Security Type Checking for MILS-AADL Specifications

Kevin van der Pol, Thomas Noll

MILS Workshop Amsterdam – January 20, 2015
Motivation

Cryptographic controller [Rus92]
- Placed between trusted system and untrusted network
- Declassifies confidential information
- Goal: verify its security
Motivation

- Cryptographic controller [Rus92]
Cryptographic controller [Rus92]
- Placed between trusted system and untrusted network
Motivation

▶ Cryptographic controller [Rus92]
▶ Placed between trusted system and untrusted network
▶ Declassifies confidential information
- Cryptographic controller [Rus92]
- Placed between trusted system and untrusted network
- Declassifies confidential information
- Goal: verify its security
# Table of Contents

1. MILS-AADL specifications
2. Non-interference
3. Type checking
4. Conclusion
MILS-AADL specifications

Non-interference

Type checking

Conclusion
Crypto controller
Crypto controller

- Crypto controller
- Split
- Bypass
- Merge
- Crypto

MILS-AADL specifications

Kevin van der Pol, Thomas Noll | MILS Workshop Amsterdam – January 20, 2015
Crypto controller

crypto controller

split

bypass

crypto

merge
Syntax

system cryptocontroller(

)
system cryptocontroller(
  inframe: in (int L, int H) (0,0)
  outframe: out (int L, enc int H L) (0, encrypt(0, k0))
)

)
Syntax

system cryptocontroller(
    inframe: in (int L, int H) (0,0)
    outframe: out (int L, enc int H L) (0, encrypt(0, k0))
    system split(…)
    system bypass(…)
    system merge(…)
    system crypto(
        inpayload: in int H 0
        outpayload: out enc int H L encrypt(0, k0)
        k: key L k0
    )
)

)
system cryptocontroller(
  inframe: in (int L, int H) (0,0)
  outframe: out (int L, enc int H L) (0, encrypt(0, k0))
  system split(...)
  system bypass(...)
  system merge(...)
  system crypto(
    inpayload: in int H 0
    outpayload: out enc int H L encrypt(0, k0)
    k: key L k0
  )
  connection (split.payload, crypto.inpayload)
  connection (crypto.outpayload, merge.payload)
):
system cryptocontroller(
    inframe: in (int L, int H) (0,0)
    outframe: out (int L, enc int H L) (0, encrypt(0, k0))
    system split(...)
    system bypass(...)
    system merge(...)
    system crypto(
        inpayload: in int H 0
        outpayload: out enc int H L encrypt(0, k0)
        k: key L k0
        m: initial mode L
        m= [then outpayload := encrypt(inpayload,k)] -> m
    )
connection (split.payload, crypto.inpayload)
connection (crypto.outpayload, merge.payload)
)

## Syntax

<table>
<thead>
<tr>
<th>Case</th>
<th>Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>$\sigma ::= H \mid L$</td>
</tr>
<tr>
<td>Basic type</td>
<td>$t ::= \text{int} \mid \text{bool} \mid \text{enc } \tau$</td>
</tr>
<tr>
<td>Security type</td>
<td>$\tau ::= t \sigma \mid \text{key } \sigma$</td>
</tr>
<tr>
<td>Case</td>
<td>Grammar</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Level</td>
<td>$\sigma ::= H \mid L$</td>
</tr>
<tr>
<td>Basic type</td>
<td>$t ::= \text{int} \mid \text{bool} \mid \text{enc } \tau$</td>
</tr>
<tr>
<td>Security type</td>
<td>$\tau ::= t \sigma \mid \text{key } \sigma$</td>
</tr>
<tr>
<td>Expression</td>
<td>$e ::= n \mid x \mid e \oplus e$</td>
</tr>
</tbody>
</table>
## Syntax

