Secure Compilation: Formal Foundations and (Some) Applications

Marco Patrignani

03 April 2024
Who Am I ?
Special Thanks to: (wrt the contents of this talk)
Special Thanks to: (wrt the contents of this talk)

please interrupt & ask questions
Special Thanks to: 
(wrt the contents of this talk)

for offline questions: I leave tomorrow
Foundations of Secure Compilation
Programming Languages: Pros and Cons

Good PLs (R, TypeScript, ..., ...) provide:

• helpful abstractions to write secure code
Programming Languages: Pros and Cons

Good PLs (R, TypeScript, ... ) provide:

• helpful abstractions to write secure code

but

• when compiled (\[\cdot\]) and linked with adversarial target code
Good PLs (R, TypeScript, Java, Python, ...) provide:

- helpful abstractions to write secure code

but

- when compiled (\([\cdot]\)) and linked with adversarial target code
- these abstractions are NOT enforced
Secure Compilation: Example

$F^*$ HACL*, Zinzindohouè et al., CCS’17

Asm

[ChaCha20] [Poly1305] [...]

ChaCha20 Poly1305
Secure Compilation: Example

F*: HACL*. Zinzindohouë et al., CCS’17

Asm

[ChaCha20] [Poly1305] [...]
Secure Compilation: Example

Preserve the security of

ChaCha20  Poly1305  ...

F*  HACL*: Zinzindohouë et al., CCS’17

Asm

[ChaCha20]  [Poly1305]  [...]
Secure Compilation: Example

Preserve the security of

ChaCha20 → Poly1305 → ...

\[ F^* \quad \text{HACL*, Zinzindohouè et al., CCS'17} \]

Asm

[ChaCha20] → [Poly1305] → [...]

when interoperating with
Secure Compilation: Example

Correct compilation

ChaCha20  Poly1305  ...

F*  HACL*: Zinzindohouë et al., CCS'17

Asm

[ChaCha20] [Poly1305] [...]
Secure Compilation: Example

Secure compilation

ChaCha20
Poly1305
...

F* HACL*: Zinzindohouë et al., CCS'17

Asm

[ChaCha20]
[Poly1305]
[...]

4/35
Secure Compilation: Example

Enable source-level security reasoning

ChaCha20  Poly1305  ...

F* HACL*: Zinzindohoue et al., CCS’17

Asm

[ChaCha20]  [Poly1305]  [...]
What does it mean for a compiler to be secure?
What does it mean for a compiler to be secure?

Analogous questions are answered for type systems, correct compilation, …
Secure Implementation of Channel Abstractions

Martín Abadi
ma@pa.dec.com
Digital Equipment Corporation
Systems Research Center

Cédric Fournet
Cedric.Fournet@inria.fr
INRIA Rocquencourt

Georges Gonthier
Georges.Gonthier@inria.fr
INRIA Rocquencourt

Abstract

Communication in distributed systems often relies on useful abstractions such as channels, remote procedure calls, and remote method invocations. The implementations of these abstractions sometimes provide security properties, in particular through encryption. In this paper, we focus on the secure implementation of channels via cryptography. Our goal is to define that their implementation of secure channels via cryptography was secure.
Once Upon a Time in Process Algebra

**Fully Abstract Compilation (FAC)**

**Theorem 1** The compositional translation is fully-abstract, up to observational equivalence: for all join-calculus processes $P$ and $Q$, 

$$ P \approx Q \quad \text{if and only if} \quad \text{Env}[\llbracket P \rrbracket] \approx \text{Env}[\llbracket Q \rrbracket] $$

Challenge: define that their implementation of secure channels via cryptography was secure
Fully Abstract Compilation Influence

Typed Closure Conversion Preserves Observational Equivalence

Authentication primitives and their compilation

Secure Compilation of Object-Oriented Components to Protected Module Architectures

Local Memory via Layout Randomization

Secure Compilation to Protected Module Architectures

Fully Abstract Compilation via Universal Embedding

Beyond Good and Evil

Formalizing the Security Guarantees of Compartmentalizing Compilation

A Secure Compiler for ML Modules

An Equivalence-Preserving CPS Translation via Multi-Language Semantics

On Modular and Fully-Abstract Compilation

Fully Abstract Compilation to JavaScript

Secure Implementations for Typed Session Abstraction:

- Ricardo Corin1,2,3 Pierre-Malo Denielou1,2 Cédric Fournet1,2
  - Karthikeyan Bhargavan1,2 James Leifer1
  - MSR-INRIA Joint Centre, Microsoft Research, University of I

On Protection by Layout Randomization

-MARTÍN ABADI, Microsoft Research, Silicon Valley
  - Santa Cruz, Collège de France
  - Gordon D. Plotkin, University of Edinburgh

Challenge: easier/more efficient/more precise alternatives preserve classes of (hyper)properties
FAC: useful for language expressiveness but complex and with an unclear security implication
- FAC: useful for **language expressiveness** but complex and with an unclear security implication
- **Challenge**: easier/more efficient/more precise alternatives
• FAC: useful for **language expressiveness** but complex and with an unclear security implication
• **Challenge**: easier/more efficient/more precise alternatives preserve classes of (hyper)properties

Clarkson & Schneider JCS ’10
Robust Compilation (RC) Criteria

Robust Relational Hyperproperty Preservation

Robust K-Relational Hyperproperty Preservation

Robust 2-Relational Hyperproperty Preservation

Robust Relational Property Preservation

Robust K-Relational Property Preservation

Robust 2-Relational Property Preservation

Robust Relational relaXed safety Preservation

Robust Finite-Relational relaXed safety Preservation

Robust K-Relational relaXed safety Preservation

Robust 2-Relational relaXed safety Preservation

Robust Relational Safety Preservation

Robust Finite-Relational Safety Preservation

Robust K-Relational Safety Preservation

Robust 2-Relational Safety Preservation

Robust Hyperproperty Preservation

Robust Subset-Closed Hyperproperty Preservation

Robust K-Subset-Closed Hyperproperty Preservation

Robust 2-Subset-Closed Hyperproperty Preservation

Robust Trace Property Preservation

Robust Trace Equivalence Preservation

Robust Hypersafety Preservation

Robust K-Hypersafety Preservation

Robust 2-Hypersafety Preservation

Robust Safety Property Preservation

Robust Termination-Insensitive Noninterference Preservation

Tradeoffs for code efficiency, security guarantees, proof complexity
Robust Compilation (RC) Criteria

Tradeoffs for code efficiency, security guarantees, proof complexity
Robust Criteria: Intuition

Each point has two equivalent criteria:

• Property – ful:
  + clearly tells what class it preserves
Robust Criteria: Intuition

