Secure Compilation: Formal Foundations and (Some) Applications

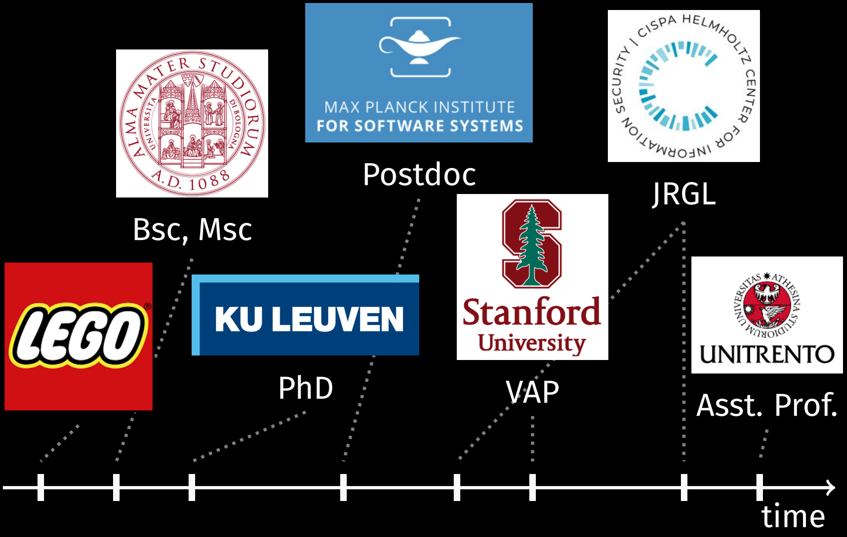
Marco Patrignani

UNIVERSITÀ DI TRENTO

03 April 2024

Who Am I ?

Marco Patrignani


LEGO
ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA A.D. 1088
KU LEUVEN
MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS
Stanford University
CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY
UNITRENTO

Bsc, Msc

PhD

Postdoc

VAP

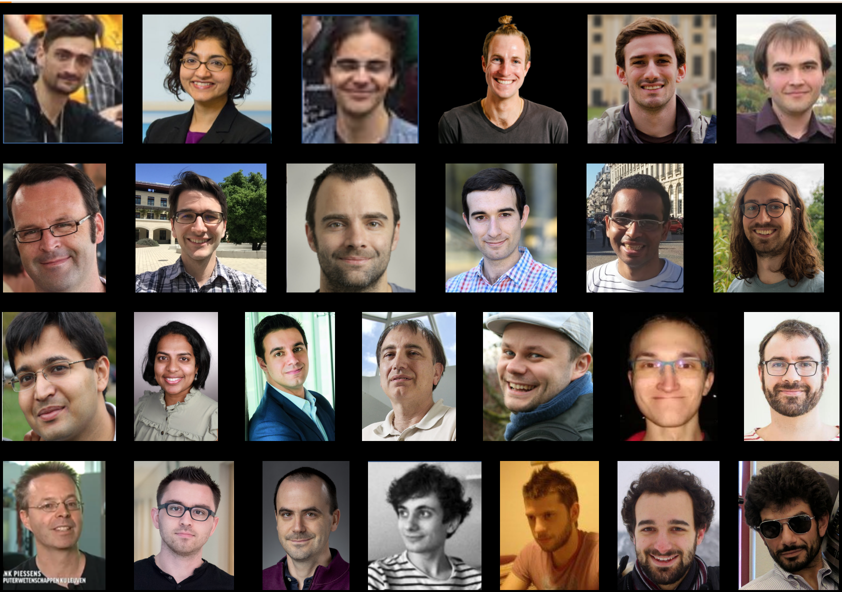
JRGL

Asst. Prof.

time

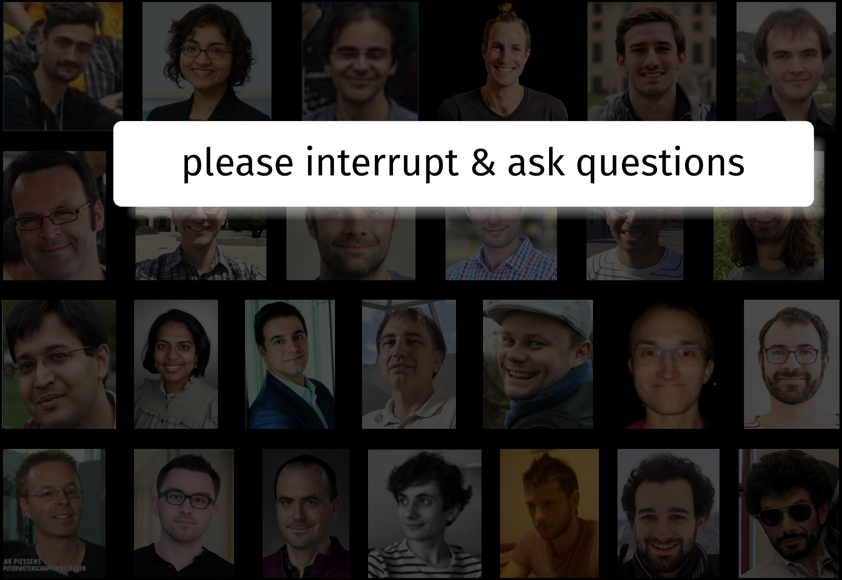
Special Thanks to:

(wrt the contents of this talk)



Special Thanks to:

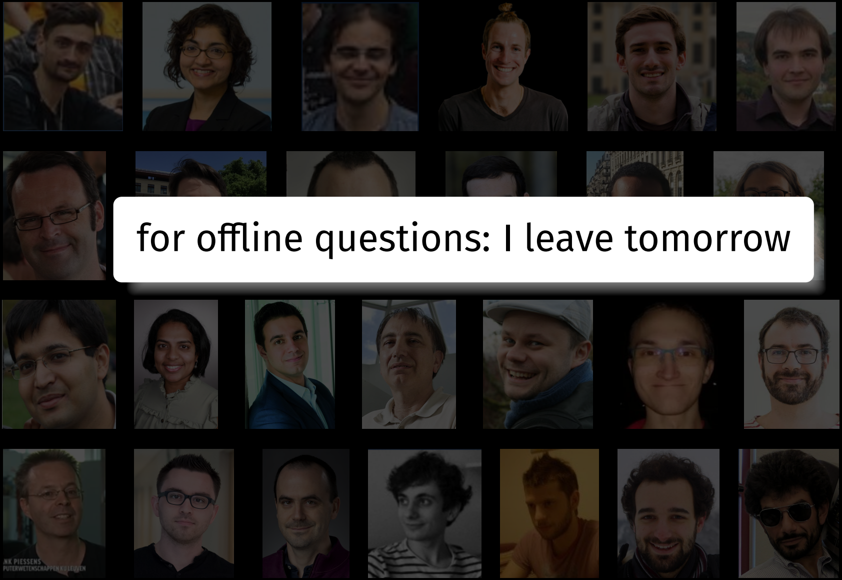
(wrt the contents of this talk)



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, >λ=, F*, …) provide:

- helpful abstractions to write secure code

Programming Languages: Pros and Cons

Good PLs (R, TypeScript, λ, F, …) provide:

- helpful abstractions to write secure code

but

- when compiled ($\llbracket \cdot \rrbracket$) and linked with adversarial target code

Programming Languages: Pros and Cons

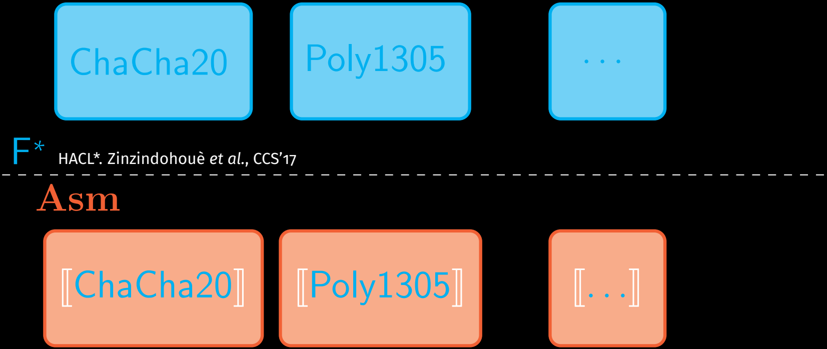
Good PLs (R, TypeScript, >λ=, ★, …) provide:

- helpful abstractions to write secure code

but

- when compiled ($[\![\cdot]\!]$) and linked with adversarial target code
- these abstractions are NOT enforced

Secure Compilation: Example



ChaCha20

Poly1305

$\cdots$

F^*

HACL*. Zinzindohouè *et al.*, CCS'17

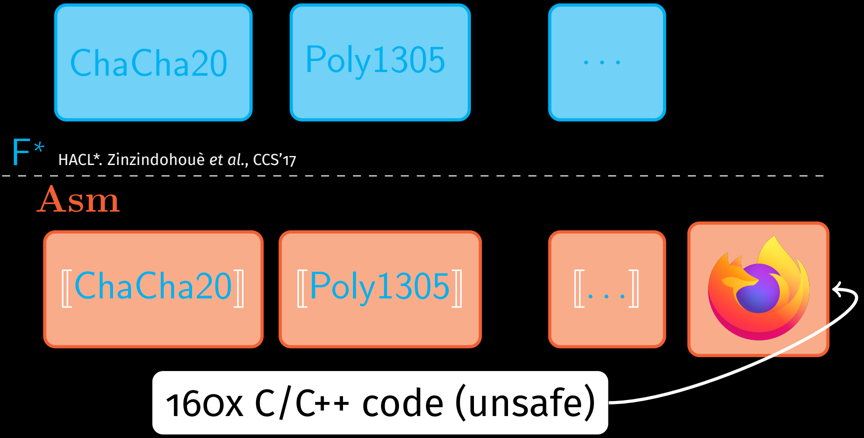
Asm

$[\![\text{ChaCha20}]\!]$

$[\![\text{Poly1305}]\!]$

$[\![\ldots]\!]$

Secure Compilation: Example



ChaCha20

Poly1305

. . .

F*

HACL*. Zinzindohouè *et al.*, CCS'17

Asm

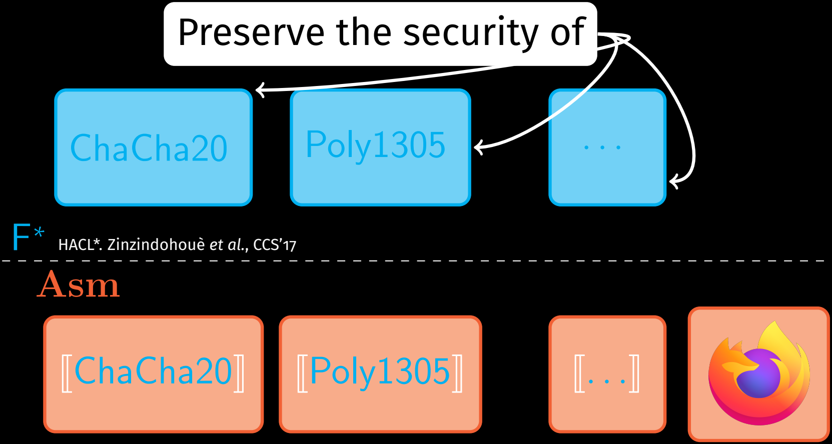
⟦ChaCha20⟧

⟦Poly1305⟧

⟦. . .⟧

160x C/C++ code (unsafe)

Secure Compilation: Example



Preserve the security of

ChaCha20

Poly1305

$\cdots$

F^*

HACL*. Zinzindohouè *et al.*, CCS'17

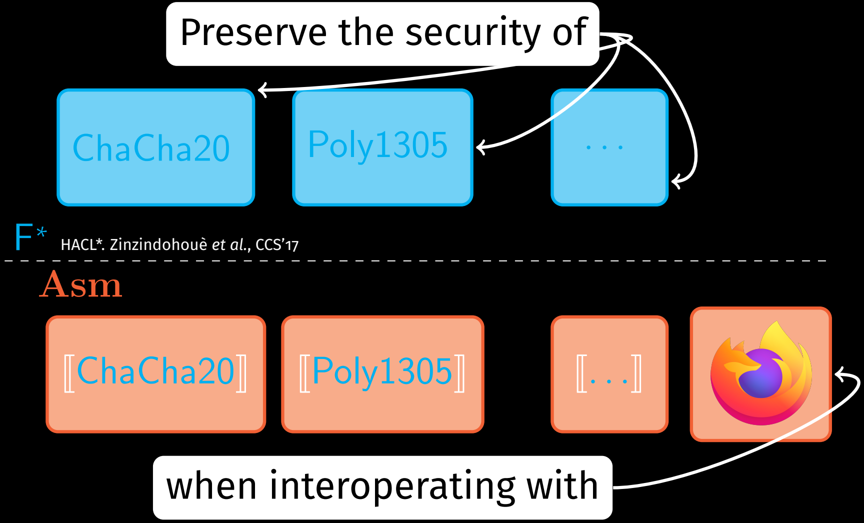
Asm

$[\![\text{ChaCha20}]\!]$

$[\![\text{Poly1305}]\!]$

$[\![\ldots]\!]$

Secure Compilation: Example



Preserve the security of

ChaCha20

Poly1305

$\dots$

F^*

HACL*. Zinzindohouè *et al.*, CCS'17

Asm

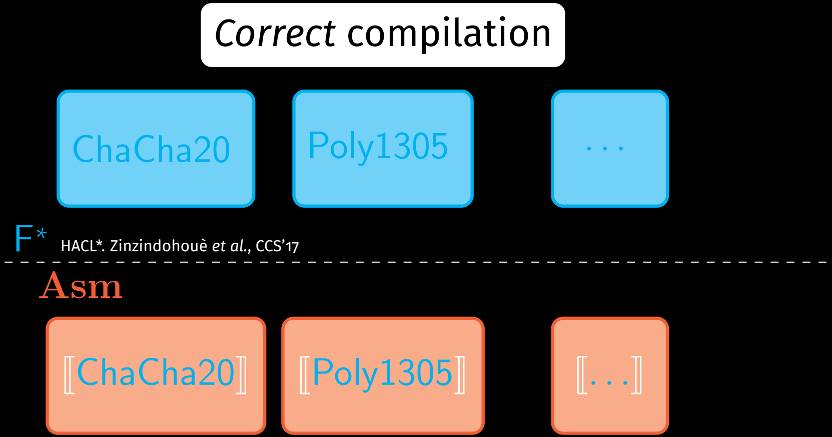
$[\![\text{ChaCha20}]\!]$

$[\![\text{Poly1305}]\!]$

$[\![\dots]\!]$

when interoperating with

Secure Compilation: Example



Correct compilation

ChaCha20

Poly1305

...