<table>
<thead>
<tr>
<th>Case</th>
<th>Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>$\sigma ::= H \mid L$</td>
</tr>
<tr>
<td>Basic type</td>
<td>$t ::= \text{int} \mid \text{bool} \mid \text{enc}\ \tau$</td>
</tr>
<tr>
<td>Security type</td>
<td>$\tau ::= t\ \sigma \mid \text{key}\ \sigma$</td>
</tr>
<tr>
<td>Expression</td>
<td>$e ::= n \mid x \mid e \oplus e$</td>
</tr>
<tr>
<td>System</td>
<td>$S ::= \text{system}\ s(S^<em>\ P^</em>\ C^<em>\ V^</em>\ M^<em>\ T^</em>)$</td>
</tr>
<tr>
<td>Case</td>
<td>Grammar</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Level</td>
<td>$\sigma ::= H \mid L$</td>
</tr>
<tr>
<td>Basic type</td>
<td>$t ::= \text{int} \mid \text{bool} \mid \text{enc } \tau$</td>
</tr>
<tr>
<td>Security type</td>
<td>$\tau ::= t \sigma \mid \text{key } \sigma$</td>
</tr>
<tr>
<td>Expression</td>
<td>$e ::= n \mid x \mid e \oplus e$</td>
</tr>
<tr>
<td>System</td>
<td>$S ::= \text{system } s(S^* P^* C^* V^* M^* T^*)$</td>
</tr>
<tr>
<td>Port</td>
<td>$P ::= p : (\text{in} \mid \text{out})(\text{event } \sigma \mid \text{data } \tau \ e)$</td>
</tr>
</tbody>
</table>
## Syntax

<table>
<thead>
<tr>
<th>Case</th>
<th>Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>$\sigma ::= \text{H} \mid \text{L}$</td>
</tr>
<tr>
<td>Basic type</td>
<td>$t ::= \text{int} \mid \text{bool} \mid \text{enc } \tau$</td>
</tr>
<tr>
<td>Security type</td>
<td>$\tau ::= t ; \sigma \mid \text{key } \sigma$</td>
</tr>
<tr>
<td>Expression</td>
<td>$e ::= n \mid x \mid e \oplus e$</td>
</tr>
<tr>
<td>System</td>
<td>$S ::= \text{system } s(S^* ; P^* ; C^* ; V^* ; M^* ; T^*)$</td>
</tr>
<tr>
<td>Port</td>
<td>$P ::= p:(\text{in} \mid \text{out})(\text{event } \sigma \mid \text{data } \tau ; e)$</td>
</tr>
<tr>
<td>Connection</td>
<td>$C ::= ([s.]p, [s.]p)$</td>
</tr>
</tbody>
</table>
## Syntax

<table>
<thead>
<tr>
<th>Case</th>
<th>Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>( \sigma ::= H \mid L )</td>
</tr>
<tr>
<td>Basic type</td>
<td>( t ::= \text{int} \mid \text{bool} \mid \text{enc} \ \tau )</td>
</tr>
<tr>
<td>Security type</td>
<td>( \tau ::= t \ \sigma \mid \text{key} \ \sigma )</td>
</tr>
<tr>
<td>Expression</td>
<td>( e ::= n \mid x \mid e \oplus e )</td>
</tr>
<tr>
<td>System</td>
<td>( S ::= \text{system} \ s (S^* \ P^* \ C^* \ V^* \ M^* \ T^*) )</td>
</tr>
<tr>
<td>Port</td>
<td>( P ::= p : (\text{in} \mid \text{out})(\text{event} \ \sigma \mid \text{data} \ \tau \ e) )</td>
</tr>
<tr>
<td>Connection</td>
<td>( C ::= ([s.]p, [s.]p) )</td>
</tr>
<tr>
<td>Variable</td>
<td>( V ::= x : \tau \ e )</td>
</tr>
</tbody>
</table>
## Syntax

<table>
<thead>
<tr>
<th>Case</th>
<th>Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Level</strong></td>
<td>( \sigma ::= H \mid L )</td>
</tr>
<tr>
<td><strong>Basic type</strong></td>
<td>( t ::= \text{int} \mid \text{bool} \mid \text{enc } \tau )</td>
</tr>
<tr>
<td><strong>Security type</strong></td>
<td>( \tau ::= t \sigma \mid \text{key } \sigma )</td>
</tr>
<tr>
<td><strong>Expression</strong></td>
<td>( e ::= n \mid x \mid e \oplus e )</td>
</tr>
<tr>
<td><strong>System</strong></td>
<td>( S ::= \text{system } s(S^* P^* C^* V^* M^* T^*) )</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>( P ::= p : (\text{in} \mid \text{out})(\text{event } \sigma \mid \text{data } \tau \ e) )</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>( C ::= ([s.]p, [s.]p) )</td>
</tr>
<tr>
<td><strong>Variable</strong></td>
<td>( V ::= x : \tau \ e )</td>
</tr>
<tr>
<td><strong>Mode</strong></td>
<td>( M ::= m : [\text{initial}] \text{mode } \sigma )</td>
</tr>
</tbody>
</table>
Syntax

