Foundations and Applications of Secure Compilation (includes wip)

Marco Patrignani[1,2]

20th March 2020

1 Stanford University 2 CISPA Helmholtz Center for Information Security

Talk Outline

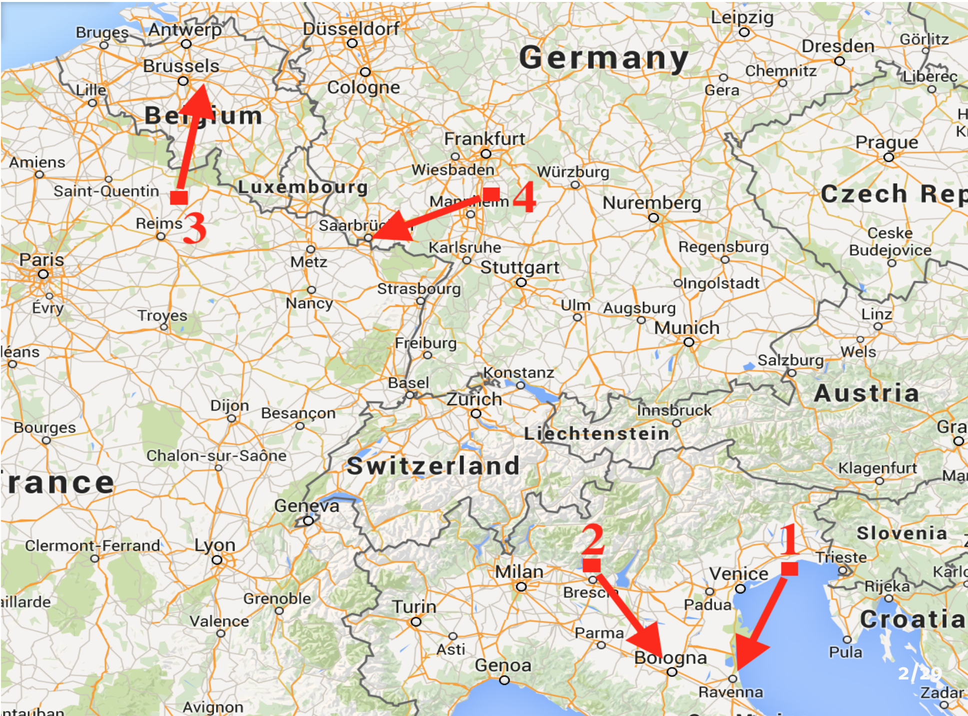
Who Am I ?

Foundations of Secure Compilation

Exorcising Spectres with Secure Compilers WIP

Future Outlook

Who Am I ?



Bruges
Antwerp
Düsseldorf
Leipzig
Germany
Dresden
Görlitz
Brussels
Cologne
Chemnitz
Gera
Liberec
Lille
Belgium
Frankfurt
Prague
Amiens
Wiesbaden
Würzburg
Saint-Quentin
Luxembourg
Czech Rep
Mannheim
4
Nuremberg
Reims
3
Saarbrücken
Karlsruhe
Regensburg
Ceske Budejovice
Paris
Metz
Stuttgart
Évry
Strasbourg
Ingolstadt
Nancy
Ulm
Augsburg
Linz
Troyes
Munich
Orléans
Freiburg
Wels
Salzburg
Konstanz
Basel
Zurich
Austria
Dijon
Besançon
Innsbruck
Bourges
Liechtenstein
Graz
Chalon-sur-Saône
France
Switzerland
Klagenfurt
Maribor
Geneva
2
1
Slovenia
Clermont-Ferrand
Lyon
Milan
Trieste
Venice
Karlovac
Brescia
Rijeka
Grenoble
Padua
Turin
Croatia
Valence
Parma
Asti
Bologna
Pula
Genoa
Ravenna
Zadar
Avignon
Montauban

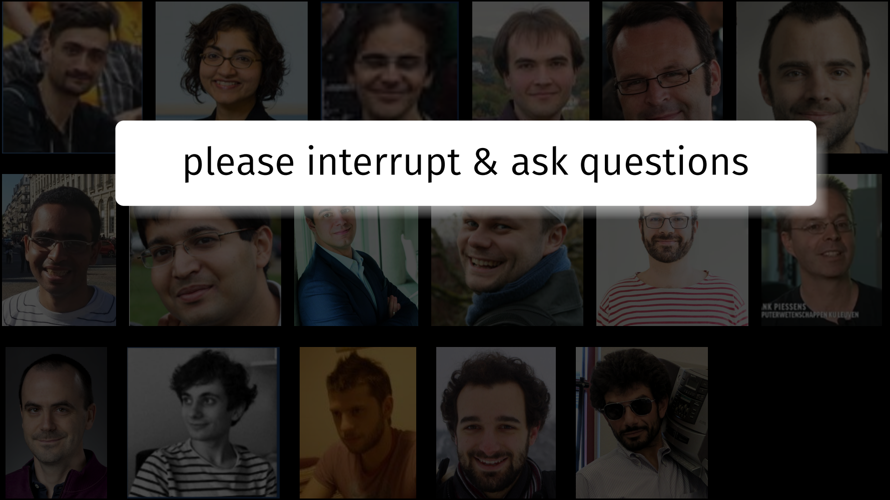
KU LEUVEN
MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS
CISPA-Stanford Center FOR CYBERSECURITY
ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA
1
2
3
4
Germany
Belgium
Luxembourg
France
Czech Rep
Slovenia
Croatia
Brussels
Saarbrücken
Bologna
Ravenna

Special Thanks to:

(wrt the contents of this talk)

Special Thanks to:

(wrt the contents of this talk)



please interrupt & ask questions

Foundations of Secure Compilation

Programming Languages: Pros and Problems

Good PLs (, , , , …) provide:

- helpful abstractions to write secure code

Programming Languages: Pros and Problems

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

- helpful abstractions to write secure code

but

- when compiled ($[\![\cdot]\!]$) and linked with adversarial target code

Programming Languages: Pros and Problems

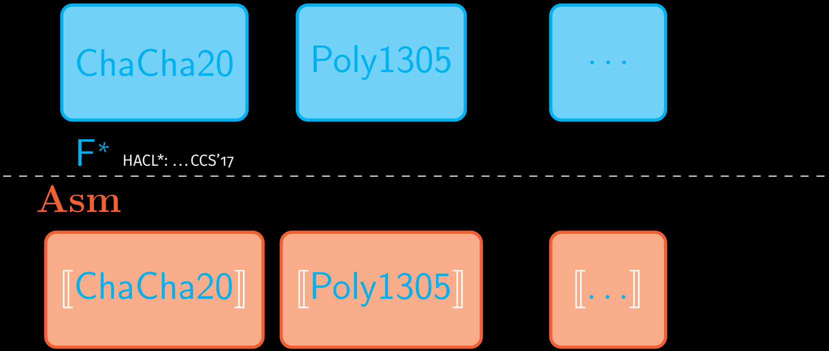
Good PLs (, TypeScript, , , …) provide:

- helpful abstractions to write secure code

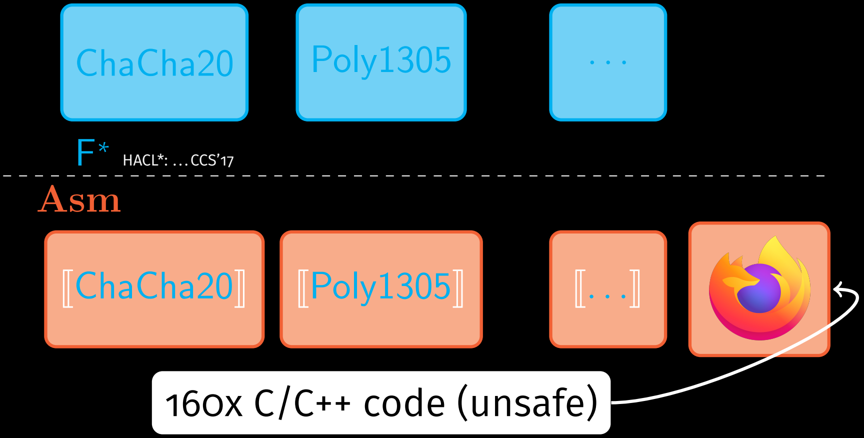
but

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

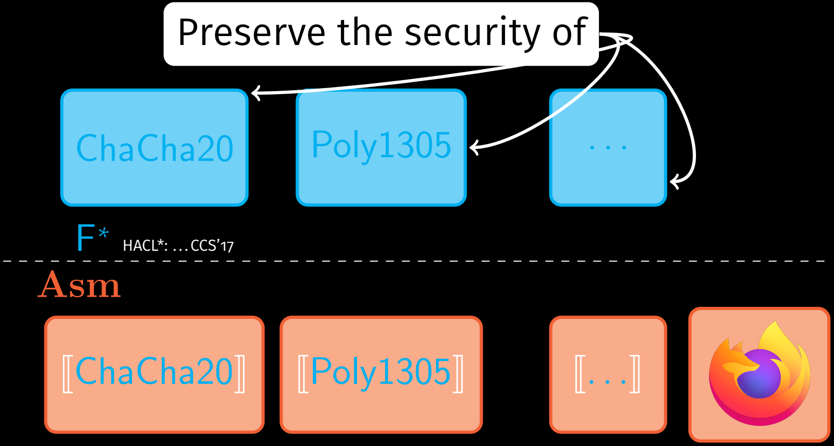
Secure Compilation: Example



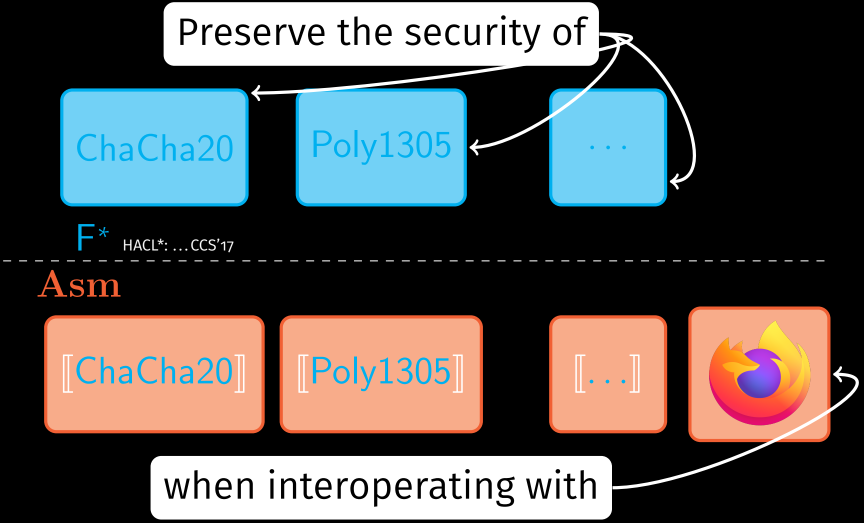
Secure Compilation: Example



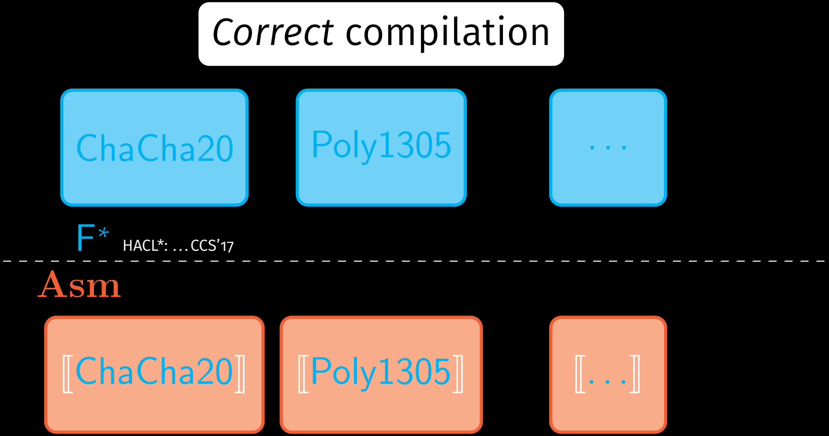
Secure Compilation: Example



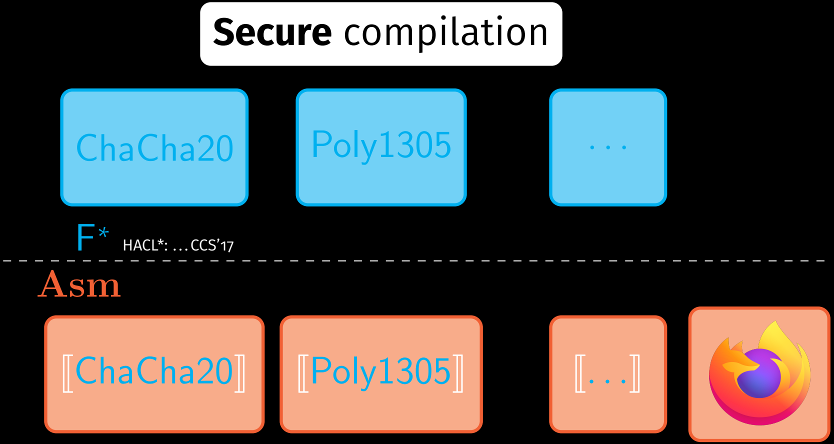
Secure Compilation: Example



Secure Compilation: Example

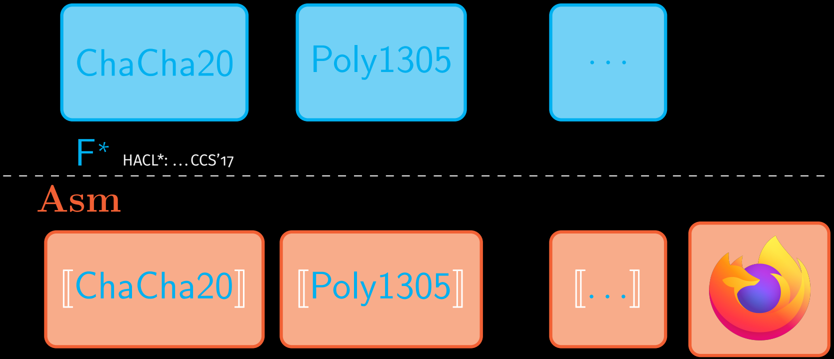


Secure Compilation: Example



Secure Compilation: Example

Enable source-level security reasoning



Quest for Foundations

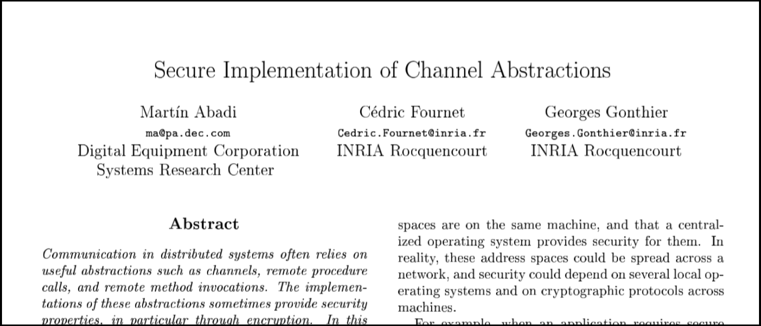
What does it mean for a compiler to be secure?

Quest for Foundations

What does it mean for a compiler to be secure?

Known for type systems, CC but not for SC

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

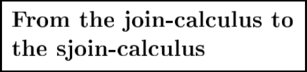
Georges Gonthier
Georges.Gonthier@inria.fr
INRIA Rocquencourt

Abstract

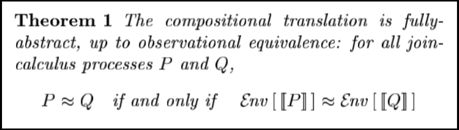
Communication in distributed systems often relies on useful abstractions such as channels, remote procedure calls, and remote method invocations. The implementations of these abstractions sometimes provide security properties, in particular through encryption. In this

