Confidentiality analyses correct wrt. computational semantics

Peeter Laud
peeter.l@ut.ee

Tartu Ülikool
Cybernetica AS
Overview

- Computationally secure information flow.
- A program analysis, correct wrt. above.
- Confidentiality in cryptographic protocols.
- A very simple analysis.
- Using the def. of secure encryption.
Problem statement

Inputs come from a known source, i.e. the distribution of inputs is known.

- Public outputs should be independent of secret inputs.
- We want tools checking that.
- The input to these tools is the program text.
  - ... possibly also the description of input distribution.
Programming language — syntax

The WHILE-language (simple imperative language).

\[
P ::= \begin{align*}
  & x := o(x_1, \ldots, x_k) \\
  | & \text{skip} \\
  | & P_1; P_2 \\
  | & \text{if } b \text{ then } P_1 \text{ else } P_2 \\
  | & \text{while } b \text{ do } P'
\end{align*}
\]

\(b, x, x_1, \ldots, x_k \in \text{Var. } o \in \text{Op. } \text{Enc, Gen} \in \text{Op.}\)
Programming language — semantics

Denotational semantics: $[P] : \text{State} \rightarrow \text{State}_\bot$.

State = Var $\rightarrow$ Val.

State$_\bot$ has an extra element $\bot$, denoting nontermination.

For each $o \in \text{Op}$ with arity $k$, a function $[o] : \text{Val}^k \rightarrow \text{Val}$ is given.

Semantics is defined inductively over program structure.

This is the traditional setup...
Cryptographic considerations

- Security definitions in theoretical cryptography require primitives with probabilistic functionality;
- the security parameter.
- Also, all values are bit-strings.

Therefore:
- \([P] = \{[P]_n\}_{n \in \mathbb{N}}\);
- \([P]_n : \text{State}_n \rightarrow \mathcal{D}(\text{State}_{n \perp})\);
- \(\text{State}_n = \text{Var} \rightarrow \text{Val}_n\);
- \(\text{Val}_n = \{0, 1\}^*\).

Also, \([o] = \{[o]_n\}_{n \in \mathbb{N}}, [o]_n : \text{Val}_n^k \rightarrow \mathcal{D}(\text{Val}_n)\).
Computationally secure information flow

A program has CSIF, if its public outputs are computationally independent from its secret inputs.

- **Secret inputs** — initial values of variables in $\text{Var}_S \subseteq \text{Var}$.
- **Public outputs** — final values of variables in $\text{Var}_P \subseteq \text{Var}$.

Let $D_n \in \mathcal{D}(\text{State}_n)$ be the distribution of input states for security parameter $n$. Computational independence means:

$$\{ (s_n|\text{Var}_S, t_n|\text{Var}_P) : s_n \leftarrow D_n, t_n \leftarrow [P]_n(s_n) \} \approx$$

$$\{ (s_n|\text{Var}_S, t'_n|\text{Var}_P) : s_n, s'_n \leftarrow D_n, t'_n \leftarrow [P]_n(s'_n) \}$$
Programs running in polynomial time

This def. is good for programs running in expected polynomial time.

If a program leaks information only after exponentially long time, then the previous definition still considers it insecure.

Let \( P^\ell \) be a program that makes at most \( \ell(n) \) steps of \( P \). If \( P \) has not stopped, then \( P^\ell \) stops in a special state \( \bot \). (\( \ell \) — a polynomial)

\( P^\ell \) can be expressed in the WHILE-language.

The rewrite of \( P \) to \( P^\ell \) is quite simple.

\[ P \text{ is secure} \iff \forall \ell : P^\ell \text{ is secure.} \]
Timing-insensitive def.

Definition on previous slide is timing-sensitive.
- This is good.

Sometimes we do not want timing sensitivity.
- Good timing-sensitive analyses are hard to construct.
- Timing issues seem to be orthogonal to computational issues.