F^*

HACL*. Zinzindohouè *et al.*, CCS’17

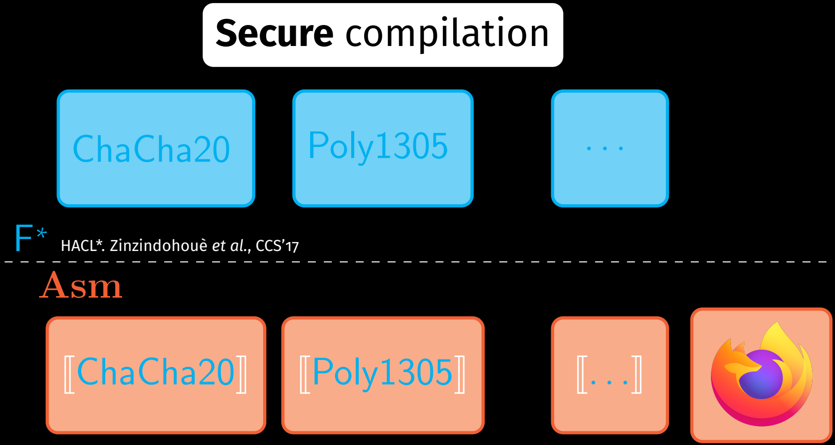
Asm

⟦ChaCha20⟧

⟦Poly1305⟧

⟦...⟧

Secure Compilation: Example



Secure compilation

ChaCha20

Poly1305

. . .

F*

HACL*. Zinzindohouè *et al.*, CCS'17

Asm

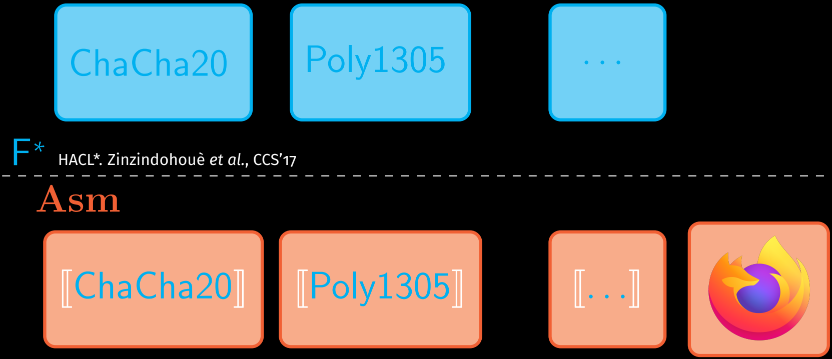
$[\![\text{ChaCha20}]\!]$

$[\![\text{Poly1305}]\!]$

$[\![\ldots]\!]$

Secure Compilation: Example

Enable source-level security reasoning



ChaCha20

Poly1305

. . .

F*

HACL*. Zinzindohouè *et al.*, CCS'17

Asm

⟦ChaCha20⟧

⟦Poly1305⟧

⟦. . .⟧

Quest for Foundations

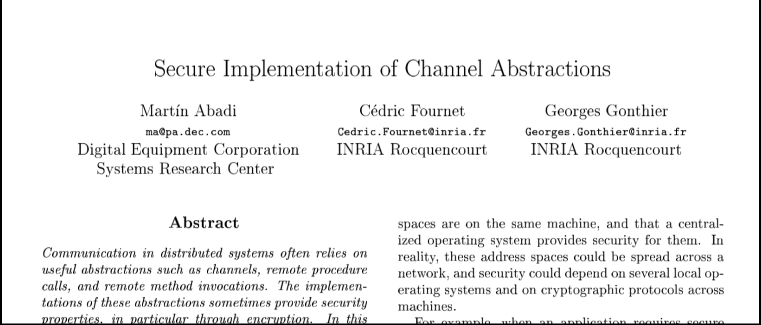
What does it mean for a compiler to be secure?

Quest for Foundations

What does it mean for a compiler to be secure?

Analogous questions are answered for type systems, correct compilation, ...

Once Upon a Time in Process Algebra



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

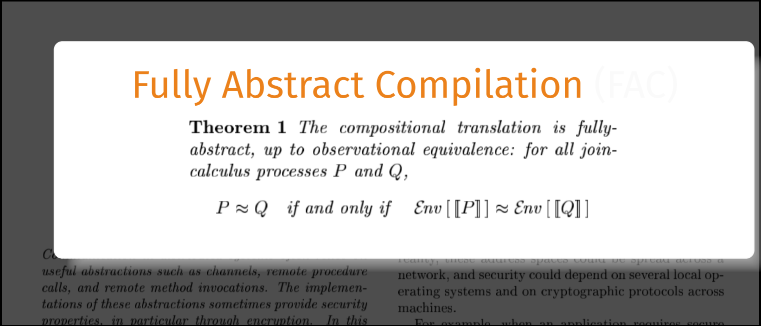
Communication in distributed systems often relies on useful abstractions such as channels, remote procedure calls, and remote method invocations. The implementations of these abstractions sometimes provide security properties, in particular through encryption. In this

spaces are on the same machine, and that a centralized operating system provides security for them. In reality, these address spaces could be spread across a network, and security could depend on several local operating systems and on cryptographic protocols across machines.

For example, when an application requires secure

Challenge: 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,*

$$P \approx Q \quad \textit{if and only if} \quad \mathcal{E}nv\,[\,[\![P]\!]\,] \approx \mathcal{E}nv\,[\,[\![Q]\!]\,]$$

Co 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

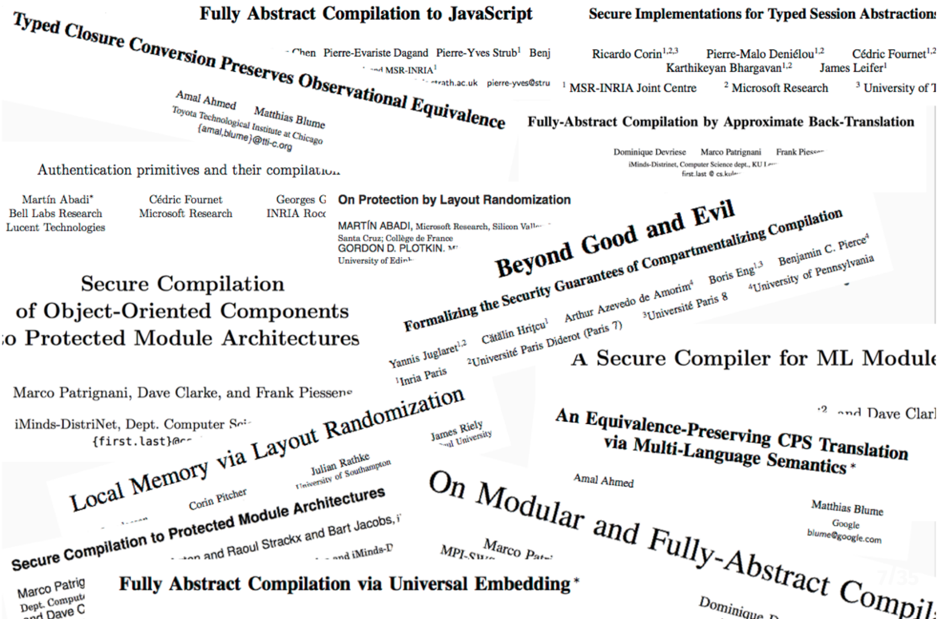
reality, these address spaces could be spread across a network, and security could depend on several local operating systems and on cryptographic protocols across machines.

For example, when an application requires secure

Challenge: define that their implementation of secure channels via cryptography was secure

Fully Abstract Compilation Influence

ACM CSUR'19



Typed Closure Conversion Preserves Observational Equivalence

Amal Ahmed Matthias Blume
Toyota Technological Institute at Chicago
{amal,blume}@tti-c.org

Fully Abstract Compilation to JavaScript

-Chen Pierre-Evariste Dagand Pierre-Yves Strub[1] Benj
-nd MSR-INRIA[1]
-trath.ac.uk pierre-yves@stru

Secure Implementations for Typed Session Abstractions

Ricardo Corin[1,2,3] Pierre-Malo Deniélou[1,2] Cédric Fournet[1,2]
Karthikeyan Bhargavan[1,2] James Leifer[1]
[1] MSR-INRIA Joint Centre [2] Microsoft Research [3] University of T

Fully-Abstract Compilation by Approximate Back-Translation

Dominique Devriese Marco Patrignani Frank Piess-
iMinds-Distrinet, Computer Science dept., KU L-
first.last @ cs.kule-

Authentication primitives and their compilation

Martín Abadi* Cédric Fournet Georges G
Bell Labs Research Microsoft Research INRIA Rocq
Lucent Technologies

On Protection by Layout Randomization

MARTÍN ABADI, Microsoft Research, Silicon Vall-
Santa Cruz; Collège de France
GORDON D. PLOTKIN, M-
University of Edin-

Beyond Good and Evil

Formalizing the Security Guarantees of Compartmentalizing Compilation

Yannis Juglaret[1,2] Cătălin Hriţcu[1] Arthur Azevedo de Amorim[4] Boris Eng[1,3] Benjamin C. Pierce[4]
[1]Inria Paris [2]Université Paris Diderot (Paris 7) [3]Université Paris 8 [4]University of Pennsylvania

Secure Compilation of Object-Oriented Components to Protected Module Architectures

Marco Patrignani, Dave Clarke, and Frank Piessens

iMinds-DistriNet, Dept. Computer Sci-
{first.last}@c-

A Secure Compiler for ML Module

-2 and Dave Clar

Local Memory via Layout Randomization

Corin Pitcher Julian Rathke James Riely
University of Southampton -ul University

An Equivalence-Preserving CPS Translation via Multi-Language Semantics *

Amal Ahmed Matthias Blume
Google
blume@google.com

Secure Compilation to Protected Module Architectures

-on and Raoul Strackx and Bart Jacobs, i
-nd iMinds-D

On Modular and Fully-Abstract Compil

Marco Patr-
MPI-SWS

Dominique D

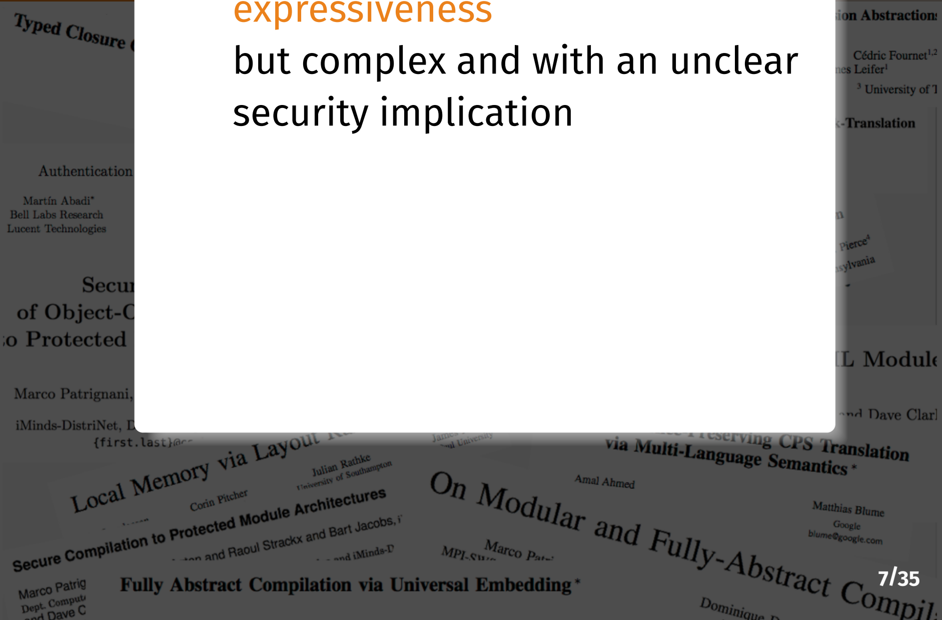
Marco Patrig
Dept. Comput
and Dave C

Fully Abstract Compilation via Universal Embedding *

Fully A

CSUR'19

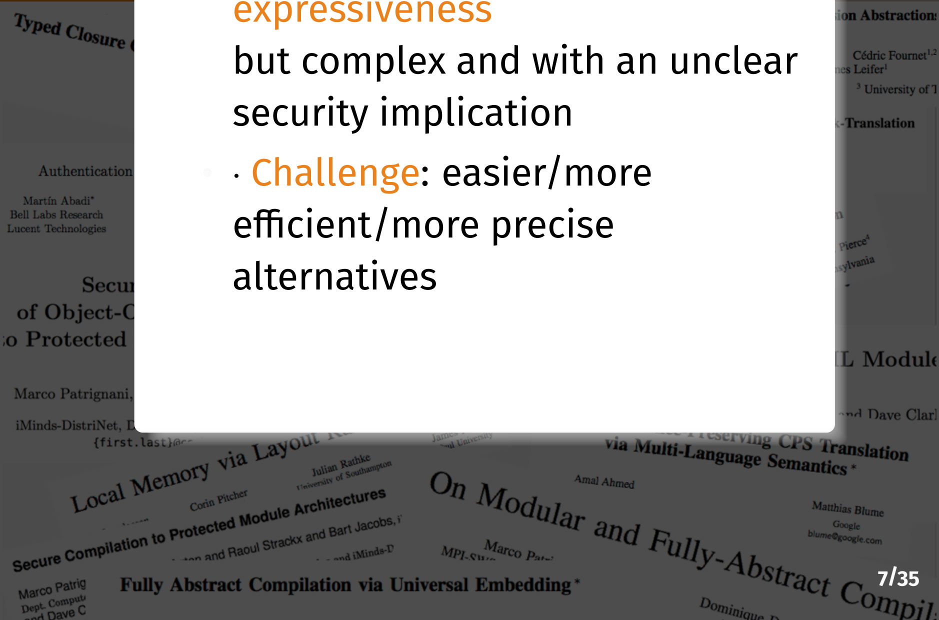
- FAC: useful for language expressiveness
  but complex and with an unclear security implication



