Confidentiality analyses correct wrt. computational semantics

Peeter Laud

peeter_l@ut.ee

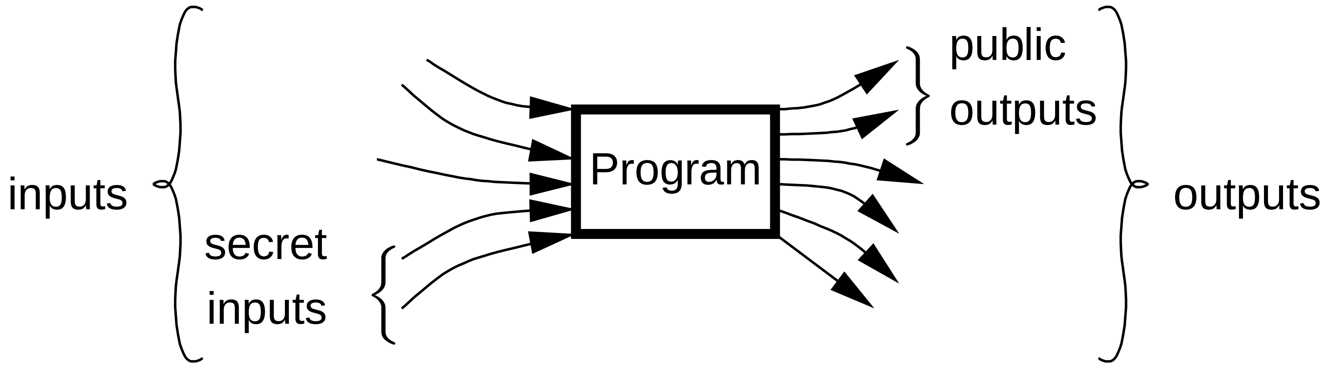
Tartu Ülikool

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).

$$\begin{array}{lcl} \mathsf{P} & ::= & \mathtt{x} := o(\mathtt{x}_1, \ldots, \mathtt{x}_k) \\ & \mid & \mathit{skip} \\ & \mid & \mathsf{P}_1; \mathsf{P}_2 \\ & \mid & \mathit{if}\ \mathtt{b}\ \mathit{then}\ \mathsf{P}_1\ \mathit{else}\ \mathsf{P}_2 \\ & \mid & \mathit{while}\ \mathtt{b}\ \mathit{do}\ \mathsf{P}' \end{array}$$

$\mathtt{b}, \mathtt{x}, \mathtt{x}_1, \ldots, \mathtt{x}_k \in \mathbf{Var}$. $o \in \mathbf{Op}$. $\mathcal{E}nc, \mathcal{G}en \in \mathbf{Op}$.

Programming language — semantics

Denotational semantics: $[\![\mathrm{P}]\!] : \mathbf{State} \to \mathbf{State}_\perp$.

$\mathbf{State} = \mathbf{Var} \to \mathbf{Val}$.

$\mathbf{State}_\perp$ has an extra element $\perp$, denoting nontermination.

For each $o \in \mathbf{Op}$ with arity k, a function $[\![o]\!] : \mathbf{Val}^k \to \mathbf{Val}$ 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:

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

Also, $[\![o]\!] = \{[\![o]\!]_n\}_{n\in\mathbb{N}}$, $[\![o]\!]_n : \mathbf{Val}_n^k \rightarrow \mathcal{D}(\mathbf{Val}_n)$.

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 $\mathbf{Var}_S \subseteq \mathbf{Var}$.
- Public outputs — final values of variables in $\mathbf{Var}_P \subseteq \mathbf{Var}$.

