# Privacy and Computer Science (ECI 2015)
# Day 5 - Formal Approaches for Elaborated Protocols
# Logics

Frédéric Prost
Frederic.Prost@ens-lyon.fr

Ecole Normale Supérieure de Lyon

July 2015

# Introduction

- Cryptographic primitives are usually only a part of wider schemes.
- In order to study the security of whole protocols it is often advantageous to have an abstract view of cryptographic operations.

# Introduction

- Cryptographic primitives are usually only a part of wider schemes.
- In order to study the security of whole protocols it is often advantageous to have an abstract view of cryptographic operations.

  $\Longrightarrow$ the aim is to work with a high-level description of what encryption primitives are supposed to achieve.
- It is similar to high-level programming approach to programming vs circuit design or TM programming...

# Introduction

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

# Introduction

- There are two main ways to "prove" security from an abstract point of view:

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

# Introduction

- There are two main ways to "prove" security from an abstract point of view:

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

- How do they relate to one another?

# Introduction

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

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

- How do they relate to one another?

$\Longrightarrow$ Computational soundness.

# Symbolic vs Computational approaches

- Computational is more “solid”.

# Symbolic vs Computational approaches

- Computational is more “solid”.
- Computational is more “artistic”: for each protocol, cryptographic functions, one has to build a specific proof.

# Symbolic vs Computational approaches

- Computational is more "solid".
- Computational is more "artistic": for each protocol, cryptographic functions, one has to build a specific proof.
- Symbolic allows to make more elaborated proofs: protocols are more and more complex and built as subtle combinations of basic cryptographic primitives.

# Symbolic vs Computational approaches

- Computational is more "solid".
- Computational is more "artistic": for each protocol, cryptographic functions, one has to build a specific proof.
- Symbolic allows to make more elaborated proofs: protocols are more and more complex and built as subtle combinations of basic cryptographic primitives.
- Symbolic allows automated proof approaches.

# The Formal View

- Cryptographic operations are seen as purely formal: $\{M\}_K$
  $M$ and $K$ are formal expressions, not sequences of bits.
- An algebra among such formal terms can be applied: typically $\{\{M\}_K\}_{\overline{K}} = M$
  All-or-nothing kind of approach (no probability or incomplete leakages).

# The Formal View

- Cryptographic operations are seen as purely formal: $\{M\}_K$
  $M$ and $K$ are formal expressions, not sequences of bits.
- An algebra among such formal terms can be applied: typically
  $\{\{M\}_K\}_{\overline{K}} = M$
  All-or-nothing kind of approach (no probability or incomplete leakages).
  Counter example ? $\{M\}_K = M + K$ and $K$ used twice...
- Starts with the work of Dolev and Yao [Dolev and Yao, 1983] extensively used to prove the safety of some protocols and also to discover many attacks.
- Leads to the development of effective methods and automatic tools for automated protocol analysis.

$\Longrightarrow$ There is a gap between the ideal represenation of encryption in a formal model and its concrete implementation.

# The Computational View

- Based on complexity theory.
- A proponent of this approach would say that formal approaches are naïve and disconnected from the reality.
- Here, keys, messages are just srtings of bits. Encryption is just an algorithm. The adversary is a Turing Machine.
- Good protocols are the one in which adversaries cannot do "something bad" too often and efficiently enough.
- Example the notion of advantage gained.

# Plan

# The Dolev-Yao Formal Model of Security[Dolev and Yao, 1983]

- In the Needham-Schroeder protocol [Needham and Schroeder, 1978] of identification flaws were found after the publication of the paper. It triggered the interest for formal security protocol analysis tools.
- The Dolev-Yao model is the first proposal.
- The original model is very constrained and does not allow to describe many interesting protocols. Still it is interesting because:
  - First proposition of formal model
  - Restriction are mostly on the honest protocol participants and security goal. Adversaries are quite general.
  - Restricting the class of target protocols allows interesting results like: security is decidable in polynomial time, it can be automated.

# What is Looked for in a Formal Model?

If you want to unambiguously answer to the question: is this protocol secure or not ? What do you need ?

# What is Looked for in a Formal Model?

If you want to unambiguously answer to the question: is this protocol secure or not ? What do you need ?

- Precise language for descriptions of protocols.
- Formal execution model (kind of operational semantics of the protocol), possibly in the presence of an adversary. It includes a descrption of adversary's capabilities: typically, starting the execution of an arbitrary instances of the protocol among anyone (honnest players and adversary).
- A formal language for specifying desired security protocols.

# Dolev-Yao Model

- Focus is on two party protocols and secrecy properties.

# Dolev-Yao Model

- Focus is on two party protocols and secrecy properties.
- Important features of DY model:
  1. **Secrecy properties:** The only goal is to send $M$ in a secret way.

# Dolev-Yao Model

- Focus is on two party protocols and secrecy properties.
- Important features of DY model:
  1. **Secrecy properties:** The only goal is to send $M$ in a secret way.
  2. **Stateles parties:** The messages transmitted by a party at every step of the protocol are a function of theier initial knowledge and the message they just received. In particular, parties cannot use information collected from previous messages.

# Dolev-Yao Model

- Focus is on two party protocols and secrecy properties.

- Important features of DY model:

  1. **Secrecy properties:** The only goal is to send $M$ in a secret way.
  2. **Stateles parties:** The messages transmitted by a party at every step of the protocol are a function of theier initial knowledge and the message they just received. In particular, parties cannot use information collected from previous messages.
  3. **Concurrent execution:** The adversary can start an arbitrary number of protocol executions, involving different sets of parties, where each player can partecipate in several concurrent executions.

# Dolev-Yao Model

- Focus is on two party protocols and secrecy properties.
- Important features of DY model:
  1. **Secrecy properties:** The only goal is to send $M$ in a secret way.
  2. **Stateles parties:** The messages transmitted by a party at every step of the protocol are a function of theier initial knowledge and the message they just received. In particular, parties cannot use information collected from previous messages.
  3. **Concurrent execution:** The adversary can start an arbitrary number of protocol executions, involving different sets of parties, where each player can partecipate in several concurrent executions.
  4. **Public Key cryptography and infrastructure:** It is assumed that a public table $(X, E_X)$ containing the name and public key of every user is publicly available. The initial knowledge of each user consists of this table, plus the user secret decryption key $D_X$.