P is secure \iff \exists \ell_0 \forall \ell \geq \ell_0 : P^\ell \text{ is secure.}

To analyse P, we analyse P^\ell.
- ...but the number of executed steps is only checked at loop heads.
Having secure information flow is uncomputable in general.

Description of inputs — whatever is known about $D$.

... and expressible in the domain of the analysis.
Domain of the analysis

Analysis maps the description of the input distribution to the description of the output distribution.

Description of $D = \{D_n\}_{n \in \mathbb{N}}$ is $(\mathcal{X}, \mathcal{K}) \in \mathcal{P}(\mathcal{P}(\text{Var}) \times \mathcal{P}(\text{Var})) \times \mathcal{P}(\text{Var})$.

- $(X, Y) \in \mathcal{X}$, if $X$ and $Y$ are independent in $D$.
- $k \in \mathcal{K}$, if (the value of) $k$ is distributed like a key.

Assume the program does not change the variables in $\text{Var}_S$.

If $(\text{Var}_S, \text{Var}_P) \in \mathcal{X}_{\text{output}}$, then the program has secure information flow.

The analysis is defined inductively over the program structure.
Example: analysing assignments

Consider the program $x := o(x_1, \ldots, x_k)$.

If $(X \cup \{x_1, \ldots, x_k\}, Y) \in X_{\text{input}}$

then $(X \cup \{x_1, \ldots, x_k, x\}, Y) \in X_{\text{output}}$. 
Let $k$ be distributed like a key in $D_{\text{input}}$. 

- Consider the program $l := k + 1$. Then $\{l\}$ is not independent of $\{k\}$ in $D_{\text{output}}$. 

- Consider the program $x := \text{Enc}(k, y)$. Then $\{x\}$ is not independent of $\{k\}$ in $D_{\text{output}}$. 

- To check whether $x$ and $k$ come from the same or from different samples of $D_{\text{output}}$, try to decrypt $x$ with $k$.

These two cases should be distinguished as $l$ is usable for decryption but $x$ is not.
Encrypting black boxes

Let \( k \in \text{Var} \). Let \( S_n \) be a program state.

- \( S_n([k]\mathcal{E}) \) denotes a black box that encrypts with \( k \). I.e.
  - \( S_n([k]\mathcal{E}) \) has an input tape and an output tape;
  - When a bit-string \( w \) is written on the its tape,

\[
\begin{align*}
\mathcal{E}_{\text{nc}}(S_n(k), w)
\end{align*}
\]

is invoked and the result written to the output tape.

- Indistinguishability can be defined for distributions over black boxes.
  - Independence can be defined, too.

- Security of \( (\mathcal{G}_{\text{en}}, \mathcal{E}_{\text{nc}}) \) is defined as the indistinguishability of certain black boxes.
Security of encryption

- \((G, E)\) is secure against CPA, iff

\[
\{ [E_k(\cdot) : k \leftarrow G] \} \approx \{ [E_k(0) : k \leftarrow G] \}
\]

- \((G, E)\) is which-key concealing, iff

\[
\{ ([E_k(\cdot), E_{k'}(\cdot)] : k, k' \leftarrow G] \} \approx \{ ([E_k(\cdot), E_k(\cdot)] : k \leftarrow G] \}
\]

\([\text{Gen}, [\text{Enc}]]\) must satisfy both.
Modified domain of the analysis

Let \( \widetilde{\text{Var}} = \text{Var} \uplus \{ [x]_\mathcal{E} : x \in \text{Var} \} \).

Description of a distribution \( D \) is

\[
(\mathcal{X}, \mathcal{K}) \in \mathcal{P}(\mathcal{P}(\widetilde{\text{Var}}) \times \mathcal{P}(\widetilde{\text{Var}})) \times \mathcal{P}(\text{Var}) .
\]

\( (X, Y) \in \mathcal{X} \) if \( X \) and \( Y \) are independent in \( D \).

\( k \in \mathcal{K} \), if the distribution of \([k]_\mathcal{E}\) according to \( D \) is indistinguishable from \([\mathcal{E}nc]_k(\cdot)\).
Analysing encryptions

Consider the program \( x := \text{Enc}(k, y) \).