<table>
<thead>
<tr>
<th>Case</th>
<th>Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>$\sigma ::= H \mid L$</td>
</tr>
<tr>
<td>Basic type</td>
<td>$t ::= \text{int} \mid \text{bool} \mid \text{enc} \tau$</td>
</tr>
<tr>
<td>Security type</td>
<td>$\tau ::= t \sigma \mid \text{key} \sigma$</td>
</tr>
<tr>
<td>Expression</td>
<td>$e ::= n \mid x \mid e \oplus e$</td>
</tr>
<tr>
<td>System</td>
<td>$S ::= \text{system} , s(S^* , P^* , C^* , V^* , M^* , T^*)$</td>
</tr>
<tr>
<td>Port</td>
<td>$P ::= p : (\text{in} \mid \text{out})(\text{event} , \sigma \mid \text{data} , \tau , e)$</td>
</tr>
<tr>
<td>Connection</td>
<td>$C ::= ([s.]p, [s.]p)$</td>
</tr>
<tr>
<td>Variable</td>
<td>$V ::= x : \tau , e$</td>
</tr>
<tr>
<td>Mode</td>
<td>$M ::= m : [\text{initial}] , \text{mode} , \sigma$</td>
</tr>
<tr>
<td>Transition</td>
<td>$T ::= m - [ [p] , [\text{when} , e] , [\text{then} , x := e] ] \rightarrow m'$</td>
</tr>
</tbody>
</table>
Table of Contents

1 MILS-AADL specifications
2 Non-interference
3 Type checking
4 Conclusion
Lattice of confidentiality levels
Non-interference

- Lattice of confidentiality levels
- This presentation: high and low ($H \sqsubseteq L$)
Non-interference

- Lattice of confidentiality levels
- This presentation: high and low ($H \sqsubseteq L$)
- Non-interference: low outputs not affected by high inputs
Lattice of confidentiality levels

This presentation: high and low ($H \sqsubseteq L$)

Non-interference: low outputs not affected by high inputs

Non-interference

Kevin van der Pol, Thomas Noll | MILS Workshop Amsterdam – January 20, 2015
Lattice of confidentiality levels

This presentation: high and low ($H \subseteq L$)

Non-interference: low outputs not affected by high inputs
Non-interference

- Lattice of confidentiality levels
- This presentation: high and low ($H \sqsubseteq L$)
- Non-interference: low outputs not affected by high inputs
Lattice of confidentiality levels

This presentation: high and low ($\mathbb{H} \sqsubseteq \mathbb{L}$)

Non-interference: low outputs not affected by high inputs

When varying secret variables and ports, reachable values of public variables and ports are indistinguishable
Explicit data flow

Implicit data flow

if high then
    low := 1
else
    low := 2
endif
Unwanted data flows

- Explicit data flow
  - physical connection from high to low ports
Explicit data flow
  ▶ physical connection from high to low ports
  ▶ low := high

Implicit data flow
  ▶ data leak through control flow
  ▶ if high
  ▶ then
  ▶ low := 1
  ▶ else
  ▶ low := 2
  ▶ endif
Unwanted data flows

- Explicit data flow
  - physical connection from high to low ports
  - low := high
- Implicit data flow
Unwanted data flows

- Explicit data flow
  - physical connection from high to low ports
  - low := high

- Implicit data flow
  - data leak through control flow
Unwanted data flows

- Explicit data flow
  - physical connection from high to low ports
  - low := high

- Implicit data flow
  - data leak through control flow
  - if high then low := 1 else low := 2 endif
Possibilistic non-interference

- Abstract encryption function
Abstract encryption function

Encryption breaks traditional non-interference
Abstract encryption function

Encryption breaks traditional non-interference

- Public ciphertexts do depend on confidential contents!
Abstract encryption function

- Encryption breaks traditional non-interference
  - Public ciphertexts do depend on confidential contents!

- Encryption non-deterministically calculates a ciphertext from a set
Abstract encryption function

Encryption breaks traditional non-interference
  - Public ciphertexts do depend on confidential contents!