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



From the join-calculus to the sjoin-calculus



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

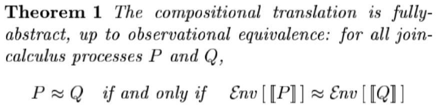
$$P \approx Q \quad \textit{if and only if} \quad \mathcal{E}nv\,[\,[\![P]\!]\,] \approx \mathcal{E}nv\,[\,[\![Q]\!]\,]$$

Once Upon a Time in Process Algebra

they needed a definition that their implementation of secure channels via cryptography was secure

Once Upon a Time in Process Algebra

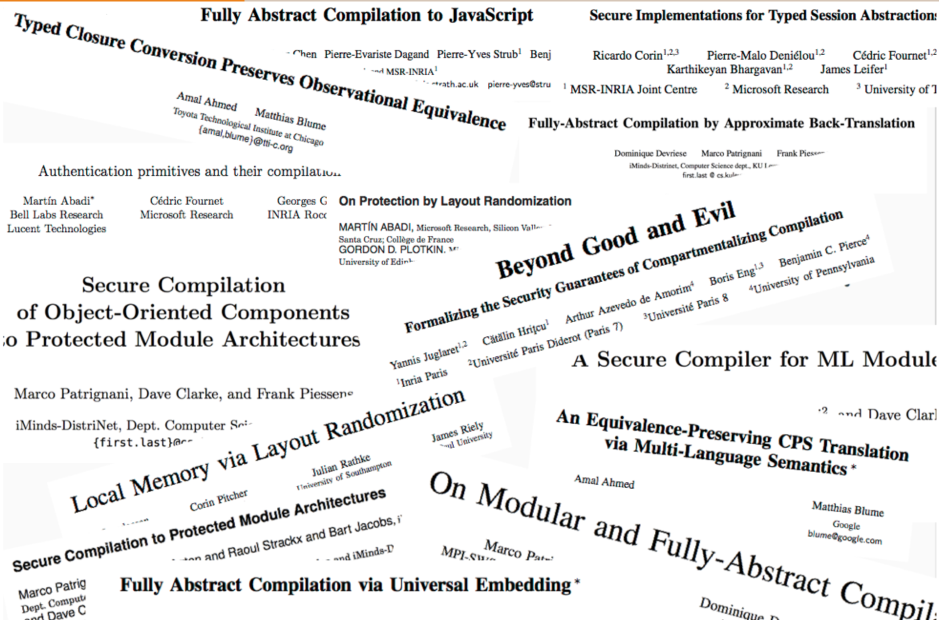
Fully Abstract Compilation (FAC)



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

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

Fully Abstract Compilation Influence



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
and MSR-INRIA[1]
strath.ac.uk pierre-yves@stru

Secure Implementations for Typed Session Abstraction

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 Pies
iMinds-Distrinet, Computer Science dept., KU L
first.last @ cs.kul

Authentication primitives and their compilation

Martín Abadi* Cédric Fournet Georges G
Bell Labs Research Microsoft Research INRIA Roc
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 Hrițcu[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 Sc
{first.last}@c