Each point has two equivalent criteria:

• Property – ful:
  + clearly tells what class it preserves
  - harder to prove
Each point has two equivalent criteria:

- **Property – ful**:  
  + clearly tells what class it preserves  
  - harder to prove

- **Property – free**:  
  + easier to prove
Robust Criteria: Intuition

Each point has two equivalent criteria:

• Property – ful:
  + clearly tells what class it preserves
  - harder to prove

• Property – free:
  + easier to prove
  - unclear what security classes are preserved
In Depth Example: RSC

\[ J \cdot K = \text{compiler} \]

\[ J \cdot K : \text{RSP} \quad \text{def} = \]

[\cdot] = compiler  \quad [\cdot] : \text{RSP} \quad \text{def} =
In Depth Example: RSC

\[
\begin{align*}
\llbracket \cdot \rrbracket : \text{RSP} & \overset{\text{def}}{=} \forall \pi \sim \pi \in \text{Safety}. \\
\pi / \pi & = \text{set of traces}
\end{align*}
\]
In Depth Example: RSC

\[ 
\cdot : \text{RSP} \overset{\text{def}}{=} \forall \pi \approx \pi \in \text{Safety}. \forall P. \\
\]

- \([\cdot] = \text{compiler}
- \pi / \pi = \text{set of traces}
- P = \text{partial program}
In Depth Example: RSC

\[ [\cdot] : \text{RSP} \xrightarrow{\text{def}} \forall \pi \approx \pi \in \text{Safety}. \forall P. \]

\[ \text{if } (\forall A, t. \]

- \([\cdot]\) = compiler
- \(\pi / \pi\) = set of traces
- \(P\) = partial program
- \(A / A\) = attacker
- \(t / t\) = trace of events
In Depth Example: RSC

\[ \mathbb{J} \cdot \mathbb{K} : \text{RSP} \overset{\text{def}}{=} \forall \pi \approx \pi \in Safety. \forall P. \text{if } (\forall A, t. A[P] \leadsto t) \]

\[ \mathbb{J} \cdot \mathbb{K} : \text{RSC} \overset{\text{def}}{=} \forall P, A, m. \text{if } A[J\ P\ K] \leadsto m \text{ then } \exists A, m \approx m. A[P] \leadsto m \]

\[ \mathbb{J} \cdot \mathbb{K} = \text{compiler} \]

\[ \pi / \pi = \text{set of traces} \]

\[ P = \text{partial program} \]

\[ A / A = \text{attacker} \]

\[ t / t = \text{trace of events} \]

\[ \mathbb{J} \cdot \mathbb{K} = \text{linking} \]

\[ \leadsto / \leadsto = \text{trace semantics} \]
In Depth Example: RSC

\[
\cdot \colon \text{RSP} \overset{\text{def}}{=} \forall \pi \approx \pi \in \text{Safety}. \forall P. 

\text{if } (\forall A, t. A[P] \leadsto t \Rightarrow t \in \pi)
\]
In Depth Example: RSC

\[ \boxed{} : \text{RSP} \overset{\text{def}}{=} \forall \pi \approx \pi \in \text{Safety}. \forall P. \]

if \((\forall A, t. A[P] \sim t \Rightarrow t \in \pi)\)

then \((\forall A, t. \)
In Depth Example: RSC

\[ \cdot : \text{RSP} \overset{\text{def}}{=} \forall \pi \approx \pi \in \text{Safety}. \forall P. \]

if \((\forall A, t. A[P] \xrightarrow{\cdot} t \Rightarrow t \in \pi)\)

then \((\forall A, t. A[[P]] \xrightarrow{\cdot} t \Rightarrow )\)
In Depth Example: RSC

\[ \cdot \] : RSP = \forall \pi \approx \pi \in Safety. \forall P.

\text{if } \left( \forall A, t. A[P] \rightsquigarrow t \Rightarrow t \in \pi \right) \text{ then } \left( \forall A, t. A[[P]] \rightsquigarrow t \Rightarrow t \in \pi \right)
In Depth Example: RSC

\[ [\cdot] : \text{RSP} \overset{\text{def}}{=} \forall \pi \approx \pi \in \text{Safety.} \forall P. \]

\[ \text{if } (\forall A, t. A [P] \leadsto t \Rightarrow t \in \pi) \]

\[ \text{then } (\forall A, t. A [\lceil P \rceil] \leadsto t \Rightarrow t \in \pi) \]

\[ [\cdot] : \text{RSC} \overset{\text{def}}{=} \]
In Depth Example: RSC

\[
\begin{align*}
\llbracket \cdot \rrbracket : RSP & \overset{\text{def}}{=} \forall \pi \simeq \pi \in Safety. \forall P. \\
& \quad \text{if } (\forall A, t. A[\llbracket P \rrbracket] \rightsquigarrow t \Rightarrow t \in \pi) \\
& \quad \text{then } (\forall A, t. A[\llbracket P \rrbracket] \rightsquigarrow t \Rightarrow t \in \pi)
\end{align*}
\]

\[
\begin{align*}
\llbracket \cdot \rrbracket : RSC & \overset{\text{def}}{=} \forall P, A, m.
\end{align*}
\]
In Depth Example: RSC

\[
\cdot : RSP \overset{\text{def}}{=} \forall \pi \approx \pi \in Safety. \forall P.
\text{if } (\forall A, t. A [P] \rightsquigarrow t \Rightarrow t \in \pi)
\text{then } (\forall A, t. A [\cdot[P]] \rightsquigarrow t \Rightarrow t \in \pi)
\]

\[
\cdot : RSC \overset{\text{def}}{=} \forall P, A, m.
\text{if } A [\cdot[P]] \rightsquigarrow m
\]
In Depth Example: RSC

\[\llbracket \cdot \rrbracket : RSP \defeq \forall \pi \approx \pi \in Safety. \forall P.\]
\[\text{if } (\forall A, t. A[P] \xrightarrow{\cdot} t \Rightarrow t \in \pi)\]
\[\text{then } (\forall A, t. A[[P]] \xrightarrow{\cdot} t \Rightarrow t \in \pi)\]

\[\llbracket \cdot \rrbracket : RSC \defeq \forall P, A, m.\]
\[\text{if } A[[P]] \xrightarrow{\cdot} m\]
\[\text{then } \exists A,\]
In Depth Example: RSC

\[ [\cdot] : \text{RSP} \overset{\text{def}}{=} \forall \pi \approx \pi \in \text{Safety}. \forall P. \]
\[ \text{if } (\forall A, t. A[P] \rightsquigarrow t \Rightarrow t \in \pi) \]
\[ \text{then } (\forall A, t. A[\llbracket P \rrbracket] \rightsquigarrow t \Rightarrow t \in \pi) \]

