether *there exists* a trace leading from a fact in X to a fact in Y, namely a trace in the form

$$\mathbf{S}_1 \longrightarrow \mathbf{S}_2 \longrightarrow \ldots \longrightarrow \mathbf{S}_n$$

where $\mathbf{S}_1 \in X$ and $\mathbf{S}_n \in Y$.

Theorem

The MSR safety problem is undecidable.

Protocols as Theories

- Finite automata and Turing machines are *sequential* models of computation. Can we somehow model **concurrent** models of computation like security protocols as theories?

Protocols as Theories

- Finite automata and Turing machines are *sequential* models of computation. Can we somehow model **concurrent** models of computation like security protocols as theories?
- Here is a recipe:
  - For every **agent** X and for every phase $i \in \mathbb{N}$ in the execution of the protocol, there is a predicate X_i capturing the fact that X is in phase i, and that it known some data, seen as a parameter to X_i.
  - The exchange of data between the parties is mediated by the **network**, and this is captured by a predicate N_i, this way paving the way to the modeling of emphattackers.

Protocols as Theories: An Example

- Consider the following, very simple, protocol:

$$\begin{aligned} A &\longrightarrow B : N_A \\ B &\longrightarrow A : N_A, N_B \\ A &\longrightarrow B : N_B \end{aligned}$$

Protocols as Theories: An Example

- Consider the following, very simple, protocol:

$$\begin{array}{rcl} A & \longrightarrow & B : N_A \\ B & \longrightarrow & A : N_A, N_B \\ A & \longrightarrow & B : N_B \end{array}$$

- We can model it as a signature on a single sort n (for nonces):
  - **Predicates**:

$$\begin{array}{lll} \mathrm{A}_0 : 1 & \mathrm{A}_1 : \mathsf{n} & \mathrm{A}_2 : \mathsf{n} \times \mathsf{n} \\ \mathrm{B}_0 : 1 & \mathrm{B}_1 : \mathsf{n} \times \mathsf{n} & \mathrm{B}_2 : \mathsf{n} \times \mathsf{n} \\ \mathrm{N}_1 : \mathsf{n} & \mathrm{N}_2 : \mathsf{n} \times \mathsf{n} & \mathrm{N}_3 : \mathsf{n} \end{array}$$

  - **Rules**:

$$\begin{array}{rcl} \mathrm{A}_0 & \longrightarrow & \exists x.\mathrm{A}_1(x), \mathrm{N}_1 x \\ \mathrm{B}_0, \mathrm{N}_1(x) & \longrightarrow & \exists y.\mathrm{B}_1(x, y), \mathrm{N}_2(x, y) \\ \mathrm{A}_1(x), \mathrm{N}_2(x, y) & \longrightarrow & \mathrm{A}_2(x, y), \mathrm{N}_3(y) \\ \mathrm{B}_1(x, y), \mathrm{N}_3(y) & \longrightarrow & \mathrm{B}_2(x, y) \end{array}$$

Protocols as Theories: An Example

- From the state $\mathrm{A}_0, \mathrm{A}_1$, we can form essentially *one* trace, namely the following one:

$$\begin{aligned}
\mathrm{A}_0, \mathrm{B}_0 &\longrightarrow \mathrm{A}_1(\mathtt{n}_A), \mathrm{N}_1(\mathtt{n}_A), \mathrm{B}_0 \\
&\longrightarrow \mathrm{A}_1(\mathtt{n}_A), \mathrm{B}_1(\mathtt{n}_A, \mathtt{n}_B), \mathrm{N}_2(\mathtt{n}_A, \mathtt{n}_B) \\
&\longrightarrow \mathrm{A}_2(\mathtt{n}_A, \mathtt{n}_B), \mathrm{B}_1(\mathtt{n}_A, \mathtt{n}_B), \mathrm{N}_3(\mathtt{n}_B) \\
&\longrightarrow \mathrm{A}_2(\mathtt{n}_A, \mathtt{n}_B), \mathrm{B}_2(\mathtt{n}_A, \mathtt{n}_B)
\end{aligned}$$

