A Type System for Computationally Secure Information Flow

Peeter Laud

Tartu University & Cybernetica AS

joint work with Varmo Vene

Problem statement

- We have a program. It takes some inputs and produces some outputs.
- Some inputs and outputs are private, others are public.
- We want the private inputs to not detectably influence the public outputs.
- The program uses cryptographic operations to hide information.
- We may allow such information flows from secret inputs to public outputs that require unreasonable resources to take advantage of.

Programs

$$
\begin{array}{rcl}
\mathsf{P} & ::= & x := o(x_1, \ldots, x_k) \\
& | & \mathit{skip} \\
& | & \mathsf{P}_1 ; \mathsf{P}_2 \\
& | & \mathit{if}\ b\ \mathit{then}\ \mathsf{P}_1\ \mathit{else}\ \mathsf{P}_2 \\
& | & \mathit{while}\ b\ \mathit{do}\ \mathsf{P}_1
\end{array}
$$

- Let $\mathbf{Var}$ be the set of variables. Let $\mathbf{Var}_\mathrm{S} \subseteq \mathbf{Var}$ be the set of initially secret variables and $\mathbf{Var}_\mathrm{P} \subseteq \mathbf{Var}$ the set of variables whose final values are made public.
- o may be $\mathcal{E}nc$, $\mathcal{G}en$ or some other operation.
  - $\mathcal{G}en$ (nullary) — generates a new encryption key;
  - $\mathcal{E}nc$ (binary) — symmetric encryption;
  - decryption is not handled specially.

Security

- A program has computationally secure information flow (csif) if its secret inputs are computationally independent from its public outputs.
  - An adversary presented with initial values of $\mathbf{Var}_{\mathrm{S}}$ and final values of $\mathbf{Var}_{\mathrm{P}}$ must be unable to decide whether these values come from the same run of the program.
- The encryption system must be
  - message-length-concealing;
  - key-identity-concealing;
  - secure against chosen plaintext attacks.

Static methods for checking csif

- Abstract interpretation
  - data-flow analysis
- Type systems
- (computer-aided) theorem proving
- ...

On typing

- A typing γ assigns a type to each variable.
  - (A part of) these types shows what kind of information may have influenced the contents of these variables.
- Types are partially ordered (the set of types is a lattice).
  - Going higher means more influences.
  - There is a type h denoting the secret information.
- There are inference rules that allow us to decide whether a given typing is correct.
- Correctness theorem: if a program has a typing γ such that $\gamma(x) \geq h$ for all $h \in \mathbf{Var}_{\mathrm{S}}$ and $\bigvee_{x \in \mathbf{Var}_{\mathrm{P}}} \gamma(x) \not\geq h$ then the program has csif.

Information types

- These types record whether a variable may contain information about secret inputs or keys.
- We also want to know, which keys a variable may depend on.
- Basic types: $\mathcal{T}_0 = \{h\} \cup \mathcal{G}$.
  - $\mathcal{G}$ is the set of all program points containing key generation statements.
  - $g \in \mathcal{G}$ corresponds to keys generated at the g-th key generation statement.
  - Cannot distinguish different keys generated at the same program point.

Information types

- $\mathcal{T}_1 = \{t_N \,|\, t \in \mathcal{T}_0, N \subseteq \mathcal{G}\}$.
- A variable of type t_N may contain the information of type t, but it is encrypted with keys generated at program points in N.
- Ordering: $t_N \leq t_{N'}$ if $N \supseteq N'$.
- The main kind of types: $\mathcal{T}_2 = \mathcal{P}(\mathcal{T}_1)$.
- If $\gamma(x) = \{t_1, \ldots, t_n\}$, where $t_1, \ldots, t_n \in \mathcal{T}_1$, then x may contain information of each of the types $t_1, \ldots, t_n$.
- Ordering: $T \leq U$ if $\forall t \in T \, \exists u \in U : t \leq u$.
  - $\leq$ is reflexive and transitive, but not antisymmetric.
  - Factorize with $\leq \cap \geq$.
  - This amounts to deleting all non-maximal elements of a type in $\mathcal{T}_2$.