A Secure Compiler for ML Module

and Dave Clar

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

Amal Ahmed Matthias Blume
Google
blume@google.com

Local Memory via Layout Randomization

Corin Pitcher Julian Rathke James Riely
University of Southampton University

Secure Compilation to Protected Module Architectures

Marco Patrig
Dept. Comput
and Dave C

and Raoul Strackx and Bart Jacobs,
and iMinds-D

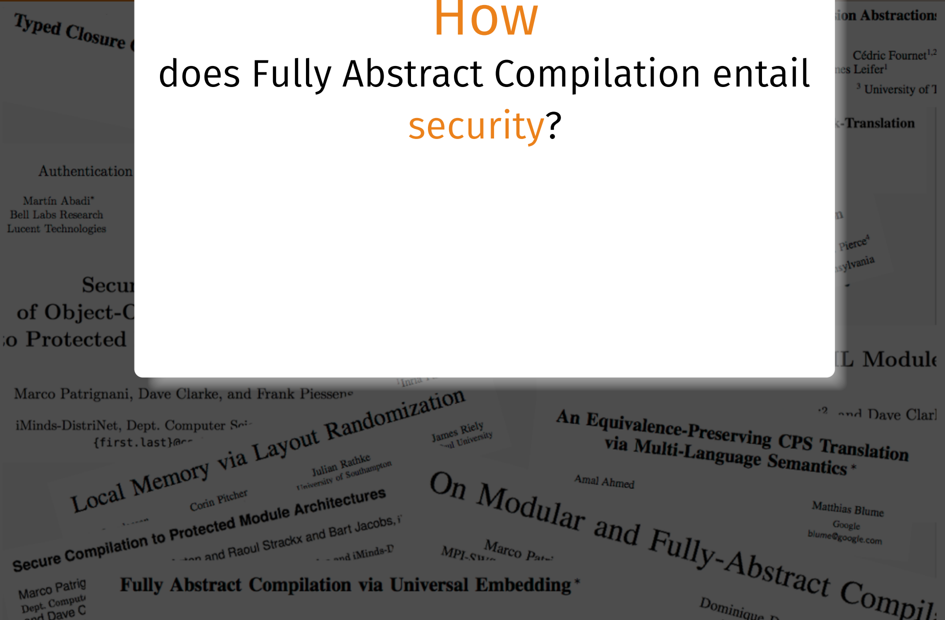
On Modular and Fully-Abstract Compil

Marco Patr
MPI-SWS

Dominique D

Fully Abstract Compilation via Universal Embedding *

Fully Abstract Compilation Influence



Typed Closure

Authentication

Martín Abadi*
Bell Labs Research
Lucent Technologies

Secur
of Object-C
o Protected

Marco Patrignani, Dave Clarke, and Frank Piessens

iMinds-DistriNet, Dept. Computer Sc

Local Memory via Layout Randomization

Corin Pitcher

Julian Rathke
University of Southampton

James Riely

Secure Compilation to Protected Module Architectures

and Raoul Strackx and Bart Jacobs

Fully Abstract Compilation via Universal Embedding*

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

Amal Ahmed

Matthias Blume
Google
blume@google.com

On Modular and Fully-Abstract Compil

Cédric Fournet

University of T

-Translation

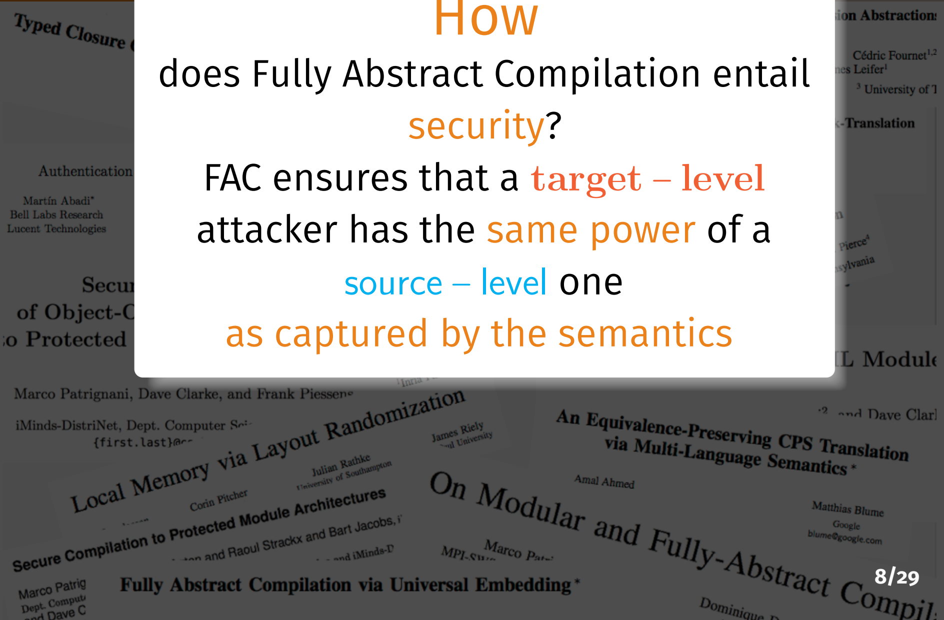
L Module

and Dave Clar

How

does Fully Abstract Compilation entail security?

Fully Abstract Compilation Influence



Typed Closure

Authentication

Martín Abadi*
Bell Labs Research
Lucent Technologies

Secu
of Object-C
o Protected

Marco Patrignani, Dave Clarke, and Frank Piessens

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

Local Memory via Layout Randomization

Corin Pitcher

Julian Rathke
University of Southampton

James Riely
DePaul University

Secure Compilation to Protected Module Architectures

and Raoul Strackx and Bart Jacobs,

and iMinds-D

Marco Patrig
Dept. Comput
and Dave C

Fully Abstract Compilation via Universal Embedding*

ion Abstractions

Cédric Fournet[1,2]
es Leifer[1]
[3] University of T

-Translation

Pierce[4]
sylvania

L Module

and Dave Clar

An Equivalence-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

Dominique D

How

does Fully Abstract Compilation entail

security?

FAC ensures that a $\mathbf{target-level}$

attacker has the same power of a

source – level one

as captured by the semantics

Fully Abstract Compilation: Definition

$$\mathsf{P}_1 \quad \simeq_{ctx} \quad \mathsf{P}_2$$

$$\Updownarrow$$

$$[\![\mathsf{P}_1]\!] \ \simeq_{ctx} \ [\![\mathsf{P}_2]\!]$$

Fully Abstract Compilation: Definition

$$\mathsf{P}_1 \quad \simeq_{ctx} \quad \mathsf{P}_2$$

$$\Uparrow$$

$$[\![\mathsf{P}_1]\!] \; \simeq_{ctx} \; [\![\mathsf{P}_2]\!]$$

Fully Abstract Compilation: Definition

$$\mathsf{P}_1 \quad \simeq_{ctx} \quad \mathsf{P}_2$$

$$\Downarrow$$

$$[\![\mathsf{P}_1]\!] \simeq_{ctx} [\![\mathsf{P}_2]\!]$$

Fully Abstract Compilation: Definition

$$\mathsf{P}_1 \quad \simeq_{ctx} \quad \mathsf{P}_2$$

$$\Downarrow$$

$$\forall \mathbf{A}.\, \mathbf{A}\left[\llbracket \mathsf{P}_1 \rrbracket\right]\Downarrow \iff \mathbf{A}\left[\llbracket \mathsf{P}_2 \rrbracket\right]\Downarrow$$

Are there Alternatives to FAC?

- FAC is not precise about security

Are there Alternatives to FAC?

- FAC is not precise about security
- this affects efficiency and proof complexity

Are there Alternatives to FAC?