If \((X, Y) \in X_{\text{input}}\)

and \(k \in K_{\text{input}}\)

and \(\{[k]_{\mathcal{E}}\}, X \cup Y \cup \{y\}\) \(\in X_{\text{input}}\)

then \((X \cup \{x\}, Y) \in X_{\text{output}}\).

Generally \((\{[k]_{\mathcal{E}}\}, \{[k]_{\mathcal{E}}\}) \in X_{\text{input}}, \text{ hence } (\{x\}, \{[k]_{\mathcal{E}}\}) \in X_{\text{output}}.\)

If we have a program \( l := k + 1 \), then \((\{1\}, \{[k]_{\mathcal{E}}\}) \not\in X_{\text{output}}.\)
On security def. of encryptions

In the definition a system is considered, consisting of
- the adversary,
- the encrypting black box,
- ...

The key is inside the black box.
- I.e. the usage of the key is quite restricted.

Programming language puts no restrictions on the usage of the variable containing the key.

Requirement \( (\{ [k] \}, X \cup Y \cup \{ y \}) \in X_{\text{input}} \) gives the necessary restrictions.
Analysing key generations

Consider the program $k := Gen()$.

If $(X, Y) \in \mathcal{X}_{\text{input}}$

then $k \in \mathcal{K}_{\text{output}}$

and $(X \cup \{[k]_{\mathcal{E}}\}, Y \cup \{[k]_{\mathcal{E}}\}) \in \mathcal{X}_{\text{output}}$. 
Analysing if-then-else

Consider the program \( \text{if } b \text{ then } P_1 \text{ else } P_2 \).

- Let \( \{x_1, \ldots, x_k\} = \text{Var}_{\text{asgn}} \subseteq \text{Var} \) be the set of variables assigned to in \( P_1 \) and \( P_2 \).

- Let \( \text{Var}' = \text{Var} \cup \{N, x_1^{\text{true}}, \ldots, x_k^{\text{true}}, x_1^{\text{false}}, \ldots, x_k^{\text{false}}\} \)

- Program at right has the same functionality.

- \( P_1^{\text{true}} \) is \( P_1 \), where each \( x_i \) is replaced with \( x_i^{\text{true}} \).

- Similarly for \( P_2^{\text{false}} \).
Consider the program $x := b \ ? \ y : z$. Let $y, z \in \mathcal{K}_{\text{input}}$.

- If $(X, Y) \in \mathcal{X}_{\text{input}}$ and $(\{y\}_E, \{z\}_E, X \cup Y \cup \{b\}) \in \mathcal{X}_{\text{input}}$
  then $(X \cup \{x\}_E, Y \cup \{x\}_E) \in \mathcal{X}_{\text{output}}$.

- If $(\{y\}_E, \{z\}_E, \{b\}) \in \mathcal{X}_{\text{input}}$ then $x \in \mathcal{K}_{\text{output}}$.

$(X_1, \ldots, X_k) \in \mathcal{X}$ means

$(X_1, X_2) \in \mathcal{X}$
$(X_1 \cup X_2, X_3) \in \mathcal{X}$

\[ \cdots \]
$(X_1 \cup \cdots \cup X_{k-1}, X_k) \in \mathcal{X}$
Analysing loops

Consider the program \( \text{while } b \text{ do } P \). Its analysis is the repeated application of the analysis of

\[
\text{if } b \text{ then } P \text{ else skip}
\]

It stabilises due to finiteness of the domain and monotonicity of the analysis.
Active adversaries — problem statement

$M$ remains confidential if for all adversaries $\mathcal{A}$, the adversary’s experience is independent of $M$. 