Protocols as Theories: An Example

- From the state $\mathrm{A}_0, \mathrm{A}_1$, we can form essentially *one* trace, namely the following one:

$$
\begin{aligned}
\mathrm{A}_0, \mathrm{B}_0 &\longrightarrow \mathrm{A}_1(\mathtt{n}_A), \mathrm{N}_1(\mathtt{n}_A), \mathrm{B}_0 \\
&\longrightarrow \mathrm{A}_1(\mathtt{n}_A), \mathrm{B}_1(\mathtt{n}_A, \mathtt{n}_B), \mathrm{N}_2(\mathtt{n}_A, \mathtt{n}_B) \\
&\longrightarrow \mathrm{A}_2(\mathtt{n}_A, \mathtt{n}_B), \mathrm{B}_1(\mathtt{n}_A, \mathtt{n}_B), \mathrm{N}_3(\mathtt{n}_B) \\
&\longrightarrow \mathrm{A}_2(\mathtt{n}_A, \mathtt{n}_B), \mathrm{B}_2(\mathtt{n}_A, \mathtt{n}_B)
\end{aligned}
$$

- Observe that the final state is $\mathrm{A}_2(\mathtt{n}_A, \mathtt{n}_B), \mathrm{B}_2(\mathtt{n}_A, \mathtt{n}_B)$: indeed, we want that the parties share the same values at the end of the protocol.
- There is no possibility for the attacker to intervene in the communication. What if we *indeed wanted* to allow the attacker to do so? How could we modify the model?

Modeling the Intruder

- In the Dolev-Yao model, the intruder (i.e. the attacker) can perform activiteis of four kinds:
  - **Read** any message, preventing it to reach its destination.
  - **Decompose** a message into parts and remember them (including decrypting a ciphertext for which it has obtained the key).
  - **Generate** fresh data.
  - **Compose** a new message from known data and send it to the network.

Modeling the Intruder

- In the Dolev-Yao model, the intruder (i.e. the attacker) can perform activiteis of four kinds:
  - **Read** any message, preventing it to reach its destination.
  - **Decompose** a message into parts and remember them (including decrypting a ciphertext for which it has obtained the key).
  - **Generate** fresh data.
  - **Compose** a new message from known data and send it to the network.
- This can be modeled by endowing the theory with:
  - A predicate O capturing what the attacker has *observed*.
  - A predicate M modeling the intruder's *memory*.
  - A predicate C which serves to model new messages the adversary has *crafted*, and which could possibly be sent.
- It is convenient that the aforementioned (unary) predicates are on a sort m of which n is a subsort, and that a binary function symbol $\langle \cdot, \cdot \rangle$ on m is available.

Modeling the Intruder

- The following rules are all natural and their role is intuitive:

$$
\begin{aligned}
\mathrm{N}_1(x) &\longrightarrow \mathrm{D}(x) & \qquad \mathrm{N}_2(x,y) &\longrightarrow \mathrm{D}(\langle x,y\rangle)\\
\mathrm{N}_3(x) &\longrightarrow \mathrm{D}(x) & \mathrm{D}(\langle x,y\rangle) &\longrightarrow \mathrm{D}(x),\mathrm{D}(y)\\
\mathrm{D}(x) &\longrightarrow \mathrm{M}(x) & \mathrm{M}(x) &\longrightarrow \mathrm{C}(x),\mathrm{M}(x)\\
\mathrm{C}(x) &\longrightarrow \mathrm{N}_1(x) & \mathrm{C}(x),\mathrm{C}(y) &\longrightarrow \mathrm{C}(\langle x,y\rangle)\\
\mathrm{C}(\langle x,y\rangle) &\longrightarrow \mathrm{N}_2(x,y) & \mathrm{C}(x) &\longrightarrow \mathrm{N}_3(x)\\
&\longrightarrow \exists x.\mathrm{M}(x) & &
\end{aligned}
$$