Normalization in $\mathcal{T}_2$

- The types $T \in \mathcal{T}_2$ must be normalized, in order to take into account that keys may be used.
- This normalization moves upwards in the lattice.
- For example: $\{h_{\{1,2\}}, 1_\emptyset\}$ can be simplified to $\{h_{\{2\}}, 1_\emptyset\}$.
  - i.e. $h_{\{2\}}$ may be added to the original set.
- More interesting cases:
  - $\{1_{\{1\}}\}$ may be simplified to $\{1_\emptyset\}$.
  - $\{1_{\{2\}}, 2_{\{3\}}, \ldots, (n-1)_{\{n\}}, n_{\{1\}}\}$ may be simplified to $\{1_\emptyset, \ldots, n_\emptyset\}$.
- These correspond to removal of encryption cycles.
- The security of encryption cycles does not follow from the security against chosen plaintext attacks.

Usage types

- We have to record whether some data can be used as an encryption key.
- Types:
  - $\mathsf{Key}_{\{i_1,\ldots,i_n\}}$ — a key created in one of the program points $i_1,\ldots,i_n$;
  - Data — a non-key.
- $\gamma(x)$ — a pair $\langle T,K\rangle$ of information type and usage type.
- $\langle T,K\rangle \leq \langle T',K'\rangle$ if
  - $K = K' = \mathsf{Data}$ and $T \leq T'$;
  - $K = \mathsf{Key}_N$, $K' = \mathsf{Key}_{N'}$, $N \subseteq N'$ and $T \leq T'$;
  - $K = \mathsf{Key}_{\{i_1,\ldots,i_n\}}$, $K' = \mathsf{Data}$ and $T \vee \{i_{1\emptyset},\ldots,i_{n\emptyset}\} \leq T'$.