Diagram: 
- Secret $M$ originates from $\mathcal{M}$ and flows to $S$.
- $S$ is connected to $A$ and $B$.
- $A$ and $B$ exchange information.
- $\mathcal{A}$ observes the interactions.
Language for protocols

A party is a sequence of statements. Statements are:

- \( k := Gen \)
- \( x := \text{random} \)
- \( x := (y_1, \ldots, y_m) \)
- \( y := \pi_i^m(x) \)
- \( x := \text{encr}_k(y) \)
- \( y := \text{decr}_k(x) \)
- \( \text{send } x \)
- \( x := \text{receive} \)
- \( \text{check}(x = y) \)

Protocol is a set of parties.

Some additional statements (generation of long-term keys) are done at the very beginning of execution.

Each variable may occur at LHS at most once.
Semantics

Protocol runs in parallel with the adversary.

- Adversary takes care of message forwarding.
- If something goes wrong during the execution of a party, then this party becomes stuck.
  - \( \text{check}(x = y) \) returns false;
  - operand types do not match the operator;
  - a message does not decrypt.

- Parties execute one statement at a time, the adversary does the scheduling.
  - When a party gets stuck, the adversary is not notified immediately.
Adversary’s experience

- Adversary learns the values of the variables $x$, where send $x$ is a statement in some party.
- No timing information is available, because the adversary schedules.
- Therefore there is again a set of public variables $\text{Var}_P$, whose values make up the entire experience.

$$\text{Var}_P = \{x \mid \text{some party contains send } x\}$$
Denning-style analysis

Suppose a statement $x := O(x_1, \ldots, x_m)$ occurs in some party.

- $x_1, \ldots, x_m$ are all variables occurring in RHS.
- $O$ can be any operation — tupling, projection, decryption, encryption.

There is information flow from $x_i$ to $x$.

- Denote $x_i \Rightarrow x$.

Protocol is insecure, if $M \Rightarrow^* x$ for some $x \in \text{Var}_P$.
- Otherwise it is secure.

An extremely conservative analysis.
Security against CCA

Encryption system \((\mathcal{G}, \mathcal{E}, \mathcal{D})\) is secure against CCA, if

\[
\left\{ \left( \mathcal{E}_k(\cdot), \mathcal{D}_k(\cdot) \right) : k \leftarrow \mathcal{G} \right\}
\]

is indistinguishable from

\[
\left\{ \left( \mathcal{E}_k(0), \mathcal{D}_k(\cdot) \right) : k \leftarrow \mathcal{G} \right\}
\]

by all adversaries that do not give the output of the left black box to the right black box.
Main idea

- Replace statements $x := \text{encr}_k(y)$ with statements $x := \text{encr}_k(Z)$, where $[Z] = 0$.

- $Z$ is a new variable.

- This makes the information flow relation $\Rightarrow$ sparser.

- The replacement is valid only when certain conditions are satisfied.

- Valid $\equiv$ does not change the adversary’s experience.
Conditions for replacing

When replacing the statement $x := \text{en}cr_k(y)\ldots$

- We must know exactly, where else the key $k$ is used.
  - The same key may occur under different names.
  - To find it out, we symbolically execute the protocol.

- When computing the values of the variables in $\text{Var}_P$, the key $k$ may only be used to encrypt and decrypt.

- We may not decrypt the ciphertexts created with key $k$.
  - We achieve this with a program transformation.
Symbolic execution of protocols

We assign a term to each variable. They terms $T$ are

- $\text{const}(x)$
- $\text{tuple}^m(T_1, \ldots, T_m)$
- $\text{secret}(M)$
- $\pi^m_i(T)$
- $\text{key}(x)$
- $\text{enctr}(l, T_k, T_y)$
- $\text{key}(l)$
- $\text{decr}(T_k, T_y)$
- $\text{random}(l)$
- $\text{received}(l)$
- $\text{stuck}$

$l$ — statement label.

$(x)$ is assigned to the variable $x$ that is initialised before the run of the protocol.

There are some obvious simplification rules.
Symbolic execution of *Check-s*

There are some rules telling us, when the bit-strings corresponding to two terms are certainly different.