Fully A

CSUR'19

- FAC: useful for language expressiveness
  but complex and with an unclear security implication
- Challenge: easier/more efficient/more precise alternatives



Typed Closure

Authentication

Martín Abadi*
Bell Labs Research
Lucent Technologies

Secu
of Object-C
o Protected

Marco Patrignani,

iMinds-DistriNet, D
{first.last}@c

ion Abstraction

Cédric Fournet1,2
nes Leifer1
3 University of T

-Translation

Pierce4
sylvania

L Module

nd Dave Clarl

Local Memory via Layout

Corin Pitcher

Julian Rathke
University of Southampton

Secure Compilation to Protected Module Architectures

on and Raoul Strackx and Bart Jacobs,
and iMinds-D

Marco Patrig
Dept. Comput
and Dave C

Fully Abstract Compilation via Universal Embedding*

On Modular and Fully-Abstract Compil

Marco Pat
MPI-SWS

Preserving CPS Translation
via Multi-Language Semantics*

Amal Ahmed

Matthias Blume
Google
blume@google.com

Dominique

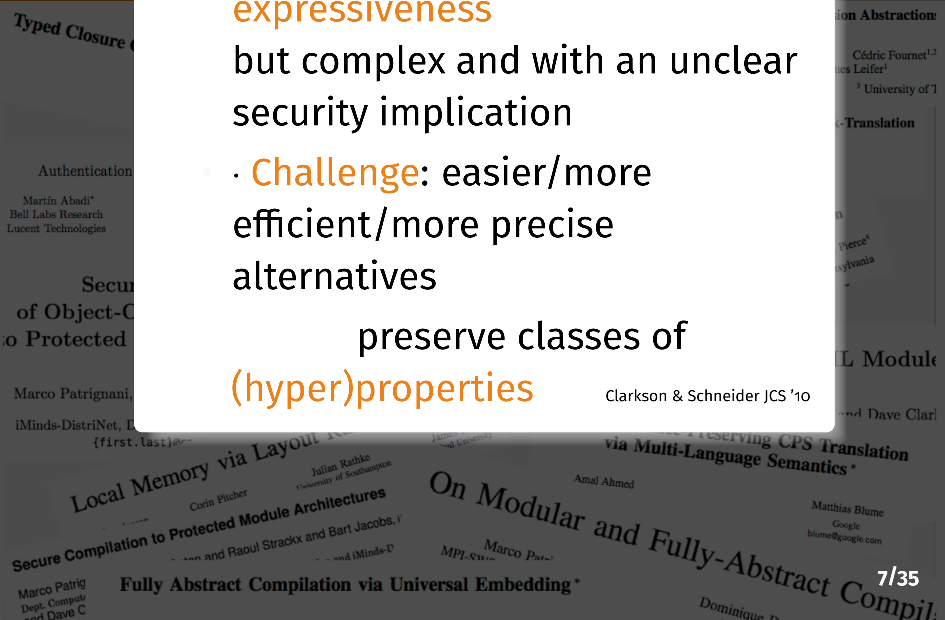
Fully A

CSUR'19

- 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



Typed Closure C

Authentication

Martín Abadi*
Bell Labs Research
Lucent Technologies

Secur
of Object-C
o Protected

Marco Patrignani,
iMinds-DistriNet, D
{first.last}@c

ion Abstraction

Cédric Fournet
es Leifer
University of T

-Translation

Pierce
sylvania

L Module

nd Dave Clar

Local Memory via Layout R

Julian Rathke
University of Southampton

Corin Pitcher

Secure Compilation to Protected Module Architectures

on and Raoul Strackx and Bart Jacobs,
and iMinds-D

Preserving CPS Translation
via Multi-Language Semantics *

Amal Ahmed

Matthias Blume
Google
blume@google.com

On Modular and Fully-Abstract Compil

Marco Pat
MPI-SWS

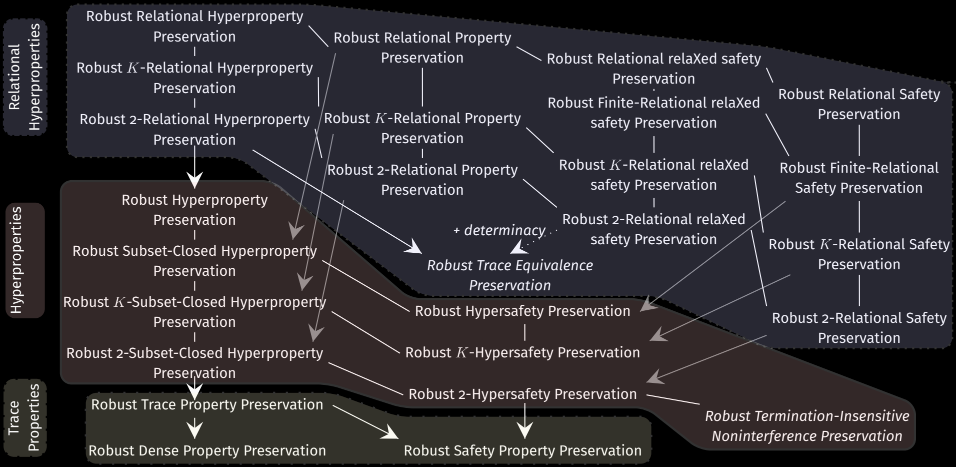
Marco Patrig
Dept. Comput
and Dave C

Fully Abstract Compilation via Universal Embedding *

Dominique D

Robust Compilation (RC) Criteria

CSF'19, ESOP'20, Toplas'21



Relational Hyperproperties

Hyperproperties

Trace Properties

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

+ determinacy

Robust Trace Equivalence Preservation

Robust Hyperproperty Preservation

Robust Subset-Closed Hyperproperty Preservation

Robust K-Subset-Closed Hyperproperty Preservation

Robust 2-Subset-Closed Hyperproperty Preservation

Robust Hypersafety Preservation

Robust K-Hypersafety Preservation

Robust 2-Hypersafety Preservation

Robust Termination-Insensitive Noninterference Preservation

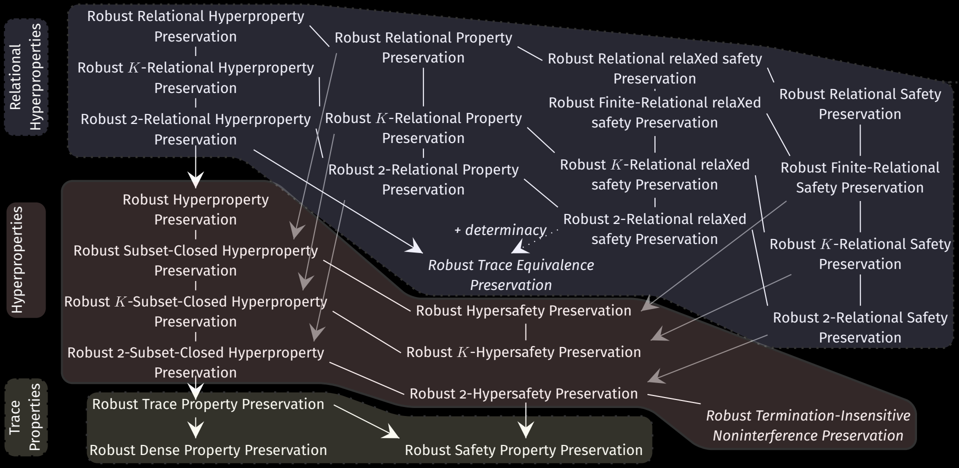
Robust Trace Property Preservation

Robust Dense Property Preservation

Robust Safety Property Preservation

Robust Compilation (RC) Criteria

CSF'19, ESOP'20, Toplas'21



Relational Hyperproperties

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

+ *determinacy*

Robust Trace Equivalence Preservation

Hyperproperties

Robust Hyperproperty Preservation

Robust Subset-Closed Hyperproperty Preservation

Robust K-Subset-Closed Hyperproperty Preservation

Robust 2-Subset-Closed Hyperproperty Preservation

Robust Hypersafety Preservation

Robust K-Hypersafety Preservation

Robust 2-Hypersafety Preservation

Robust Termination-Insensitive Noninterference Preservation

Trace Properties

Robust Trace Property Preservation

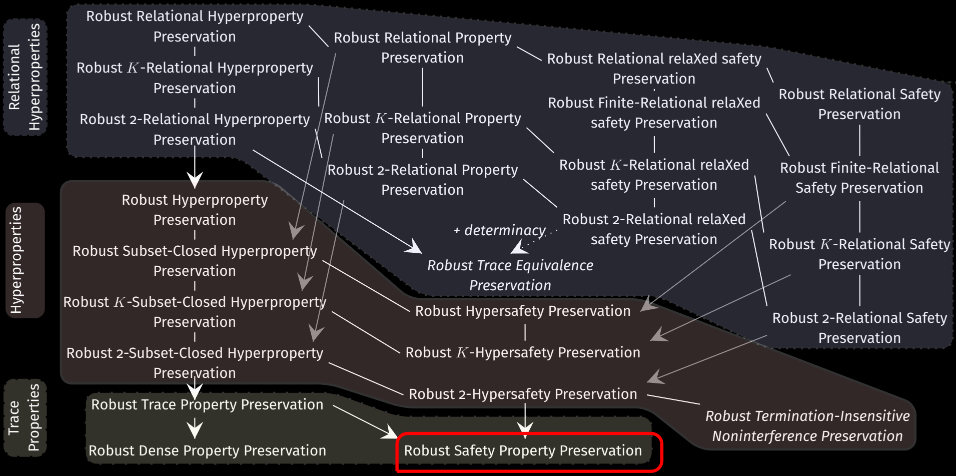
Robust Dense Property Preservation

Robust Safety Property Preservation

Tradeoffs for code efficiency, security guarantees, proof complexity

Robust Compilation (RC) Criteria

CSF'19, ESOP'20, Toplas'21



Relational Hyperproperties
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
+ determinacy
Robust Trace Equivalence Preservation
Hyperproperties
Robust Hyperproperty Preservation
Robust Subset-Closed Hyperproperty Preservation
Robust K-Subset-Closed Hyperproperty Preservation
Robust 2-Subset-Closed Hyperproperty Preservation
Robust Hypersafety Preservation
Robust K-Hypersafety Preservation
Robust 2-Hypersafety Preservation
Robust Termination-Insensitive Noninterference Preservation
Trace Properties
Robust Trace Property Preservation
Robust Dense Property Preservation
Robust Safety Property Preservation

Tradeoffs for code efficiency, security guarantees, proof complexity

Robust Criteria: Intuition

Each point has two equivalent criteria:

- $\mathrm{Property-ful}:$
  - clearly tells what class it preserves

Robust Criteria: Intuition

Each point has two equivalent criteria:

- $\mathbf{Property-ful}:$
    - + clearly tells what class it preserves
    - - harder 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

Robust Criteria: Intuition

Each point has two equivalent criteria:

- $\text{Property} - \text{ful}:$
  - + clearly tells what class it preserves
  - - harder to prove
- $\text{Property} - \text{free}:$
  - + easier to prove
  - - unclear what security classes are preserved

In Depth Example: RSC

ESOP'19, TOPLAS'21

$[\![\cdot]\!]$ = compiler $[\![\cdot]\!] : \mathsf{RSP} \overset{\text{def}}{=}$

In Depth Example: RSC

ESOP'19, TOPLAS'21

$[\![\cdot]\!]$ = compiler

π / π = set of traces

$$[\![\cdot]\!] : \mathrm{RSP} \overset{\mathrm{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.$$

In Depth Example: RSC

ESOP'19, TOPLAS'21

$\llbracket\cdot\rrbracket$ = compiler

π / π = set of traces

P = partial program

$$\llbracket\cdot\rrbracket : \mathsf{RSP} \overset{\mathsf{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathsf{P}.$$