Encryption non-deterministically calculates a ciphertext from a set
Look at sets of possibilities
Possibilistic non-interference

- Abstract encryption function
- Encryption breaks traditional non-interference
  - Public ciphertexts do depend on confidential contents!
- Encryption non-deterministically calculates a ciphertext from a set
- Look at sets of possibilities
- Naive approach: all ciphertexts are indistinguishable
Abstract encryption function

Encryption breaks traditional non-interference
  - Public ciphertexts do depend on confidential contents!

Encryption non-deterministically calculates a ciphertext from a set

Look at sets of possibilities

Naive approach: all ciphertexts are indistinguishable

Cannot distinguish calculating new ciphertext or re-using same one
Possibilistic non-interference

- Abstract encryption function
- Encryption breaks traditional non-interference
  - Public ciphertexts do depend on confidential contents!

- Encryption non-deterministically calculates a ciphertext from a set
- Look at sets of possibilities
- Naive approach: all ciphertexts are indistinguishable
- Cannot distinguish calculating new ciphertext or re-using same one
  - low1 := encrypt(v,k); low2 := low1
Possibilistic non-interference

- Indistinguishability equivalence $\equiv$ on ciphertexts [AHS08]
Possibilistic non-interference

- Indistinguishability equivalence $\equiv$ on ciphertexts [AHS08]
  - safe usage
Possibilistic non-interference

- Indistinguishability equivalence $\equiv$ on ciphertexts [AHS08]
  - safe usage
  - prevent implicit flow
Possibilistic non-interference

- Indistinguishability equivalence $\equiv$ on ciphertexts [AHS08]
  - safe usage
  - prevent implicit flow
- Realistic for common encryption classes (IND-CPA, INT-PTXT) [Lau08]
Possibilistic non-interference

- Indistinguishability equivalence $\equiv$ on ciphertexts [AHS08]
  - safe usage
  - prevent implicit flow
- Realistic for common encryption classes (IND-CPA, INT-PTXT) [Lau08]
- Possibilistic non-interference
Non-interference non-compositionality
Non-interference non-compositionality
Non-interference non-compositionality

Kevin van der Pol, Thomas Noll | MILS Workshop Amsterdam – January 20, 2015
Non-interference non-compositionality

Kevin van der Pol, Thomas Noll | MILS Workshop Amsterdam – January 20, 2015
Restrictions

non-determinism
Restrictions

non-determinism

branching on secrets
Restrictions

non-determinism

branching on secrets

secret-Zeno
Restrictions

non-determinism

branching on secrets

secret-Zeno

public output from secret region
MILS-AADL specifications

Non-interference

Type checking

Conclusion
$T$: local variables and data ports $\rightarrow$ declared type
$T$: local variables and data ports $\rightarrow$ declared type

- Modes and event ports get a confidentiality level
Type checking

- $T$: local variables and data ports $\rightarrow$ declared type
- Modes and event ports get a confidentiality level
- Type rules in context of $T$
Type checking

- $T$: local variables and data ports $\rightarrow$ declared type
- Modes and event ports get a confidentiality level
- Type rules in context of $T$
- Goal: type each subsystem, connection and transition

Theorem
If the system is typable, it is non-interfering
Type checking

- $T$: local variables and data ports $\rightarrow$ declared type
- Modes and event ports get a confidentiality level
- Type rules in context of $T$
- Goal: type each subsystem, connection and transition
Type checking

- $T$: local variables and data ports $\rightarrow$ declared type
- Modes and event ports get a confidentiality level
- Type rules in context of $T$
- Goal: type each subsystem, connection and transition