For \( \text{check}(x = y) \), we check whether terms assigned to \( x \) and \( y \) are certainly different.

- If yes, the protocol party is stuck.
- If no, then we replace the more complex term with the simpler one everywhere.

Complexity is the same as size.

But: the terms containing subterms \( \text{received}(l) \) are the most complex.

We consider the key corresponding to \( \text{key}(l) \) to be used exactly where the subterm \( \text{key}(l) \) occurs.
Replacing decryptions

Let $k$ be used for encryption at statements

\[ x_1 := encr_{k_1}(y_1), \ldots, x_m := encr_{k_m}(y_m) \]

Replace $z := decr_k(w)$ by

\[ z := case \ w \ of \]
\[ x_1 \rightarrow y_1 \]
\[ \ldots \ldots \ldots \ldots \]
\[ x_m \rightarrow y_m \]
\[ else \rightarrow decr_k(w) \]

No change to adversary's view
Semantics of *case*-constructs

- $z$ is assigned the first $y_i$, where $x_i$ matches $w$.
- If this $y_i$ has not been defined yet, then the protocol party gets stuck.
  - This never happens in our transformed protocols.
- A yet undefined $x_i$ never matches.
Ciphertext integrity

An encryption system $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ has ciphertext integrity, if:

No PPT algorithm $\mathcal{A}$ with access to oracles $\mathcal{E}_k(\cdot)$ and $\mathcal{D}_k(\cdot)$ can submit to $\mathcal{D}_k(\cdot)$ a bit-string $y$, such that

- $\mathcal{D}_k(y)$ exists, i.e. $y$ is a valid ciphertext;
- $y$ was not an output of $\mathcal{E}_k(\cdot)$.

i.e. we need no else-clause.

If nothing matches in a case-statement, then the protocol party gets stuck.

The replacement — wrap-up

- Do the symbolic execution.
- Choose a key $\text{key}(x)$ or $\text{key}(l)$, such that
  - In terms assigned to $y \in \text{Var}_P$, this $\text{key}(\ldots)$ occurs only as the key in en-/decryptions.
- Replace the decryption statements $z := \text{decr}_k(y)$, where the term assigned to $k$ is this $\text{key}(\ldots)$.
  - Replace them with $\text{case}$-statements.
- Replace the encryption statements $x := \text{encr}_k(y)$, where the term assigned to $k$ is this $\text{key}(\ldots)$.
  - Replace them with $x := \text{encr}_k(Z)$.
Getting rid of *case*-statements

\[ z := \text{case } w \text{ of } x_1 \to y_1 \cdots x_m \to y_m, \]

where

\[ x_1 := \text{encr}_{k_1}(y_1), \ldots, \quad x_m := \text{encr}_{k_m}(y_m) \]

is replaced by

\[
\begin{align*}
\text{wait}(s) \\
\text{check}(w = x_i) \\
z := y_i
\end{align*}
\]

and \textit{signal}(s) is added after \( x_i := \text{encr}_{k_i}(y_i) \).

\( i \) is chosen non-deterministically (we get \( m \) new protocols).

\( s \) is a new semaphore.

Executing \textit{wait}(s) before \textit{signal}(s) gets stuck.
Handling *wait*-s and *signal*-s

- In the next round, the symbolic execution must proceed in an order consistent with *wait*-s and *signal*-s.
- We may have to do simultaneous symbolic execution of the parties.
- If there are cyclic dependencies, then the statements in and after the cycle are stuck.
Conclusions

- Cryptographic effects can be faithfully abstracted away.
- Resulting analyses are not overwhelmingly complex.
Future work

- Track the keys in the first analysis presented.
- Do not track the keys in an analysis with active adversaries.
  - Assume that keys are never sent out.
- More expressive language for the second analysis.
- More cryptographic primitives.
  - Public key encryption, digital signatures, ... 
- Other security properties. (Integrity)
- Different security definitions for cryptographic primitives.
  - Encryption as a PRP ...
- One-way functions.
  - New confidentiality definition is necessary.