In Depth Example: RSC

ESOP'19, TOPLAS'21

$[\![\cdot]\!]$ = compiler

π/π = set of traces

P = partial program

A/**A** = attacker

t/**t** = trace of events

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

$$\text{if } (\forall \mathsf{A}, \mathsf{t}.$$

In Depth Example: RSC

ESOP'19, TOPLAS'21

$[\![\cdot]\!]$ = compiler

π / π = set of traces

P = partial program

A / **A** = attacker

t / **t** = trace of events

$[\cdot]$ = linking

$\rightsquigarrow$ / $\rightsquigarrow$ = trace semantics

$$[\![\cdot]\!] : \mathrm{RSP} \overset{\mathrm{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathrm{P}.$$

$$\text{if } (\forall \mathrm{A}, \mathrm{t}.\ \mathrm{A}\left[\mathrm{P}\right] \rightsquigarrow \mathrm{t}$$

In Depth Example: RSC

ESOP'19, TOPLAS'21

$\llbracket\cdot\rrbracket$ = compiler
π/π = set of traces
P = partial program
A/$\mathbf{A}$ = attacker
t/$\mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow$/$\rightsquigarrow$ = trace semantics

$$\llbracket\cdot\rrbracket : \mathrm{RSP} \overset{\text{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathrm{P}.$$

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

In Depth Example: RSC

ESOP'19, TOPLAS'21

$[\![\cdot]\!]$ = compiler

π / π = set of traces

P = partial program

A / $\mathbf{A}$ = attacker

t / $\mathbf{t}$ = trace of events

$[\cdot]$ = linking

$\rightsquigarrow / \rightsquigarrow$ = trace semantics

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

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

$$\text{then } (\forall \mathbf{A}, \mathbf{t}.$$

In Depth Example: RSC

ESOP'19, TOPLAS'21

$[\![\cdot]\!]$ = compiler

π / π = set of traces

P = partial program

A / **A** = attacker

t / **t** = trace of events

$[\cdot]$ = linking

$\rightsquigarrow$ / $\rightsquigarrow$ = trace semantics

$$[\![\cdot]\!] : \mathrm{RSP} \overset{\text{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathsf{P}.$$

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

$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}\left[[\![\mathsf{P}]\!]\right] \rightsquigarrow \mathbf{t} \Rightarrow$$

In Depth Example: RSC

ESOP'19, TOPLAS'21

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program
$A / \mathbf{A}$ = attacker
$t / \mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow / \rightsquigarrow$ = trace semantics

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

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

$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[[\![P]\!]] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \pi)$$

In Depth Example: RSC

ESOP'19, TOPLAS'21

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program
A / $\mathbf{A}$ = attacker
t / $\mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow$ / $\rightsquigarrow$ = trace semantics

$$[\![\cdot]\!] : \mathrm{RSP} \overset{\text{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathsf{P}.$$

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

$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[[\![\mathsf{P}]\!]] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \pi)$$

$$[\![\cdot]\!] : \mathrm{RSC} \overset{\text{def}}{=}$$

In Depth Example: RSC

ESOP’19, TOPLAS’21

$\llbracket\cdot\rrbracket$ = compiler
$\pi / \boldsymbol{\pi}$ = set of traces
P = partial program
$\mathsf{A} / \mathbf{A}$ = attacker
$\mathsf{t} / \mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow / \rightsquigarrow$ = trace semantics
$\mathsf{m} / \mathbf{m}$ = prefix of a trace

$$\llbracket\cdot\rrbracket : \mathsf{RSP} \overset{\text{def}}{=} \forall \pi \approx \boldsymbol{\pi} \in \mathit{Safety}.\ \forall \mathsf{P}.$$

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

$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[\llbracket\mathsf{P}\rrbracket] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \boldsymbol{\pi})$$

$$\llbracket\cdot\rrbracket : \mathsf{RSC} \overset{\text{def}}{=} \forall \mathsf{P}, \mathbf{A}, \mathbf{m}.$$

In Depth Example: RSC

ESOP'19, TOPLAS'21

$\llbracket\cdot\rrbracket$ = compiler

π/$\boldsymbol{\pi}$ = set of traces

P = partial program

A/$\mathbf{A}$ = attacker

t/$\mathbf{t}$ = trace of events

$[\cdot]$ = linking

$\rightsquigarrow$/$\boldsymbol{\rightsquigarrow}$ = trace semantics

m/$\mathbf{m}$ = prefix of a trace

$$\llbracket\cdot\rrbracket : \mathrm{RSP} \overset{\text{def}}{=} \forall \pi \approx \boldsymbol{\pi} \in \mathit{Safety}.\ \forall \mathsf{P}.$$

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

$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}\left[\llbracket\mathsf{P}\rrbracket\right] \boldsymbol{\rightsquigarrow} \mathbf{t} \Rightarrow \mathbf{t} \in \boldsymbol{\pi})$$

$$\llbracket\cdot\rrbracket : \mathrm{RSC} \overset{\text{def}}{=} \forall \mathsf{P}, \mathbf{A}, \mathbf{m}.$$

$$\text{if } \mathbf{A}\left[\llbracket\mathsf{P}\rrbracket\right] \boldsymbol{\rightsquigarrow} \mathbf{m}$$

In Depth Example: RSC

ESOP'19, TOPLAS'21

$[\![\cdot]\!]$ = compiler

π / $\boldsymbol{\pi}$ = set of traces

P = partial program

A / $\mathbf{A}$ = attacker

t / $\mathbf{t}$ = trace of events

$[\cdot]$ = linking

$\rightsquigarrow$ / $\boldsymbol{\rightsquigarrow}$ = trace semantics

m / $\mathbf{m}$ = prefix of a trace

$$[\![\cdot]\!] : \mathrm{RSP} \overset{\text{def}}{=} \forall \pi \approx \boldsymbol{\pi} \in \mathit{Safety}.\ \forall \mathrm{P}.$$

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

$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[[\![\mathrm{P}]\!]] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \boldsymbol{\pi})$$

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

$$\text{if } \mathbf{A}[[\![\mathrm{P}]\!]] \rightsquigarrow \mathbf{m}$$

$$\text{then } \exists \mathrm{A},$$

In Depth Example: RSC

ESOP’19, TOPLAS’21

$\llbracket\cdot\rrbracket$ = compiler
π/$\boldsymbol{\pi}$ = set of traces
P = partial program
A/$\mathbf{A}$ = attacker
t/$\mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow$/$\boldsymbol{\rightsquigarrow}$ = trace semantics
m/$\mathbf{m}$ = prefix of a trace

$$\llbracket\cdot\rrbracket : \mathrm{RSP} \overset{\text{def}}{=} \forall \pi \approx \boldsymbol{\pi} \in \mathit{Safety}.\ \forall \mathsf{P}.$$

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

$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}\left[\llbracket \mathsf{P} \rrbracket\right] \boldsymbol{\rightsquigarrow} \mathbf{t} \Rightarrow \mathbf{t} \in \boldsymbol{\pi})$$

$$\llbracket\cdot\rrbracket : \mathrm{RSC} \overset{\text{def}}{=} \forall \mathsf{P}, \mathbf{A}, \mathbf{m}.$$

$$\text{if } \mathbf{A}\left[\llbracket \mathsf{P} \rrbracket\right] \boldsymbol{\rightsquigarrow} \mathbf{m}$$

$$\text{then } \exists \mathsf{A}, \mathsf{m} \approx \mathbf{m}.$$

In Depth Example: RSC

ESOP'19, TOPLAS'21

$[\![\cdot]\!]$ = compiler

π/$\boldsymbol{\pi}$ = set of traces

P = partial program

A/$\mathbf{A}$ = attacker

t/$\mathbf{t}$ = trace of events

$[\cdot]$ = linking

$\rightsquigarrow$/$\rightsquigarrow$ = trace semantics

m/$\mathbf{m}$ = prefix of a trace

$$[\![\cdot]\!] : \mathrm{RSP} \overset{\text{def}}{=} \forall \pi \approx \boldsymbol{\pi} \in \mathit{Safety}.\ \forall \mathsf{P}.$$

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

$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[[\![\mathsf{P}]\!]] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \boldsymbol{\pi})$$

$$[\![\cdot]\!] : \mathrm{RSC} \overset{\text{def}}{=} \forall \mathsf{P}, \mathbf{A}, \mathbf{m}.$$

$$\text{if } \mathbf{A}[[\![\mathsf{P}]\!]] \rightsquigarrow \mathbf{m}$$

$$\text{then } \exists \mathsf{A}, \mathsf{m} \approx \mathbf{m}.\ \mathsf{A}[\mathsf{P}] \rightsquigarrow \mathsf{m}$$

Secure Compilation Threat Model

- robust, active attacker ($\forall \mathbf{A}$)

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

Secure Compilation Threat Model

- robust, active attacker ($\forall \mathbf{A}$)

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 ($\forall\,\mathbf{A}$)

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

- in-language expressible attacker

- trace-based security behaviour ($\mathsf{m}/\mathbf{m}$)

Secure Compilation Threat Model

- robust, active attacker ($\forall \mathbf{A}$)

'20

- i

What can we do with these foundations?

- trace-based security behaviour ($\mathsf{m}/\mathbf{m}$)

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

POPL’23

Memory Safety (Untyped, Intra-Object)

Memory Safety (Untyped, Intra-Object)

- add colours+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

F	F	F	F	F	F	F

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

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)
```

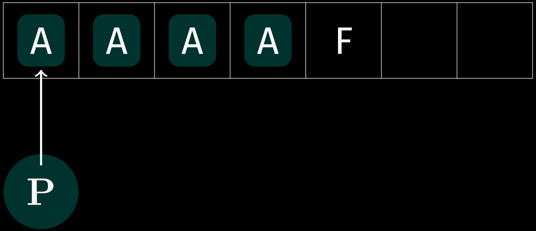
A	A	A	A	F		

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)
```

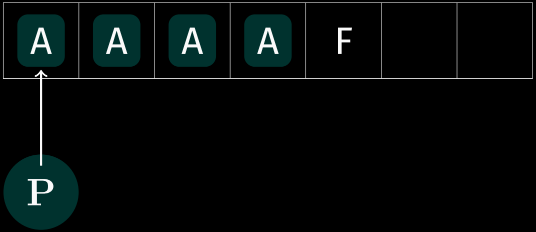


Memory Safety (Untyped, Intra-Object)

- 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)
```

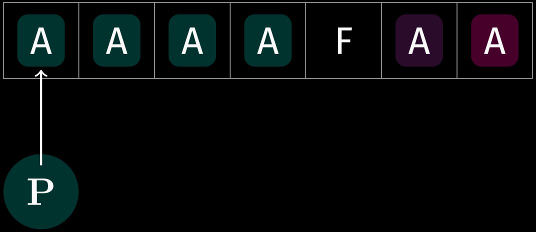


Memory Safety (Untyped, Intra-Object)

- 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)
```

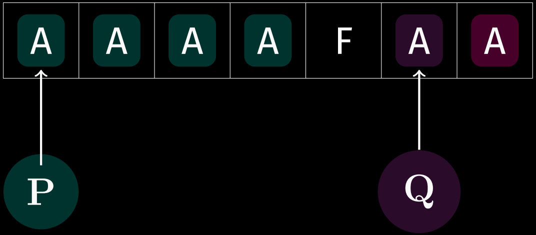


Memory Safety (Untyped, Intra-Object)

- 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)
```

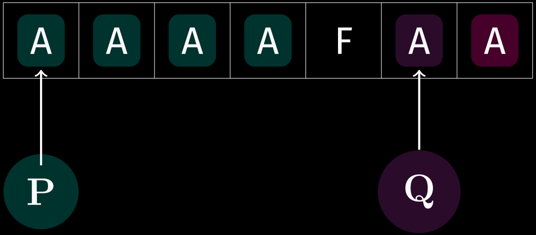


Memory Safety (Untyped, Intra-Object)

- 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)
 read(P)
```



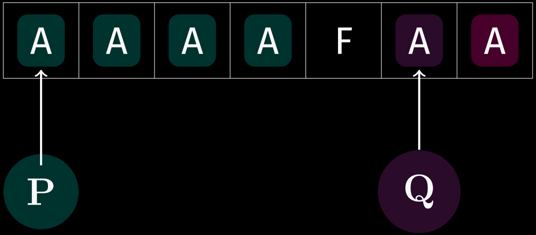
Memory Safety (Untyped, Intra-Object)

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

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

ok

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

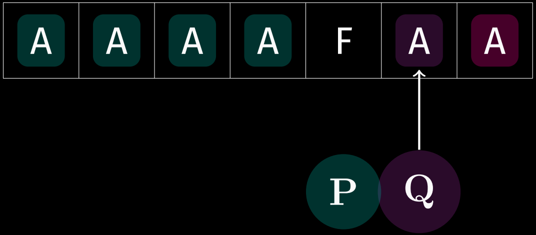


Memory Safety (Untyped, Intra-Object)

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

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

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



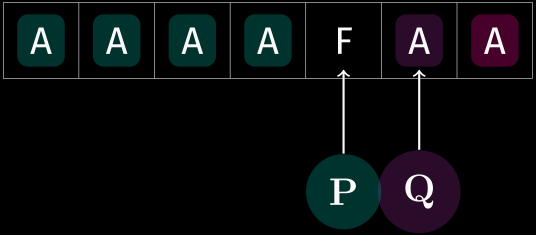
Memory Safety (Untyped, Intra-Object)

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

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

NO

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

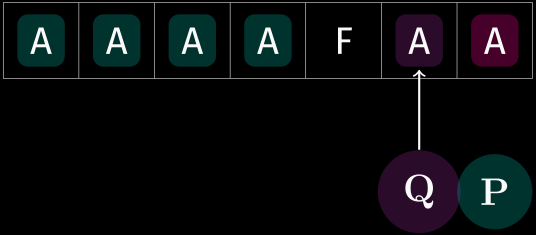


Memory Safety (Untyped, Intra-Object)