# Example 1:

- Alice sends an encrypted message to Bob and waits for an echo in acknowledgment:

$$\begin{array}{ll} 1. & A \rightarrow B: \quad \{M\}_B \\ 2. & B \rightarrow A: \quad \{M\}_A \end{array}$$

# Example 1:

- Alice sends an encrypted message to Bob and waits for an echo in acknowledgment:

$$\begin{array}{lll} 1. & A \rightarrow B: & \{M\}_B \\ 2. & B \rightarrow A: & \{M\}_A \end{array}$$

- This protocol is insecure. Formal attack goes like this:

$$\begin{array}{llll} 1. & A \rightarrow Z: & \{M\}_B & Z \text{ intercepts the message} \end{array}$$

# Example 1:

- Alice sends an encrypted message to Bob and waits for an echo in acknowledgment:

$$\begin{array}{lll} 1. & A \rightarrow B: & \{M\}_B \\ 2. & B \rightarrow A: & \{M\}_A \end{array}$$

- This protocol is insecure. Formal attack goes like this:

$$\begin{array}{llll} 1. & A \rightarrow Z: & \{M\}_B & Z \text{ intercepts the message} \\ 2. & Z \rightarrow B: & \{M\}_B & \end{array}$$

# Example 1:

- Alice sends an encrypted message to Bob and waits for an echo in acknowledgment:

$$\begin{array}{lll} 1. & A \to B: & \{M\}_B \\ 2. & B \to A: & \{M\}_A \end{array}$$

- This protocol is insecure. Formal attack goes like this:

$$\begin{array}{llll} 1. & A \to Z: & \{M\}_B & Z \text{ intercepts the message} \\ 2. & Z \to B: & \{M\}_B & \\ 3. & B \to Z: & \{M\}_Z & \text{since } B \text{ follows the protocol, } Z \text{ can recover } M \end{array}$$

# Example 1:

- Alice sends an encrypted message to Bob and waits for an echo in acknowledgment:

$$\begin{array}{lll} 1. & A \rightarrow B: & \{M\}_B \\ 2. & B \rightarrow A: & \{M\}_A \end{array}$$

- This protocol is insecure. Formal attack goes like this:

$$\begin{array}{llll} 1. & A \rightarrow Z: & \{M\}_B & Z \text{ intercepts the message} \\ 2. & Z \rightarrow B: & \{M\}_B & \\ 3. & B \rightarrow Z: & \{M\}_Z & \text{since } B \text{ follows the protocol, } Z \text{ can recover } M \\ 4. & Z \rightarrow A: & \{M\}_A & \text{optional so that even } A \\ & & & \text{does'nt notice the protocol has been broken} \end{array}$$

# Example 2:

- Let's try to fix the protocol by adding the name and an extra layer of encryption:

$$\begin{array}{lll} 1. & A \rightarrow B: & \{\{M\}_B; A\}_B \\ 2. & B \rightarrow A: & \{\{M\}_A; B\}_A \end{array}$$

# Example 2:

- Let's try to fix the protocol by adding the name and an extra layer of encryption:

$$
\begin{array}{lll}
1. & A \rightarrow B: & \{\{M\}_B; A\}_B \\
2. & B \rightarrow A: & \{\{M\}_A; B\}_A
\end{array}
$$

is it secure (from DY point of view) ?

# Example 2:

- Let's try to fix the protocol by adding the name and an extra layer of encryption:

$$\begin{array}{lll} 1. & A \rightarrow B: & \{\{M\}_B; A\}_B \\ 2. & B \rightarrow A: & \{\{M\}_A; B\}_A \end{array}$$

  is it secure (from DY point of view) ?

- No! Here is a (formal) attack.

# Example 2:

1. $Z$ intercepts a protocol execution between $A$ and $B$ with message $M$, and intercepts the last message $\{\text{M'}\}_A where \text{M'} = \{\text{M}\}_A; B$ (as before).

# Example 2:

1. $Z$ intercepts a protocol execution between $A$ and $B$ with message $M$, and intercepts the last message $\{\text{M'}\}_A where \text{M'} = \{\text{M}\}_A; B$ (as before).
2. $Z$ starts another protocol between $Z$ and $A$ with message $M'$, using its knowledge of $\{M'\}_A$:

$$\begin{array}{lll} 1. & Z \rightarrow A: & \{\{M'\}_A; Z\}_A \\ 2. & A \rightarrow Z: & \{\{M'\}_Z; A\}_Z \end{array}$$

Now $Z$ can decrypt and recover $M' = \{M\}_A; B$. Dropping the last $B$, this gives $\{M\}_A$.

# Example 2:

1. $Z$ intercepts a protocol execution between $A$ and $B$ with message $M$, and intercepts the last message $\{\text{M'}\}_A where \text{M'} = \{\text{M}\}_A; B$ (as before).
2. $Z$ starts another protocol between $Z$ and $A$ with message $M'$, using its knowledge of $\{M'\}_A$:

$$\begin{array}{lll} 1. & Z \to A: & \{\{M'\}_A; Z\}_A \\ 2. & A \to Z: & \{\{M'\}_Z; A\}_Z \end{array}$$

   Now $Z$ can decrypt and recover $M' = \{M\}_A; B$. Dropping the last $B$, this gives $\{M\}_A$.
3. $Z$ starts another interaction with $A$:

$$\begin{array}{lll} 1. & Z \to A: & \{\{M\}_A; Z\}_A \\ 2. & A \to Z: & \{\{M\}_Z; A\}_Z \end{array}$$

   At this point, $Z$ can decrypt and recover the original message $M$ which was intented for $B$ only

# DY Model: Protocols considered

The DY model considered focuses on 2 party protocols, executed concurrently in a network with an arbitrary number of partecipants.

- The protocol involves two parties: $S$ (the sender) and $R$ (the receiver) $S(M, R)$ takes an input message $M$, and an identity $R$ of the party $S$ wants to send the message $M$ to.
- The receiver is ready to engage in a protocol execution with any sender.
- Each protocol step is modeled as a function mapping the last received message to a new message to be transmitted. These functions can be the composition of any number of basic functions chosen from a given set $F_X$ of basic functions available to user $X$.