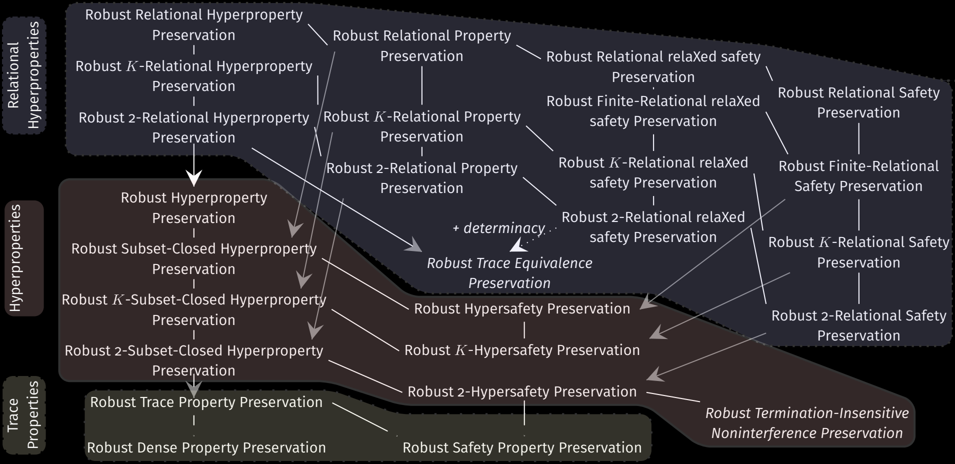
- FAC is not precise about security
- this affects efficiency and proof complexity
- in certain cases we want easier/more efficient alternatives

Are there Alternatives to FAC?

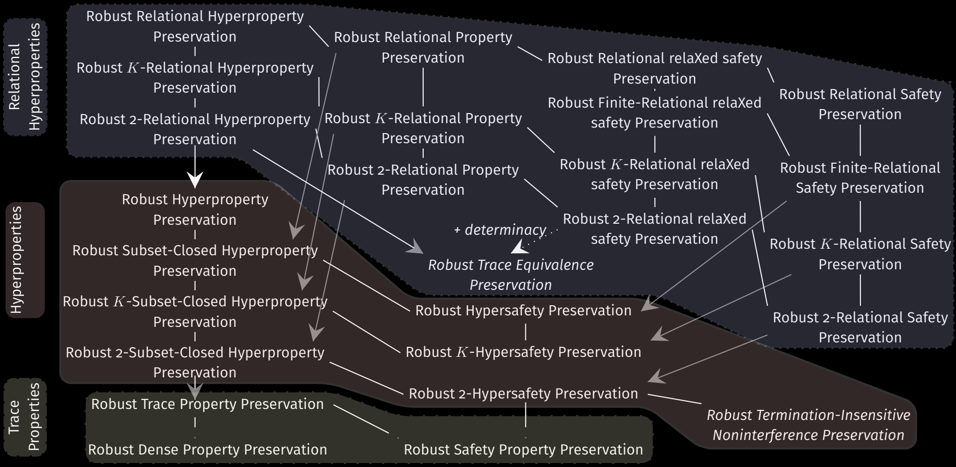
- FAC is not precise about security
- this affects efficiency and proof complexity
- in certain cases we want easier/more efficient alternatives

preserve classes of security (hyper)properties

Robust Compilation Criteria "Journey Beyond Full Abstraction …" CSF'19

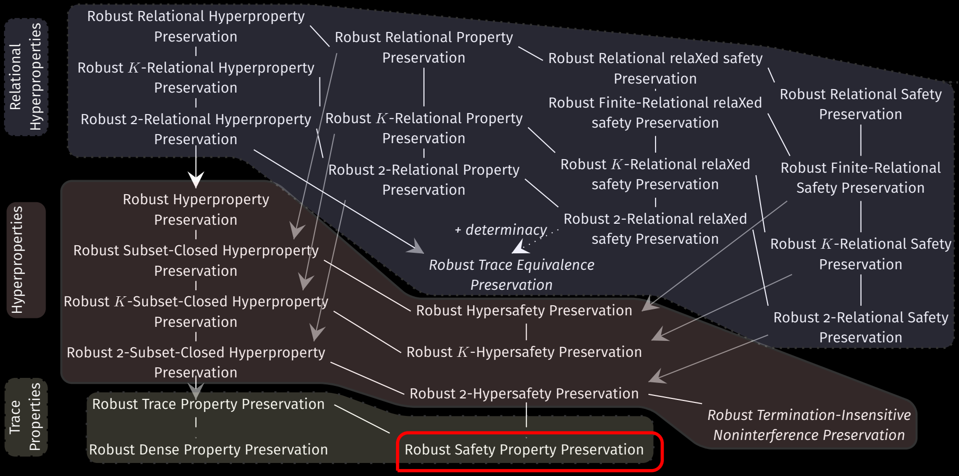

Relational Hyperproperties
Hyperproperties
Trace Properties
Robust Relational Hyperproperty Preservation
Robust K-Relational Hyperproperty Preservation
Robust 2-Relational Hyperproperty Preservation
Robust Relational Property Preservation
Robust K-Relational Property Preservation
Robust 2-Relational 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 Criteria “Journey Beyond Full Abstraction …” CSF’19


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

Tradeoffs for code efficiency, security guarantees, proof complexity

Robust Compilation Criteria "Journey Beyond Full Abstraction ..." CSF'19


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

Tradeoffs for code efficiency, security guarantees, proof complexity

Robust Criteria: Intuition

Each point has two equivalent criteria:

- $\textbf{Property – ful}:$
  - + 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:

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

Robust Criteria: Intuition

Each point has two equivalent criteria:

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

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
  - = akin to some crypto statements (UC)

In Depth Example: RSC "Robustly-Safe Compilation ..." ESOP'19

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

In Depth Example: RSC “Robustly-Safe Compilation …” ESOP’19

$[\![\cdot]\!]$ = compiler
π / π = set of traces

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

In Depth Example: RSC “Robustly-Safe Compilation …” ESOP’19

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program

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

In Depth Example: RSC “Robustly-Safe Compilation …” ESOP’19

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

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

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

In Depth Example: RSC "Robustly-Safe Compilation ..." ESOP'19

$⟦\cdot⟧$ = 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

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

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

In Depth Example: RSC “Robustly-Safe Compilation ...” ESOP’19

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

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

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

In Depth Example: RSC “Robustly-Safe Compilation ...” ESOP’19

$[\![\cdot]\!]$ = 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 “Robustly-Safe Compilation …” ESOP’19

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

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

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

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

In Depth Example: RSC “Robustly-Safe Compilation …” ESOP’19

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

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

In Depth Example: RSC “Robustly-Safe Compilation …” ESOP’19

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

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

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

In Depth Example: RSC “Robustly-Safe Compilation …” ESOP’19

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

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

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

In Depth Example: RSC "Robustly-Safe Compilation …" ESOP'19

$[\![\cdot]\!]$ = 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
m / $\mathbf{m}$ = prefix of a trace