**Theorem**

If the system is typable, it is non-interfering
Types of expressions

<table>
<thead>
<tr>
<th>Case</th>
<th>Type rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic types</td>
<td>$T \vdash n : \text{int} \ L$ $T \vdash b : \text{bool} \ L$</td>
</tr>
<tr>
<td>Variable</td>
<td>$T(x) = \tau$</td>
</tr>
<tr>
<td></td>
<td>$T \vdash x : \tau$</td>
</tr>
<tr>
<td>Operator</td>
<td>$T \vdash e_1 : t_1 \sigma_1$ $T \vdash e_2 : t_2 \sigma_2$ $\oplus : t_1 \times t_2 \rightarrow t$</td>
</tr>
<tr>
<td></td>
<td>$T \vdash e_1 \oplus e_2 : t \left(\sigma_1 \sqcup \sigma_2\right)$</td>
</tr>
<tr>
<td>Encryption</td>
<td>$T \vdash e_1 : \tau$ $T \vdash e_2 : \text{key} \ L$</td>
</tr>
<tr>
<td></td>
<td>$T \vdash \text{encrypt}(e_1, e_2) : \text{enc} \tau \ L$</td>
</tr>
<tr>
<td>Decryption</td>
<td>$T \vdash e_1 : \text{enc} \tau \sigma$ $T \vdash e_2 : \text{key} \ H$</td>
</tr>
<tr>
<td></td>
<td>$T \vdash \text{decrypt}(e_1, e_2) : \tau^\sigma$</td>
</tr>
</tbody>
</table>
system crypto(
    inpayload: int H 0
    outpayload: out enc int H L encrypt(0, k0)
    k: key L k0
    m: initial mode L
    m - [then outpayload := encrypt(inpayload, k)] -> m
)

- Type of encrypt(inpayload, k)?

\[
T(\text{inpayload}) = \text{int H} \quad T(k) = \text{key L} \\
T \vdash \text{inpayload : int H} \quad T \vdash k : \text{key L} \\
T \vdash \text{encrypt(inpayload, k) : enc int H L}
\]
If $\tau_e \ll \tau_x$, assignment $x := e$ is valid

- Traditional type safety
- Confidentiality restriction
If $\tau_e <: \tau_x$, assignment $x := e$ is valid

- Traditional type safety
- Confidentiality restriction

<table>
<thead>
<tr>
<th>Case</th>
<th>Subtype rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic types</td>
<td>$\sigma \sqsubseteq \sigma'$</td>
</tr>
<tr>
<td></td>
<td>$\text{int } \sigma &lt;: \text{int } \sigma'$</td>
</tr>
<tr>
<td></td>
<td>$\sigma \sqsubseteq \sigma'$</td>
</tr>
<tr>
<td></td>
<td>$\text{bool } \sigma &lt;: \text{bool } \sigma'$</td>
</tr>
<tr>
<td></td>
<td>$\text{key } \sigma &lt;: \text{key } \sigma$</td>
</tr>
</tbody>
</table>
If $\tau_e \ <: \ \tau_x$, assignment $x := e$ is valid

- Traditional type safety
- Confidentiality restriction

### Subtyping

<table>
<thead>
<tr>
<th>Case</th>
<th>Subtype rule</th>
</tr>
</thead>
</table>
| Basic types     | $\sigma \sqsubseteq \sigma'$  
                  | int $\sigma \ <: \ $int $\sigma'$  
                  | bool $\sigma \ <: \ $bool $\sigma'$  
                  | key $\sigma \ <: \ $key $\sigma'$  |
| Encryption      | $\tau \ <: \ \tau'$  
                  | $\sigma \sqsubseteq \sigma'$  
                  | enc $\tau \sigma \ <: \ $enc $\tau' \sigma'$  |
Subtyping

system crypto(
    inpayload: int H 0
    outpayload: out enc int H L encrypt(0, k0)
    k: key L k0
    m: initial mode L
    m-[then outpayload:=encrypt(inpayload,k)]->m
  )

▶ Is outpayload:=encrypt(inpayload,k) valid?
system crypto(
    inpayload: int H 0
    outpayload: out enc int H L encrypt(0, k0)
    k: key L k0
    m: initial mode L
    m- [then outpayload := encrypt(inpayload,k)] -> m
)

▶ Is outpayload:=encrypt(inpayload,k) valid?
  (let τ abbreviate enc int H L)
system crypto(
    inpayload: int H 0
    outpayload: out enc int H L encrypt(0, k0)
    k: key L k0
    m: initial mode L
    m [then outpayload := encrypt(inpayload, k)] -> m
)

▶ Is outpayload := encrypt(inpayload, k) valid?
(let τ abbreviate enc int int H L)

\[
\begin{align*}
T(\text{outpayload}) &= \tau \\
T \vdash \text{outpayload} : \tau & \quad T \vdash \text{encrypt}(\text{inpayload}, k) : \tau \\
\tau & <: \tau \\
T \vdash \text{outpayload := encrypt}(\text{inpayload}, k) : \tau
\end{align*}
\]
### Type of transitions