\[ [\cdot] : \text{RSC} \overset{\text{def}}{=} \forall P, A, m. \]
\[ \text{if } A[\llbracket P \rrbracket] \rightsquigarrow m \]
\[ \text{then } \exists A, m \approx m. \]
In Depth Example: RSC

\[ [\cdot] : \text{RSP} \overset{\text{def}}{=} \forall \pi \approx \pi \in \text{Safety}. \forall P. \]

\[
\text{if } (\forall A, t. A[P] \rightsquigarrow t \Rightarrow t \in \pi) \]

\[ \text{then } (\forall A, t. A[\lbrack P \rbrack] \rightsquigarrow t \Rightarrow t \in \pi) \]

\[ [\cdot] : \text{RSC} \overset{\text{def}}{=} \forall P, A, m. \]

\[
\text{if } A[\lbrack P \rbrack] \rightsquigarrow m
\]

\[ \text{then } \exists A, m \approx m. A[P] \rightsquigarrow m \]
Secure Compilation Threat Model

- robust, active attacker (\(\forall A\))

robust safety works, e.g., Swasey et al. OOPSLA'17, Sammler et al. POPL'20
Secure Compilation Threat Model

• robust, active attacker ($\forall A$)
  robust safety works, e.g., Swasey et al. OOPSLA'17, Sammler et al. POPL'20

• in-language expressible attacker
Secure Compilation Threat Model

- robust, active attacker ($\forall A$)

  robust safety works, e.g., Swasey et al. OOPSLA'17, Sammler et al. POPL'20

- in-language expressible attacker

- trace-based security behaviour ($m/m$)
Secure Compilation Threat Model

- robust, active attacker (\( \forall A \))
- in-language expressible attacker
- trace-based security behaviour (\( m/m \))

What can we do with these foundations?
Talk Outline

Robust Memory Safety  
POPL’23

Robust Cryptographic Constant Time  
(wip)

Micro-architectural Attacks (Spectre)  
CCS’21

Security Architectures  
(e.g., Cheri/ARM Morello, Sancus/Intel SGX, …)  
Toplas’15, CSF’21, …

Mechanise Cryptographic Proofs  
CSF’24 + wip

Conclusion
Robust Memory Safety
Memory Safety (Untyped, Intra-Object)

- Add colours+shades to pointers & memory
- Check colour+shade when using pointers

Memarian et al.
POPL'19, Azevedo de Amorim et al.
POST'18
Memory Safety (Untyped, Intra-Object)

• add colors+shades to pointers & memory
Memory Safety (Untyped, Intra-Object)

- add **colours+shades** to pointers & memory
- **check** colour+shade when using pointers

Memarian et al. POPL’19, Azevedo de Amorim et al. POST’18
Memory Safety (Untyped, Intra-Object)

- add **colours+shades** to pointers & memory
- check colour+shade when using pointers

Memarian et al. POPL’19, Azevedo de Amorim et al. POST’18
Memory Safety (Untyped, Intra-Object)

- add **colours+shades** to pointers & memory
- **check** colour+shade when using pointers

Memarian et al. POPL’19, Azevedo de Amorim et al. POST’18

```
alloc(4)
```

```
F F F F F F F F F
```
Memory Safety (Untyped, Intra-Object)

- add **colours+shades** to pointers & memory
- **check** colour+shade when using pointers

 alloc(4)

Memarian et al. POPL’19, Azevedo de Amorim et al. POST’18
Memory Safety (Untyped, Intra-Object)

- add **colours+shades** to pointers & memory
- check colour+shade when using pointers

Memarian et al. POPL’19, Azevedo de Amorim et al. POST’18

```plaintext
alloc(4)
```

```
A A A A F
```

```plaintext
read(P)
ok
write(P)
NO
```
Memory Safety (Untyped, Intra-Object)

- add **colours+shades** to pointers & memory
- check colour+shade when using pointers

Memarian et al. POPL’19, Azevedo de Amorim et al. POST’18

```
alloc(4)
alloc(1+1)
```
Memory Safety (Untyped, Intra-Object)

- add **colours+shades** to pointers & memory
- check colour+shade when using pointers

Memarian et al. POPL’19, Azevedo de Amorim et al. POST’18

alloc(4)
alloc(1+1)

```
A   A   A   A   F   A   A
```

P
Memory Safety (Untyped, Intra-Object)

- add **colours+shades** to pointers & memory
- check colour+shade when using pointers

Memarian et al. POPL’19, Azevedo de Amorim et al. POST’18

```
alloc(4)
alloc(1+1)
```

```
A A A A A F A A
```

alloc(4)
alloc(1+1)

```
P Q
```
Memory Safety (Untyped, Intra-Object)

• add **colours**+**shades** to pointers & memory
• check colour+shade when using pointers

Memarian et al. POPL’19, Azevedo de Amorim et al. POST’18

alloc(4)
alloc(1+1)
read(P)
Memory Safety (Untyped, Intra-Object)

- add **colours+shades** to pointers & memory
- check colour+shade when using pointers

Memarian et al. POPL’19, Azevedo de Amorim et al. POST’18

```
allocate(4)
allocate(1+1)
read(P)
```

```
A A A A A F A A
```

```
P
```

```
Q
```

ok
Memory Safety (Untyped, Intra-Object)

- add **colours+shades** to pointers & memory
- check colour+shade when using pointers

Memarian et al. POPL’19, Azevedo de Amorim et al. POST’18

```
alloc(4)
alloc(1+1)
read(P)
```

Diagram:

```
A A A A A F A A
P Q
```
Memory Safety (Untyped, Intra-Object)

- add **colours+shades** to pointers & memory
- **check** colour+shade when using pointers

Memarian *et al.* POPL’19, Azevedo de Amorim *et al.* POST’18

```markdown
alloc(4)
alloc(1+1)
read(P)
read(P)
```

![Diagram](image.png)
Memory Safety (Untyped, Intra-Object)

- add **colours+shades** to pointers & memory
- check colour+shade when using pointers

Memarian et al. POPL’19, Azevedo de Amorim et al. POST’18
Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian et al. POPL’19, Azevedo de Amorim et al. POST’18

```
alloc(4)  \  
alloc(1+1)  \  
read(P) \  
write(P) \  
```

```
<table>
<thead>
<tr>
<th>A</th>
<th>A</th>
<th>A</th>
<th>A</th>
<th>F</th>
</tr>
</thead>
</table>
```

```
<table>
<thead>
<tr>
<th>A</th>
<th>A</th>
</tr>
</thead>
</table>
```

```
Q  P
```

NO
Memory Safety (Untyped, Intra-Object)