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

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

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



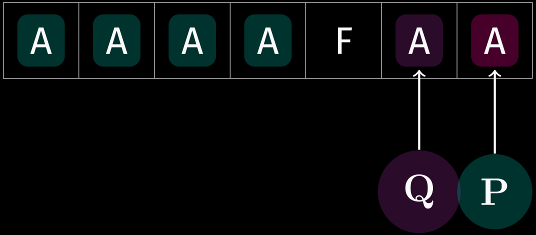
Memory Safety (Untyped, Intra-Object)

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

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

NO

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

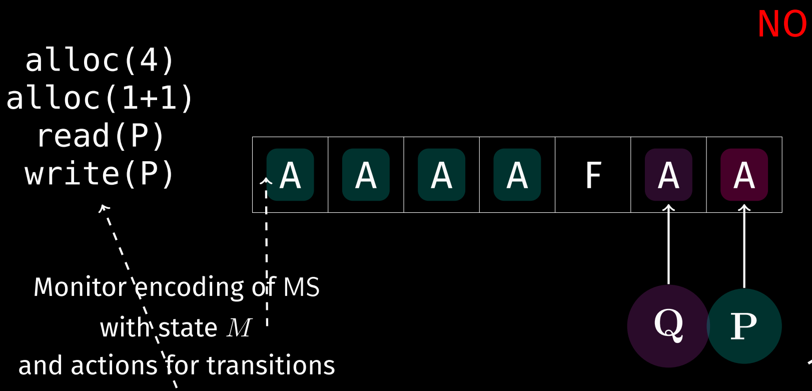


Memory Safety (Untyped, Intra-Object)