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

$$\{\!|(s_n|_{\mathbf{Var}_S}, t_n|_{\mathbf{Var}_P}) \;:\; s_n \leftarrow D_n, t_n \leftarrow [\![\mathsf{P}]\!]_n(s_n)|\!\} \approx \{\!|(s_n|_{\mathbf{Var}_S}, t'_n|_{\mathbf{Var}_P}) \;:\; s_n, s'_n \leftarrow D_n, t'_n \leftarrow [\![\mathsf{P}]\!]_n(s'_n)|\!\}$$

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^ℓ be a program that makes at most $\ell(n)$ steps of P. If P has not stopped, then P^ℓ stops in a special state $\perp$. (ℓ — a polynomial)
- P^ℓ can be expressed in the WHILE-language.
  - The rewrite of P to P^ℓ is quite simple.

$$\mathsf{P}\text{ is secure} :\Longleftrightarrow \forall \ell : \mathsf{P}^\ell \text{ is secure.}$$

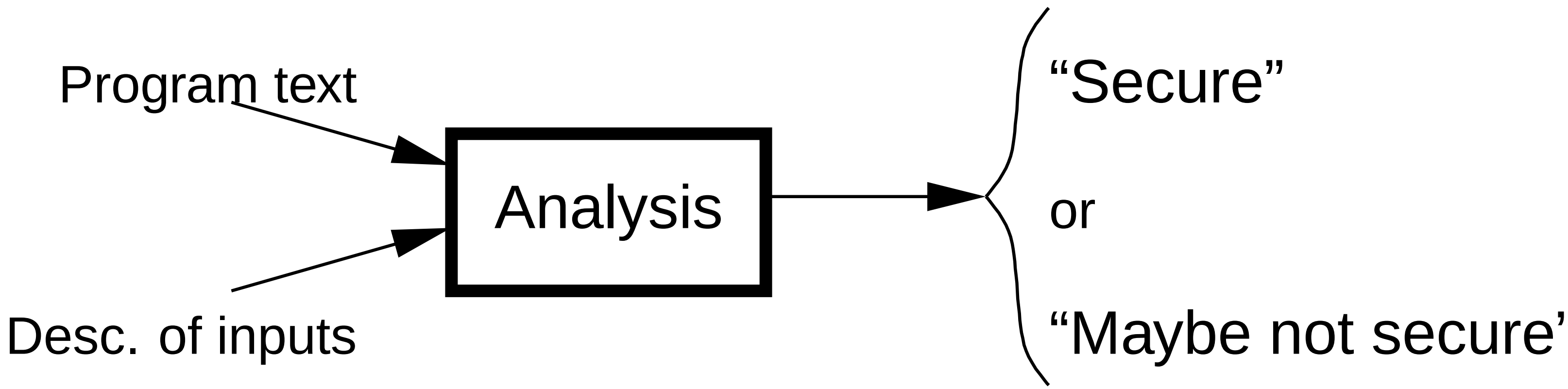
Timing-insensitive def.

- 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.

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

- To analyse P, we analyse P^{ℓ}.
  - ... but the number of executed steps is only checked at loop heads.

Program analysis's approach



- 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 $(\mathcal{X}, \mathcal{K}) \in \mathcal{P}(\mathcal{P}(\mathbf{Var}) \times \mathcal{P}(\mathbf{Var})) \times \mathcal{P}(\mathbf{Var})$.
  - $(X, Y) \in \mathcal{X}$, if X and Y are independent in D.
  - $\mathtt{k} \in \mathcal{K}$, if (the value of) $\mathtt{k}$ is distributed like a key.
- Assume the program does not change the variables in $\mathbf{Var}_S$.
- If $(\mathbf{Var}_S, \mathbf{Var}_P) \in \mathcal{X}_{\mathrm{output}}$, then the program has secure information flow.
- The analysis is defined inductively over the program structure.

Example: analysing assignments

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

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

then $(X \cup \{\mathtt{x_1}, \ldots, \mathtt{x_k}, \mathtt{x}\}, Y) \in \mathcal{X}_{\text{output}}$.

Analysing encryptions — problems

Let $\mathtt{k}$ be distributed like a key in D_{input}.

- Consider the program $\mathtt{l} := \mathtt{k} + 1$.

  Then $\{\mathtt{l}\}$ is not independent of $\{\mathtt{k}\}$ in D_{output}.

- Consider the program $\mathtt{x} := \mathcal{E}nc(\mathtt{k}, \mathtt{y})$.

  Then $\{\mathtt{x}\}$ is not independent of $\{\mathtt{k}\}$ in D_{output}.