- add **colours+shades** to pointers & memory
- check **colour+shade** when using pointers

Memarian et al. POPL’19, Azevedo de Amorim et al. POST’18
Memory-Safe WebAssembly (MSWAsm)

- WAsm:
  - inter-sandboxes MS

Watson et al. S&P'15

handles: \langle base, length, offset, isCorrupted, id \rangle

segment instructions:
- segment _alloc,
- segment _free
- segment _read,
- segment _write
- handle _add,
- handle _slice
Memory-Safe WebAssembly (MSWAsm)

- **WAsm:**
  - inter-sandboxes MS
  - intra-sandbox **vulnerability**
Memory-Safe WebAssembly (MSWAsm)

- WAsm:
  - inter-sandboxes MS
  - intra-sandbox vulnerability

- MSWAsm: segment memory indexed by Cheri-like pointers

Watson et al. S&P’15
Memory-Safe WebAssembly (MSWAsm)

- **WAsm:**
  - inter-sandboxes MS
  - intra-sandbox *vulnerability*
- **MSWAsm:** segment memory indexed by Cheri-like pointers
  - **handles:**
    - \( \langle \text{base}, \text{length}, \text{offset}, \text{isCorrupted}, \text{id} \rangle \)

Watson et al. S&P’15
Memory-Safe WebAssembly (MSWAsm)

- WAsm:
  - inter-sandboxes MS
  - intra-sandbox vulnerability

- MSWAsm: segment memory indexed by Cheri-like pointers
  - handles:
    \((\text{base}, \text{length}, \text{offset}, \text{isCorrupted}, \text{id})\)

- segment instructions:
  - segment_alloc, segment_free

Watson et al. S&P’15
Memory-Safe WebAssembly (MSWAsm)

- **WAsm:**
  - inter-sandboxes MS
  - intra-sandbox *vulnerability*
- **MSWAsm:** segment memory indexed by Cheri-like pointers
  - **handles:**
    \[
    \langle \text{base}, \text{length}, \text{offset}, \text{isCorrupted}, \text{id} \rangle
    \]
- **segment instructions:**
  - `segment_alloc`, `segment_free`
  - `segment_read`, `segment_write`

Watson et al. S&P’15
Memory-Safe WebAssembly (MSWAsm)

- **WAsm:**
  - inter-sandboxes MS
  - intra-sandbox vulnerability

- **MSWAsm:** segment memory indexed by Cheri-like pointers
  - handles:
    \[
    \langle \text{base}, \text{length}, \text{offset}, \text{isCorrupted}, \text{id} \rangle
    \]

- **Segment instructions:**
  - `segment_alloc`, `segment_free`
  - `segment_read`, `segment_write`
  - `handle_add`, `handle_slice`

Watson et al. S&P'15
Compiling from C to MSWAsm

- pointer becomes handle
Compiling from C to MSWAsm

- pointer becomes handle
- dereference becomes segment_read
Compiling from C to MSWAsm

- **pointer** becomes **handle**
- **dereference** becomes **segment_read**
- **write** becomes **segment_write**
Compiling from C to MSWAsm

- pointer becomes handle
- dereference becomes segment_read
- write becomes segment_write
- pointer arithmetic becomes handle_add
Compiling from C to MSWAsm

- pointer becomes handle
- dereference becomes segment_read
- write becomes segment_write
- pointer arithmetic becomes handle_add
- field access becomes handle_slice
Compiler Properties

$\Omega = \text{source state}$

$\alpha / \alpha = \text{trace action}$
Compiler Properties

Ω = source state
Ω = compiled state
α / α = trace action

PRO: proved
MS: preservation,
MS: enforcement

CON: not really
RSC (no ∀ A)

Challenge: how to ensure A actions do not affect MS?
Compiler Properties

\( \Omega = \text{source state} \)
\( \Omega' = \text{compiled state} \)
\( \alpha / \alpha = \text{trace action} \)
\( \delta = \text{partial bijection} \)

\( \pi \): proved
\( \text{MS} \): preservation, enforcement

\( \text{CON} \): not really
\( \text{RSC} \): (no \( \forall A \))

Challenge: how to ensure \( A \) actions do not affect \( \text{MS} \)?
Compiler Properties

\( \Omega = \text{source state} \)
\( \Omega' = \text{compiled state} \)
\( \alpha / \alpha = \text{trace action} \)
\( \delta = \text{partial bijection} \)

\( \Omega = \text{source state} \)
\( \Omega' = \text{compiled state} \)
\( \alpha / \alpha = \text{trace action} \)
\( \delta = \text{partial bijection} \)

\[ \begin{align*}
\Omega & \rightarrow \alpha \rightarrow \Omega' \\
\Omega' & \rightarrow \alpha' \rightarrow \Omega'
\end{align*} \]

\[ \begin{align*}
\Omega & \rightarrow \delta \rightarrow \Omega \\
\Omega' & \rightarrow \delta' \rightarrow \Omega'
\end{align*} \]

\[ \begin{align*}
\Omega & \rightarrow \star \rightarrow \Omega' \\
\Omega' & \rightarrow \star \rightarrow \Omega'
\end{align*} \]

**Challenge:** how to ensure \( A \) actions do not affect \( MS \)?
Compiler Properties

- $\Omega =$ source state
- $\Omega' =$ compiled state
- $\alpha / \alpha =$ trace action
- $a / a =$ allocator
- $\delta =$ partial bijection

PRO: proved

MS preservation, MS enforcement

CON: not really

RSC (no $\forall A$)

Challenge: how to ensure $A$ actions do not affect $\text{MS}$?
Compiler Properties

Ω = source state
Ω' = compiled state
α / α = trace action
a / a = allocator
δ = partial bijection

\[ \Omega = \text{source state} \]
\[ \Omega' = \text{compiled state} \]
\[ \alpha / \alpha = \text{trace action} \]
\[ a / a = \text{allocator} \]
\[ \delta = \text{partial bijection} \]

PRO: proved
MS: preservation, MS enforcement
CON: not really
RSC: (no ∀ A)

Challenge: how to ensure A actions do not affect MS?
Compiler Properties

Ω = source state
Ω = compiled state
α/α = trace action
a/a = allocator
δ = partial bijection
M = monitor state
α = monitor action

PRO: proved
MS preservation, MS enforcement
CON: not really
RSC (no ∀ A)

Challenge: how to ensure A actions do not affect MS?
Compiler Properties

\( \Omega \) = source state
\( \Omega' \) = compiled state
\( \alpha / \alpha \) = trace action
\( a / a \) = allocator
\( \delta \) = partial bijection
\( M \) = monitor state
\( \alpha \) = monitor action

\[ \Omega = \text{source state} \]
\[ \Omega' = \text{compiled state} \]
\[ \alpha / \alpha = \text{trace action} \]
\[ a / a = \text{allocator} \]
\[ \delta = \text{partial bijection} \]
\[ M = \text{monitor state} \]
\[ \alpha = \text{monitor action} \]
Compiler Properties