# DY Model: Basic Operations

DY considers two kinds of protocols (corresponding to two sets of basic functions) called cascade protocols and namestamp protocols. The latter is a generalization of the first one, so we concentrate on namestamp protocols. The basic operations available to party $X$ are:

- $D_X$ (decryption under $X$'s secret key)
- $Ey$ (encryption under any user $Y$'s public key)
- $i_y$ (append identifier $y$ to the message)
- $d_y$ (delete identifier $y$ from the end of the message). If input message does not end in $y$, then abort.
- $d$ (delete identifier at end of message)

# DY model: Formal Description of a Protocol

A two party protocol is formally described as a sequence of strings $f[1], f[2], ..., f[k]$ where for any $i$, $f[2i+1]$ is a string over the function symbols available to $S$, and $f[2i]$ is a string over the function symbols available to $R$.

# DY model: Formal Description of a Protocol

A two party protocol is formally described as a sequence of strings $f[1], f[2], ..., f[k]$ where for any $i$, $f[2i+1]$ is a string over the function symbols available to $S$, and $f[2i]$ is a string over the function symbols available to $R$.

- $f[1]$ is the function applied by the sender to the input message $M$ to determine the first message sent to $R$.
- $f[2i]$ is the function applied by $R$ to the *ith* received message to determine the next message to be transmitted to $S$.
- $f[2i+1]$ is the function applied by S to the *ith* received message to determine the next message to be transmitted to $R$.

$S$ and $R$ in the above description are two generic party names, and the protocol can be instantiated replacing $S$ and $R$ with any other pair of parties. Replacing $S$ and $R$ in $f[i]$ with $A$ and $B$ is denoted $f[i]\{A, B\}$.

# DY model: Composition and Cancellation Rules

- For any $i$, let $F[i](M) = f[i](f[i-1](...f[2](f[1](M))...)$ be the composition of the first $i$ functions.

  The sequence of message transmitted during the execution of protocol on input $M$ are $F[1](M), ...F[k](M)$.

# DY model: Composition and Cancellation Rules

- For any $i$, let $F[i](M) = f[i](f[i-1](\ldots f[2](f[1](M))\ldots)$ be the composition of the first $i$ functions.

  The sequence of message transmitted during the execution of protocol on input $M$ are $F[1](M), \ldots F[k](M)$.

- Strings of function symbols are interpreted modulo the following cancellation rules:

$$\begin{array}{lcl} D_x E_x & = & \epsilon \\ E_x D_x & = & \epsilon \\ d_x i_x & = & \epsilon \\ d i_x & = & \epsilon \end{array}$$

  where $\epsilon$ is the empty string, representing the identity function.

# DY model: Composition and Cancellation Rules

- This set of operations can be easily generalized. E.g., strings are taken to represent functions, and in particular, the set of cancellation rules should satisfy the property that if $fw = gw$ for any string $w$, then $f$ and $g$ are the same function (symbol).

- The above rules satisfy these properties.

  An immediate consequence is that if $f$ has both a left and right inverse $lf = fr = id$, then $l = r$ and this inverse is unique.

# Examples

- Example 1:

$$\begin{array}{lll} 1. & S \rightarrow R: & \{M\}_R \\ 2. & R \rightarrow S: & \{M\}_S \end{array}$$

  is modeled by the sequence of strings:

# Examples

- Example 1:

$$\begin{array}{ll} 1. & S \to R : \quad \{M\}_R \\ 2. & R \to S : \quad \{M\}_S \end{array}$$

is modeled by the sequence of strings:

$$\begin{array}{ll} 1. & E_R \\ 2. & E_S D_R \end{array}$$

# Examples

- Example 1:

$$\begin{array}{ll} 1. & S \rightarrow R: \quad \{M\}_R \\ 2. & R \rightarrow S: \quad \{M\}_S \end{array}$$

  is modeled by the sequence of strings:

$$\begin{array}{ll} 1. & E_R \\ 2. & E_S D_R \end{array}$$

- Example 2:

$$\begin{array}{ll} 1. & S \rightarrow R: \quad \{\{M\}_R; S\}_R \\ 2. & R \rightarrow S: \quad \{\{M\}_S; R\}_S \end{array}$$

  is modeled by the sequence of strings:

# Examples

- Example 1:

$$\begin{array}{lll} 1. & S \rightarrow R: & \{M\}_R \\ 2. & R \rightarrow S: & \{M\}_S \end{array}$$

  is modeled by the sequence of strings:

$$\begin{array}{ll} 1. & E_R \\ 2. & E_S D_R \end{array}$$

- Example 2:

$$\begin{array}{lll} 1. & S \rightarrow R: & \{\{M\}_R; S\}_R \\ 2. & R \rightarrow S: & \{\{M\}_S; R\}_S \end{array}$$

  is modeled by the sequence of strings:

$$\begin{array}{ll} 1. & E_R i_S E_R \\ 2. & E_S i_R E_S D_R d_S D_R \end{array}$$

# Formal Execution Model

DY considers a model where an active attacker can interfere with the concurrent execution of an arbitrary number of protocol executions.

- Let $U$ be a potentially infinite pool of user names. Some of the users in $U$ are honest ($H$) and some are corrupted ($C$). The attacker can start an arbitrary number of protocol executions between parties in $U$, honest and dishonest ones.
- The goal of the adversary is to recover the message $M$ underlying a protocol execution between two honest parties A and $B$.
- The attacker is assumed to have total control of the network: in other words, the adversary **IS** the network.

# Formal Excution Model: Adversary Functions

Under the DY execution model the adversary has access to the following functions:

- $f[i]$ where $i \geq 1$ and $S, R$ are replaced by any pair of distinct parties in $U$.
- $E_X, i_X, d_X$ and $d$ for any party $X$ in $U$.
- $D_X$ for any dishonest party $X$ in $C$

Moreover, the adversary can obtain the value $f[1]\{A, B\}(M)$ for any honest parties $A, B$.