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

These two cases should be distinguished as $\mathtt{l}$ is usable for decryption but $\mathtt{x}$ is not.

Encrypting black boxes

- Let $\mathtt{k} \in \mathbf{Var}$. Let S_n be a program state.
- $S_n([\mathtt{k}]_{\mathcal{E}})$ denotes a black box that encrypts with $\mathtt{k}$. I.e.
  - $S_n([\mathtt{k}]_{\mathcal{E}})$ has an input tape and an output tape;
  - When a bit-string w is written on the its tape,

$$[\![\mathcal{E}nc]\!]_n(S_n(\mathtt{k}), w)$$

    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 $([\![\mathcal{G}en]\!], [\![\mathcal{E}nc]\!])$ is defined as the indistinguishability of certain black boxes.

Security of encryption

- $(\mathcal{G}, \mathcal{E})$ is secure against CPA, iff

$$\{\!|\, \boxed{\mathcal{E}_k(\cdot)} \;:\; k \leftarrow \mathcal{G} \,|\!\} \approx \{\!|\, \boxed{\mathcal{E}_k(\mathbf{0})} \;:\; k \leftarrow \mathcal{G} \,|\!\}$$

- $(\mathcal{G}, \mathcal{E})$ is which-key concealing, iff

$$\{\!|\, (\boxed{\mathcal{E}_k(\cdot)}, \boxed{\mathcal{E}_{k'}(\cdot)}) \;:\; k, k' \leftarrow \mathcal{G} \,|\!\} \approx \{\!|\, (\boxed{\mathcal{E}_k(\cdot)}, \boxed{\mathcal{E}_k(\cdot)}) \;:\; k \leftarrow \mathcal{G} \,|\!\}$$

$([\![\mathcal{G}en]\!], [\![\mathcal{E}nc]\!])$ must satisfy both.

Modified domain of the analysis

- Let $\widetilde{\mathbf{Var}} = \mathbf{Var} \uplus \{[\mathtt{x}]_{\mathcal{E}} \,:\, \mathtt{x} \in \mathbf{Var}\}$.
- Description of a distribution D is

$$(\mathcal{X}, \mathcal{K}) \in \mathcal{P}(\mathcal{P}(\widetilde{\mathbf{Var}}) \times \mathcal{P}(\widetilde{\mathbf{Var}})) \times \mathcal{P}(\mathbf{Var}) \quad .$$

  - $(X, Y) \in \mathcal{X}$ if X and Y are independent in D.
  - $\mathtt{k} \in \mathcal{K}$, if the distribution of $[\mathtt{k}]_{\mathcal{E}}$ according to D is indistinguishable from $\boxed{[\![\mathcal{E}nc]\!]_k(\cdot)}$.

Analysing encryptions

Consider the program $\mathtt{x} := \mathcal{E}nc(\mathtt{k}, \mathtt{y})$.

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

and $\mathtt{k} \in \mathcal{K}_{\text{input}}$

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

then $(X \cup \{\mathtt{x}\}, Y) \in \mathcal{X}_{\text{output}}$.

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

If we have a program $\mathtt{l} := \mathtt{k} + 1$, then $(\{\mathtt{l}\}, \{[\mathtt{k}]_{\mathcal{E}}\}) \notin \mathcal{X}_{\text{output}}$.

On security def. of encryptions

- In the definition a system is considered, consisting of
  - the adversary,
  - the encrypting black box,
  - ...
- The key is <u>inside</u> 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 $(\{[\mathtt{k}]_{\mathcal{E}}\}, X \cup Y \cup \{\mathtt{y}\}) \in \mathcal{X}_{\mathrm{input}}$ gives the necessary restrictions.

Analysing key generations

Consider the program $\texttt{k} := \mathcal{G}en()$.

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

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

and $(X \cup \{[\texttt{k}]_{\mathcal{E}}\}, Y \cup \{[\texttt{k}]_{\mathcal{E}}\}) \in \mathcal{X}_{\text{output}}$.

Analysing if-then-else

Consider the program *if* $\mathtt{b}$ *then* P_1 *else* P_2.