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

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

In Depth Example: RSC "Robustly-Safe Compilation ..." ESOP'19

$[\![\cdot]\!]$ = compiler
π / π = 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

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

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

In Depth Example: RSC "Robustly-Safe Compilation …" ESOP'19

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

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

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

In Depth Example: RSC "Robustly-Safe Compilation …" ESOP'19

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

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

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

Understanding RSC

RSP/RSC:

- adaptable to reason about complex features: concurrency, undefined behaviour

Understanding RSC

RSP/RSC:

- adaptable to reason about complex features: concurrency, undefined behaviour

RSP:

- provable if source is robustly-safe

Understanding RSC

RSP/RSC:

- adaptable to reason about complex features: concurrency, undefined behaviour

RSP:

- provable if source is robustly-safe

RSC:

- easiest backtranslation proof

RSC - FAC

Both:

RSC - FAC

Both:

- robust ($\forall \mathbf{A}$)

RSC - FAC

Both:

- robust ($\forall \mathbf{A}$)
- rely on program semantics ($\rightsquigarrow$ builds on $\Downarrow$)

RSC - FAC

Both:

- robust ($\forall \mathbf{A}$)
- rely on program semantics ($\rightsquigarrow$ builds on $\Downarrow$)

FAC:

RSC - FAC

Both:

- robust ($\forall \mathbf{A}$)
- rely on program semantics ($\rightsquigarrow$ builds on $\Downarrow$)

FAC:

- yields a language result

RSC - FAC

Both:

- robust ($\forall \mathbf{A}$)
- rely on program semantics ($\rightsquigarrow$ builds on $\Downarrow$)

FAC:

- yields a language result

RSC/RSP:

- extends the semantics ($\rightsquigarrow$) to focus on security

Is There More?

Is There More?

Some still unknown foundations include:

Is There More?

Some still unknown foundations include:

- optimisation

Is There More?

Some still unknown foundations include:

- optimisation
- composition (multipass & linking)

Is There More?

Some still unknown foundations include:

- optimisation
- composition (multipass & linking)
- arbitrary side-channels

Is There More?

Some still unknown foundations include:

- optimisation
- composition (multipass & linking)
- arbitrary side-channels

Is There More?

Some still unknown foundations include:

- optimisation
- composition (multipass & linking)
- arbitrary side-channels

Exorcising Spectres with Secure Compilers

WIP

Speculative execution + branch prediction

Size of array **A**

```
if (x < A_size)
    y = B[A[x]]
```

Speculative execution + branch prediction

Size of array **A**

```
if (x < A_size)
  y = B[A[x]]
```

Speculative execution + branch prediction

Size of array **A**

```
if (x < A_size)
    y = B[A[x]]
```

HELP

Branch predictor

Speculative execution + branch prediction

Prediction based on ***branch history*** & ***program structure***

Size of array **A**

```
if (x < A_size)
    y = B[A[x]]
```

Branch predictor

Speculative execution + branch prediction

Prediction based on ***branch history*** & ***program structure***

Size of array **A**

```
if (x < A_size)
    y = B[A[x]]
```

Branch predictor

Speculative execution + branch prediction

Size of array **A**

Prediction based on ***branch history*** & ***program structure***

```
if (x < A_size)
    y = B[A[x]]
```

Branch predictor

Wrong predicton? ***Rollback changes***!

✓ Architectural (ISA) state

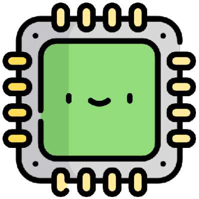
✗ Microarchitectural state

Spectre V1

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

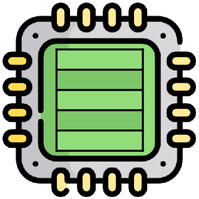
Spectre V1

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```



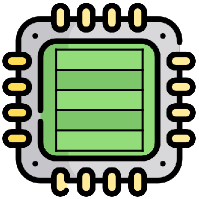
Spectre V1

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

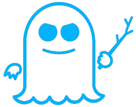


Spectre V1

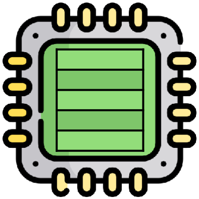
```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```



Spectre V1

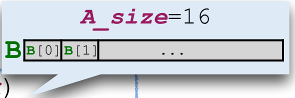


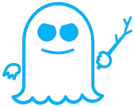
```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

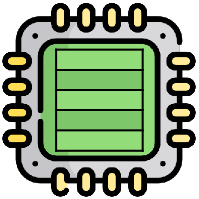


Spectre V1

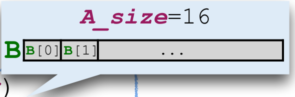
```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```



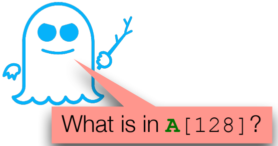


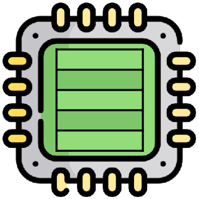


Spectre V1



```
void f(int x)
  if (x < A_size)
      y = B[A[x]]
```



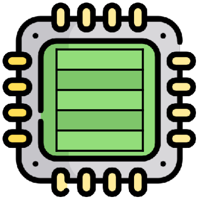


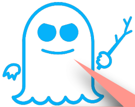
Spectre V1

A_size=16

B | B[0] | B[1] | ...

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```





What is in A[128]?

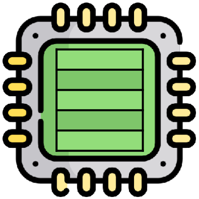
1) Training

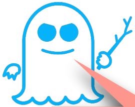
Spectre V1

A_size=16

B | B[0] | B[1] | ...

```
void f(int x)
   if (x < A_size)
      y = B[A[x]]
```





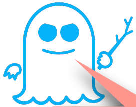
What is in A[128]?

1) Training

Spectre V1

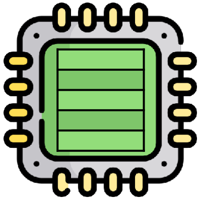
```
void f(int x)
  if (x < A_size)
      y = B[A[x]]
```

```
A_size=16
B B[0] B[1] ...
```



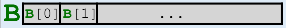
What is in A[128]?

1) Training `f(0);`

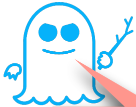


Spectre V1

```
A_size=16
```

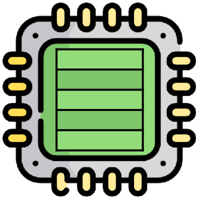


```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```



What is in A[128]?

1) Training `f(0);f(1);`

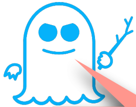


Spectre V1

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

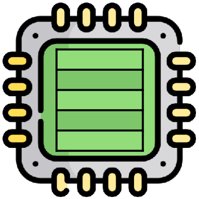
A_size=16

B B[0] B[1] ...