Ω = source state
Ω = compiled state
α / α = trace action
a / a = allocator
δ = partial bijection
M = monitor state
α = monitor action

PRO: proved
MS: preservation, MS: enforcement

CON: not really
RSC: (no ∀ A)

Challenge: how to ensure A actions do not affect MS?
PRO: proved MS preservation, MS enforcement
PRO: proved MS preservation, MS enforcement

CON: not really RSC (no \( \forall A \))
PRO: proved MS preservation, MS enforcement

CON: not really RSC (no $\forall A$)

Challenge: how to ensure $A$ actions do not affect MS?
Robust Cryptographic Constant Time
• larger trace model than MS:
• larger trace model than MS:
  • memory accesses (as for MS)
  • and timing-relevant operations
(Robust) Cryptographic Constant Time

- larger trace model than MS:
  - memory accesses (as for MS)
  - and timing-relevant operations

- (in)formally RCT: ...
(Robust) Cryptographic Constant Time

• larger trace model than MS:
  • memory accesses (as for MS)
  • and timing-relevant operations

• (in)formally RCT: …
  no secret-dependent operations

Bernstein '15, Barbosa et al. S&P'21
• **Goal:** protect a crypto library from *any application* using it
• **Goal:** protect a crypto library from any application using it

• crypto developers *already* zero out memory before calling apps (e.g., Libsodium)
Compiler Preserving RCT

• **Goal**: protect a crypto library from *any application* using it

• crypto developers *already* zero out memory before calling apps (e.g., Libsodium)

• **Challenge**: crypto devs must make their code CT
• **Goal**: protect a crypto library from any application using it

• crypto developers already zero out memory before calling apps (e.g., Libsodium)

• **Challenge**: crypto devs must make their code CT

• **Solution**: devise CT code  
  e.g., Bacelar Almeida et al. CCS’17
Compiler Preserving RCT

- **Goal**: protect a crypto library from any application using it
- Crypto developers already zero out memory before calling apps (e.g., Libsodium)
- **Challenge**: crypto devs must make their code CT
- **Solution**: devise CT code (e.g., Bacelar Almeida et al. CCS'17)
- **Challenge**: crypto devs do not know where their code is used
Compiler Preserving RCT

- **Goal**: protect a crypto library from any application using it

- crypto developers **already** zero out memory before calling apps (e.g., Libsodium)

- **Challenge**: crypto devs must make their code CT

- **Solution**: devise CT code  
  e.g., Bacelar Almeida et al. CCS’17

- **Challenge**: crypto devs do not know where their code is used

- **Solution**: use a compiler that preserves RCT
Micro-architectural Attacks (Spectre)
void f (int x) ↦ if(x < A.size) {y = B[A[x]]}
void f (int x) \rightarrow \text{if}(x < A.\text{size}) \{ y = B[A[x]] \}
run 1: A.\text{size} = 16, A[128] = 3

call f 128

\text{if} (128 < 16) \{ y = B[A[128]] \}

\Rightarrow \text{SNI violation}
void f (int x) ⇔ if(x < A.size) \{ y = B[A[x]] \}


call f 128

if (128 < 16) \{ y = B[A[128]] \}

skip

⇒ different traces

⇒ SNI violation
void f (int x) ↦ if(x < A.size) { y = B[A[x]]}

call f 128


void f (int x) ↦ if (x < A.size) { y = B[A[x]] }

A program is SNI (\(\vdash P : SNI\)) if, given two runs from low-equivalent states:
• assuming the non-speculative traces are low-equivalent
• then the speculative traces are also low-equivalent
void f (int x) ↦ if(x < A.size) { y = B[A[x]]


run 2: A[128] = 7 different H values

call f 128


skip


rd A[128]

y = B[ 3 ]

rd B[3]

y = _

⇒ SNI violation

A program is SNI (⊢ P : SNI) if, given two runs from low-equivalent states:

• assuming the non-speculative traces are low-equivalent
• then the speculative traces are also low-equivalent
void f (int x) ↦ if (x < A.size) \{ y = B[A[x]] \}


```c
y = B[A[128]]
```


different H values

```
if (128 < 16) { y = B[A[128]] }
```

different traces
⇒ SNI violation

A program is SNI (\( \vdash P : \text{SNI} \)) if, given two runs from low-equivalent states:

• assuming the non-speculative traces are low-equivalent
• then the speculative traces are also low-equivalent
Speculative Semantics & SNI

**void f (int x) ↦ if (x < A.size) { y = B[A[x]] }**


call f 128

if (128 < 16) { y = B[A[128]] }

skip


⇒ SNI violation
void f (int x) ↦ if(x < A.size) {y = B[A[x]]}
run 2: A[128] = 7 different H values
void f (int x) ↦ if(x < A.size) { y = B[A[x]]}

run 2: A[128] = 7 different H values

call f 128


different traces ⇒ SNI violation
**Speculative Semantics & SNI**

```
void f (int x) ↦ if (x < A.size) {y = B[A[x]]}
run 2: A[128] = 7 different H values
```

call f 128

```
if (128 < 16) { y = B[A[128]]}
```
	race 1:

```
```
**void f (int x) \mapsto if(x < A.size) \{ y = B[A[x]] \}\right\}**

run 1: $A$ size = 16, $A[128] = 3$

run 2: $A[128] = 7$ different H values

call f 128


different traces

trace 2: \texttt{rd A[128] rd B[7]}  \Rightarrow SNI violation
A program is SNI (\(\neg P : \text{SNI}\)) if, given two runs from low-equivalent states:

- assuming the non-speculative traces are low-equivalent
- then the speculative traces are also low-equivalent

trace 1: \(\text{rd } A[128]\) \(\text{rd } B[3]\) different traces \(\Rightarrow\) SNI violation

trace 2: \(\text{rd } A[128]\) \(\text{rd } B[7]\)
void f (int x) ↦ if(x < A.size) { y = B[A[x]]}
run 2: A[128] = 7 different H values

call f 128

if (128 < 16) { y = B[A[128]]}

Problem: Proving compiler preserves SNI is hard
Problem: Proving compiler preserves SNI is hard