Inference rules

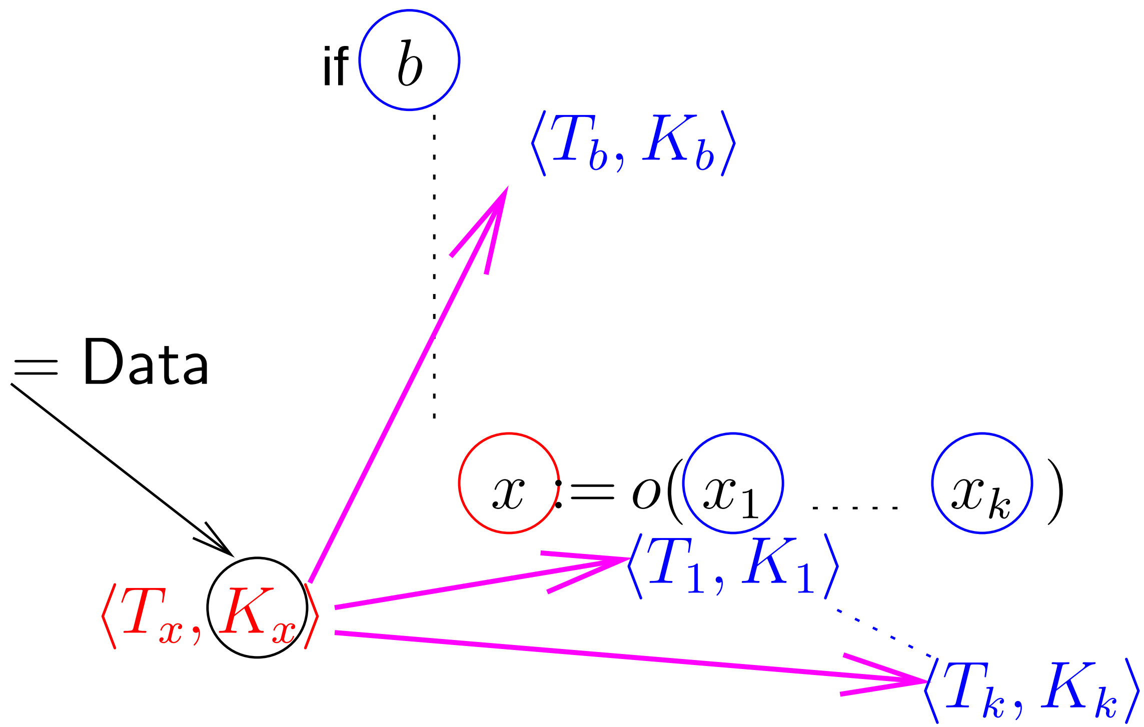
- Something like

$$\frac{\gamma(x) = \langle T, \mathsf{Data} \rangle \qquad \gamma(x_i) \leq \langle T, \mathsf{Data} \rangle}{\gamma \vdash x := o(x_1, \ldots, x_k) : T\ cmd}$$

  i.e. type of the RHS must be smaller than the type of the LHS; the result is a program that does not assign to variables with types less than T.

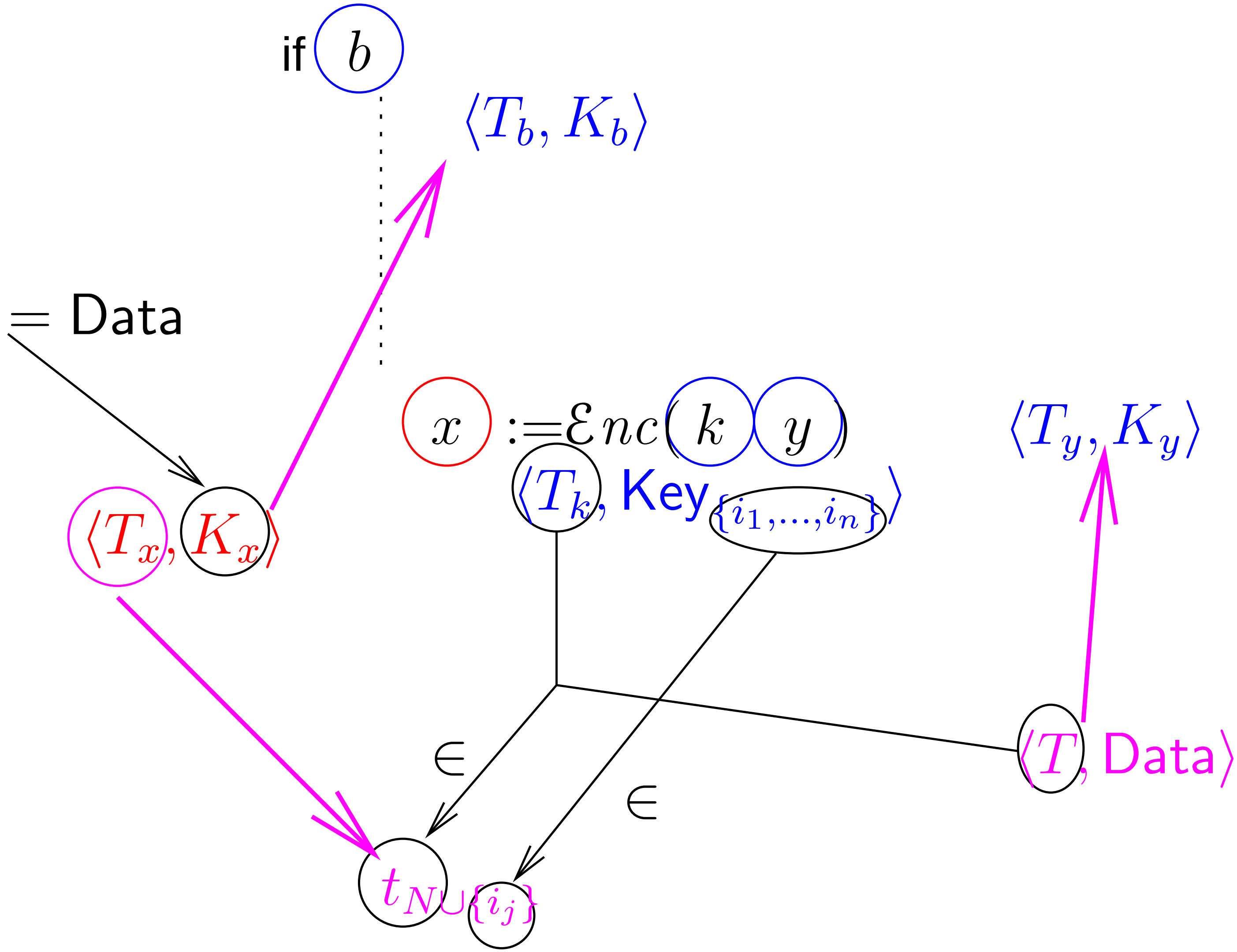
- We will not present all the inference rules here.
- Instead, we show what constraints these rules impose upon γ.
  - This could probably be developed to a type inference algorithm.