- Let $\{\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{k}\} = \mathbf{Var}_{\text{asgn}} \subseteq \mathbf{Var}$ be the set of variables assigned to in P_1 and P_2.
- Let $\mathbf{Var}' = \mathbf{Var} \,\dot\cup\, \{N, \mathtt{x}_1^{\mathsf{true}}, \ldots, \mathtt{x}_\mathtt{k}^{\mathsf{true}}, \mathtt{x}_1^{\mathsf{false}}, \ldots, \mathtt{x}_\mathtt{k}^{\mathsf{false}}\}$
- Program at right has the same functionality.
- $\mathsf{P}_1^{\mathsf{true}}$ is P_1, where each $\mathtt{x}_\mathtt{i}$ is replaced with $\mathtt{x}_\mathtt{i}^{\mathsf{true}}$.
- Similarly for $\mathsf{P}_2^{\mathsf{false}}$.

$\mathtt{N} := \mathtt{b}$
$\mathtt{x}_1^{\mathsf{true}} := \mathtt{x}_1$
$\mathtt{x}_1^{\mathsf{false}} := \mathtt{x}_1$
...
$\mathtt{x}_\mathtt{k}^{\mathsf{true}} := \mathtt{x}_\mathtt{k}$
$\mathtt{x}_\mathtt{k}^{\mathsf{false}} := \mathtt{x}_\mathtt{k}$
$\mathsf{P}_1^{\mathsf{true}}$
$\mathsf{P}_2^{\mathsf{false}}$
$\mathtt{x}_1 := \mathtt{N} \,?\, \mathtt{x}_1^{\mathsf{true}} : \mathtt{x}_1^{\mathsf{false}}$
...
$\mathtt{x}_\mathtt{k} := \mathtt{N} \,?\, \mathtt{x}_\mathtt{k}^{\mathsf{true}} : \mathtt{x}_\mathtt{k}^{\mathsf{false}}$

Analysing $?:$

Consider the program $\mathtt{x} := \mathtt{b}\,?\,\mathtt{y} : \mathtt{z}$. Let $\mathtt{y}, \mathtt{z} \in \mathcal{K}_{\mathrm{input}}$.

- If $(X, Y) \in \mathcal{X}_{\mathrm{input}}$ and $(\{[\mathtt{y}]_{\mathcal{E}}\}, \{[\mathtt{z}]_{\mathcal{E}}\}, X \cup Y \cup \{\mathtt{b}\}) \in \mathcal{X}_{\mathrm{input}}$ then $(X \cup \{[\mathtt{x}]_{\mathcal{E}}\}, Y \cup \{[\mathtt{x}]_{\mathcal{E}}\}) \in \mathcal{X}_{\mathrm{output}}$.
- If $(\{[\mathtt{y}]_{\mathcal{E}}, [\mathtt{z}]_{\mathcal{E}}\}, \{\mathtt{b}\}) \in \mathcal{X}_{\mathrm{input}}$ then $\mathtt{x} \in \mathcal{K}_{\mathrm{output}}$.

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

$$
\begin{gathered}
(X_1, X_2) \in \mathcal{X} \\
(X_1 \cup X_2, X_3) \in \mathcal{X} \\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
(X_1 \cup \cdots \cup X_{k-1}, X_k) \in \mathcal{X}
\end{gathered}
$$

Analysing loops

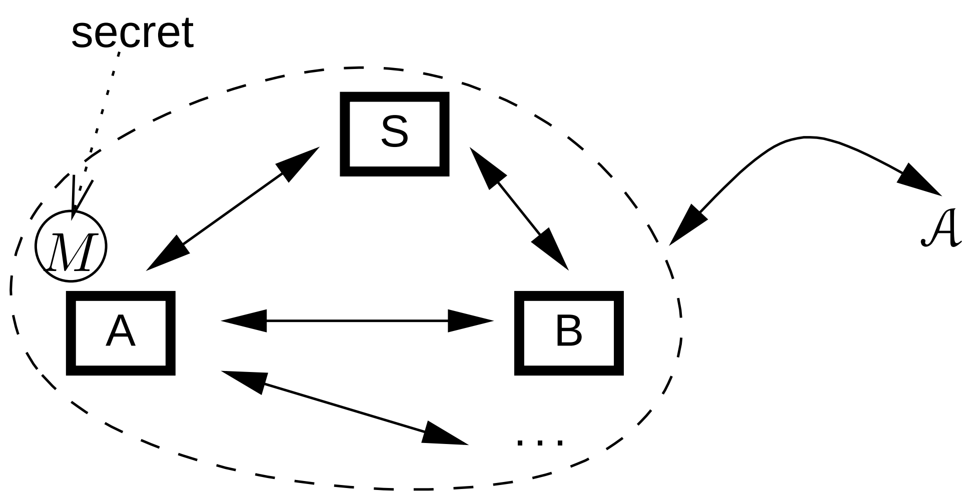
Consider the program $\textit{while}$ `b` $\textit{do}$ P.