Modeling the Intruder

- The following rules are all natural and their role is intuitive:

$$
\begin{aligned}
\mathrm{N}_1(x) &\longrightarrow \mathrm{D}(x) & \mathrm{N}_2(x,y) &\longrightarrow \mathrm{D}(\langle x,y\rangle) \\
\mathrm{N}_3(x) &\longrightarrow \mathrm{D}(x) & \mathrm{D}(\langle x,y\rangle) &\longrightarrow \mathrm{D}(x),\mathrm{D}(y) \\
\mathrm{D}(x) &\longrightarrow \mathrm{M}(x) & \mathrm{M}(x) &\longrightarrow \mathrm{C}(x),\mathrm{M}(x) \\
\mathrm{C}(x) &\longrightarrow \mathrm{N}_1(x) & \mathrm{C}(x),\mathrm{C}(y) &\longrightarrow \mathrm{C}(\langle x,y\rangle) \\
\mathrm{C}(\langle x,y\rangle) &\longrightarrow \mathrm{N}_2(x,y) & \mathrm{C}(x) &\longrightarrow \mathrm{N}_3(x) \\
&\longrightarrow \exists x.\mathrm{M}(x) & &
\end{aligned}
$$

- Using the theory above, it is possible to prove that the attacker can intercept the messages flowing on the network, then impersonating one of the honest parties.

Modeling the Intruder

- The following rules are all natural and their role is intuitive:

$$
\begin{aligned}
\mathrm{N}_1(x) &\longrightarrow \mathrm{D}(x) & \mathrm{N}_2(x,y) &\longrightarrow \mathrm{D}(\langle x,y\rangle)\\
\mathrm{N}_3(x) &\longrightarrow \mathrm{D}(x) & \mathrm{D}(\langle x,y\rangle) &\longrightarrow \mathrm{D}(x),\mathrm{D}(y)\\
\mathrm{D}(x) &\longrightarrow \mathrm{M}(x) & \mathrm{M}(x) &\longrightarrow \mathrm{C}(x),\mathrm{M}(x)\\
\mathrm{C}(x) &\longrightarrow \mathrm{N}_1(x) & \mathrm{C}(x),\mathrm{C}(y) &\longrightarrow \mathrm{C}(\langle x,y\rangle)\\
\mathrm{C}(\langle x,y\rangle) &\longrightarrow \mathrm{N}_2(x,y) & \mathrm{C}(x) &\longrightarrow \mathrm{N}_3(x)\\
&\longrightarrow \exists x.\mathrm{M}(x) & &
\end{aligned}
$$

- Using the theory above, it is possible to prove that the attacker can intercept the messages flowing on the network, then impersonating one of the honest parties.

Fact

There is a trace starting in the A_0, B_0 and ending in a state containing A_2 and B_2 in which the parties do not share the same data.

Towards the Needham-Schroeder's Protocol

- Now, suppose we are interested in treating not the toy protocol we have dealt with so far, but the "real" Needham-Schroeder's Protocol.

Towards the Needham-Schroeder's Protocol

- Now, suppose we are interested in treating not the toy protocol we have dealt with so far, but the "real" Needham-Schroeder's Protocol.
- We can follow the same recipe we followed in the case of the toy protocol.

Towards the Needham-Schroeder's Protocol

- Now, suppose we are interested in treating not the toy protocol we have dealt with so far, but the “real” Needham-Schroeder's Protocol.
- We can follow the same recipe we followed in the case of the toy protocol.
- There will be not two but *three* roles.