- 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)
read(P)
write(P)
```



Memory-Safe WebAssembly (`MSWAsm`)

- WAsm:
    - inter-sandboxes MS

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

  Watson *et al.* S&P'15
  - **handles**:

    $\langle \mathbf{base}, \mathbf{length}, \mathbf{offset}, \mathbf{isCorrupted}, \mathbf{id} \rangle$

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
  - handles:
    $\langle \mathbf{base}, \mathbf{length}, \mathbf{offset}, \mathbf{isCorrupted}, \mathbf{id} \rangle$
- segment instructions:
  - **segment_alloc**, **segment_free**

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
  - **handles**:

    $\langle \mathbf{base}, \mathbf{length}, \mathbf{offset}, \mathbf{isCorrupted}, \mathbf{id} \rangle$
- segment instructions:
  - **segment_alloc**, **segment_free**
  - **segment_read**, **segment_write**

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
  - handles:

    $\langle \textbf{base}, \textbf{length}, \textbf{offset}, \textbf{isCorrupted}, \textbf{id} \rangle$
- segment instructions:
  - segment_alloc, segment_free
  - segment_read, segment_write
  - handle_add, handle_slice

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

Ω = source state

α/α = trace action

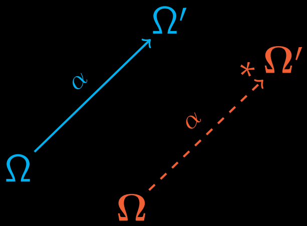
$$\begin{tikzcd}
 & \Omega' \\
\Omega \arrow[ur, "\alpha"] &
\end{tikzcd}$$

Compiler Properties

Ω = source state

$\mathbf{\Omega}$ = compiled state

α/α = trace action



Compiler Properties

Ω = source state

$\boldsymbol{\Omega}$ = compiled state

α/α = trace action

δ = partial bijection

$$\begin{tikzcd}
\Omega \arrow[r, "\alpha"] \arrow[d, dash, "=_{\delta}"'] & \Omega' \\
\boldsymbol{\Omega} \arrow[r, dashed, "\alpha", "*" near end] & \boldsymbol{\Omega}'
\end{tikzcd}$$

Compiler Properties

Ω = source state

Ω = compiled state

α/α = trace action

δ = partial bijection

$$\begin{tikzcd}
\Omega' \arrow[r, dash, dashed, "=_{\delta'}"] & \Omega' \\
\Omega \arrow[u, "\alpha"] \arrow[r, dash, "=_{\delta}"'] \arrow[ur, phantom, "\sim_{\delta'}"] & \Omega \arrow[u, dashed, "\alpha"', "*"]
\end{tikzcd}$$

Compiler Properties

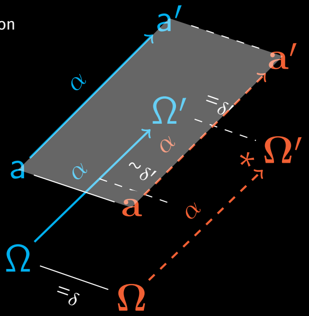
Ω = source state

$\boldsymbol{\Omega}$ = compiled state

α/α = trace action

a/**a** = allocator

δ = partial bijection



Compiler Properties

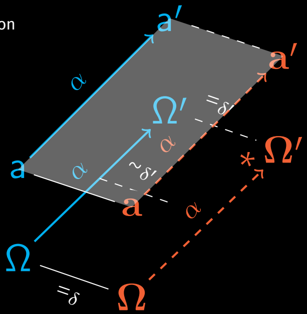
Ω = source state

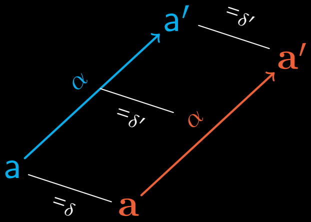
$\boldsymbol{\Omega}$ = compiled state

α/α = trace action

a/$\mathbf{a}$ = allocator

δ = partial bijection





Compiler Properties

Ω = source state

$\boldsymbol{\Omega}$ = compiled state

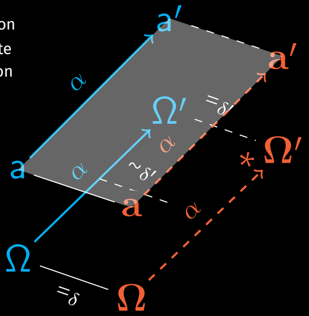
α/α = trace action

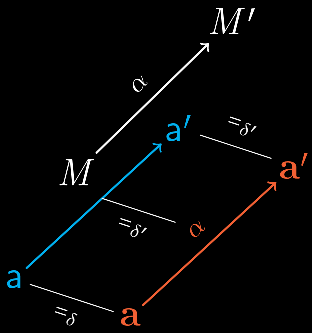
a/**a** = allocator

δ = partial bijection

M = monitor state

α = monitor action





Compiler Properties

Ω = source state

$\boldsymbol{\Omega}$ = compiled state

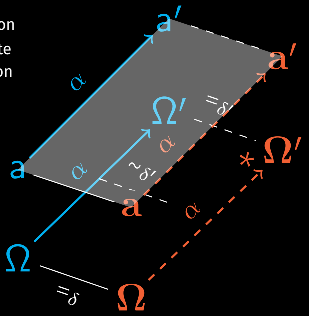
α/α = trace action

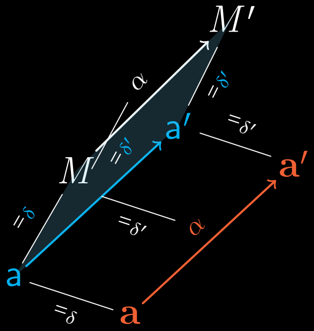
a/**a** = allocator

δ = partial bijection

M = monitor state

α = monitor action





Compiler Properties

Ω = source state

$\boldsymbol{\Omega}$ = compiled state

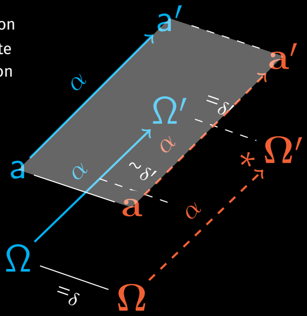
α/α = trace action

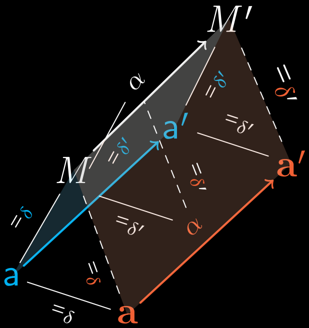
a/**a** = allocator

δ = partial bijection

M = monitor state

α = monitor action





Compiler Properties

PRO: proved MS preservation, MS enforcement

Ω = source state

$\mathbf{\Omega}$ = compiled state

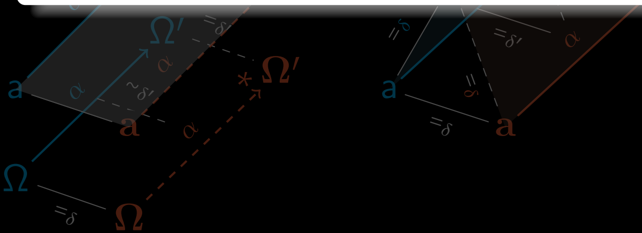
α/$\boldsymbol{\alpha}$ = trace action

a/**a** = allocator

δ = partial bijection

M = monitor state

α = monitor action



Compiler Properties

PRO: proved MS preservation, MS enforcement

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

Ω = source state

$\boldsymbol{\Omega}$ = compiled state

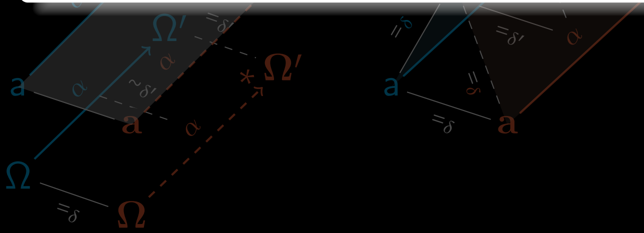
α/$\boldsymbol{\alpha}$ = trace action

a/**a** = allocator

δ = partial bijection

M = monitor state

α = monitor action



Compiler Properties

PRO: proved MS preservation, MS enforcement

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

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

Ω = source state

$\boldsymbol{\Omega}$ = compiled state

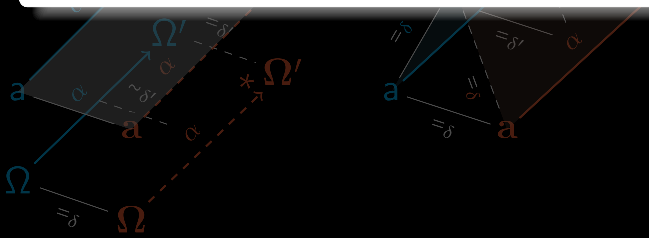
α/$\boldsymbol{\alpha}$ = trace action

a/**a** = allocator

δ = partial bijection

M = monitor state

α = monitor action



Robust Cryptographic Constant Time

(wip)

(Robust) Cryptographic Constant Time

- larger trace model than MS:

(Robust) Cryptographic Constant Time

- 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

Compiler Preserving RCT

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

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)

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

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

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)

CCS'21

Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

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

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

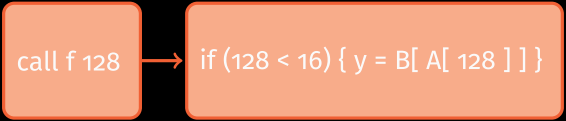
call f 128

Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

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

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

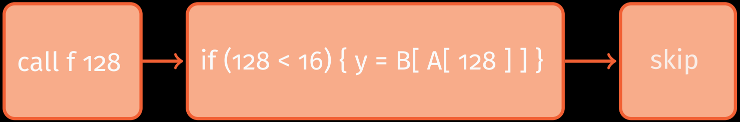


Speculative Semantics & SNI

Guarnieri *et al.* S&P’21

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

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



call f 128

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

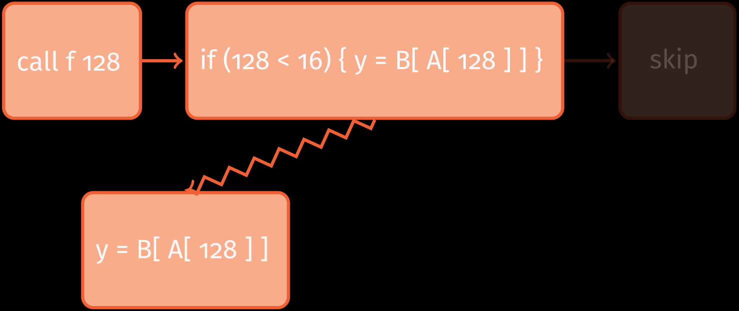
skip

Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

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

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

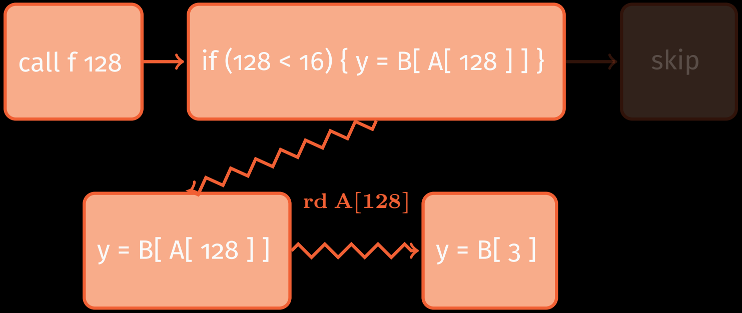


Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

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

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

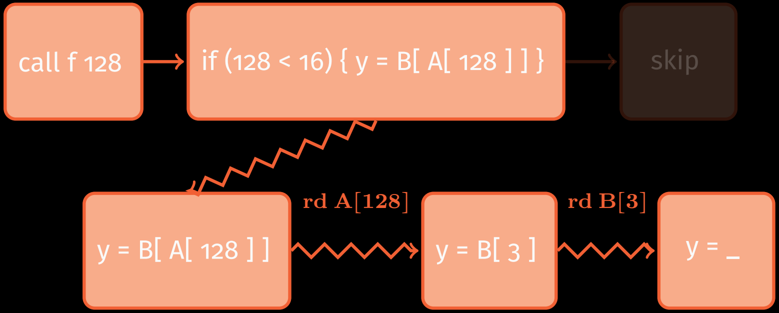


Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

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

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

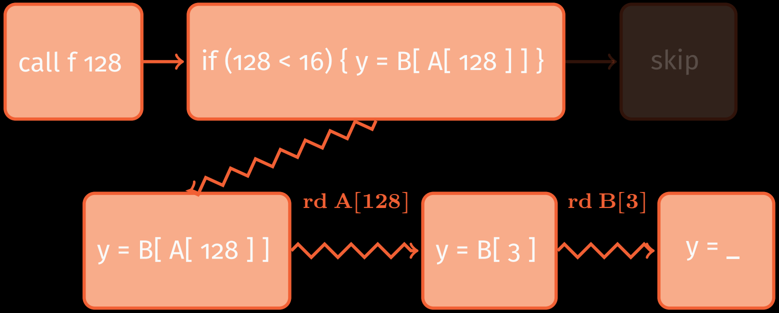


Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

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

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

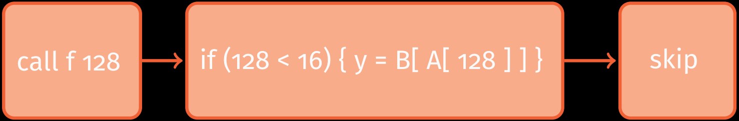


Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

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

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



trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$

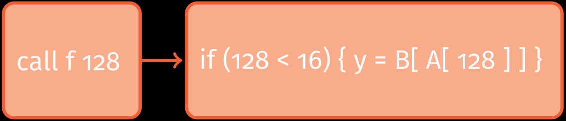
Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

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

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

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



call f 128

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

trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$

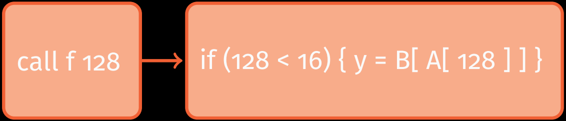
Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

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

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

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



call f 128

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

trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$

$\mathbf{rd\ A[128]}$

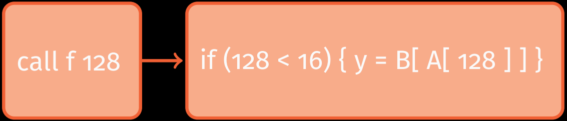
Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

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

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

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



call f 128

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

trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$

$\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[7]}$

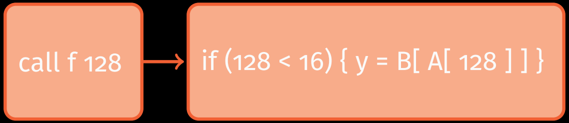
Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

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

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

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



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

trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$ different traces

trace 2: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[7]}$ ⇒ SNI violation

Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

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

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

call f

trace 1: $\mathbf{rd}\ \mathrm{A}[128]$ $\mathbf{rd}\ \mathrm{B}[3]$ different traces

trace 2: $\mathbf{rd}\ \mathrm{A}[128]$ $\mathbf{rd}\ \mathrm{B}[7]$ ⇒ SNI violation

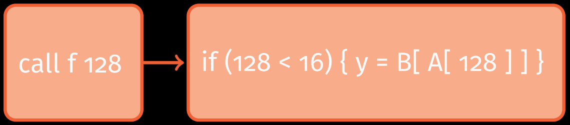
Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

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

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

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



trace 1: **rd A[128]** **rd B[3]** different traces

trace 2: **rd A[128]** **rd B[7]** ⇒ SNI violation

Problems Problems Problems ...

Problem: Proving compiler preserves SNI is hard

Problems Problems Problems …

Problem: Proving compiler preserves SNI is hard

Solution: overapproximate SNI with a novel property: speculative safety (SS)

Speculative Safety (SS): Taint Tracking

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

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

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

call f 128