<table>
<thead>
<tr>
<th>Case</th>
<th>Type rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transition</td>
<td></td>
</tr>
<tr>
<td>$T \vdash p : \tau_p$</td>
<td></td>
</tr>
<tr>
<td>$T \vdash g : \tau_g$</td>
<td>$\tau_x &lt;: \tau_e$</td>
</tr>
<tr>
<td>$T \vdash x : \tau_x$</td>
<td>$lvl(m) \sqsubseteq lvl(\tau_p)$</td>
</tr>
<tr>
<td>$T \vdash e : \tau_e$</td>
<td>$lvl(m) \sqsubseteq lvl(\tau_x)$</td>
</tr>
<tr>
<td>$T \vdash m - [ p \text{ when } g \text{ then } x := e ] \rightarrow m' : \sigma$</td>
<td></td>
</tr>
</tbody>
</table>
### Type of transitions

<table>
<thead>
<tr>
<th>Case</th>
<th>Type rule</th>
</tr>
</thead>
</table>
| Transition | $T \vdash p : \tau_p$               
|           | $T \vdash g : \tau_g$  
|           | $\tau_x <: \tau_e$  
|           | $lvl(\tau_x) \sqsubseteq \sigma$  
|           | $T \vdash x : \tau_x$  
|           | $lvl(m) \sqsubseteq lvl(\tau_p)$  
|           | $\sigma \sqsubseteq lvl(\tau_p)$  
|           | $T \vdash e : \tau_e$  
|           | $lvl(m) \sqsubseteq lvl(\tau_x)$  
|           | $\sigma \sqsubseteq lvl(\tau_g)$  
|           | $T \vdash m - [ p \ when \ g \ then \ x := e ] \rightarrow m' : \sigma$  

### Type of transitions

<table>
<thead>
<tr>
<th>Case</th>
<th>Type rule</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transition</strong></td>
<td></td>
</tr>
<tr>
<td>$T \vdash p : \tau_p$</td>
<td>$T \vdash g : \tau_g$ $\tau_x \ll \tau_e$ $lvl(\tau_x) \sqsubseteq \sigma$</td>
</tr>
<tr>
<td>$T \vdash x : \tau_x$ $lvl(m) \sqsubseteq lvl(\tau_p)$ $\sigma \sqsubseteq lvl(\tau_p)$</td>
<td>$T \vdash e : \tau_e$ $lvl(m) \sqsubseteq lvl(\tau_x)$ $\sigma \sqsubseteq lvl(\tau_g)$</td>
</tr>
<tr>
<td>$T \vdash m - [ p \ when \ g \ then \ x := e ] - \rightarrow m' : \sigma$</td>
<td></td>
</tr>
</tbody>
</table>

- Effect is well typed
### Type of transitions

<table>
<thead>
<tr>
<th>Case</th>
<th>Type rule</th>
</tr>
</thead>
</table>
| Transition | $T \vdash p : \tau_p$
|          | $T \vdash g : \tau_g$
|          | $\tau_x <: \tau_e$
|          | $\text{lvl}(\tau_x) \sqsubseteq \sigma$
|          | $\text{lvl}(\tau_p) \sqsubseteq \text{lvl}(\tau_g)$
|          | $\sigma \sqsubseteq \text{lvl}(\tau_p)$
|          | $\text{lvl}(\tau_x) \sqsubseteq \sigma$
|          | $\text{lvl}(\tau_p) \sqsubseteq \text{lvl}(\tau_g)$
|          | $\text{lvl}(\tau_x) \sqsubseteq \sigma$
|          | $\text{lvl}(\tau_p) \sqsubseteq \text{lvl}(\tau_g)$

- Effect is well typed
- No public outputs from secret region (mode)
## Type of transitions

<table>
<thead>
<tr>
<th>Case</th>
<th>Type rule</th>
</tr>
</thead>
</table>
| **Transition** | $T \vdash p : \tau_p$  
$T \vdash g : \tau_g$  
$\tau_x \ll : \tau_e$  
$T \vdash x : \tau_x$  
$\text{lvl}(m) \sqsubseteq \text{lvl}(\tau_p)$  
$\sigma \sqsubseteq \text{lvl}(\tau_p)$  
$T \vdash e : \tau_e$  
$\text{lvl}(m) \sqsubseteq \text{lvl}(\tau_x)$  
$\sigma \sqsubseteq \text{lvl}(\tau_g)$ |