# Formal Excution Model: Secure Protocol Definition

The goal of the adversary is to recover $M$. Equivalently, the adversary's goal is to find a sequence of functions $[g_1, ...., g_k]$ such that $g_k \circ ...g_2 circg_1 of[1]\{A, B\} = id$for some honest parties $A$ and $B$. Hence the definion:

**Definition**

Let $f[1], ..., f[r]$ be a two party protocol between a sender $S$ and receiver $R$. The protocol is insecure if and only if for some honest parties $A, B$, the adversary has access to a sequence of functions $g_1, ..., g_k$ such that $g_k o...g_2 og_1 of[1]\{A, B\} = id$.

# Formal Excution Model: Secure Protocol Definition

The goal of the adversary is to recover $M$. Equivalently, the adversary's goal is to find a sequence of functions $[g_1, ...., g_k]$ such that $g_k \circ ...g_2 circg_1 of[1]\{A, B\} = id$for some honest parties $A$ and $B$. Hence the definion:

## Definition

Let $f[1], ..., f[r]$ be a two party protocol between a sender $S$ and receiver $R$. The protocol is insecure if and only if for some honest parties $A, B$, the adversary has access to a sequence of functions $g_1, ..., g_k$ such that $g_k o...g_2 og_1 of[1]\{A, B\} = id$.

The remaining question: Can security be decided? Efficiently decided?

# Formal Excution Model: Reduction Theorem

The answer is yes but there are problems due to the unbounded number of participants. One has to show that we can always restrict the number of parties to 3: 2 honest parties $A, B$ and the adversary $Z$.

# Formal Excution Model: Reduction Theorem

The answer is yes but there are problems due to the unbounded number of participants. One has to show that we can always restrict the number of parties to 3: 2 honest parties $A, B$ and the adversary $Z$.

**Theorem**

*Let $f[1], ..., f[r]$ be a DY protocol. If the protocol is insecure, then there is a sequence of functions $[g_1, ..., g_k]$ and pair of parties $A, B$ demonstrating the insecurity, where all the parties involved in the functions are from $A, B$ and $Z$.*

# Formal Excution Model: Reduction Theorem

Proof sketch:
Assume $g_k \circ ... g_2 \circ g_1 \circ f[1]\{A, B\} = id$ is an attack. We obtain an attack involving only $A, B$ and $Z$ by replacing all identifiers different from $A$ and $B$ with $Z$. Since the substitution can only give more cancellations, we still have $g'_k o ... o g'_1 o f[1]\{A, B\} = id$.

# Formal Excution Model: Reduction Theorem

Proof sketch:
Assume $g_k \circ ... g_2 \circ g_1 \circ f[1]\{A, B\} = id$ is an attack. We obtain an attack involving only $A, B$ and $Z$ by replacing all identifiers different from $A$ and $B$ with $Z$. Since the substitution can only give more cancellations, we still have $g'_k o ... o g'_1 o f[1]\{A, B\} = id$.

- If $g_k$ is $D_X, E_X, i_X, d_X$ or $d$, for some $X$ different from $A, B$, then the resulting function is $D_Z, E_Z, i_Z, d_Z, d$ and adversary $Z$ is allowed to use this function.

# Formal Excution Model: Reduction Theorem

Proof sketch:

Assume $g_k \circ ... g_2 \circ g_1 \circ f[1]\{A, B\} = id$ is an attack. We obtain an attack involving only $A, B$ and $Z$ by replacing all identifiers different from $A$ and $B$ with $Z$. Since the substitution can only give more cancellations, we still have $g'_k o ... o g'_1 o f[1]\{A, B\} = id$.

- If $g_k$ is $D_X, E_X, i_X, d_X$ or $d$, for some $X$ different from $A, B$, then the resulting function is $D_Z, E_Z, i_Z, d_Z, d$ and adversary $Z$ is allowed to use this function.
- If $g_k$ is $f[i]\{A, B\}$ or $f[i]\{B, A\}$, then $g'_k = g_k$ is an allowed function

# Formal Excution Model: Reduction Theorem

Proof sketch:

Assume $g_k \circ ... g_2 \circ g_1 \circ f[1]\{A, B\} = id$ is an attack. We obtain an attack involving only $A, B$ and $Z$ by replacing all identifiers different from $A$ and $B$ with $Z$. Since the substitution can only give more cancellations, we still have $g'_k o ... o g'_1 o f[1]\{A, B\} = id$.

- If $g_k$ is $D_X, E_X, i_X, d_X$ or $d$, for some $X$ different from $A, B$, then the resulting function is $D_Z, E_Z, i_Z, d_Z, d$ and adversary $Z$ is allowed to use this function.
- If $g_k$ is $f[i]\{A, B\}$ or $f[i]\{B, A\}$, then $g'_k = g_k$ is an allowed function
- If $g_k$ is $f[i]\{A, C\}, f[i]\{B, C\}, f[i]\{C, A\}$ or $f[i]\{C, B\}$ for some $C$ different from $A$ and $B$, then the new function $g'_k$ is identical to $g_k$, except for replacing $C$ with $Z$.

# Formal Excution Model: Reduction Theorem

Proof sketch:
Assume $g_k \circ ... g_2 \circ g_1 \circ f[1]\{A, B\} = id$ is an attack. We obtain an attack involving only $A, B$ and $Z$ by replacing all identifiers different from $A$ and $B$ with $Z$. Since the substitution can only give more cancellations, we still have $g'_k o ... o g'_1 o f[1]\{A, B\} = id$.

- If $g_k$ is $D_X, E_X, i_X, d_X$ or $d$, for some $X$ different from $A, B$, then the resulting function is $D_Z, E_Z, i_Z, d_Z, d$ and adversary $Z$ is allowed to use this function.
- If $g_k$ is $f[i]\{A, B\}$ or $f[i]\{B, A\}$, then $g'_k = g_k$ is an allowed function
- If $g_k$ is $f[i]\{A, C\}, f[i]\{B, C\}, f[i]\{C, A\}$ or $f[i]\{C, B\}$ for some $C$ different from $A$ and $B$, then the new function $g'_k$ is identical to $g_k$, except for replacing $C$ with $Z$.
- If $g_k$ is $f[i]\{C, D\}$ for two parties $C, D$ not in $\{A, B\}$, then $g'_k = f[i]\{Z, Z\}$ is the composition of functions of the form $D_Z, E_Z, i_Z, d_Z, d$, which are all allowed.