Its analysis is the repeated application of the analysis of

$$\textit{if}\ \mathtt{b}\ \textit{then}\ \mathsf{P}\ \textit{else}\ \textit{skip}$$

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.

Language for protocols

- A party is a sequence of statements. Statements are:

$$\begin{array}{ll}
\mathtt{k} := \mathcal{G}en & \mathtt{x} := \mathbf{random} \\
\mathtt{x} := (\mathtt{y}_1, \ldots, \mathtt{y}_\mathtt{m}) & \mathtt{y} := \pi_i^m(\mathtt{x}) \\
\mathtt{x} := encr_\mathtt{k}(\mathtt{y}) & \mathtt{y} := decr_\mathtt{k}(\mathtt{x}) \\
\mathbf{send}\ \mathtt{x} & \mathtt{x} := \mathbf{receive} \\
\mathbf{check}(\mathtt{x} = \mathtt{y}) &
\end{array}$$

- 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.
  - $\mathbf{check}(\mathrm{x} = \mathrm{y})$ 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 $\mathtt{x}$, where $\mathbf{send}\ \mathtt{x}$ is a statement in some party.
- No timing information is available, because the adversary schedules.
- Therefore there is again a set of public variables $\mathbf{Var}_P$, whose values make up the entire experience.

$$\mathbf{Var}_P = \{\mathtt{x} \mid \text{some party contains } \mathbf{send}\ \mathtt{x}\}$$

Denning-style analysis

- Suppose a statement $\mathtt{x} := O(\mathtt{x_1}, \ldots, \mathtt{x_m})$ occurs in some party.
  - $\mathtt{x_1}, \ldots, \mathtt{x_m}$ are <u>all</u> variables occuring in RHS.
  - O can be any operation — tupling, projection, decryption, encryption.
- There is information flow from $\mathtt{x_i}$ to $\mathtt{x}$.
  - Denote $\mathtt{x_i} \Rightarrow \mathtt{x}$.
- Protocol is insecure, if $M \overset{*}{\Rightarrow} \mathtt{x}$ for some $\mathtt{x} \in \mathbf{Var}_P$.
  - Otherwise it is secure.

An extremely conservative analysis.

Security against CCA

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

$$\{\!\!\{(\boxed{\mathcal{E}_k(\cdot)}, \boxed{\mathcal{D}_k(\cdot)}) \ : \ k \leftarrow \mathcal{G}\}\!\!\}$$

is indistinguishable from

$$\{\!\!\{(\boxed{\mathcal{E}_k(\mathbf{0})}, \boxed{\mathcal{D}_k(\cdot)}) \ : \ k \leftarrow \mathcal{G}\}\!\!\}$$

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

Main idea

- Replace statements $\mathtt{x} := encr_{\mathtt{k}}(\mathtt{y})$ with statements $\mathtt{x} := encr_{\mathtt{k}}(\mathtt{Z})$, where $[\![\mathtt{Z}]\!] = \mathbf{0}$.
  - $\mathtt{Z}$ 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 $\mathtt{x} := encr_{\mathtt{k}}(\mathtt{y})\ldots$

- We must know exactly, where else the key $\mathtt{k}$ 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 $\mathbf{Var}_P$, the key $\mathtt{k}$ may only be used to encrypt and decrypt.
- We may not decrypt the ciphertexts created with key $\mathtt{k}$.
  - We achieve this with a program transformation.

