Typing Computationally Secure Information Flow in Jif

Peeter Laud
peeter.laud@ut.ee
http://www.cs.ut.ee/~peeter_l

Tartu University & Cybernetica AS & Inst. of Cybernetics

joint work with Liisi Haav

Jif

- An extension of Java
- Extends types with security labels

$$\text{int}\{\text{A} \rightarrow \text{B}; \text{B} \rightarrow \text{A}, \text{C}; \text{A} \leftarrow \text{C}, \text{D}\}\ x;$$

  - $\text{B} \rightarrow \text{A}, \text{C}$: B allows A and C (and B) to read the given data.
  - $\text{A} \leftarrow \text{C}, \text{D}$: A thinks that only C and D (and A) can affect that data.
- Jif statically enforces those policies
  - Declassification and endorsement are possible
- acts-for hierarchy of principals. $\top$ and $\bot$ principal.
- Operations with principals:
  - Conjunction: $A\&B$ acts for both A and B
  - Disjunction: Both A and B act for A, B

A type system for CSIF

- Security types for a simple imperative language with key generation and symmetric encryption. [Laud&Vene, FCT 2005]
- The information type of a variable gives its potential dependencies of sensitive data
- Sensitive data — secrets h and key generations
  - $\mathcal{G}$ — the set of all program points with key generations
  - $\mathcal{T}_0 = \{\mathsf{h}\} \cup \mathcal{G}$
  - $\mathcal{T}_1 = \{t_N \,|\, t \in \mathcal{T}_0, N \subseteq \mathcal{G}\}$ $\quad t_N \leq t_{N'}$ iff $N \supseteq N'$
  - $\mathcal{T}_2 = \wp(\mathcal{T}_1)/\equiv$ information types
    - $\equiv$ contains $\leq \cap \geq$
    - If $t_{N\cup\{i\}} \in T$ and $i_\emptyset \in T$ then $T \equiv T \cup \{t_N\}$
- The usage type of a variable is either Data or Key_N for $N \subseteq \mathcal{G}$.
  - $\langle T, \mathsf{Data}\rangle \leq \langle T', \mathsf{Data}\rangle$ if $T \leq T'$
  - $\langle T, \mathsf{Key}_N\rangle \leq \langle T', \mathsf{Key}_{N'}\rangle$ if $T \leq T'$ and $N \subseteq N'$
  - $\langle T, \mathsf{Key}_N\rangle \leq \langle T \cup N, \mathsf{Data}\rangle$ if $T \leq T'$

Typing constraints

- $if\ b\ \ldots\ x := E(\ldots, y, \ldots)$
  - $\mathsf{use}(\gamma(x)) = \mathsf{Data};\ \gamma(x) \geq \gamma(y);\ \gamma(x) \geq \gamma(b)$
- $if\ b\ \ldots\ x := \mathsf{keygen}^{g}$
  - $\gamma(x) \geq \langle \mathsf{info}(\gamma_{\mathsf{Data}}(b)), \mathsf{Key}_{\{g\}} \rangle$
- $if\ b\ \ldots\ x := y$
  - $\gamma(x) \geq \langle \mathsf{info}(\gamma(y)) \cup \mathsf{info}(\gamma_{\mathsf{Data}}(b)), \mathsf{use}(\gamma(y)) \rangle;$
- $if\ b\ \ldots\ x := \mathsf{enc}_k(y)$
  - let $\mathsf{Key}_M = \mathsf{use}(\gamma(k))$
  - let $T = \mathsf{info}(\gamma_{\mathsf{Data}}(y)) \cup \mathsf{info}(\gamma(k))$
  - $\mathsf{info}(\gamma(x)) \geq \{t_{N \cup \{i\}} \mid t_N \in T, i \in M\}$
  - $\gamma(x) \geq \gamma_{\mathsf{Data}}(b)$
- The lub of information types of all public variables must not be $\geq$ h.

Ideas for modelling the types in Jif

- A principal for each key generation.
- Use declassification to get the right result type of encryption.
- Define a `Key`-class.
  - Implements the methods for key generation, encryption...
  - Stores the usage type of the key.
- Information type $\leftrightarrow$ Jif label
- Upper bound on public variables' types $\leftrightarrow$ OutputStream's policy
- Try to overlook the integrity policies in Jif.

Ideas for modelling the types in Jif

- A principal for each key generation.
- Use declassification to get the right result type of encryption.
- Define a `Key`-class.
  - Implements the methods for key generation, encryption...
  - Stores the usage type of the key.
- Information type $\leftrightarrow$ Jif label
- Upper bound on public variables' types $\leftrightarrow$ OutputStream's policy
- Try to overlook the integrity policies in Jif.

