Computationally Secure Information Flow

Promotionskolloquium, 16.09.2002

Peeter Laud
peeter_l@ut.ee

Universität des Saarlandes
Tartu Ülikool
Cybernetica AS
Structure of the talk

- Background
  - What the problem is, how could we handle it.

- Problem statement
  - What to protect against, definitions.

- Our contribution
  - Program analysis for computationally secure information flow.

- Using a weaker cryptographic primitive

- Conclusions
Background

Programs may
- run in networked computers;
- access confidential data;
- communicate with other programs over the network.
  - some of them may be hostile.
⇒ leak confidential data.
Programs may
- run in networked computers;
- access confidential data;
- communicate with other programs over the network.
  - some of them may be hostile.

⇒ leak confidential data.

How can we find out, whether a program may leak confidential data?

- Cannot test for it.
  - One can test for properties of program runs.
Background

Programs may

- run in networked computers;
- access confidential data;
- communicate with other programs over the network.
  - some of them may be hostile.

⇒ leak confidential data.

How can we find out, whether a program may leak confidential data?

- Cannot test for it.
  - One can test for properties of program runs.
  - Confidentiality — all program runs are similar.
Program Analysis

- Analyse the text of the program.
  - Try to prove that it preserves confidentiality.
- Try to automate the analysis.
- The question of preserving confidentiality is uncomputable.
Program Analysis

- Analyse the text of the program.
  - Try to prove that it preserves confidentiality.
- Try to automate the analysis.
- The question of preserving confidentiality is uncomputable.
- An automatic analysis must have
  - False positives — labeling a secure program insecure.
    - inconvenient, but causes no leaks.
  - False negatives — labeling an insecure program secure.
    - unsafe.
Program Analysis

Devise an analysis with no false negatives:

Reality:

\[
\text{secure} \quad | \quad \text{insecure}
\]

Set of programs

Analysis:

\[
\text{definitely secure} \quad | \quad \text{may be insecure}
\]

and with as few false positives as possible.
Structure of the talk

- **Background**
  - What the problem is, how could we handle it.

- **Problem statement**
  - What to protect against, definitions.

- **Our contribution**
  - Program analysis for computationally secure information flow.

- Using a weaker cryptographic primitive

- Conclusions
On the Attackers

Some communication partners of the program are hostile.

What are their capabilities?

The security of the program depends on them.

Two main categories of attackers:

Passive.

- Can read from the network.
- Cannot send any new data to the network.

Active.

- Can read from the network.
- Can also send data to the network.

Active attackers are stronger than passive attackers.
Only Passive Attackers

- We only consider security against passive attackers. In this case
  - The program has no dialogue with the environment.
  - The system may be modeled as follows:
    - The program is given its inputs. Some of the inputs are confidential.
    - The program processes the inputs and produces some outputs.
    - Some of these outputs are made public.
- This is the usual problem of secure information flow in programs.
- If we want to handle active attackers, we have to know, how to handle passive ones.
Illustration

The attacker does not control the inputs, which are defined by a probability distribution $D$. Secure information flow ensures the independence of secret and public outputs.
Illustration

The attacker does not control the inputs. The source of inputs defines a probability distribution $D$ on inputs. The computationally secure information flow ensures computational independence of secret inputs and public outputs.
What do Compl.-Theor. Def.s Give?

- Allow to model cryptographic primitives more intuitively.
- We use complexity-theoretic definitions of secure cryptographic primitives.
- No efficient algorithm can break the primitive.

For example — symmetric encryption: $x = E_k(y)$

- Information-theoretically: $x$ is not independent of $y$.
  - At least when $k$ is shorter than $y$.
- Computationally: $x$ is independent of $y$.
  - As long as $y$ does not depend on $k$. 
What do Compl.-Theor. Def.s Give?

- Allow to model cryptographic primitives more intuitively.
- We use complexity-theoretic definitions of secure cryptographic primitives.
- No efficient algorithm can break the primitive.

For example — symmetric encryption: $x = \mathcal{E}_k(y)$

- Information-theoretically: $x$ is not independent of $y$.
- At least when $k$ is shorter than $y$.
- Computationally: $x$ is independent of $y$.
- As long as $y$ does not depend on $k$.

Actually, the last condition is:

$y$ is independent of $\rightarrow \mathcal{E}_k \rightarrow$

Then also $x$ is independent of $\rightarrow \mathcal{E}_k \rightarrow$. 
Structure of the talk

- Background
  - What the problem is, how could we handle it.

- Problem statement
  - What to protect against, definitions.