Solution: overapproximate SNI with a novel property: speculative safety (SS)
Speculative Safety ($SS$): Taint Tracking

```c
void f (int x) ↔ if (x < A.size) { y = B[A[x]] }
```

only 1 run needed: A.size=16, A[128]=3

**integrity lattice:** $S \subset U \quad S \cap U = S \quad U$ does not flow to $S$

---

call f 128

pc : $S$
void f (int x) ↔ if(x < A.size) {y = B[A[x]]}
only 1 run needed: A.size=16, A[128]=3
integrity lattice: $S \subset U$  $S \cap U = S$  $U$ does not flow to $S$

```
call f 128
pc : S

```

```
128 : S
pc : S

rd A[128] :: S
y = _
```

```
rd B[3] :: U
```

```
A program is SS ($\vdash P : SS$) if its traces do not contain $U$ actions
### Speculative Safety ($SS$): Taint Tracking

**Function Definition:**

```c
void f (int x) ↦ if (x < A.size) { y = B[A[x]] }
```

**Run Example:**
- **Call:** `call f 128`
- **If Condition:** `(128 < 16) { y = B[A[128]] }`

**Integrity Lattice:**

- $S \subset U$
- $S \cap U = S$
- $U$ does not flow to $S$

**Conclusion:**

A program is $SS$ ($\vdash P : SS$) if its traces do not contain $U$ actions.
Speculative Safety ($SS$): Taint Tracking

```c
void f (int x) ↦ if(x < A.size) { y = B[A[x]] }
```

only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U$ $S \cap U = S$ $U$ does not flow to $S$

Diagram:
- Call $f$ 128
  - $pc : S$
- If (128 < 16) { y = B[A[128]] }
  - $pc : S$
- $128 : S$
  - $y = B[A[128]]$
  - $pc : U$
Speculative Safety ($SS$): Taint Tracking

void f (int x) ⇔ if(x < A.size) {y = B[A[x]]}
only 1 run needed: A.size=16, A[128]=3
integrity lattice: $S \subset U$ $S \cap U = S$ $U$ does not flow to $S$
void f (int x) \iff (x < A.size) \{ y = B[A[x]] \} \\
only 1 run needed: A.size=16, A[128]=3 \\
integrity lattice: $S \subset U$ \ $S \cap U = S$ \ $U$ does not flow to $S$

![Diagram](image-url)
Speculative Safety ($SS$): Taint Tracking

void f (int x) ↔ if (x < A.size) { y = B[A[x]]} 
only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U$ \hspace{1cm} S \cap U = S \hspace{1cm} U$ does not flow to $S$

call f 128 pc : S

if (128 < 16) { y = B[A[128]]} pc : S

skip pc : S

Speculative Safety ($SS$): Taint Tracking

void f (int x) \mapsto \text{if}(x < A.\text{size}) \{ y = B[A[x]] \}

only 1 run needed: A.\text{size}=16, A[128]=3

A program is $SS$ ($\vdash P : SS$) if its traces do not contain $U$ actions
Secure Compilation Framework for Spectre

⊢ $P \vdash SS$

⊢ $J \vdash PK$

$\forall P \in \text{source} \vdash JPK \vdash SNI$

⊢ $J \cdot K \vdash RSP$

$\overapprox{\vdash J \cdot K \vdash RSC}$

dashed premises are already discharged

• to show security: simply prove
∀P ∈ source
⊢ P : SS

⊢ [P] : SS

⊢ [P] : RSP

criteria equality

⊢ [P] : RSC

overapproximation

⊢ [P] : SNI
Secure Compilation Framework for Spectre

∀ P ∈ source
⊢ P : SS

⊢ [P] : RSC

⊢ [P] : RSP

• dashed premises are already discharged
Secure Compilation Framework for Spectre

∀P ∈ source
⊢ P : SS

⊢ [·] : RSC
criteria equality

⊢ [·] : RSP

⊢ [P] : SS

⊢ [P] : SNI

• dashed premises are already discharged
• to show security: simply prove RSC
void f(int x) \mapsto \text{if}(x < A.size)\{y = B[A[x]]\} \quad \text{// A.size=16, A[128]=3}

\text{lfence}; y = B[A[x]]
void f(int x) \Rightarrow \text{if}(x < A\text{.size})\{y = B[A[x]]\} \quad // A\text{.size}=16, A[128]=3

\text{lfence} = \text{void f(int x) } \Rightarrow \text{if}(x < A\text{.size})\{\text{lfence}; y = B[A[x]]\}
RSC for `lfence`

```c
void f(int x) \mapsto \text{if}(x < A.\text{size})\{y = B[A[x]]\}
// A.\text{size}=16, A[128]=3
```

```
\text{\triangleright\triangleright\triangleright} \text{void f(int x) \mapsto \text{if}(x < A.\text{size})\{lfence; y = B[A[x]]\}}
```

call f 128

```
call f 128
pc : S
```

```
pc : S
```

```
lfence; y = B[ A[ 128 ] ]
pc : U
```

RSC for `lfence`

```c
void f(int x) { return if (x < A.size) { y = B[A[x]] } } // A.size=16, A[128]=3
```