# Decidability of Formal Execution Model

- We use the Theorem to reduce the problem of testing the security of a protocol to a special case of the same problem where the number of parties is bounded by 3 and the adversary has access only to a finite number of functions.

# Decidability of Formal Execution Model

- We use the Theorem to reduce the problem of testing the security of a protocol to a special case of the same problem where the number of parties is bounded by 3 and the adversary has access only to a finite number of functions.
  Though the length of the attack is still potentially unbounded!
  $\Longrightarrow$ so it is not clear if the problem can be solved algorithmically.

# Decidability of Formal Execution Model

- We use the Theorem to reduce the problem of testing the security of a protocol to a special case of the same problem where the number of parties is bounded by 3 and the adversary has access only to a finite number of functions.
  Though the length of the attack is still potentially unbounded!
  $\Longrightarrow$ so it is not clear if the problem can be solved algorithmically.
- DY shows that the problem is indeed decidable, and moreover, there is an efficient (polynomial time) decision procedure.
- The running time of the decision procedure of DY is $n^3$.

# Decidability Procedure of Formal Execution Model

1. Consider the set of all words over the alphabet $\{E_A, E_B, E_Z, D_A, D_B, D_Z, i_A, i_B, i_Z, d_A, d_B, d_Z, d\}$ that simplify to the empty string using the cancellation rules $D_X E_X = E_X D_X = d_X i_X = di_X = \epsilon$.
2. This set of words is context free and can be generated by a context free grammar with rules $S \rightarrow \epsilon | D_X S E_X S | \ldots$ and so on for all cancellation rules.

# Decidability Procedure of Formal Execution Model

1. Consider the set of all words over the alphabet $\{E_A, E_B, E_Z, D_A, D_B, D_Z, i_A, i_B, i_Z, d_A, d_B, d_Z, d\}$ that simplify to the empty string using the cancellation rules $D_X E_X = E_X D_X = d_X i_X = di_X = \epsilon$.
2. This set of words is context free and can be generated by a context free grammar with rules $S \to \epsilon | D_X S E_X S | \ldots$ and so on for all cancellation rules.
   The grammar can be easily converted into an equivalent Push Down Automaton. Notice that the size of this automaton is constant because it does not depend on the protocol.

# Decidability Procedure of Formal Execution Model

1. Consider the set of all words over the alphabet $\{E_A, E_B, E_Z, D_A, D_B, D_Z, i_A, i_B, i_Z, d_A, d_B, d_Z, d\}$ that simplify to the empty string using the cancellation rules $D_X E_X = E_X D_X = d_X i_X = di_X = \epsilon$.
2. This set of words is context free and can be generated by a context free grammar with rules $S \rightarrow \epsilon | D_X S E_X S | \dots$ and so on for all cancellation rules.
   The grammar can be easily converted into an equivalent Push Down Automaton. Notice that the size of this automaton is constant because it does not depend on the protocol.
3. Next we build a nondeterministic finite automaton accepting all the strings of the form $g_k \circ \dots o g_1 of[1]A, B$ where each $g_i$ is one of the finitely many functions the adversary has access to.

# Decidability Procedure of Formal Execution Model

4. The resulting automaton has a number of states proportional to $n$.

# Decidability Procedure of Formal Execution Model

4. The resulting automaton has a number of states proportional to $n$.

5. Finally, we combine the PDA and NFA using a cartesian product construction to obtain a new PDA that accepts the intersection of the two languages.

   At this point we are left with the problem of deciding if the language of a PDA is empty or not.

# Decidability Procedure of Formal Execution Model

4. The resulting automaton has a number of states proportional to $n$.

5. Finally, we combine the PDA and NFA using a cartesian product construction to obtain a new PDA that accepts the intersection of the two languages.

   At this point we are left with the problem of deciding if the language of a PDA is empty or not.

   $\Longrightarrow$ This can be done in $O(n^3)$.

# Plan

1 Dolev-Yao Model

2 Computational and Logical Approaches to Cryptography
- Symetric Encryption, Passive Attacker

3 Conclusion

# Plan

1. Dolev-Yao Model
2. Computational and Logical Approaches to Cryptography
   - Symetric Encryption, Passive Attacker
3. Conclusion

# The Computational Soundness of Formal Encryption [Abadi and Rogaway, 2000]

- Expressions represent data used in messages in security protocols: they are built from bits and keys by pairing and encryption.
- An equivalence relation is denfined to capture the idea that "data look the same" to an adversary that has no prior knowledge of the keys used in the data.
- For instance an adversary cannot obtain $K$ from $\{0\}_K$ and $\{1\}_K$.
- Similarly the pairs $(0, \{0\}_K)$ and $(0, \{1\}_K)$ are equivalent.
- On the other hand pairs $(K, \{0\}_K)$ and $(K, \{1\}_K)$ are not.

# Formal encryption and expression equivalence

- Symmetric encryption.
- Expressions are built from bits and keys.
- Expressions represent data exchanged in security protocols.
- An equivalenc relation is built: when two expressions "look the same" to the eyes of an adveresary.
  - Adversary cannot get $K$ from $\{1\}_K$ or $\{0\}_K$.
  - $(0, \{0\}_K)$ is the same as $(0, \{1\}_K)$.
  - $(K, \{0\}_K)$ not is the same as $(K, \{1\}_K)$.

# Expressions

$$
\begin{array}{lcll}
M, N & := & K & (K \in Keys) \\
 & & i & (i \in \mathbf{Bool}) \\
 & & (M, N) & \\
 & & \{M\}_K & (K \in Keys)
\end{array}
$$

- Cyclic terms forbidden, e.g.: $\{K\}_K, (\{K'\}_K, \{K\}_{K'})$
- possible extensions:
  - Possibility to use arbitrary expressions as keys.: $\{N\}_M$.

# Expressions

$$
\begin{array}{lll}
M, N := & K & (K \in Keys) \\
& i & (i \in \mathbf{Bool}) \\
& (M, N) & \\
& \{M\}_K & (K \in Keys)
\end{array}
$$

- Cyclic terms forbidden, e.g.: $\{K\}_K, (\{K'\}_K, \{K\}_{K'})$
- possible extensions:
  - Possibility to use arbitrary expressions as keys.: $\{N\}_M$.
  - Distinguishing encryption and decryption keys.