General assignments

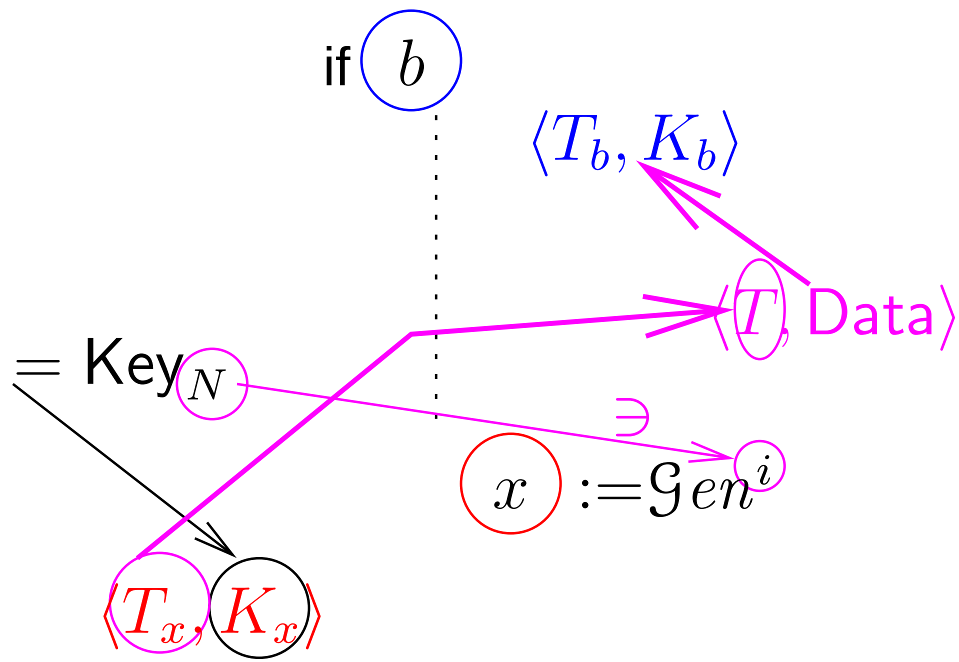


- Here $\longrightarrow$ means $\geq$.
- The next slides will present special cases. These are alternatives to the general scheme.