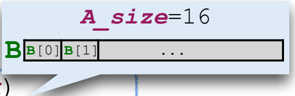
What is in A[128]?

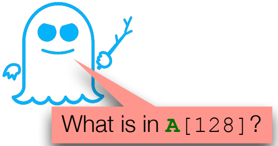
1) Training `f(0);f(1);f(2); …`

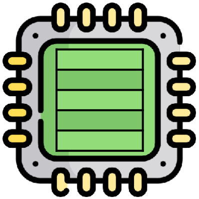


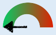
Spectre V1

```
void f(int x)
   if (x < A_size)
      y = B[A[x]]
```







1) Training  `f(0);f(1);f(2); …`

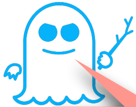
2) Prepare cache

Spectre V1

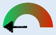
```
void f(int x)
   if (x < A_size)
      y = B[A[x]]
```

`A_size=16`

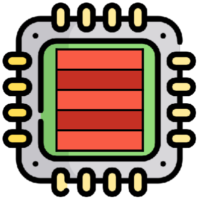
`B` | `B[0]` | `B[1]` | `...`



What is in `A[128]`?

1) Training  `f(0);f(1);f(2); …`

2) Prepare cache

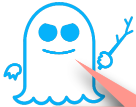


Spectre V1

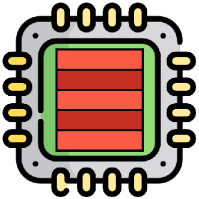
A_size=16

B B[0] B[1] ...

```
void f(int x)
   if (x < A_size)
      y = B[A[x]]
```



What is in A[128]?



1) Training f(0);f(1);f(2); …

2) Prepare cache

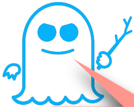
3) Run with x = 128

Spectre V1

A_size=16

B B[0] B[1] ...

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

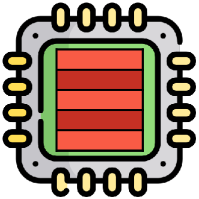


What is in A[128]?

1) Training f(0);f(1);f(2); …

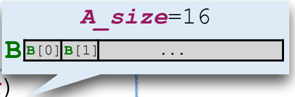
2) Prepare cache

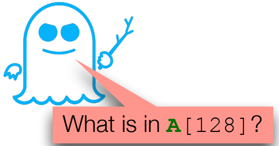
3) Run with x = 128

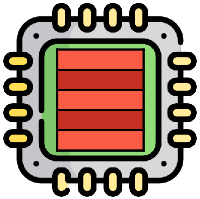


Spectre V1

```
void f(int x)
  if (x < A_size)
      y = B[A[x]]
```







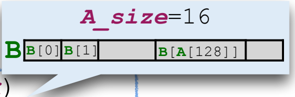
1) Training `f(0);f(1);f(2); …`

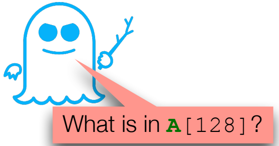
2) Prepare cache

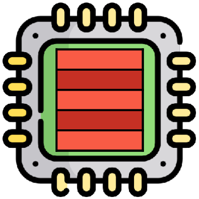
3) Run with `x = 128`

Spectre V1

```
void f(int x)
  if (x < A_size)
      y = B[A[x]]
```







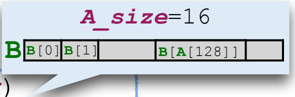
1) Training `f(0);f(1);f(2); …`

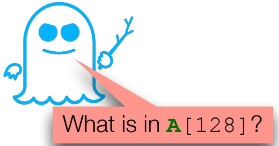
2) Prepare cache

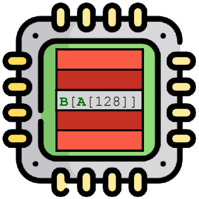
3) Run with `x = 128`

Spectre V1

```
void f(int x)
  if (x < A_size)
      y = B[A[x]]
```





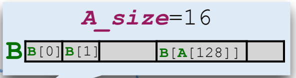


1) Training `f(0);f(1);f(2); …`

2) Prepare cache

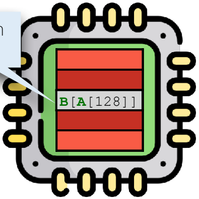
3) Run with `x = 128`

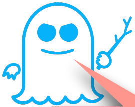
Spectre V1



```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

Depends on A[128]





What is in A[128]?

1) Training f(0);f(1);f(2); …

2) Prepare cache

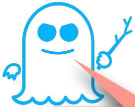
3) Run with x = 128

Spectre V1

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

A_size=16

B B[0] B[1] B[A[128]]



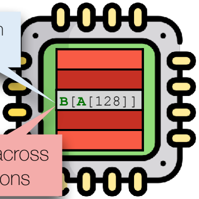
What is in A[128]?

1) Training f(0);f(1);f(2); …

2) Prepare cache

3) Run with x = 128

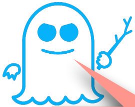
Depends on A[128]



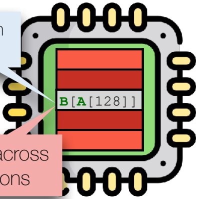
Persistent across speculations

Spectre V1

```
void f(int x)
  if (x < A_size)
      y = B[A[x]]
```



What is in A[128]?



1) Training `f(0);f(1);f(2); …`

2) Prepare cache

3) Run with *x* = 128

4) Extract from cache

Compiler-level countermeasures

Compiler-level countermeasures

For ***Spectre V1***

Injecting speculation barriers

```
if (x < A_size)
    y = B[A[x]]
```

⇓

```
if (x < A_size)
  lfence
  y = B[A[x]]
```

- In x86, ***LFENCE*** act as ***speculation barrier***
- Compiler injects LFENCE after each branch instruction
  - Microsoft Visual C++
  - Intel ICC
- Effectively ***stop speculative execution***!

Speculative load-hardening (SLH)