- Our contribution
  - Program analysis for computationally secure information flow.
  - Using a weaker cryptographic primitive

- Conclusions
Our Contribution

- Definition of computationally secure information flow.
- Static program analysis for a simple imperative programming language.
  - Contains assignments (with computations in RHS)
  - sequences of statements
  - if-then-else-branches
  - while-loops
- The analysis handles symmetric encryption.
- Proof of correctness of the analysis.
  - Cannot use standard results about fix-point approximation.
- A practical implementation of the analysis.
Domain of the Analysis

Given a program $P$, the analysis
- Takes a description of the distribution of inputs.
- Returns a description of the distribution of outputs.

Description of distribution — set of pairs of variables $(X, Y)$.
- (Values of) variables in $X$ are independent of variables in $Y$.

Analysis is a function with domain and range $\mathcal{P}(\mathcal{P}(\text{Var}) \times \mathcal{P}(\text{Var}))$. 
Domain of the Analysis

- Given a program \( P \), the analysis
  - Takes a description of the distribution of inputs.
  - Returns a description of the distribution of outputs.
- Description of distribution — set of pairs of variables and encrypting black boxes (EBB) \((X, Y)\).
  - (Values of) variables and EBBs in \( X \) are independent of variables and EBBs in \( Y \).
- Analysis is a function with domain and range \( \mathcal{P}(\mathcal{P}(\text{Var} \uplus \text{Var}) \times \mathcal{P}(\text{Var} \uplus \text{Var})) \).

Actually, we also have encrypting black boxes.
Consider the statement $x = o(x_1, \ldots, x_k)$

- Let $X$ be a set of variables and EBBs.
- Suppose that $\{x_1, \ldots, x_k\}$ is independent of $X$ before the statement.
- Then $x$ is independent of $X$ after the statement.
Requirements for the Encryption

Encryption must hide the identities of plaintexts and keys:

- $\mathcal{E}$ must be *repetition-concealing*.
  - Let $x_1 = \mathcal{E}_k(y_1)$ and $x_2 = \mathcal{E}_k(y_2)$.
  - From $x_1, x_2$ impossible to find, whether $y_1 = y_2$.
  - For this, $\mathcal{E}_k$ must be probabilistic.

- $\mathcal{E}$ must be *which-key concealing*.
  - Let $x = \mathcal{E}_k(y)$ and $x' = \mathcal{E}_{k'}(y')$.
  - From $x, x'$ impossible to find, whether $k = k'$.
Requirements for the Encryption

Encryption must hide the identities of plaintexts and keys:

- $E$ must be *repetition-concealing*.
  - Let $x_1 = E_k(y_1)$ and $x_2 = E_k(y_2)$.
  - From $x_1, x_2$ impossible to find, whether $y_1 = y_2$.
  - For this, $E_k$ must be probabilistic.
  - A standard property.

- $E$ must be *which-key concealing*.
  - Let $x = E_k(y)$ and $x' = E_{k'}(y')$.
  - From $x, x'$ impossible to find, whether $k = k'$.
  - A nonstandard property.
  - Some standard constructions achieve it.
Analysing the Encryption

Consider the statement \( x = \mathcal{E}_k(y) \)

Let \( X \) be a set of variables and EBBs.

Suppose that \( \mathcal{E}_k \) is independent of \( X \cup \{y\} \) before the statement.

Note that \( y \) may be dependent of \( X \).

Then \( x \) is independent of \( X \) after the statement.

Consider the statement \( k = \text{Generate}_\text{Key}() \)

Then \( \mathcal{E}_k \) is independent of \( \mathcal{E}_k \) after the statement.
Structure of the talk

- Background
  - What the problem is, how could we handle it.

- Problem statement
  - What to protect against, definitions.

- Our contribution
  - Program analysis for computationally secure information flow.

- Using a weaker cryptographic primitive

- Conclusions
More Primitive Encryption

- Which-key and repetition concealing encryption primitives are usually constructed from more primitive operations.
- These operations are assumed to be *pseudorandom permutations* (PRP).
- Directly handling pseudorandom permutations may help efficiency.
Our Contribution

- Analysis for secure information flow for programs without loops.
  - The encryption is assumed to be a PRP.
- Additionally: means for checking, whether the outputs of two programs have “the same” distribution.
  - For comparing our results with earlier ones.
- We can automatically deduce the security of some block-ciphers’ modes of operation.
Earlier work

- Programs without loops
- Which-key and repetition concealing encryption
- Cannot analyse *encryption cycles*

$$E_{k_1}(k_2), E_{k_2}(k_3), \ldots, E_{k_{n-1}}(k_n), E_{k_n}(k_1)$$

- Neither can we, when analysing PRPs.
Conclusions

In this thesis we

- gave an analysis for secure information flow, which can analyse encryption operations;

- showed that this analysis can be implemented efficiently;

- (probably) started the study of automated reasoning about systems containing pseudorandom permutations.