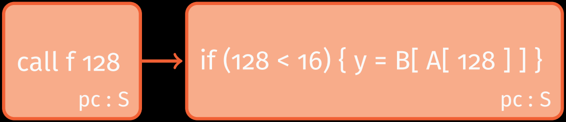
pc : S

Speculative Safety (SS): Taint Tracking

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

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

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

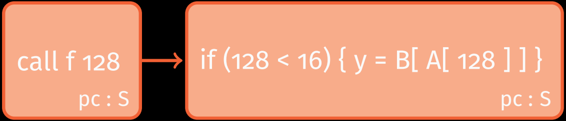


Speculative Safety (SS): Taint Tracking

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

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

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

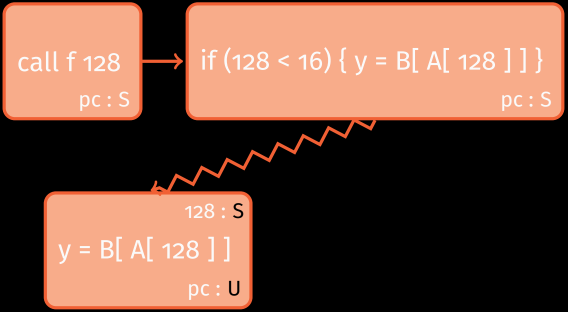


Speculative Safety (SS): Taint Tracking

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

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

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

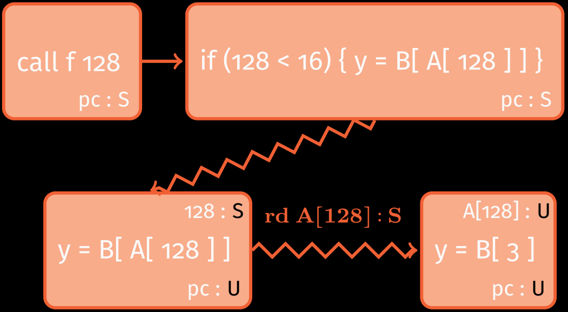


Speculative Safety (SS): Taint Tracking

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

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

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

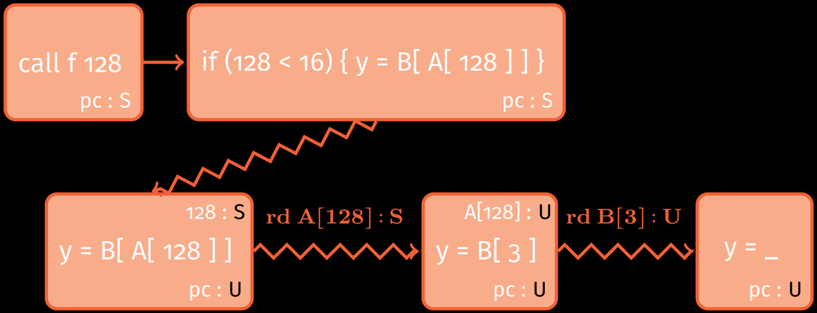


Speculative Safety (SS): Taint Tracking

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

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

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



call f 128

pc : S

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

pc : S

128 : S

y = B[A[128]]

pc : U

$\mathbf{rd\ A[128] : S}$

A[128] : U

y = B[3]

pc : U

$\mathbf{rd\ B[3] : U}$

y = _

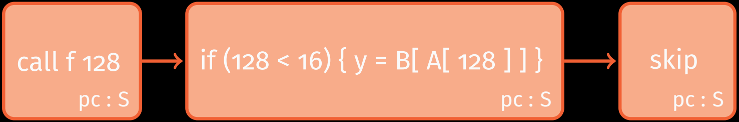
pc : U

Speculative Safety (SS): Taint Tracking

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

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

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



rd A[128] : S

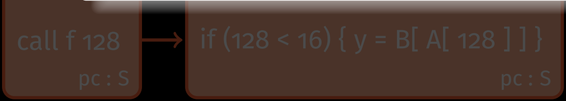
rd B[3] : U

Speculative Safety (SS): Taint Tracking

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

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

A program is SS ($\vdash \mathbf{P} : \mathbf{SS}$) if its traces do not contain $\mathbf{U}$ actions



call f 128

pc : S

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

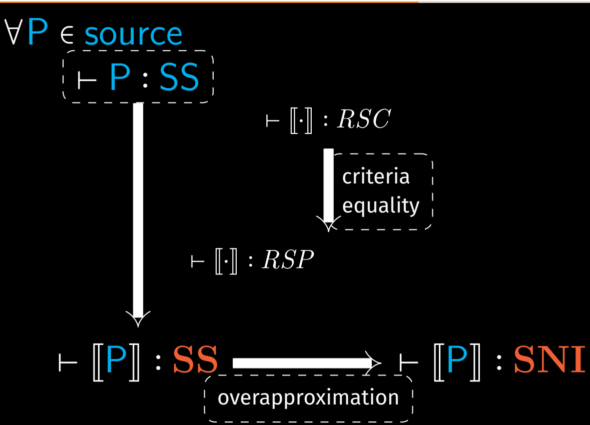
pc : S

$\mathbf{rd\ A[128] : S}$

$\mathbf{rd\ B[3] : U}$

Secure Compilation Framework for Spectre

Secure Compilation Framework for Spectre



$\forall P \in$ source

$\vdash P : SS$

$\vdash [\![\cdot]\!] : RSC$

criteria
equality

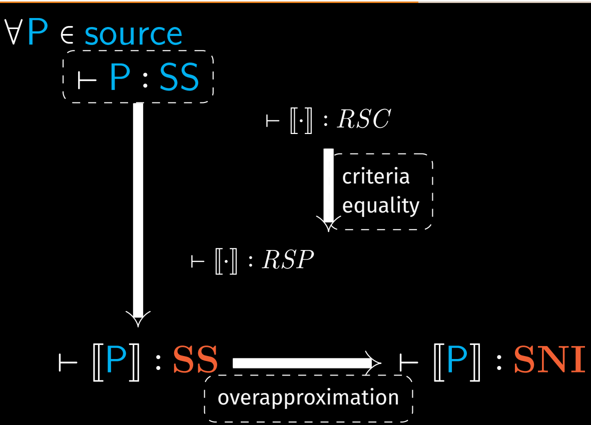
$\vdash [\![\cdot]\!] : RSP$

$\vdash [\![P]\!] : SS$

$\vdash [\![P]\!] : SNI$

overapproximation

Secure Compilation Framework for Spectre



$\forall$P $\in$ source

$\vdash$ P : SS

$\vdash [\![\cdot]\!] : RSC$

criteria equality

$\vdash [\![\cdot]\!] : RSP$

$\vdash [\![\mathrm{P}]\!] : \mathbf{SS}$

overapproximation

$\vdash [\![\mathrm{P}]\!] : \mathbf{SNI}$

- dashed premises are already discharged

Secure Compilation Framework for Spectre

$\forall P \in \text{source}$

$$\begin{tikzcd}
\vdash P : \mathrm{SS} \arrow[dd] & & \vdash \llbracket\cdot\rrbracket : RSC \arrow[d, "\text{criteria equality}"] \\
& \vdash \llbracket\cdot\rrbracket : RSP & \\
\vdash \llbracket P \rrbracket : \mathrm{SS} \arrow[rr, "\text{overapproximation}"'] & & \vdash \llbracket P \rrbracket : \mathbf{SNI}
\end{tikzcd}$$

- dashed premises are already discharged
- to show security: simply prove RSC

RSC **for** `lfence`

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

$[\![\cdot]\!] = \mathbf{void\ f(int\ x)} \mapsto \mathbf{if(x < A.size)\{lfence; y = B[A[x]]\}}$

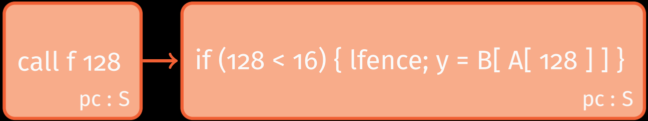
call f 128

pc : S

RSC for `lfence`

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

$[\![\cdot]\!] = \mathbf{void\ f(int\ x)} \mapsto \mathbf{if}(\mathbf{x} < \mathbf{A.size})\{\mathbf{lfence; y} = \mathbf{B}[\mathbf{A}[\mathbf{x}]]\}$



call f 128

pc : S

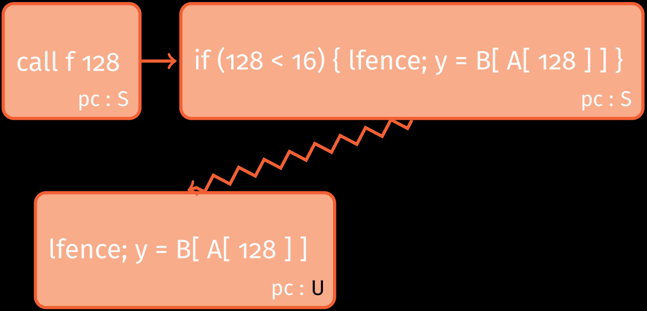
if (128 < 16) { lfence; y = B[A[128]] }

pc : S

RSC **for** `lfence`

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

$[\![\cdot]\!] = \mathbf{void\ f}(\mathbf{int\ x}) \mapsto \mathbf{if}(\mathbf{x} < \mathbf{A.size})\{\mathbf{lfence; y} = \mathbf{B}[\mathbf{A}[\mathbf{x}]]\}$

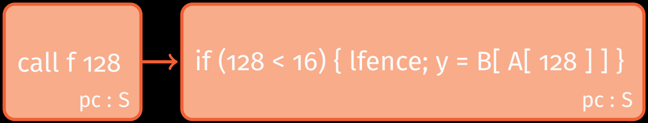


call f 128
pc : S
if (128 < 16) { lfence; y = B[A[128]] }
pc : S
lfence; y = B[A[128]]
pc : U

RSC for `lfence`

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

$[\![\cdot]\!] = \mathbf{void\ f(int\ x)} \mapsto \mathbf{if(x < A.size)\{lfence; y = B[A[x]]\}}$



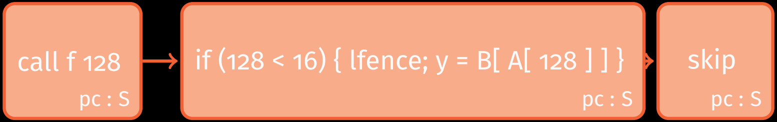
call f 128
pc : S

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

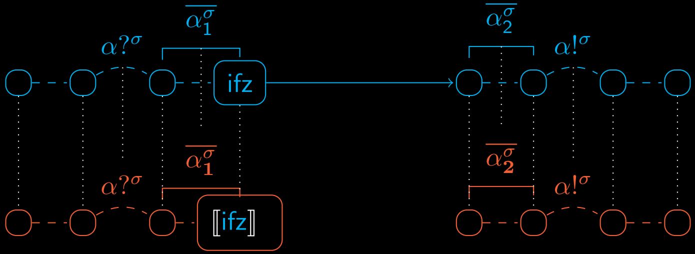
RSC for `lfence`

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

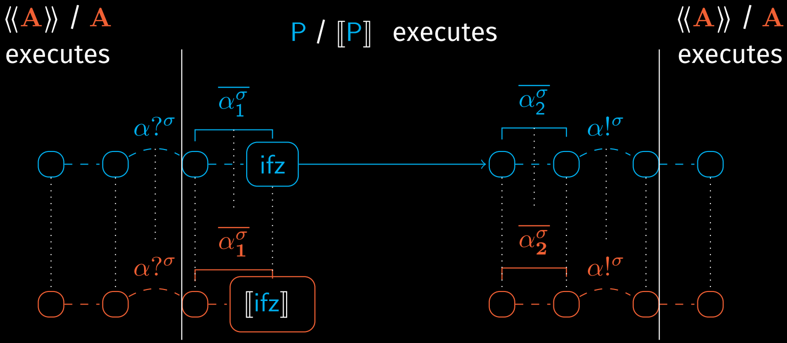
$[\![\cdot]\!] = \mathbf{void\ f(int\ x)} \mapsto \mathbf{if}(\mathbf{x} < \mathbf{A.size})\{\mathbf{lfence; y} = \mathbf{B}[\mathbf{A}[\mathbf{x}]]\}$



Proofs Insight

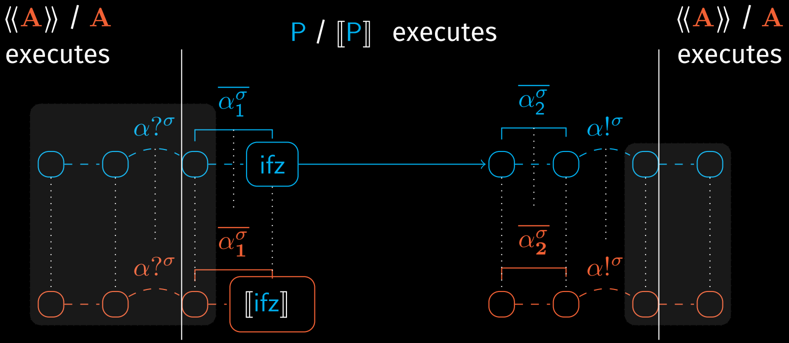

$\overline{\alpha_1^\sigma}$
$\alpha?^\sigma$
ifz
$\overline{\alpha_2^\sigma}$
$\alpha!^\sigma$
$\overline{\alpha_{\mathbf{1}}^\sigma}$
$\alpha?^\sigma$
$[\![\text{ifz}]\!]$
$\overline{\alpha_{\mathbf{2}}^\sigma}$
$\alpha!^\sigma$

Proofs Insight



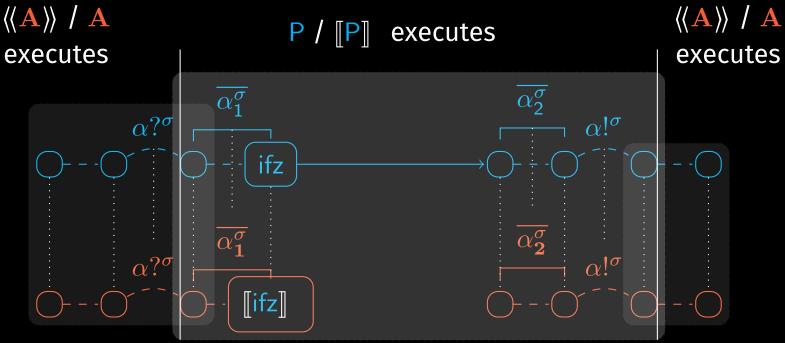
$\langle\!\langle \mathbf{A} \rangle\!\rangle$ / $\mathbf{A}$ executes
P / ⟦P⟧ executes
$\langle\!\langle \mathbf{A} \rangle\!\rangle$ / $\mathbf{A}$ executes
$\overline{\alpha_1^\sigma}$
$\overline{\alpha_2^\sigma}$
$\alpha?^\sigma$
$\alpha!^\sigma$
ifz
$\overline{\alpha_{\mathbf{1}}^\sigma}$
$\overline{\alpha_{\mathbf{2}}^\sigma}$
$\alpha?^\sigma$
$\alpha!^\sigma$
⟦ifz⟧

Proofs Insight



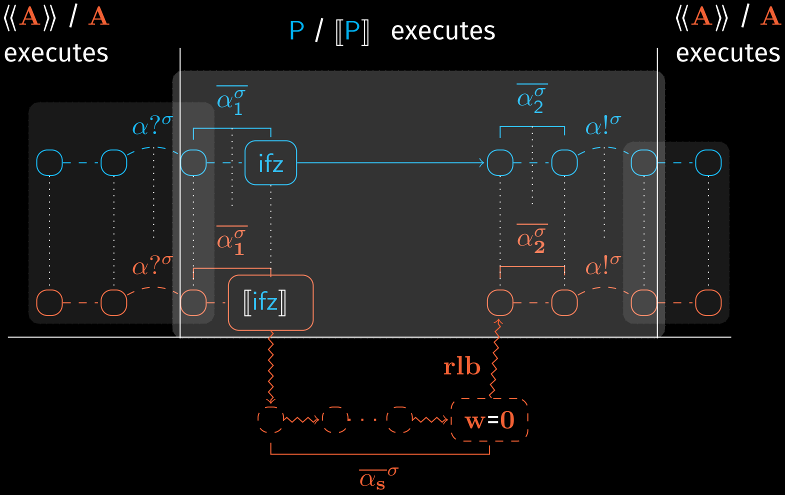
$\langle\!\langle \mathbf{A} \rangle\!\rangle$ / $\mathbf{A}$ executes
P / ⟦P⟧ executes
$\langle\!\langle \mathbf{A} \rangle\!\rangle$ / $\mathbf{A}$ executes
$\overline{\alpha_1^\sigma}$
$\overline{\alpha_2^\sigma}$
$\alpha?^\sigma$
$\alpha!^\sigma$
ifz
$\overline{\alpha_1^\sigma}$
$\overline{\alpha_2^\sigma}$
$\alpha?^\sigma$
$\alpha!^\sigma$
⟦ifz⟧

Proofs Insight



$\langle\!\langle \mathbf{A} \rangle\!\rangle$ / $\mathbf{A}$
executes
P / ⟦P⟧ executes
$\langle\!\langle \mathbf{A} \rangle\!\rangle$ / $\mathbf{A}$
executes
$\overline{\alpha_1^\sigma}$
$\overline{\alpha_2^\sigma}$
$\alpha?^\sigma$
ifz
$\alpha!^\sigma$
$\overline{\alpha_{\mathbf{1}}^\sigma}$
$\overline{\alpha_{\mathbf{2}}^\sigma}$
$\alpha?^\sigma$
⟦ifz⟧
$\alpha!^\sigma$

Proofs Insight



$\langle\!\langle \mathbf{A} \rangle\!\rangle$ / $\mathbf{A}$ executes

P / $[\![\mathrm{P}]\!]$ executes

$\langle\!\langle \mathbf{A} \rangle\!\rangle$ / $\mathbf{A}$ executes

$\overline{\alpha_1^\sigma}$

$\overline{\alpha_2^\sigma}$

$\alpha?^\sigma$

$\alpha!^\sigma$

ifz

$\overline{\alpha_{\mathbf{1}}^\sigma}$

$\overline{\alpha_{\mathbf{2}}^\sigma}$

$\alpha?^\sigma$

$\alpha!^\sigma$

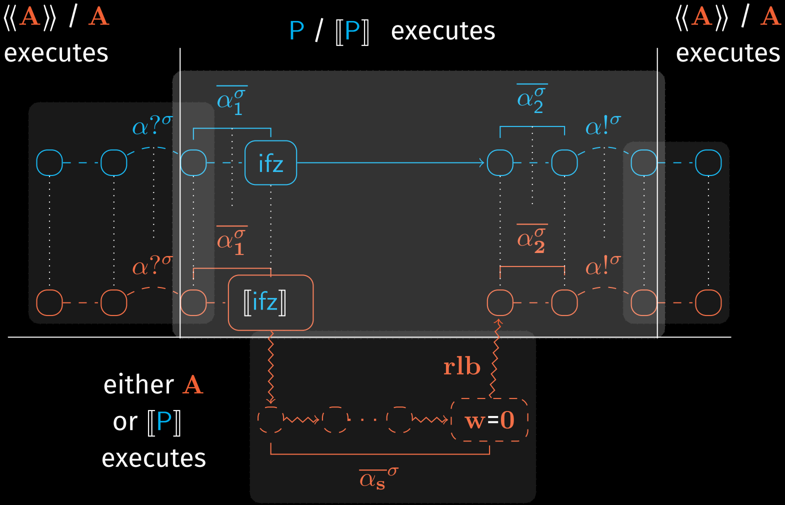
$[\![\mathrm{ifz}]\!]$

rlb

w=0

$\overline{\alpha_{\mathbf{s}}}^\sigma$

Proofs Insight



$\langle\!\langle \mathbf{A} \rangle\!\rangle$ / $\mathbf{A}$ executes
P / ⟦P⟧ executes
$\langle\!\langle \mathbf{A} \rangle\!\rangle$ / $\mathbf{A}$ executes
$\overline{\alpha_1^\sigma}$
$\overline{\alpha_2^\sigma}$
$\alpha?^\sigma$
$\alpha!^\sigma$
ifz
$\overline{\alpha_1^\sigma}$
$\overline{\alpha_2^\sigma}$
$\alpha?^\sigma$
$\alpha!^\sigma$
⟦ifz⟧
either $\mathbf{A}$ or ⟦P⟧ executes
rlb
$\mathbf{w}=\mathbf{0}$
$\overline{\alpha_\mathbf{s}}^\sigma$

What Then?

- SNIv1, SNIv2, SNIv4, SNIv5

Kocher *et al.* S&P'19

What Then?

- SNIv1, SNIv2, SNIv4, SNIv5

Kocher *et al.* S&P'19

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

What Then?

- SNIv1, SNIv2, SNIv4, SNIv5

Kocher *et al.* S&P’19

- Challenge: can the lfence compiler “mess” with SNIv2?

- Challenge: can we compose lfence(SNIv1) and retpoline(SNIv5)?

Security Architectures

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

Mechanise Cryptographic Proofs

CSF’24 + wip

Robust Hyperproperty Preservation

$$\llbracket\cdot\rrbracket \vdash \mathsf{RHP} \overset{\mathsf{def}}{=} \forall \mathsf{P}, \mathbf{A}.\ \exists \mathsf{A}.\ \forall t.$$

$$\mathbf{A}\left[\llbracket \mathsf{P} \rrbracket\right] \rightsquigarrow t \iff \mathsf{A}\left[\mathsf{P}\right] \rightsquigarrow t$$

Robust Hyperproperty Preservation

$$\begin{tikzcd}
& t & & & & t & \\
\llbracket \mathsf{P} \rrbracket & \bowtie \arrow[u, rightsquigarrow] & \mathbf{A} & \iff & \mathsf{P} & \bowtie \arrow[u, rightsquigarrow] & \mathsf{A}
\end{tikzcd}$$

$$\llbracket \cdot \rrbracket \vdash \mathsf{RHP} \overset{\mathsf{def}}{=} \forall \mathsf{P}, \mathbf{A}.\ \exists \mathsf{A}.\ \forall t.$$

$$\mathbf{A}\left[\llbracket \mathsf{P} \rrbracket\right] \rightsquigarrow t \iff \mathsf{A}\left[\mathsf{P}\right] \rightsquigarrow t$$

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: UC

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

Canetti '01, Liao *et al.* PLDI'19, Camenisch *et al.* Asiacrypt'19

Universal Composability: UC

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

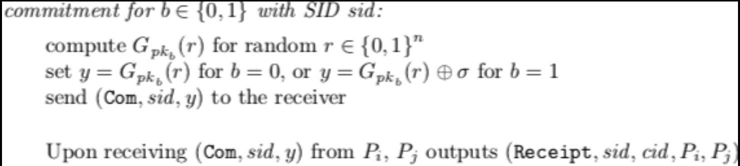
Canetti '01, Liao *et al.* PLDI'19, Camenisch *et al.* Asiacrypt'19

This talk: generic flavour, geared towards the newer theories

UC Base Notions: ITMs

Canetti and Fischlin Crypto'01

- protocols Π (using concrete crypto)



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

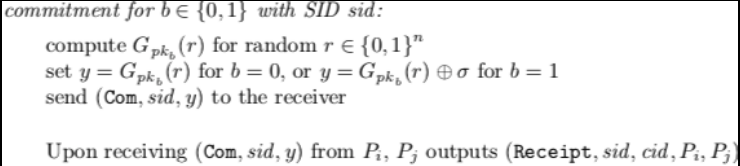
```
    compute $G_{pk_b}(r)$ for random $r \in \{0,1\}^n$
    set $y = G_{pk_b}(r)$ for $b = 0$, or $y = G_{pk_b}(r) \oplus \sigma$ for $b = 1$
    send $(\texttt{Com}, sid, y)$ to the receiver