# Entailment Relation

- $M \vdash N$: $N$ can be computed from $M$

# Entailment Relation

- $M \vdash N$: $N$ can be computed from $M$
- Rules:

$$\frac{}{M \vdash 0} \qquad \frac{}{M \vdash M} \qquad \frac{}{M \vdash 1}$$

$$\frac{M \vdash N_1 \quad M \vdash N_2}{M \vdash (N_1, N_2)} \qquad \frac{M \vdash N \quad M \vdash K}{M \vdash \{M\}_K}$$

$$\frac{M \vdash (N_1, N_2)}{M \vdash N_1} \qquad \frac{M \vdash (N_1, N_2)}{M \vdash N_2}$$

$$\frac{M \vdash K \quad M \vdash \{N\}_K}{M \vdash N}$$

# Entailment Relation: Examples

- The entailment relation models what an attacker can obtain from $M$ without any prior knowledge of the keys used in $M$.

# Entailment Relation: Examples

- The entailment relation models what an attacker can obtain from $M$ without any prior knowledge of the keys used in $M$.
- Correct Entailments:

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

# Entailment Relation: Examples

- The entailment relation models what an attacker can obtain from $M$ without any prior knowledge of the keys used in $M$.
- Correct Entailments:

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

- Incorrect Entailment:

$$(\{\{K1\}_{K_2}\}_{K_3}, K_3) \vdash K_1 \quad \text{false}$$

# Patterns

- Extension of expressions. Introduction of $\square$: undecypherable expression.

# Patterns

- Extension of expressions. Introduction of $\Box$: undecypherable expression.
- Patterns are used to model formally the fact that an attacker cannot decypher unless he has the key.

# Patterns

- Extension of expressions. Introduction of $\square$: undecypherable expression.
- Patterns are used to model formally the fact that an attacker cannot decypher unless he has the key.
- Patterns:

$$
\begin{array}{rcll}
P, Q & := & K & (K \in Keys) \\
 & & i & (i \in \mathbf{Bool}) \\
 & & (P, Q) & \\
 & & \{P\}_K & (K \in Keys) \\
 & & \square &
\end{array}
$$

# Reduction to a Pattern

- Intuitively: the pattern that can be seen in $M$ knowing set of keys $T$.

# Reduction to a Pattern

- Intuitively: the pattern that can be seen in $M$ knowing set of keys $T$.
- $\pi$ inductively defined by:

$$
\begin{array}{lcll}
\pi(K, T) & = & K & (K \in Keys) \\
\pi((M, N), T) & = & (\pi(M, T), \pi(N, T)) & \\
\pi(i, T) & = & i & (i \in \mathbf{Bool}) \\
\pi(\{M\}_K) & = & \begin{cases} \pi(M, T) & \text{if } K \in T \\ \square & \text{otherwise} \end{cases} &
\end{array}
$$

# Reduction to a Pattern

- Intuitively: the pattern that can be seen in $M$ knowing set of keys $T$.
- $\pi$ inductively defined by:

$$
\begin{array}{lcll}
\pi(K,T) & = & K & (K \in Keys) \\
\pi((M,N),T) & = & (\pi(M,T),\pi(N,T)) & \\
\pi(i,T) & = & i & (i \in \mathbf{Bool}) \\
\pi(\{M\}_K) & = & \left\{ \begin{array}{ll} \pi(M,T) & \text{if } K \in T \\ \Box & \text{otherwise} \end{array} \right. &
\end{array}
$$

- pattern of a term $M$: $\pi(M) = \pi(M, \{K \in Keys \mid M \vdash K\})$

# Reduction to a Pattern

- Intuitively: the pattern that can be seen in $M$ knowing set of keys $T$.
- $\pi$ inductively defined by:

$$
\begin{array}{lcll}
\pi(K,T) & = & K & (K \in \mathit{Keys}) \\
\pi((M,N),T) & = & (\pi(M,T),\pi(N,T)) & \\
\pi(i,T) & = & i & (i \in \mathbf{Bool}) \\
\pi(\{M\}_K) & = & \left\{ \begin{array}{ll} \pi(M,T) & \text{if } K \in T \\ \square & \text{otherwise} \end{array} \right. &
\end{array}
$$

- pattern of a term $M$: $\pi(M) = \pi(M, \{K \in \mathit{Keys} \mid M \vdash K\})$
- Equivalent terms : $M \equiv N$ iff $\pi(M) = \pi(N)$

# Reduction to a Pattern

- Intuitively: the pattern that can be seen in $M$ knowing set of keys $T$.
- $\pi$ inductively defined by:

$$
\begin{array}{lcll}
\pi(K,T) & = & K & (K \in Keys) \\
\pi((M,N),T) & = & (\pi(M,T),\pi(N,T)) & \\
\pi(i,T) & = & i & (i \in \mathbf{Bool}) \\
\pi(\{M\}_K) & = & \left\{ \begin{array}{ll} \pi(M,T) & \text{if } K \in T \\ \Box & \text{otherwise} \end{array} \right. &
\end{array}
$$

- pattern of a term $M$: $\pi(M) = \pi(M, \{K \in Keys \mid M \vdash K\})$
- Equivalent terms : $M \equiv N$ iff $\pi(M) = \pi(N)$
- Equivalence up to key renaming ($\sigma$ bijection on $Keys$): $M \simeq N$ iff $M \equiv \sigma(N)$

# Equivalent up to renaming Terms Examples

$$(\{\{K1\}_{K_2}\}_{K_3}, K_3) \equiv (\{\{0\}_{K_2}\}_{K_3}, K_3)$$

# Equivalent up to renaming Terms Examples

$$(\{\{K1\}_{K_2}\}_{K_3}, K_3) \equiv (\{\{0\}_{K_2}\}_{K_3}, K_3)$$
$$\{0\}_K \simeq \{1\}_K$$

# Equivalent up to renaming Terms Examples

$$(\{\{K1\}_{K_2}\}_{K_3}, K_3) \equiv (\{\{0\}_{K_2}\}_{K_3}, K_3)$$
$$\{0\}_K \simeq \{1\}_K$$
$$(K, \{0\}_K) \not\simeq (K, \{1\}_K)$$