- Main difference between Jif's labels and CSIF type system's types:
  - types describe, how things are
  - labels describe, how things must be

Ideas, more precisely

- Principals G and $\overline{\mathrm{G}}$ for each key generation g.
  - G is allowed to know the keys generated at g.
  - $\overline{\mathrm{G}}$ surely does not know the keys generated at g.
  - $\mathrm{G} \& \overline{\mathrm{G}} \equiv \top$
- If x may be read by P and k is generated at g, then $\mathrm{enc}_k(x)$ may be read by $\mathrm{P}, \overline{\mathrm{G}}$.
- At $k := \mathrm{keygen}^g$, variable k may be read by $\bot$. But G will be stored in the created key object.
  - To use the value of the key, use a method `value()`. The label of the return value of this method is strengthened to include G.

The implementation

```
Class Key [covariant label l1, covariant label l2] {
    final byte[ ]{this} key;

    Key( ) {
        this.key = real_keygen( );
    }

    String{pt meet l2} encrypt{this}(principal p, String pt)
            where {pt, this} ≤ {p → ⊤; p ← ⊤}, caller(p) {
        String r = real_encrypt(key,pt);
        return declassify(r, {pt meet l2; p ← ⊤});
    }

    String{this ; l1} value( ) {
        return new String(key) ;
    }
}
```

The implementation

Class Key [covariant label l1, covariant label l2] {

final byte[ ]{$this$} key;

$l1 \equiv \{p \rightarrow \mathrm{G}_1 \& \cdots \& \mathrm{G}_n; p \leftarrow \top\}$

$l2 \equiv \{p \rightarrow \overline{\mathrm{G}}_1 \& \cdots \& \overline{\mathrm{G}}_n; p \leftarrow \top\}$

Key( ) {

$this.key$ = real_keygen( );

}

String{pt **meet** l2} encrypt{$this$}(**principal** p, String pt)

where $\{pt, this\} \leq \{p \rightarrow \top; p \leftarrow \top\}$, **caller**($p$) {

String r = real_encrypt(key,pt);

return declassify(r, {pt **meet** l2; $p \leftarrow \top$});

}

String{$this$; l1} value( ) {

return **new** String(key) ;

}

}

Example

public static void main$\{p \leftarrow \top\}$(**principal** p, String $args[\,]$)
where caller(p) {
PrintStream$[\{p \rightarrow \overline{\mathrm{G}_1}; p \leftarrow \top\}]$ $out = \ldots;$
Key$[\{p \rightarrow \mathrm{G}_1; p \leftarrow \top\},\{p \rightarrow \overline{\mathrm{G}_1}; p \leftarrow \top\}]$ $k =$
new Key$[\{p \rightarrow \mathrm{G}_1; p \leftarrow \top\},\{p \rightarrow \overline{\mathrm{G}_1}; p \leftarrow \top\}]( );$
String$\{p \rightarrow \mathrm{H}; p \leftarrow \top\}$ $pt = \ldots;$
String $x = k$.encrypt(p,pt);
out.println(x);
}

Example

public static void main$\{p \leftarrow \top\}$(**principal** p, String $args[\,]$)
where caller(p) {
PrintStream$[\{p \rightarrow \overline{\mathrm{G}_1}; p \leftarrow \top\}]$ $out = \ldots;$
Key$[\{p \rightarrow \mathrm{G}_1; p \leftarrow \top\},\{p \rightarrow \overline{\mathrm{G}_1}; p \leftarrow \top\}]$ $k =$
new Key$[\{p \rightarrow \mathrm{G}_1; p \leftarrow \top\},\{p \rightarrow \overline{\mathrm{G}_1}; p \leftarrow \top\}]( );$
String$\{p \rightarrow \mathrm{H}; p \leftarrow \top\}$ $pt = \ldots;$
String $x = k$.encrypt(p,pt);
out.println(x);
}

Denote $p \rightarrow \mathrm{P}; p \leftarrow \top$ with P.

Example

public static void main$\{p \leftarrow \top\}$(**principal** p, String $args[]$)
where caller(p) {
PrintStream[$\{\overline{\mathrm{G_1}}\}$] $out = \ldots$;
Key[$\{\mathrm{G_1}\}$,$\{\overline{\mathrm{G_1}}\}$] $k =$ **new** Key[$\{\mathrm{G_1}\}$,$\{\overline{\mathrm{G_1}}\}$]( );
String$\{\mathrm{H}\}$ $pt = \ldots$;
String $x = k$.encrypt(p,pt);
out.println(x);
}

- Label at the declaration of out must be of the form $\{\mathrm{G_{i_1}} \& \cdots \& \mathrm{G_{i_k}} \& \overline{\mathrm{G_{j_1}}} \& \cdots \& \overline{\mathrm{G_{j_l}}}\}$ with $\{i_1, \ldots, i_k\} \cap \{j_1, \ldots, j_l\} = \emptyset$.