Towards the Needham-Schroeder's Protocol

- Now, suppose we are interested in treating not the toy protocol we have dealt with so far, but the "real" Needham-Schroeder's Protocol.
- We can follow the same recipe we followed in the case of the toy protocol.
- There will be not two but *three* roles.
- We have to deal with private encryption, which requires some substantial change.
  - We need new sorts for messages, keys, etc.
  - We need a function symbol `enc`.
  - Crucially, we need a a couple of new rules modeling the intruder, namely the following one:

$$
\begin{aligned}
\mathrm{D}(\texttt{enc}(k, m)), \mathrm{M}(k) &\longrightarrow \mathrm{D}(m) \\
\mathrm{C}(m), \mathrm{C}(k) &\longrightarrow \mathrm{C}(\texttt{enc}(k, m))
\end{aligned}
$$

Towards the Needham-Schroeder's Protocol

- Now, suppose we are interested in treating not the toy protocol we have dealt with so far, but the "real" Needham-Schroeder's Protocol.
- We can follow the same recipe we followed in the case of the toy protocol.
- There will be not two but *three* roles.
- We have to deal with private encryption, which requires some substantial change.
  - We need new sorts for messages, keys, etc.
  - We need a function symbol `enc`.
  - Crucially, we need a a couple of new rules modeling the intruder, namely the following one:

$$
\begin{aligned}
\mathrm{D}(\mathtt{enc}(k,m)), \mathrm{M}(k) &\longrightarrow \mathrm{D}(m) \\
\mathrm{C}(m), \mathrm{C}(k) &\longrightarrow \mathrm{C}(\mathtt{enc}(k,m))
\end{aligned}
$$

- Finally, we also need a mechanism to allow distinct sessions of the same protocol to be executed concurrently.

Part III

Relating the Two Models

Is the Symbolic Model Computationally Sound?

- As far as the adversary's capabilities are concerned, the symbolic model seems to be **more permissive** than the computational one:
  - On the one hand, the adversary is not necessarily *efficient*.
  - On the other hand, any trace leading to an unsafe state is considered a break, independently on its *likelihood*.

Is the Symbolic Model Computationally Sound?

- As far as the adversary's capabilities are concerned, the symbolic model seems to be **more permissive** than the computational one:
  - On the one hand, the adversary is not necessarily *efficient*.
  - On the other hand, any trace leading to an unsafe state is considered a break, independently on its *likelihood*.
- Starting from the late 1990s, researchers have been trying to understand whether the security guarantees provided by the symbolic model can be brought back to the computational model.
  - We take a look at the first such result in the following.

A *Simple* Symbolic Model

- We start with the set $\mathbb{B} = \{0, 1\}$ of booleans, and a set $\mathbb{K}$ of key symbols.
  - The elements of $\mathbb{K}$ must not be confused with binary strings: they are atomic symbols, with no internal structure.
- The **expressions** of the model are nothing else than the expressions produced by the following grammar:

$$M, N ::= K \;\Big|\; i \;\Big|\; \langle M, N \rangle \;\Big|\; \{M\}_K$$

  where $i \in \mathbb{B}$ and $K \in \mathbb{K}$
- We note that there is no ambiguity in the expressions. For example, $\{M\}_K$ e $\langle N, L \rangle$ are always different.

Implication between Expressions

- A crucial concept in the formal model is the **implication** between expressions.
  - Informally, an expression M *implies* N when knowing M allows to reconstruct N.
  - We denote that M implies N by $M \vdash N$.

Implication between Expressions

- A crucial concept in the formal model is the **implication** between expressions.
  - Informally, an expression M *implies* N when knowing M allows to reconstruct N.
  - We denote that M implies N by $M \vdash N$.
- The formal rules are as follows:

$$\frac{}{M \vdash 0} \quad \frac{}{M \vdash 1} \quad \frac{}{M \vdash M} \quad \frac{M \vdash N \quad M \vdash L}{M \vdash (N, L)} \quad \frac{M \vdash \langle N, L \rangle}{M \vdash N}$$

$$\frac{M \vdash \langle N, L \rangle}{M \vdash L} \qquad \frac{M \vdash N \quad M \vdash K}{M \vdash \{N\}_K} \qquad \frac{M \vdash \{N\}_K \quad M \vdash K}{M \vdash N}$$