# Equivalent up to renaming Terms Examples

$$(\{\{K1\}_{K_2}\}_{K_3}, K_3) \equiv (\{\{0\}_{K_2}\}_{K_3}, K_3)$$
$$\{0\}_K \simeq \{1\}_K$$
$$(K, \{0\}_K) \not\simeq (K, \{1\}_K)$$
$$(K, \{(\{0\}_{K'}, 0)\}_K) \simeq (K, \{(\{1\}_{K'}, 0)\}_K)$$

# Equivalent up to renaming Terms Examples

$$(\{\{K1\}_{K_2}\}_{K_3}, K_3) \equiv (\{\{0\}_{K_2}\}_{K_3}, K_3)$$
$$\{0\}_K \simeq \{1\}_K$$
$$(K, \{0\}_K) \not\simeq (K, \{1\}_K)$$
$$(K, \{(\{0\}_{K'}, 0)\}_K) \simeq (K, \{(\{1\}_{K'}, 0)\}_K)$$
$$K \not\equiv K' \text{ but } K \simeq K'.$$

# Equivalent up to renaming Terms Examples

$$(\{\{K1\}_{K_2}\}_{K_3}, K_3) \equiv (\{\{0\}_{K_2}\}_{K_3}, K_3)$$
$$\{0\}_K \simeq \{1\}_K$$
$$(K, \{0\}_K) \not\simeq (K, \{1\}_K)$$
$$(K, \{(\{0\}_{K'}, 0)\}_K) \simeq (K, \{(\{1\}_{K'}, 0)\}_K)$$
$$K \not\equiv K' \text{ but } K \simeq K'.$$
$$\{((1,1),(1,1))\}_K \simeq \{0\}_K$$

# Equivalent up to renaming Terms Examples

$$(\{\{K1\}_{K_2}\}_{K_3}, K_3) \equiv (\{\{0\}_{K_2}\}_{K_3}, K_3)$$
$$\{0\}_K \simeq \{1\}_K$$
$$(K, \{0\}_K) \not\simeq (K, \{1\}_K)$$
$$(K, \{(\{0\}_{K'}, 0)\}_K) \simeq (K, \{(\{1\}_{K'}, 0)\}_K)$$
$$K \not\equiv K' \text{ but } K \simeq K'.$$
$$\{((1,1),(1,1))\}_K \simeq \{0\}_K$$
$$(\{0\}_K, \{0\}_K) \simeq (\{0\}_K, \{1\}_K)$$

# Equivalent up to renaming Terms Examples

$$(\{\{K1\}_{K_2}\}_{K_3}, K_3) \equiv (\{\{0\}_{K_2}\}_{K_3}, K_3)$$
$$\{0\}_K \simeq \{1\}_K$$
$$(K, \{0\}_K) \not\simeq (K, \{1\}_K)$$
$$(K, \{(\{0\}_{K'}, 0)\}_K) \simeq (K, \{(\{1\}_{K'}, 0)\}_K)$$
$$K \not\equiv K' \text{ but } K \simeq K'.$$
$$\{((1,1),(1,1))\}_K \simeq \{0\}_K$$
$$(\{0\}_K, \{0\}_K) \simeq (\{0\}_K, \{1\}_K)$$
$$(\{0\}_K, \{0\}_K) \simeq (\{0\}_K, \{1\}_{K'})$$

# Computational View of an Encryption Scheme

- Plain texts (*Plaintext*), Keys (*Keys*) and Cipher texts (*Ciphertext*) are strings of bits.
- Let *Coins* be a synonym for $\{0,1\}^\omega$, and *Parameter* be a synonym for $1^*$.
- Let $\emptyset$ denotes a particular string that is the decryption of the encryption of a string not in *Plaintext*.

# Computational View of an Encryption Scheme

- Plain texts (*Plaintext*), Keys (*Keys*) and Cipher texts (*Ciphertext*) are strings of bits.
- Let *Coins* be a synonym for $\{0,1\}^\omega$, and *Parameter* be a synonym for $1^*$.
- Let $\emptyset$ denotes a particular string that is the decryption of the encryption of a string not in *Plaintext*.
- An encryption scheme $\Pi$ is $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ with:
  - $\mathcal{K} : \mathit{Parameter} \times \mathit{Coins} \rightarrow \mathit{Keys}$
  - $\mathcal{E} : \mathit{Keys} \times \mathit{String} \times \mathit{Coins} \rightarrow \mathit{Ciphertext}$
  - $\mathcal{D} \mathit{Keys} \times \mathit{String} \rightarrow \mathit{Plaintext}$

  such that if $m \in \mathit{Plaintext}$ then $\mathcal{D}_k(\mathcal{E}_k(m, r)) = m$ otherwise $\mathcal{D}_k(\mathcal{E}_k(m, r)) = \emptyset$

# Computational View of an Encryption Scheme

- Plain texts (*Plaintext*), Keys (*Keys*) and Cipher texts (*Ciphertext*) are strings of bits.
- Let *Coins* be a synonym for $\{0,1\}^\omega$, and *Parameter* be a synonym for $1^*$.
- Let $\emptyset$ denotes a particular string that is the decryption of the encryption of a string not in *Plaintext*.
- An encryption scheme Π is $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ with:
  - $\mathcal{K} : \mathit{Parameter} \times \mathit{Coins} \rightarrow \mathit{Keys}$
  - $\mathcal{E} : \mathit{Keys} \times \mathit{String} \times \mathit{Coins} \rightarrow \mathit{Ciphertext}$
  - $\mathcal{D} \mathit{Keys} \times \mathit{String} \rightarrow \mathit{Plaintext}$

  such that if $m \in \mathit{Plaintext}$ then $\mathcal{D}_k(\mathcal{E}_k(m, r)) = m$ otherwise $\mathcal{D}_k(\mathcal{E}_k(m, r)) = \emptyset$

$\Longrightarrow$ definition for probabilistic, stateless encryption.