```c
[·] = void f(int x) { lfence; y = B[A[x]] }
```

call `f 128`

```
if (128 < 16) { lfence; y = B[A[128]] }
```
void f(int x) ⇔ if(x < A.size)\{y = B[A[x]]\} \quad // A.size=16, A[128]=3
\[
\cdot \cdot = \text{void f}(\text{int x}) ⇔ \text{if}(x < \text{A.size})\{\text{lfence}; y = B[A[x]]\}
\]

call f 128
\begin{array}{c}
\text{if (128 < 16) \{ lfence; y = B[ A[ 128 ] ] \}}
\end{array}
\begin{array}{c}
\text{skip}
\end{array}
Proofs Insight

\[ \langle A \rangle / A \text{ executes} \]

\[ \langle A \rangle / [P] \text{ executes} \]

\[ \langle A \rangle / A \text{ executes} \]

\[ \alpha?^{\sigma} \quad \alpha?^{\sigma} \quad \alpha?^{\sigma} \]

\[ \overline{\alpha}_{1}^{\sigma} \quad \overline{\alpha}_{1}^{\sigma} \quad \overline{\alpha}_{2}^{\sigma} \]

\[ \text{ifz} \quad \text{ifz} \quad \alpha!^{\sigma} \]

\[ \alpha^{\sigma} \quad \alpha^{\sigma} \quad \alpha^{\sigma} \]
Proofs Insight

\[
\langle \langle A \rangle \rangle / A \quad \text{executes} \\
\]

\[
P / [P] \quad \text{executes} \\
\]

\[
\langle \langle A \rangle \rangle / A \quad \text{executes} \\
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]

\[
\]
Proofs Insight

\[\langle A \rangle / A \]
executes

\[P / [P] \]
executes

\[\langle A \rangle / A \]
executes

\[\alpha \Rightarrow \sigma\]
\[\alpha \Rightarrow \sigma\]
\[\alpha \Rightarrow \sigma\]
\[\alpha \Rightarrow \sigma\]

\[\text{ifz}\]

\[\alpha \Rightarrow \sigma\]
\[\alpha \Rightarrow \sigma\]
\[\alpha \Rightarrow \sigma\]
\[\alpha \Rightarrow \sigma\]

\[\alpha \Rightarrow \sigma\]
\[\alpha \Rightarrow \sigma\]
\[\alpha \Rightarrow \sigma\]
\[\alpha \Rightarrow \sigma\]

\[\alpha \Rightarrow \sigma\]
\[\alpha \Rightarrow \sigma\]
\[\alpha \Rightarrow \sigma\]
\[\alpha \Rightarrow \sigma\]

\[\alpha \Rightarrow \sigma\]
\[\alpha \Rightarrow \sigma\]
\[\alpha \Rightarrow \sigma\]
\[\alpha \Rightarrow \sigma\]
Proofs Insight

\[\langle A \rangle / A\] executes

\[P / [P]\] executes

\[\langle A \rangle / A\] executes

\[w = 0\]
Proofs Insight

\[ \langle A \rangle / A \text{ executes} \]

\[ P / [P] \text{ executes} \]

\[ \langle A \rangle / A \text{ executes} \]

either \( A \) or \([P]\) executes

\[ w = 0 \]

\[ rlb \]

\[ \alpha_{s} \sigma \]

\[ \alpha? \sigma \]

\[ \alpha! \sigma \]

\[ \alpha? \sigma \]

\[ \alpha! \sigma \]

\[ \alpha? \sigma \]

\[ \alpha! \sigma \]
What Then?

- SNIV1, SNIV2, SNIV4, SNIV5

Kocher et al. S&P'19
What Then?

• SNlv1, SNlv2, SNlv4, SNlv5

• Challenge: can the `lfence` compiler "mess" with SNlv2?
What Then?

- SNIv1, SNIv2, SNIv4, SNIv5

  Challenge: can the `lfence` compiler "mess" with SNIv2?

  Challenge: can we compose `lfence(SNIv1)` and `retptoline(SNIv5)`?

Kocher et al. S&P’19
Security Architectures
(e.g., Cheri/ARM Morello, Sancus/Intel SGX, ...) Toplas’15, CSF’21, ...
Mechanise Cryptographic Proofs

CSF’24 + wip
Robust Hyperproperty Preservation

\[ \cdot \vdash \text{RHP} \overset{\text{def}}{=} \forall P, A. \exists A. \forall t. \]

\[
A[[P]] \sim t \iff A[P] \sim t
\]
Robust Hyperproperty Preservation

\[ [\cdot] \vdash \text{RHP} \overset{\text{def}}{=} \forall P, A. \exists A. \forall t. \]

\[ A[[P]] \sim t \iff A[P] \sim t \]
Universal Composability: $UC$

- **gold standard** for proving security of crypto protocols under concurrent composition
Universal Composability: $UC$

- gold standard for proving security of crypto protocols under concurrent composition
- overcome main drawback in protocol vulnerabilities: composition
Universal Composability: \textit{UC}

- **gold standard** for proving security of crypto protocols under concurrent composition
- overcome main drawback in protocol vulnerabilities: \textit{composition}
- many flavours: \textit{UC}, SaUCy, iUC, ...

Canetti '01, Liao \textit{et al.} PLDI'19, Camenisch \textit{et al.} Asiacrypt'19
Universal Composability:  \textit{UC}

- \textbf{gold standard} for proving security of crypto protocols under concurrent composition
- overcome main drawback in protocol vulnerabilities: \textit{composition}
- many flavours: \textit{UC}, \textit{SaUCy}, \textit{iUC}, …

Canetti ’01, Liao et al. PLDI’19, Camenisch et al. Asiacrypt’19

This talk: generic flavour, geared towards the newer theories
UC Base Notions: ITMs

Canetti and Fischlin Crypto’01

• protocols $\Pi$ (using concrete crypto)

commitment for $b \in \{0, 1\}$ with SID $sid$:

- compute $G_{pk_b}(r)$ for random $r \in \{0, 1\}^n$
- set $y = G_{pk_b}(r)$ for $b = 0$, or $y = G_{pk_b}(r) \oplus \sigma$ for $b = 1$
- send $(Com, sid, y)$ to the receiver

Upon receiving $(Com, sid, y)$ from $P_i$, $P_j$ outputs $(Receipt, sid, cid, P_i, P_j)$


**UC Base Notions: ITMs**

- **protocols** $\Pi$ (using concrete crypto)

  commitment for $b \in \{0, 1\}$ with SID $sid$:
  
  compute $G_{pk_b}(r)$ for random $r \in \{0, 1\}^n$
  
  set $y = G_{pk_b}(r)$ for $b = 0$, or $y = G_{pk_b}(r) \oplus \sigma$ for $b = 1$
  
  send $(Com, sid, y)$ to the receiver

  Upon receiving $(Com, sid, y)$ from $P_i$, $P_j$ outputs $(Receipt, sid, cid, P_i, P_j)$

- **functionalities** $F$ (using abstract notions)

  1. Upon receiving a value $(Commit, sid, P_i, P_j, b)$ from $P_i$, where $b \in \{0, 1\}$, record the value $b$ and send the message $(Receipt, sid, P_i, P_j)$ to $P_j$ and $S$. Ignore any subsequent Commit messages.
UC Base Notions: ITMs

- **protocols** $\Pi$ (using concrete crypto)
  
  commitment for $b \in \{0, 1\}$ with SID sid:
  
  compute $G_{pk_b}(r)$ for random $r \in \{0, 1\}^n$
  
  set $y = G_{pk_b}(r)$ for $b = 0$, or $y = G_{pk_b}(r) \oplus \sigma$ for $b = 1$
  
  send $(Com, sid, y)$ to the receiver
  
  Upon receiving $(Com, sid, y)$ from $P_i$, $P_j$ outputs $(Receipt, sid, cid, P_i, P_j)$

- **functionalities** $F$ (using abstract notions)
  