Implication between Expressions

- A crucial concept in the formal model is the **implication** between expressions.
  - Informally, an expression M *implies* N when knowing M allows to reconstruct N.
  - We denote that M implies N by $M \vdash N$.
- The formal rules are as follows:

$$\frac{}{M \vdash 0} \quad \frac{}{M \vdash 1} \quad \frac{}{M \vdash M} \quad \frac{M \vdash N \quad M \vdash L}{M \vdash (N, L)} \quad \frac{M \vdash \langle N, L \rangle}{M \vdash N}$$

$$\frac{M \vdash \langle N, L \rangle}{M \vdash L} \qquad \frac{M \vdash N \quad M \vdash K}{M \vdash \{N\}_K} \qquad \frac{M \vdash \{N\}_K \quad M \vdash K}{M \vdash N}$$

- For example,

$$(\{\{K_1\}_{K_2}\}_{K_3}, K_3) \vdash K_3 \qquad (\{\{K_1\}_{K_2}\}_{K_3}, K_3) \vdash \{K_1\}_{K_2}$$

Implication and Equivalences

- The implication relation is a good way of modelling **the adversary's capabilities** in the formal model.
  - If $\mathcal{E}$ is an expressions set, $\{M \mid \exists N \in \mathcal{E}.N \vdash M\}$ is what the adversary can compute from $\mathcal{E}$.
  - At each step, the adversary may compute any expression among those in the set and use it to construct a attack.

Implication and Equivalences

- The implication relation is a good way of modelling **the adversary's capabilities** in the formal model.
  - If $\mathcal{E}$ is an expressions set, $\{M \mid \exists N \in \mathcal{E}.N \vdash M\}$ is what the adversary can compute from $\mathcal{E}$.
  - At each step, the adversary may compute any expression among those in the set and use it to construct a attack.
- In order to compare the formal model and computational one, it is worth talking about **equivalences** between expressions.
  - Two expressions are considered equivalent if they are indistinguishable with respect to an adversary whose task is to "separate" them.
  - For example, the two expressions

$$(0, \{0\}_{K_1}) \qquad (0, \{1\}_{K_2})$$

    are to be considered equivalent.
  - How to formalise this notion?

Patterns

- The way an adversary "sees" an expression is captured by the notion of **pattern**:

$$P, Q ::= K \;\Big|\; i \;\Big|\; \langle P, Q \rangle \;\Big|\; \{P\}_K \;\Big|\; \Box$$

- The pattern to which an expression corresponds , depends on the set of keys $\mathcal{T}$ available to the the adversary:

$$\begin{aligned}
p(K, \mathcal{T}) &= K \\
p(i, \mathcal{T}) &= i \\
p(\langle M, N \rangle, \mathcal{T}) &= \langle p(M, \mathcal{T}), p(N, \mathcal{T}) \rangle \\
p(\{M\}_K, \mathcal{T}) &= \begin{cases} \{p(M, \mathcal{T})\}_K & \text{if } K \in \mathcal{T} \\ \Box & \text{otherwise} \end{cases}
\end{aligned}$$

- Finally, the way an adversary sees an expression M is

$$\mathit{pattern}(M) = p(M, \{K \in \mathbb{K} \mid M \vdash K\}).$$

An Equivalence

- Two expressions M and N are **equivalent**, and we will write $M \equiv N$, iff

  $$pattern(M) = pattern(N),$$

  that is, if the two expressions are indistinguishable for every adversary.