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

UC Base Notions: ITMs

Canetti and Fischlin Crypto'01

- protocols Π (using concrete crypto)

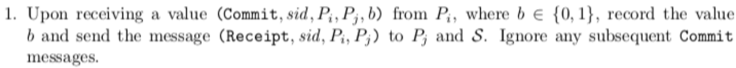


commitment for $b \in \{0,1\}$ *with SID* sid*:*

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 $(\texttt{Com}, sid, y)$ to the receiver

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

- functionalities F (using abstract notions)

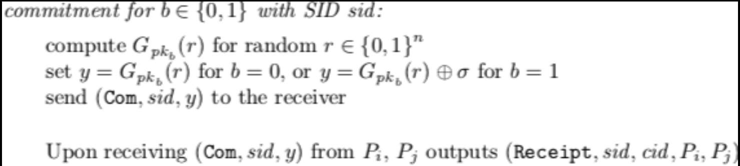


1. Upon receiving a value $(\texttt{Commit}, sid, P_i, P_j, b)$ from P_i, where $b \in \{0,1\}$, record the value b and send the message $(\texttt{Receipt}, sid, P_i, P_j)$ to P_j and $\mathcal{S}$. Ignore any subsequent Commit messages.

UC Base Notions: ITMs

Canetti and Fischlin Crypto'01

- protocols Π (using concrete crypto)

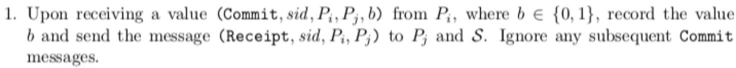


commitment for $b \in \{0,1\}$ with SID sid:

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 $(\texttt{Com}, sid, y)$ to the receiver

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

- functionalities F (using abstract notions)



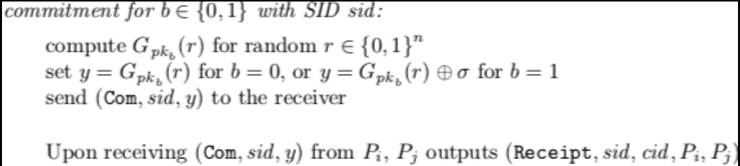
1. Upon receiving a value $(\texttt{Commit}, sid, P_i, P_j, b)$ from P_i, where $b \in \{0,1\}$, record the value b and send the message $(\texttt{Receipt}, sid, P_i, P_j)$ to P_j and $\mathcal{S}$. Ignore any subsequent Commit messages.

- attackers A & S (corrupting parties etc.)

UC Base Notions: ITMs

Canetti and Fischlin Crypto'01

- protocols Π (using concrete crypto)

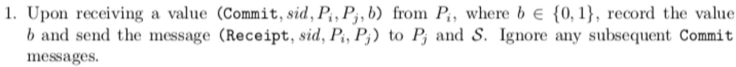


commitment for $b \in \{0,1\}$ with SID sid:

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 $(\texttt{Com}, sid, y)$ to the receiver

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

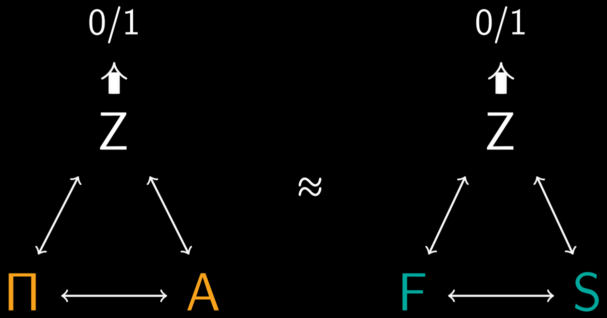
- functionalities F (using abstract notions)



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

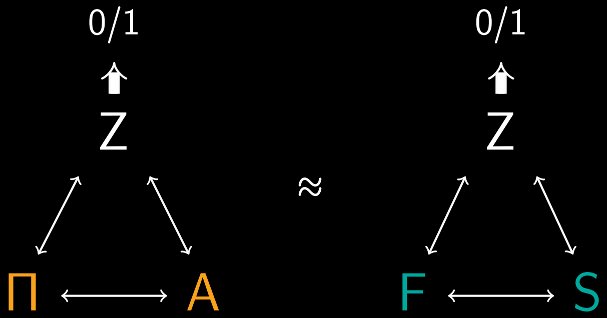
- attackers A & S (corrupting parties etc.)
- environments Z (objective witness)

UC (Semi-formally)



↔ represent communication channels

UC (Semi-formally)



↔ represent communication channels

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

A Closer Look

$$\forall \text{poly } A, \exists S, \forall Z.$$

$$\begin{tikzcd}
 & 0/1 & & & & 0/1 & \\
 & Z \arrow[u, Rightarrow] \arrow[dl, leftrightarrow] \arrow[dr, leftrightarrow] & & & & Z \arrow[u, Rightarrow] \arrow[dl, leftrightarrow] \arrow[dr, leftrightarrow] & \\
\Pi \arrow[rr, leftrightarrow] & & A & \approx & F \arrow[rr, leftrightarrow] & & S
\end{tikzcd}$$

$$\forall P, \mathbf{A}. \exists A. \forall t.$$

$$\begin{tikzcd}
 & t & & & & t & \\
[\![P]\!] & \bowtie \arrow[u, squiggly] & \mathbf{A} & \iff & P & \bowtie \arrow[u, squiggly] & A
\end{tikzcd}$$

A Closer Look

$\forall \text{poly } A, \exists S, \forall Z.$

$$\begin{tikzcd}
 & 0/1 & & & & 0/1 & \\
 & Z \arrow[u, Rightarrow] & & & & Z \arrow[u, Rightarrow] & \\
\Pi \arrow[ur, leftrightarrow] \arrow[rr, leftrightarrow] & & A \arrow[ul, leftrightarrow] & \approx & F \arrow[ur, leftrightarrow] \arrow[rr, leftrightarrow] & & S \arrow[ul, leftrightarrow]
\end{tikzcd}$$

$\forall P, \mathbf{A}. \exists A. \forall t.$

$$\begin{tikzcd}
 & t & & & & t & \\
\llbracket P \rrbracket & \bowtie \arrow[u, rightsquigarrow] & \mathbf{A} & \Longleftrightarrow & P & \bowtie \arrow[u, rightsquigarrow] & A
\end{tikzcd}$$

Isabelle’d both perfect and computational UC

Analogy

UC		*SC*	
protocol	Π	$\llbracket P \rrbracket$	compiled program
concrete attacker	A	$\mathbf{A}$	target context
ideal functionality	F	P	source program
simulator	S	A	source context
environment, output	Z, $0/1$	t, $\rightsquigarrow$	trace, semantics
communication	$\leftrightarrow$	$[\,]$	linking
probabilistic equiv.	$\approx$	$\Longleftrightarrow$	trace equality

Analogy

UC		SC	
protocol	Π	$[\![P]\!]$	compiled program
concrete attacker	A	$\mathbf{A}$	target context
ideal functionality	F	P	source program
simulator	S	A	source context
environment, output	Z, 0/1	t, $\rightsquigarrow$	trace, semantics
communication	$\leftrightarrow$	$[\,]$	linking
probabilistic equiv.	$\approx$	$\iff$	trace equality
human translation $\Pi \to F$		$[\![\cdot]\!]$: $P \to \mathbf{P}$ compiler	
general composition result		$\ldots$	

Analogy Results

- transfer UC results from ITMs to any S/$\mathbf{T}$

Analogy Results

- transfer UC results from ITMs to any S/$\mathbf{T}$
- mechanise UC results as RHP results

Analogy Results

- transfer UC results from ITMs to any S/$\mathbf{T}$
- mechanise UC results as RHP results known in computer-aided crypto Haagh *et al.* CSF'18

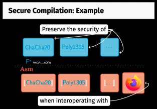
Analogy Results

- transfer UC results from ITMs to any S/$\mathbf{T}$
- mechanise UC results as RHP results known in computer-aided crypto Haagh *et al.* CSF'18
- Mechanised UC for 1-Bit Commitment in `Deepsec` submission
- Mechanised UC for 1/2 Wireguard in `Cryptoverif` CSF'24

Conclusion

Conclusion

- secure compilation threat model



Secure Compilation: Example

Preserve the security of

ChaCha20

Poly1305

· · ·

F*

HACL*: ... CCS'17

Asm

[ChaCha20]

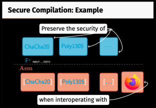
[Poly1305]

[. . .]

when interoperating with

Conclusion

- secure compilation threat model

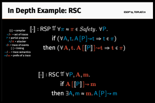


Secure Compilation: Example

Preserve the security of

when interoperating with

- formal foundations: RSC, RHP



In Depth Example: RSC

ESOP'19, TOPLAS'21

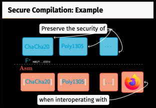
⟦·⟧ = compiler
π / π = set of traces
P = partial program
A / A = attacker
t / t = trace of events
[·] = linking
⇝ / ⇝ = trace semantics
m / m = prefix of a trace

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

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

Conclusion

- secure compilation threat model

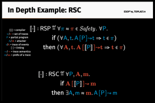


Secure Compilation: Example

Preserve the security of

when interoperating with

- formal foundations: RSC, RHP



In Depth Example: RSC

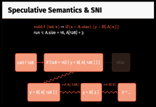
ESOP'19, TOPLAS'21

[·] = compiler
π/π = set of traces
P = partial program
A/A = attacker
t/t = trace of events
[·] = linking
⇝/⇝ = trace semantics
m/m = prefix of a trace

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

$$[\![\cdot]\!] : \mathrm{RSC} \overset{\mathrm{def}}{=} \forall \mathrm{P}, \mathrm{A}, \mathrm{m}.$$
$$\text{if } \mathrm{A}[[\![\mathrm{P}]\!]] \rightsquigarrow \mathrm{m}$$
$$\text{then } \exists \mathrm{A}, \mathrm{m} \approx \mathrm{m}.\ \mathrm{A}[\mathrm{P}] \rightsquigarrow \mathrm{m}$$

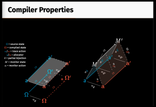
- robust compilation use-cases (MS, CT, SNI)



Speculative Semantics & SNI

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

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

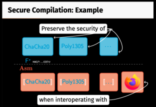


Compiler Properties

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

Conclusion

- secure compilation threat model

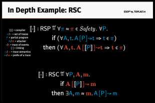


Secure Compilation: Example

Preserve the security of

when interoperating with

- formal foundations: RSC, RHP



In Depth Example: RSC

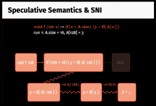
ESOP'19, TOPLAS'21

⟦·⟧ = compiler
π / π = set of traces
P = partial program
A / A = attacker
t / t = trace of events
[·] = linking
⇝ / ⇝ = trace semantics
m / m = prefix of a trace

$$
\begin{aligned}
&[\![\cdot]\!] : \mathrm{RSP} \overset{\text{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathrm{P}.\\
&\quad \text{if } (\forall \mathrm{A}, \mathrm{t}.\ \mathrm{A}[\mathrm{P}] \rightsquigarrow \mathrm{t} \Rightarrow \mathrm{t} \in \pi)\\
&\quad \text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[[\![\mathrm{P}]\!]] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \pi)
\end{aligned}
$$

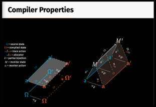
$$
\begin{aligned}
&[\![\cdot]\!] : \mathrm{RSC} \overset{\text{def}}{=} \forall \mathrm{P}, \mathbf{A}, \mathbf{m}.\\
&\quad \text{if } \mathbf{A}[[\![\mathrm{P}]\!]] \rightsquigarrow \mathbf{m}\\
&\quad \text{then } \exists \mathrm{A}, \mathrm{m} \approx \mathbf{m}.\ \mathrm{A}[\mathrm{P}] \rightsquigarrow \mathrm{m}
\end{aligned}
$$

- robust compilation use-cases (MS, CT, SNI)



Speculative Semantics & SNI

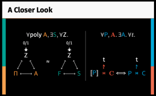
void f (int x) ↦ if(x < A.size) {y = B[A[x]]}
run 1: A.size = 16, A[128] = 3



Compiler Properties

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

- connection with UC



A Closer Look

$\forall \text{poly } A, \exists S, \forall Z.$

$\forall \mathrm{P}, \mathbf{A}.\ \exists \mathrm{A}.\ \forall t.$

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?

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?
- S
- S
- SC for different languages?
- Other UC-like connections?
- More mechanised UC protocols?

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

Questions?