  1. Upon receiving a value $(Commit, sid, P_i, P_j, b)$ from $P_i$, where $b \in \{0, 1\}$, record the value $b$ and send the message $(Receipt, sid, P_i, P_j)$ to $P_j$ and $S$. Ignore any subsequent Commit messages.

- **attackers** $A$ & $S$ (corrupting parties etc.)
UC Base Notions: ITMs

- **protocols** $\Pi$ (using concrete crypto)

  commitment for $b \in \{0, 1\}$ with SID $\text{sid}$:
  
  compute $G_{pk_b}(r)$ for random $r \in \{0, 1\}^n$
  set $y = G_{pk_b}(r)$ for $b = 0$, or $y = G_{pk_b}(r) \oplus \sigma$ for $b = 1$
  send $(\text{Com}, \text{sid}, y)$ to the receiver

  Upon receiving $(\text{Com}, \text{sid}, y)$ from $P_i$, $P_j$ outputs $(\text{Receipt}, \text{sid}, \text{cid}, P_i, P_j)$

- **functionalities** $F$ (using abstract notions)

  1. Upon receiving a value $(\text{Commit}, \text{sid}, P_i, P_j, b)$ from $P_i$, where $b \in \{0, 1\}$, record the value $b$ and send the message $(\text{Receipt}, \text{sid}, P_i, P_j)$ to $P_j$ and $S$. Ignore any subsequent Commit messages.

- **attackers** $A \& S$ (corrupting parties etc.)

- **environments** $Z$ (objective witness)
\( UC \) \( \text{(Semi-formally)} \)

\[
\begin{align*}
\Pi & \quad \uparrow \quad A \\
\downarrow & \quad \downarrow \\
Z & \quad \Rightarrow \quad 0/1
\end{align*}
\]

\[
\begin{align*}
\Pi & \quad \uparrow \quad A \\
\downarrow & \quad \downarrow \\
F & \quad \Rightarrow \quad S
\end{align*}
\]

\( \leftrightarrow \) represent communication channels
$UC$ (Semi-formally)

$\Pi \vdash_{UC} F \overset{\text{def}}{=} \forall \text{poly} \ A, \exists S, \forall Z.$

$\text{Exec}[Z, A, \Pi] \approx \text{Exec}[Z, S, F]$

$\leftrightarrow$ represent communication channels
∀poly A, ∃S, ∀Z.

∀P, A. ∃A. ∀t.
A Closer Look

∀ poly A, ∃ S, ∀ Z.

∀ P, A. ∃ A. ∀ t.

Isabelle’d both perfect and computational \textit{UC}
## Analogy

<table>
<thead>
<tr>
<th>$UC$</th>
<th>$SC$</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>compiled program</td>
</tr>
<tr>
<td>concrete attacker</td>
<td>target context</td>
</tr>
<tr>
<td>ideal functionality</td>
<td>source program</td>
</tr>
<tr>
<td>simulator</td>
<td>source context</td>
</tr>
<tr>
<td>environment, output</td>
<td>trace, semantics</td>
</tr>
<tr>
<td>communication</td>
<td>linking</td>
</tr>
<tr>
<td>probabilistic equiv.</td>
<td>trace equality</td>
</tr>
</tbody>
</table>

- $\mathsf{protocol} \: \mathsf{\Pi} \: [\mathsf{P}]$  
- $\mathsf{concrete\: attacker} \: \mathsf{A} \: \mathsf{A}$  
- $\mathsf{ideal\: functionality} \: \mathsf{F} \: \mathsf{P}$  
- $\mathsf{simulator} \: \mathsf{S} \: \mathsf{A}$  
- $\mathsf{environment,\: output} \: \mathsf{Z,\: 0/1} \: \mathsf{t,} \: \rightsquigarrow$  
- $\mathsf{communication} \: \leftrightarrow \: \mathsf{[]}$  
- $\mathsf{probabilistic\: equiv.} \: \approx \: \leftrightarrow$
## Analogy

<table>
<thead>
<tr>
<th>UC</th>
<th>SC</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>( \Pi )</td>
</tr>
<tr>
<td>concrete attacker</td>
<td>( A )</td>
</tr>
<tr>
<td>ideal functionality</td>
<td>( F )</td>
</tr>
<tr>
<td>simulator</td>
<td>( S )</td>
</tr>
<tr>
<td>environment, output</td>
<td>( Z, \ 0/1 )</td>
</tr>
<tr>
<td>communication</td>
<td>( \leftrightarrow )</td>
</tr>
<tr>
<td>probabilistic equiv.</td>
<td>( \approx )</td>
</tr>
<tr>
<td>human translation</td>
<td>( \Pi \rightarrow F )</td>
</tr>
<tr>
<td>general composition result</td>
<td>([\cdot\cdot]): ( P \rightarrow P ) compiler</td>
</tr>
<tr>
<td></td>
<td>( \ldots )</td>
</tr>
</tbody>
</table>

- \( \Pi \rightarrow F \): human translation
- \([\cdot\cdot]\): compiled program
- \( [P] \): compiled program
- \( A \): target context
- \( A \): source context
- \( P \): source program
- \( \approx \): trace equality
- \( \leftrightarrow \): linking
- \( t, \sim \): trace, semantics
Analogy Results

• **transfer** $UC$ results from ITMs to any $S/T$
Analogy Results

- transfer $UC$ results from ITMs to any $S/T$
- mechanise $UC$ results as RHP results
Analogy Results

- transfer UC results from ITMs to any S/T
- mechanise UC results as RHP results known in computer-aided crypto

Haagh et al. CSF'18
Analogy Results

- transfer $UC$ results from ITMs to any $S/T$
- mechanise $UC$ results as RHP results known in computer-aided crypto
  Haagh et al. CSF’18
- Mechanised $UC$ for 1-Bit Commitment in Deepsec
- Mechanised $UC$ for 1/2 Wireguard in Cryptooverif
  CSF’24
Conclusion
Conclusion

- secure compilation threat model
Conclusion

• secure compilation threat model

• formal foundations: RSC, RHP
Conclusion

• secure compilation **threat model**

• formal foundations: **RSC, RHP**

• robust compilation **use-cases** (MS, CT, SNI)
Conclusion

- secure compilation threat model
- formal foundations: RSC, RHP
- robust compilation use-cases (MS, CT, SNI)
- connection with UC
Future

- More foundations questions?
Future

• More foundations questions?
• SC for emerging security archs?
Future

• More foundations questions?
• SC for emerging security archs?
• SC for more properties?
Future

• More foundations questions?
• SC for emerging security archs?
• SC for more properties?
• SC for different languages?

Come to PRISC'25, co-located with POPL'25.
Future

• More foundations questions?
• SC for emerging security archs?
• SC for more properties?
• SC for different languages?
• Other UC-like connections?
Future

- More foundations questions?
- SC for emerging security archs?
- SC for more properties?
- SC for different languages?
- Other $UC$-like connections?
- More mechanised $UC$ protocols?
Future

• More foundations questions?
• SC for emerging security archs?
• SC for more properties?
• SC for different languages?
• Other $UC$-like connections?
• More mechanised $UC$ protocols?

Come to PRISC’25, co-located with POPL’25.
Questions?