An Equivalence

- Two expressions M and N are **equivalent**, and we will write $M \equiv N$, iff

$$pattern(M) = pattern(N),$$

  that is, if the two expressions are indistinguishable for every adversary.
- The equivalence should then be *weakened* by considering

$$M \cong N \Longleftrightarrow M \equiv N\sigma$$

  where σ is a bijection on $\mathbb{K}$.
- **Examples**:

$$\begin{array}{lll} 0 \cong 0 & \{0\}_K \cong \{1\}_K & K_1 \cong K_2 \\ 1 \not\cong 0 & \{0\}_K \cong \{K\}_K & (K, \{0\}_K) \not\cong (K, \{1\}_K) \end{array}$$

Relating Formal and Computational Models

- A natural question is: in which way is the formal model an abstraction of the computational model?

Relating Formal and Computational Models

- A natural question is: in which way is the formal model an abstraction of the computational model?
- We must first understand what an expression M corresponds to in the computational model, given an encryption scheme $\Pi = (Gen, Enc, Dec)$.
  - It will correspond to a *family of distributions*, parameterized on a security parameter n.
  - First of all, we match each **key** $K \in \mathbb{K}$ that occurs in M with the key obtained from $Gen(1^n)$.
  - Then, we match 0 and 1 with a string encoding this boolean value.
  - $\langle N, L\rangle$ pairs in M will be appropriately encoded as binary strings.
  - A **ciphertext** $\{N\}_K$ that occurs in M will be handled by invoking Enc.

Relating Formal and Computational Models

- A natural question is: in which way is the formal model an abstraction of the computational model?
- We must first understand what an expression M corresponds to in the computational model, given an encryption scheme $\Pi = (Gen, Enc, Dec)$.
  - It will correspond to a *family of distributions*, parameterized on a security parameter n.
  - First of all, we match each **key** $K \in \mathbb{K}$ that occurs in M with the key obtained from $Gen(1^n)$.
  - Then, we match 0 and 1 with a string encoding this boolean value.
  - $\langle N, L \rangle$ pairs in M will be appropriately encoded as binary strings.
  - A **ciphertext** $\{N\}_K$ that occurs in M will be handled by invoking Enc.
- The family of distributions to which M corresponds is denoted with $[\![M]\!]_\Pi$.

Relating Formal and Computational Models

- A natural question is: in which way is the formal model an abstraction of the computational model?
- We must first understand what an expression M corresponds to in the computational model, given an encryption scheme $\Pi = (Gen, Enc, Dec)$.
  - It will correspond to a *family of distributions*, parameterized on a security parameter n.
  - First of all, we match each **key** $K \in \mathbb{K}$ that occurs in M with the key obtained from $Gen(1^n)$.
  - Then, we match 0 and 1 with a string encoding this boolean value.
  - $\langle N, L\rangle$ pairs in M will be appropriately encoded as binary strings.
  - A **ciphertext** $\{N\}_K$ that occurs in M will be handled by invoking Enc.
- The family of distributions to which M corresponds is denoted with $[\![M]\!]_\Pi$.
- When are two such families of distributions equivalent *from a computational point of view*?

Relating Formal and Computational Models

Definition

Two families of distributions $\mathcal{D} = \{D_n\}_{n\in\mathbb{N}}$ and $\mathcal{E} = \{E_n\}_{n\in\mathbb{N}}$ are called *computationally indistinguishable* iff for each PPT adversary A there exists a negligible function $\varepsilon \in \mathcal{NGL}$ such that

$$|Pr(A(1^n, D_n) = 1) - Pr(A(1^n, E_n) = 1)| \leq \varepsilon(n)$$

In this case we write $\mathcal{D} \sim \mathcal{E}$.

- An expression M is said to be *cyclic* iff for every subexpression $\{N\}_K$ of M, the key K does not occur in N.

Theorem (Abadi&Rogaway)

If Π is secure and M, N are acyclic, then $M \cong N$ implies $[\![M]\!]_\Pi \sim [\![N]\!]_\Pi$.

Part IV

ProVerif:
a Symbolic Verification Tool

ProVerif

- The `ProVerif` tool is a formal cryptography tool developed by INRIA.