$T \vdash m - [ p \text{ when } g \text{ then } x := e ] \rightarrow m' : \sigma$

- Effect is well typed
- No public outputs from secret region (mode)
- Guard or event high $\rightarrow$ transition high
## Type of transitions

<table>
<thead>
<tr>
<th>Case</th>
<th>Type rule</th>
</tr>
</thead>
</table>
| Transition   | $T \vdash p : \tau_p$
               | $T \vdash g : \tau_g$
               | $\tau_x < : \tau_e$
               | $lvl(\tau_x) \sqsubseteq \sigma$
               | $lvl(\tau) (\tau_x)$
               | $\sigma \sqsubseteq lvl(\tau_p)$
               | $\sigma \sqsubseteq lvl(\tau_g)$
               | $T \vdash m - [p \text{ when } g \text{ then } x := e ] \rightarrow m' : \sigma$

- Effect is well typed
- No public outputs from secret region (mode)
- Guard or event high $\rightarrow$ transition high
- No public outputs from secret region (transition)
Example

system crypto(
        inpayload: int H 0
    outpayload: out enc int H L encrypt(0, k0)
    k: key L k0
    m: initial mode L
    m- [then outpayload := encrypt(inpayload,k)] -> m
)

▶ Is the transition well typed?
Example

```
system crypto(
    inpayload: int H 0
    outpayload: out enc int H L encrypt(0, k0)
    k: key L k0
    m: initial mode L
    m = [then outpayload := encrypt(inpayload, k)] -> m
)

► Is the transition well typed?
  ► Effect is well typed: ✓
```
Example

system crypto(
    inpayload: int H 0
    outpayload: out enc int H L encrypt(0, k0)
    k: key L k0
    m: initial mode L
    m- [then outpayload := encrypt(inpayload,k)] -> m
)

▶ Is the transition well typed?
  ▶ Effect is well typed: ✓
  ▶ No public outputs from secret region (mode): ✓
Example

```plaintext
system crypto(
   inpayload: int H 0
   outpayload: out enc int H L encrypt(0, k0)
   k: key L k0
   m: initial mode L
   m -[then outpayload := encrypt(inpayload,k)] -> m
)

▶ Is the transition well typed?
   ▶ Effect is well typed: ✓
   ▶ No public outputs from secret region (mode): ✓
   ▶ Guard or event high → transition high: ✓
```
system crypto(
    inpayload: int H 0
    outpayload: out enc int H L encrypt(0, k0)
    k: key L k0
    m: initial mode L
    m- [then outpayload := encrypt(inpayload,k)] -> m
)

▶ Is the transition well typed?
  ▶ Effect is well typed: ✓
  ▶ No public outputs from secret region (mode): ✓
  ▶ Guard or event high → transition high: ✓
  ▶ No public outputs from secret region (transition): ✓
```
system crypto(
    inpayload: int H 0
    outpayload: out enc int H L encrypt(0, k0)
    k: key L k0
    m: initial mode L
  ) -> m
  inpayload := encrypt(inpayload, k)
```

- **Is the transition well typed?**
  - Effect is well typed: ✓
  - No public outputs from secret region (mode): ✓
  - Guard or event high → transition high: ✓
  - No public outputs from secret region (transition): ✓

- All subsystems and connections trivially well typed
Example

```
 system crypto(
   inpayload: int H 0
   outpayload: out enc int H L encrypt(0, k0)
   k: key L k0
   m: initial mode L
   m - [then outpayload := encrypt(inpayload, k)] -> m
)
```

- Is the transition well typed?
  - Effect is well typed: ✓
  - No public outputs from secret region (mode): ✓
  - Guard or event high → transition high: ✓
  - No public outputs from secret region (transition): ✓

- All subsystems and connections trivially well typed
- `crypto` system is non-interfering
Table of Contents

1 MILS-AADL specifications
2 Non-interference
3 Type checking
4 Conclusion
AADL variant with cryptographic primitives
Conclusion

- AADL variant with cryptographic primitives
- Automatic verification for possibilistic non-interference
Conclusion

- AADL variant with cryptographic primitives
- Automatic verification for possibilistic non-interference
- Future work: soundness proof, implementation, type inference