# Security Dimensions

1. Repetition concealing vs repetition revealing:
   Given $c, c'$ can one tell if their underlying plaintext are equal?

# Security Dimensions

1. Repetition concealing vs repetition revealing:
   Given $c, c'$ can one tell if their underlying plaintext are equal?
2. Which key conceiling vs which key revealing:
   If one encrypts messages under various keys can one tell which messages were encrypted the same keys?

# Security Dimensions

1. Repetition concealing vs repetition revealing:
   Given $c, c'$ can one tell if their underlying plaintext are equal?
2. Which key conceiling vs which key revealing:
   If one encrypts messages under various keys can one tell which messages were encrypted the same keys?
3. Message-length conceiling vs message-key revealing:
   Does a cyphertext reveal the length of its plaintext?

# Security Dimensions

1. Repetition concealing vs repetition revealing:
   Given $c, c'$ can one tell if their underlying plaintext are equal?
2. Which key conceiling vs which key revealing:
   If one encrypts messages under various keys can one tell which messages were encrypted the same keys?
3. Message-length conceiling vs message-key revealing:
   Does a cyphertext reveal the length of its plaintext?

- Three dimensions orthogonals: 8 possible combinations.
- Concealing is 0, revealing is 1.
- Most signficant is Repetition, least is Message-length.
- Usual approach is type-3 security [Goldwasser and Micali, 1984].

# Computable Indistinguishability

- $\epsilon : \mathbb{N} \rightarrow \mathbb{R}$ is negligible if $\forall c > 0 \exists N_c \forall \eta > N_C . \epsilon(\eta) \leq \eta^{-c}$.
- $D$ and $D'$ are indistinguishable, written $D \approx D'$, if for every probabilistic polynomial-time adversary $A$:

$$\epsilon(\eta) \stackrel{def}{=} Pr[x \stackrel{R}{\leftarrow} D_\eta : A(\eta, x) = 1] - Pr[x \stackrel{R}{\leftarrow} D'_\eta : A(\eta, x) = 1)]$$

  is negligible.
- Type-0 security for $\Pi = (Keys, \mathcal{E}, \mathcal{D})$ with parameter $\eta$:

$$\begin{aligned}\mathrm{Adv}^0_{\Pi[\eta]}(A) \quad \stackrel{def}{=} \quad & Pr[k, k' \stackrel{R}{\leftarrow} Keys(\eta) : A^{\mathcal{E}_k(\cdot), \mathcal{E}_{k'}(\cdot)}(\eta) = 1] - \\ & Pr[k, k' \stackrel{R}{\leftarrow} Keys(\eta) : A^{\mathcal{E}_k(\emptyset), \mathcal{E}_k(\emptyset)}(\eta) = 1]\end{aligned}$$

  $\Pi$ is secure if for all $A$, $\mathrm{Adv}^0_{\Pi[\eta]}(A)$ is negligible (in $\eta$).

# Computational Soundness of Formal Equivalence

- Relation between the two views on cryptography.
- For each formal expression $M$ we associate a distribution on strings $[\![M]\!]_{\pi(\eta)}$.
  - For each $K$ that occurs in $M$ : $\tau(K) \stackrel{R}{\leftarrow} Keys(\eta)$.
  - $[\![M]\!]$ is
    - if $M = K$ and $K \in \mathcal{K}$ then $\langle \tau(K), "key" \rangle$.
    - if $M = b$ and $b \in$ **Bool** then $\langle b, "bool" \rangle$.
    - if $M = (M_1, M_2)$ then $\langle [\![M_1]\!], [\![M_2]\!], "pair" \rangle$.
    - if $M = \{M_1\}_K$ then $x \stackrel{R}{\leftarrow} [\![M_1]\!]$ $\langle \mathcal{E}_{\tau(K)}(x), "ciphertext" \rangle$

# Equivalence Implies Indistinguishability

**Theorem ([Abadi and Rogaway, 2000])**

*Let $M, N$ be acyclic expressions and $\Pi$ be a type-0 secure encryption scheme. If $M \simeq N$ then $[\![M]\!]_{Pi} \approx [\![N]\!]_{\Pi}$.*

# Equivalence Implies Indistinguishability

**Theorem ([Abadi and Rogaway, 2000])**

*Let $M, N$ be acyclic expressions and $\Pi$ be a type-0 secure encryption scheme. If $M \simeq N$ then $[\![M]\!]_{Pi} \approx [\![N]\!]_{\Pi}$.*

Proof sketch:

Complicated proof based on a hybrid arguments (6 pages long).

The first part consist in renaming keys. Complicated because some keys are not directly recoverable (use of acyclicity).

Definition of patterns relating $M, N$ to their renamed version $M', N'$.

The proof is then finished by contradiction over the type-0 security by considering $[\![M']\!]_{Pi} \approx [\![N']\!]_{\Pi}$.

□

# Plan

# Conclusion

- Formal Approach is a different way to deal with security proofs:
  - Suitable to automatic proof.
  - Hypotheses on protocols have to be clear.
  - What are the ssumptions on the cryptographic functions?
  - The attacker is very powerfull in some sense.
- There are works to make both communities converge.
- There are even formal approach to Zero-Knowledge proofs citeBackesU10 (with results wrt computational proofs of security).
  - Needs to include formal randomness.
  - Needs to have a stateful execution model.

# Bibliography I

Abadi, M. and Rogaway, P. (2000).
Reconciling two views of cryptography (the computational soundness of formal encryption).
In *Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics, International Conference IFIP TCS 2000, Sendai, Japan, August 17-19, 2000, Proceedings*, pages 3–22.

Backes, M., Pfitzmann, B., and Waidner, M. (2003).
A universally composable cryptographic library.
*IACR Cryptology ePrint Archive*, 2003:15.

Backes, M. and Unruh, D. (2010).
Computational soundness of symbolic zero-knowledge proofs.
*Journal of Computer Security*, 18(6):1077–1155.

# Bibliography II


Dolev, D. and Yao, A. C. (1983).
On the security of public key protocols.
*IEEE Transactions on Information Theory*, 29(2):198–207.

Goldwasser, S. and Micali, S. (1984).
Probabilistic encryption.
*Journal of Computer and System Sciences*, 28(2):270–299.

Needham, R. M. and Schroeder, M. D. (1978).
Using encryption for authentication in large networks of computers.
*Commun. ACM*, 21(12):993–999.