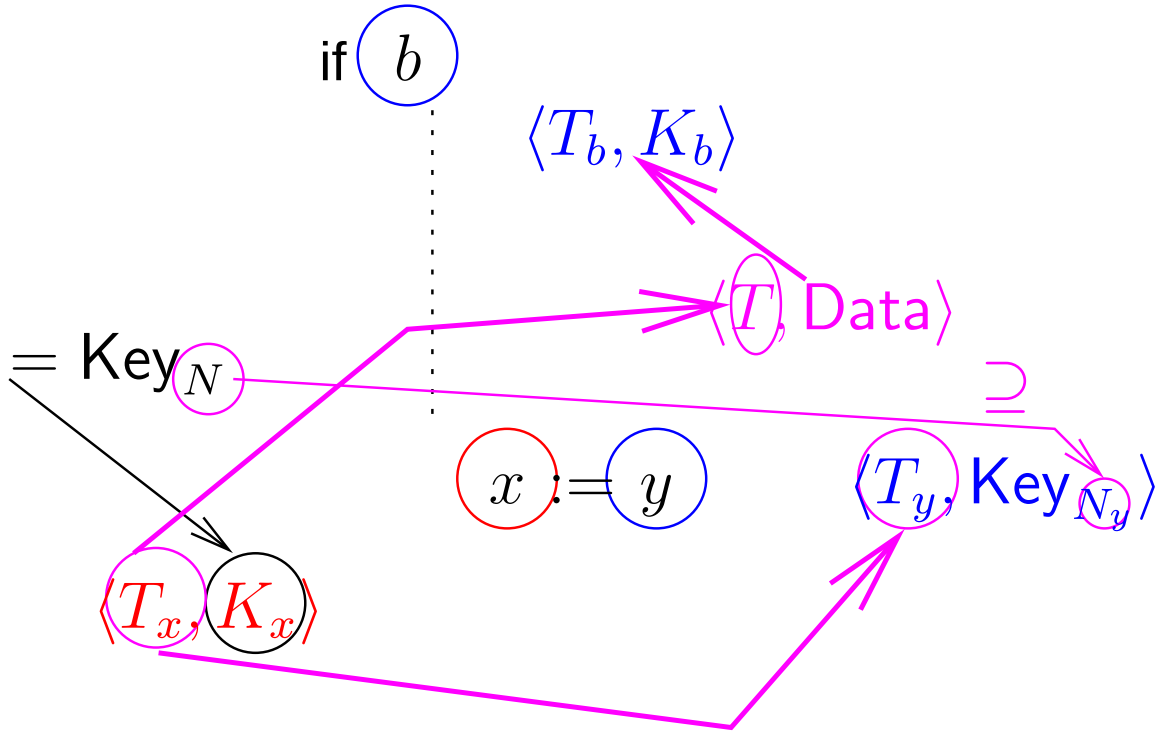
Encryptions



Key generations



Assigning one key to another



On proof of correctness

- Given a program and a correct typing we construct another program that
  - produces public outputs that are indistinguishable from the original program;
  - does not access secret inputs.
- We construct it by transforming the original program.
- Two kinds of transformations:
  - those that don’t change the semantics at all
    - proven by showing a bisimulation
  - those that correspond to the indistinguishability of certain processes according to the security definition of the encryption system.

Example

$k := \mathcal{G}en^1$

$if\ b\ then$

$\quad k' := k$

$\quad y := \mathcal{G}en^2$

$else$

$\quad k' := \mathcal{G}en^3$

$\quad y := \mathcal{G}en^4$

$z := o(y)$

$x := \mathcal{E}nc(k', z)$

$u := \mathcal{E}nc(k, z)$

Example

$$k := \mathcal{G}en^1$$
$$\mathit{if}\ b\ \mathit{then}$$
$$\quad k' := k$$
$$\quad y := \mathcal{G}en^2$$
$$\mathit{else}$$
$$\quad k' := \mathcal{G}en^3$$
$$\quad y := \mathcal{G}en^4$$
$$z := o(y)$$
$$x := \mathcal{E}nc(k', z)$$
$$u := \mathcal{E}nc(k, z)$$