- The form of key classes must be Key[$\{\mathrm{G_{i_1}} \& \cdots \& \mathrm{G_{i_k}}\}$,$\{\overline{\mathrm{G_{i_1}}} \& \cdots \& \overline{\mathrm{G_{i_k}}}\}$] with $k \geq 1$.

These are simple syntactic checks.

Example 2

```
public static void main{p ← ⊤}(principal p, String args[])
      where caller(p) {
   PrintStream[{$\overline{G_1}$&$\overline{G_2}$}] out = ...;
   Key[{G1},{$\overline{G_1}$}] k1 = new Key[{G1},{$\overline{G_1}$}]( );
   Key[{G2},{$\overline{G_2}$}] k2 = new Key[{G2},{$\overline{G_2}$}]( );
   String{H} pt = ...;
   String x = k1.encrypt(p,pt);
   String y = k2.encrypt(p,k1.value( ));
   out.println(x + " " + y);
}
```

Example 3

```
public static void main{p ← ⊤}(principal p, String args[])
      where caller(p) {
   PrintStream[{G1&¬G2}] out = ...;
   Key[{G1},{¬G1}] k1 = new Key[{G1},{¬G1}]( );
   Key[{G2},{¬G2}] k2 = new Key[{G2},{¬G2}]( );
   String{H} pt = ...;
   String x = k1.encrypt(p,pt);
   String y = k2.encrypt(p,x);
   out.println(x + " " + k1.value( ));
}
```

Example 4

public static void main$\{p \leftarrow \top\}$(**principal** p, String $args$[])
where caller(p) {
PrintStream[$\{\overline{\mathrm{G}_1 \& \mathrm{G}_2}\}$] $out = \ldots$;
Key[$\{\mathrm{G}_1\}$,$\{\overline{\mathrm{G}_1}\}$] $k_1 =$ **new** Key[$\{\mathrm{G}_1\}$,$\{\overline{\mathrm{G}_1}\}$]( );
Key[$\{\mathrm{G}_2\}$,$\{\overline{\mathrm{G}_2}\}$] $k_2 =$ **new** Key[$\{\mathrm{G}_2\}$,$\{\overline{\mathrm{G}_2}\}$]( );
Key[$\{\mathrm{G}_1 \& \mathrm{G}_2\}$,$\{\overline{\mathrm{G}_1 \& \mathrm{G}_2}\}$] $k_3 = low\ ?\ k_1 : k_2$;
String$\{\mathrm{H}\}$ $pt = \ldots$;
String $x = k_3$.encrypt(p,pt);
out.println(x);
}

Failing Example

public static void main$\{p \leftarrow \top\}$(**principal** p, String $args[]$)
where caller(p) {
PrintStream[$\{\overline{\mathrm{G_1}\&\mathrm{G_2}}\}$] $out = \ldots$;
Key[$\{\mathrm{G_1}\}$,$\{\overline{\mathrm{G_1}}\}$] $k_1 =$ **new** Key[$\{\mathrm{G_1}\}$,$\{\overline{\mathrm{G_1}}\}$]( );
Key[$\{\mathrm{G_2}\}$,$\{\overline{\mathrm{G_2}}\}$] $k_2 =$ **new** Key[$\{\mathrm{G_2}\}$,$\{\overline{\mathrm{G_2}}\}$]( );
Key[$\{\mathrm{G_1}\&\mathrm{G_2}\}$,$\{\overline{\mathrm{G_1}\&\mathrm{G_2}}\}$] $k_3 = high\ ?\ k_1 : k_2$;
String$\{\mathrm{H}\}$ $pt = \ldots$;
String $x = k_3$.encrypt(p,pt);
out.println(x);
}

- CSIF type system accepts this example.
- At $x = y$.m(...), Jif generates the constraint $\{x\} \geq \{y\}$.

Static method for encryption

```
Class Key [covariant label l1, covariant label l2] {
  final byte[ ]{this} key;

  String{pt meet l2} encrypt{this}(principal p, String pt)
        where {pt, this} ≤ {p → ⊤; p ← ⊤}, caller(p) {
    String r = real_encrypt(key,pt);
    return declassify(r, {pt meet l2; p ← ⊤});
  }

  static String{pt meet l2; k meet l2} sencrypt{this}
        (principal p, Key[l1,l2] k, String pt) throws NullPointerException
        where {pt, k} ≤ {p → ⊤; p ← ⊤}, caller(p) {
    byte[ ] kv = declassify(k, k meet l2).key;
    String r = real_encrypt(kv,pt);
    return declassify(r, {pt meet l2; k meet l2; p ← ⊤});
  }

}
```

Future work

- Implement the CSIF type system "properly", by modifying Jif and increasing the expressiveness of its policies.
  - Then the encryption cycles could be modelled, too.
- Study the information flow in a language with both declassification / endorsement and encryption.