ProVerif

- The `ProVerif` tool is a formal cryptography tool developed by INRIA.
- It is an open-source, freely usable tool developed more than 10 years ago and now in version 2.0.

ProVerif

- The `ProVerif` tool is a formal cryptography tool developed by INRIA.
- It is an open-source, freely usable tool developed more than 10 years ago and now in version 2.0.
- We can see `ProVerif` as a tool that takes in input a file in which there are:
  - The description of some **cryptographic primitives**.
  - The description of a **protocol**.
  - The description of some protocol properties, which `ProVerif` will attempt to verify and refute

  outputting an element of $\{Y, N, ?\}$, together with some other information.

ProVerif

- The ProVerif tool is a formal cryptography tool developed by INRIA.
- It is an open-source, freely usable tool developed more than 10 years ago and now in version 2.0.
- We can see ProVerif as a tool that takes in input a file in which there are:
  - The description of some **cryptographic primitives**.
  - The description of a **protocol**.
  - The description of some protocol properties, which ProVerif will attempt to verify and refute

  outputting an element of $\{Y, N, ?\}$, together with some other information.
- The *syntax* with which primitives and protocols are described is simple enough, with features from functional and concurrent programming.
- More information is available here:
  https://prosecco.gforge.inria.fr/personal/bblanche/proverif/

Input:

```
free c : channel.
free s: bitstring [private].

process
out(c,s);
0
```

Output:

Input:

```
free c : channel.
free s: bitstring [private].

query attacker(s).

process
out(c,s);
0
```

Output:

```
-- Query not attacker(s[])
Completing...
Starting query not attacker(s[])
goal reachable: attacker(s[])
RESULT not attacker(s[]) is false.
```

Input:

```
free c : channel.
free s: bitstring [private].
free p: bitstring [private].

query attacker(s).

process
out(c,p);
0
```

Output:

```
-- Query not attacker(s[])
Completing...
Starting query not attacker(s[])
RESULT not attacker(s[]) is true.
```

Input:

```
free c : channel.
free s: bitstring.
free p: bitstring [private].

query attacker(s).

process
out(c,p);
0
```

Output:

```
-- Query not attacker(s[])
Completing...
Starting query not attacker(s[])
goal reachable: attacker(s[])
RESULT not attacker(s[]) is false.
```

Input:

```
type key.

fun encrypt(bitstring, key): bitstring.
fun decrypt(bitstring, key): bitstring.
equation forall x:bitstring, y:key; decrypt(encrypt(x,y),y) = x.
equation forall x:bitstring, y:key; encrypt(decrypt(x,y),y) = x.

free c: channel.
free k: key [private].
free s: bitstring [private].

query attacker(s).

let processA =
out(c, encrypt(s,k)).

let processB =
in(c, x: bitstring);
let n = decrypt(x,k) in
0.

process
(!processA) | (!processB)
```

Output:

```
Completing equations...
Completing equations...
-- Query not attacker(s[])
Completing...
Starting query not attacker(s[])
RESULT not attacker(s[]) is true.
```

Input:

```
type key.

fun encrypt(bitstring, key): bitstring.
fun decrypt(bitstring, key): bitstring.
equation forall x:bitstring, y:key; decrypt(encrypt(x,y),y) = x.
equation forall x:bitstring, y:key; encrypt(decrypt(x,y),y) = x.

free c: channel.
free k: key [private].
free s: bitstring [private].

query attacker(s).

let processA =
out(c, encrypt(s,k)).

let processB =
in(c, x: bitstring);
let n = decrypt(x,k) in
out(c,k).

process
(!processA) | (!processB)
```

Output:

```
Completing equations...
Completing equations...
-- Query not attacker(s[])
Completing...
Starting query not attacker(s[])
goal reachable: attacker(s[])
RESULT not attacker(s[]) is false.
```