b:$\langle\{h_\emptyset\}, \mathsf{Data}\rangle$

k:$\langle\emptyset, \mathsf{Key}_{\{1\}}\rangle$

k':$\langle\{h_\emptyset\}, \mathsf{Key}_{\{1,3\}}\rangle$

y:$\langle\{h_\emptyset\}, \mathsf{Key}_{\{2,4\}}\rangle$

z:$\langle\{h_\emptyset, 2_\emptyset, 4_\emptyset\}, \mathsf{Data}\rangle$

x:$\langle\{h_{\{1\}}, h_{\{3\}}, 2_{\{1\}}, 2_{\{3\}}, 4_{\{1\}}, 4_{\{3\}}\}, \mathsf{Data}\rangle$

u:$\langle\{h_{\{1\}}, 2_{\{1\}}, 4_{\{1\}}\}, \mathsf{Data}\rangle$

Another example

$k := \mathcal{G}en^1$

$x := \mathcal{E}nc(k, y)$

$g := (\mathsf{lsb}_{10}(s) \neq \mathsf{lsb}_{10}(x))$

$while\ g\ do$

$\quad x := \mathcal{E}nc(k, y)$

$\quad g := (\mathsf{lsb}_{10}(s) \neq \mathsf{lsb}_{10}(x))$

Another example

$k := \mathcal{G}en^1$
$x := \mathcal{E}nc(k, y)$
$g := (\mathsf{lsb}_{10}(s) \neq \mathsf{lsb}_{10}(x))$
$while\ g\ do$
$\quad x := \mathcal{E}nc(k, y)$
$\quad g := (\mathsf{lsb}_{10}(s) \neq \mathsf{lsb}_{10}(x))$

$s{:}\langle \{h_\emptyset\}, \mathsf{Data}\rangle$
$y{:}\langle \emptyset, \mathsf{Data}\rangle$
$k{:}\langle \emptyset, \mathsf{Key}_{\{1\}}\rangle$
$g{:}\langle \{h_\emptyset\}, \mathsf{Data}\rangle$
$x{:}\langle \{h_\emptyset\}, \mathsf{Data}\rangle$