Symbolic execution of protocols

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

$$\begin{array}{ll}
\underline{const}(\mathtt{x}) & \underline{tuple^m}(T_1, \ldots, T_m) \\
\underline{secret}(\mathtt{M}) & \underline{\pi_i^m}(T) \\
\underline{key}(\mathtt{x}) & \underline{encr}(l, T_{\mathrm{k}}, T_{\mathrm{y}}) \\
\underline{key}(l) & \underline{decr}(T_{\mathrm{k}}, T_{\mathrm{y}}) \\
\underline{random}(l) & \underline{received}(l) \\
\underline{stuck} &
\end{array}$$

- l — statement label.
- $\underline{\ldots}(\mathtt{x})$ is assigned to the variable $\mathtt{x}$ 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 $\mathbf{check}(\mathrm{x} = \mathrm{y})$, 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 $\underline{received}(l)$ are the most complex.

We consider the key corresponding to $\underline{key}(l)$ to be used exactly where the subterm $\underline{key}(l)$ occurs.

Replacing decryptions

Let $\mathtt{k}$ be used for encryption at statements

$$\mathtt{x}_1 := encr_{\mathtt{k}_1}(\mathtt{y}_1), \quad \ldots, \quad \mathtt{x}_\mathtt{m} := encr_{\mathtt{k}_\mathtt{m}}(\mathtt{y}_\mathtt{m})$$

Replace $\mathtt{z} := decr_{\mathtt{k}}(\mathtt{w})$ by

$$\begin{array}{l} \mathtt{z} := case\ \mathtt{w}\ of \\ \qquad \begin{array}{ll} \mathtt{x}_1 & \longrightarrow \mathtt{y}_1 \\ \ldots\ldots\ldots\ldots\ldots & \\ \mathtt{x}_\mathtt{m} & \longrightarrow \mathtt{y}_\mathtt{m} \\ else & \longrightarrow decr_{\mathtt{k}}(\mathtt{w}) \end{array} \end{array}$$

No change to adversary's view

Semantics of *case*-constructs

- $\mathtt{z}$ is assigned the first $\mathtt{y_i}$, where $\mathtt{x_i}$ matches $\mathtt{w}$.
- If this $\mathtt{y_i}$ has not been defined yet, then the protocol party gets stuck.
  - This never happens in our transformed protocols.
- A yet undefined $\mathtt{x_i}$ 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 $\boxed{\mathcal{E}_k(\cdot)}$ and $\boxed{\mathcal{D}_k(\cdot)}$ can submit to $\boxed{\mathcal{D}_k(\cdot)}$ a bit-string y, such that

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

i.e. we need no $else$-clause.

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

See [Bellare and Namprempre, ASIACRYPT 2000] for constructions of encryption primitives.

The replacement — wrap-up

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

Getting rid of *case*-statements

$$\mathtt{z} := \mathit{case}\ \mathtt{w}\ \mathit{of}\ \mathtt{x}_1 \to \mathtt{y}_1\ \cdots\ \mathtt{x}_\mathtt{m} \to \mathtt{y}_\mathtt{m},$$

where

$$\mathtt{x}_1 := \mathit{encr}_{\mathtt{k}_1}(\mathtt{y}_1), \quad \ldots, \quad \mathtt{x}_\mathtt{m} := \mathit{encr}_{\mathtt{k}_\mathtt{m}}(\mathtt{y}_\mathtt{m})$$

is replaced by

$$\begin{array}{l} \mathit{wait}(s) \\ \mathbf{check}(\mathtt{w} = \mathtt{x}_\mathtt{i}) \\ \mathtt{z} := \mathtt{y}_\mathtt{i} \end{array}$$

and $\mathit{signal}(s)$ is added after $\mathtt{x}_\mathtt{i} := \mathit{encr}_{\mathtt{k}_\mathtt{i}}(\mathtt{y}_\mathtt{i})$.

i is chosen nondeterministically (we get m new protocols).
s is a new semaphore.

Executing $\mathit{wait}(s)$ before $\mathit{signal}(s)$ 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.