Another example

$k := \mathcal{G}en^1$

$x := \mathcal{E}nc(k, y)$

$g := (\mathsf{lsb}_{10}(s) \neq \mathsf{lsb}_{10}(x))$

$while\ g\ do$

$\quad x := \mathcal{E}nc(k, y)$

$\quad g := (\mathsf{lsb}_{10}(s) \neq \mathsf{lsb}_{10}(x))$

$s{:}\langle\{h_\emptyset\}, \mathsf{Data}\rangle$

$y{:}\langle\emptyset, \mathsf{Data}\rangle$

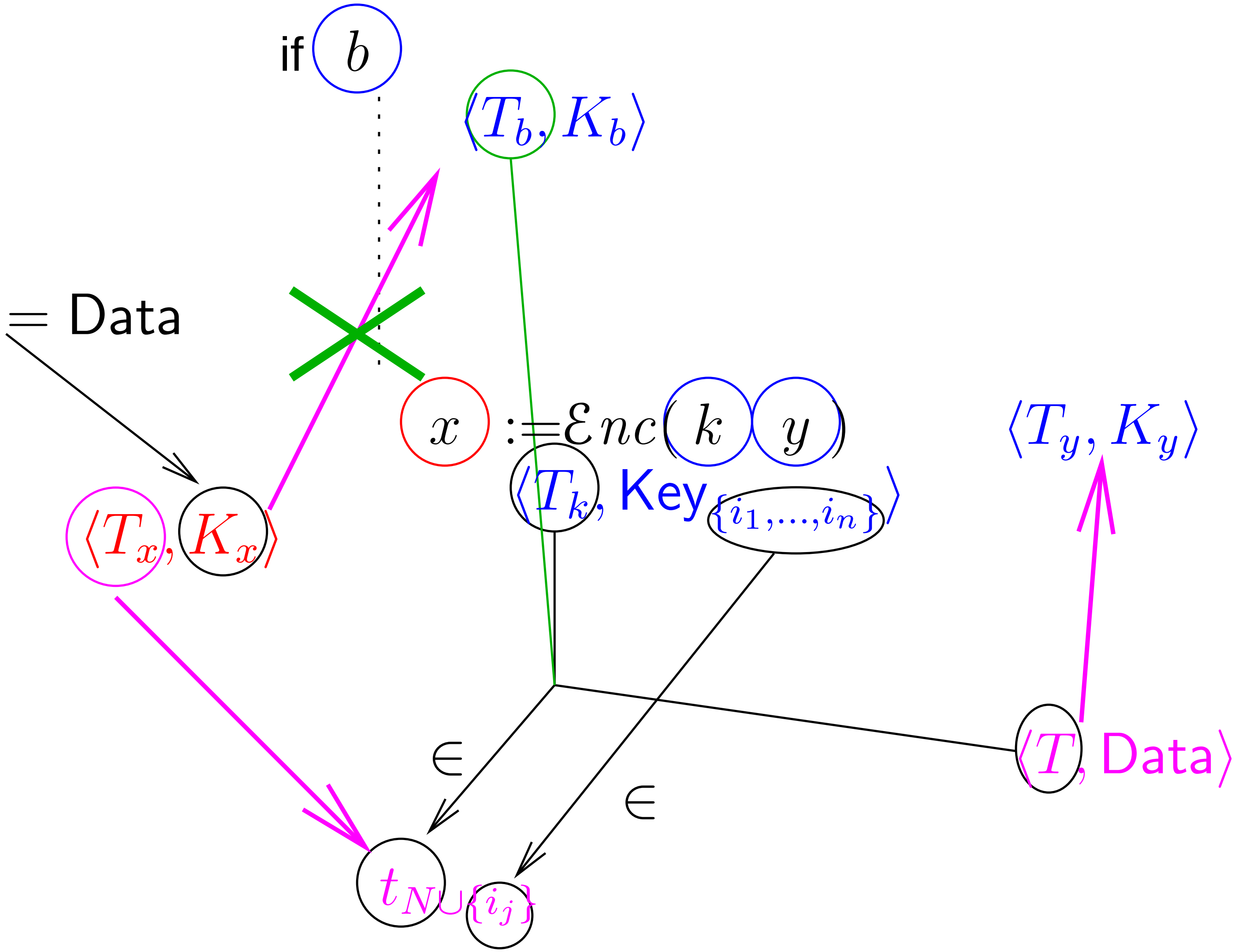
$k{:}\langle\emptyset, \mathsf{Key}_{\{1\}}\rangle$

$g{:}\langle\{h_\emptyset\}, \mathsf{Data}\rangle$

$x{:}\langle\{h_\emptyset\}, \mathsf{Data}\rangle$

If the information carried by the loop guard (g) also went encrypted into ciphertexts (x), then…

Encryptions



Another example

$k := \mathcal{G}en^1$
$x := \mathcal{E}nc(k, y)$
$g := (\mathsf{lsb}_{10}(s) \neq \mathsf{lsb}_{10}(x))$
$while\ g\ do$
$\quad x := \mathcal{E}nc(k, y)$
$\quad g := (\mathsf{lsb}_{10}(s) \neq \mathsf{lsb}_{10}(x))$

$s{:}\langle \{h_\emptyset\}, \mathsf{Data}\rangle$
$y{:}\langle \emptyset, \mathsf{Data}\rangle$
$k{:}\langle \emptyset, \mathsf{Key}_{\{1\}}\rangle$
$g{:}\langle \{h_\emptyset\}, \mathsf{Data}\rangle$
$x{:}\langle \{h_{\{1\}}\}, \mathsf{Data}\rangle$

If the information carried by the loop guard (g) also went encrypted into ciphertexts (x), then...

Conclusions

- A Type System for Computationally Secure Information Flow…
  - is easier to comprehend than data-flow analysis;
  - is easier to assist than data-flow analysis;
  - may allow separate analysis of modules.