```
if (x < A_size)
    y = B[A[x]]
```

⬇

```
if (x < A_size)
    y = B[mask(A[x])]
```

- Injects ***data dependencies*** and ***masking operations***
    - Combines ***conditional moves*** and ***binary operations***
- Stops ***speculative leaks***
- Does not block speculative execution!
- Implemented in Clang

Goal

1. formalise `lfence` & `SLH` compilers

Goal

1. formalise `lfence` & `SLH` compilers
2. $\mathbb{T}$ must capture speculative execution ($\rightsquigarrow$)

Goal

1. formalise `lfence` & `SLH` compilers
2. $\mathbf{T}$ must capture speculative execution ($\rightsquigarrow$)
3. need a safety property capturing vulnerability to Spectre v1: SS

Goal

1. formalise `lfence` & `SLH` compilers
2. $\mathbf{T}$ must capture speculative execution ($\rightsquigarrow$)
3. need a safety property capturing vulnerability to Spectre v1: SS
4. adapt RSC to preserve SS: RSSC

Goal

1. formalise `lfence` & `SLH` compilers
2. $\mathbf{T}$ must capture speculative execution ($\rightsquigarrow$)
3. need a safety property capturing vulnerability to Spectre v1: SS
4. adapt RSC to preserve SS: RSSC
5. prove the compilers attain RSSC

Goal Up Next

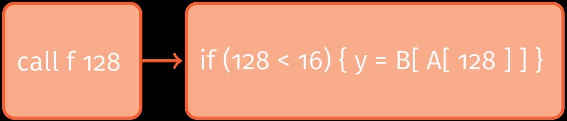
2. $\mathbf{T}$ must capture speculative execution ($\rightsquigarrow$)
3. need a safety property capturing vulnerability to Spectre v1: SS
4. adapt RSC to preserve SS: RSSC

Speculative Semantics 101 “Spectector …” S&P’20

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

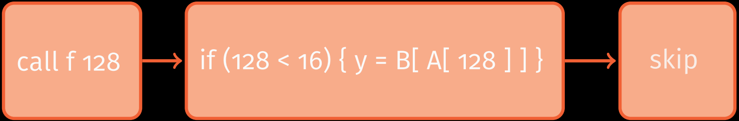
Speculative Semantics 101 "Spectector …" S&P'20

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



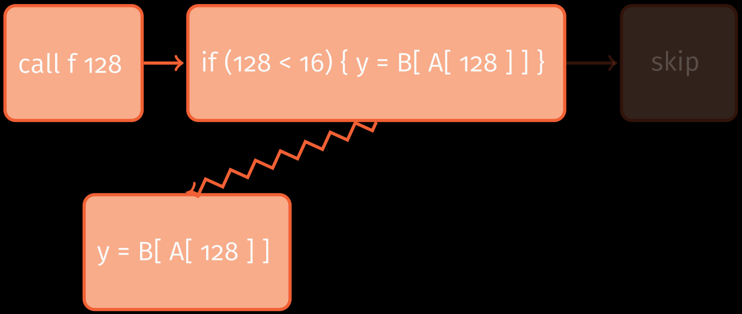
Speculative Semantics 101 "Spectector ..." S&P'20

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



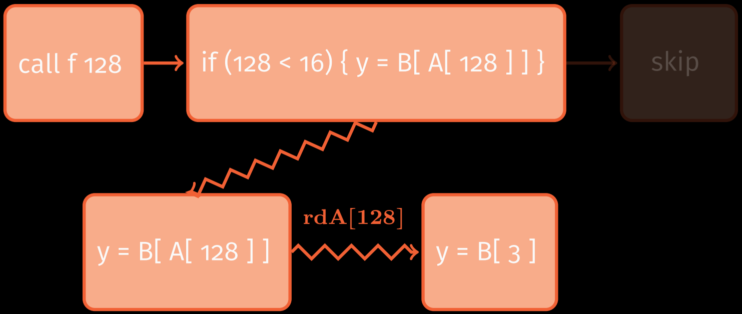
Speculative Semantics 101 "Spectector …" S&P'20

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



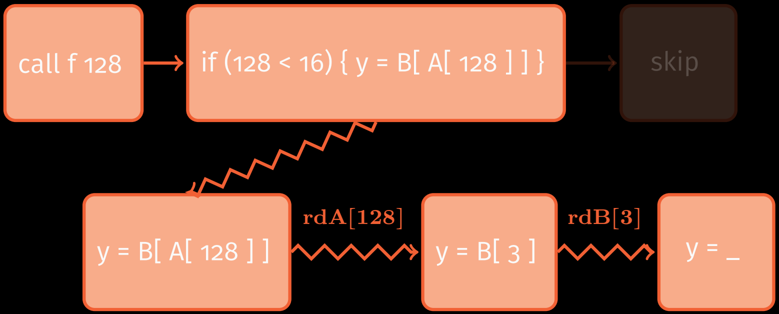
Speculative Semantics 101 "Spectector …" S&P'20

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



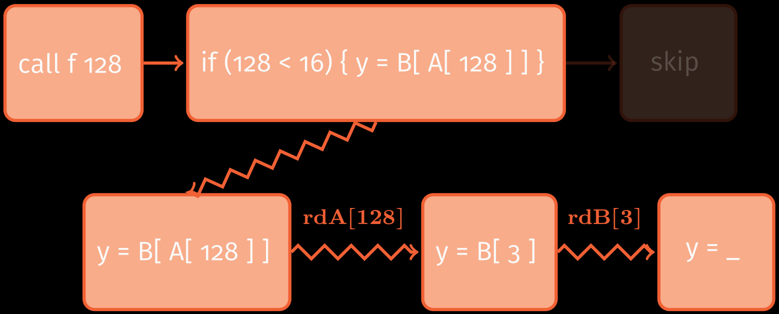
Speculative Semantics 101 “Spectector …” S&P’20

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



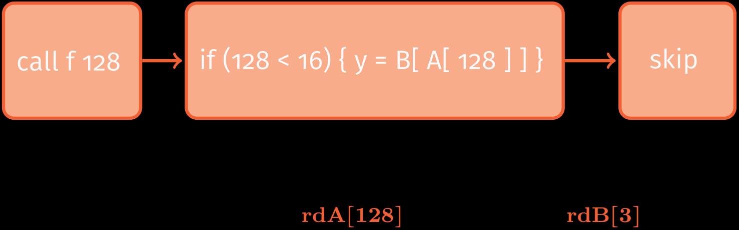
Speculative Semantics 101 “Spectector …” S&P’20

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



Speculative Semantics 101 "Spectector ..." S&P'20

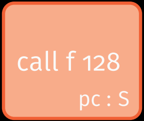
void f (int x) ↦ if (x < A.size) { y = B[A[x]] }   // A.size=16, A[128]=3



Speculative Safety (SS): Taint Tracking

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

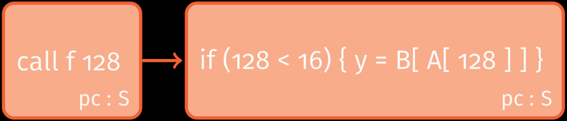
integrity lattice: $S \subset U$ $S \sqcap U = S$ U does not flow to S



Speculative Safety (SS): Taint Tracking

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

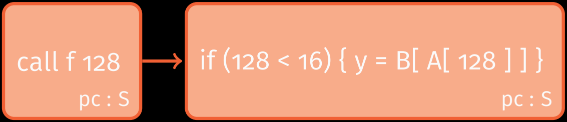
integrity lattice: $S \subset U$ $S \sqcap U = S$ U does not flow to S



Speculative Safety (SS): Taint Tracking

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

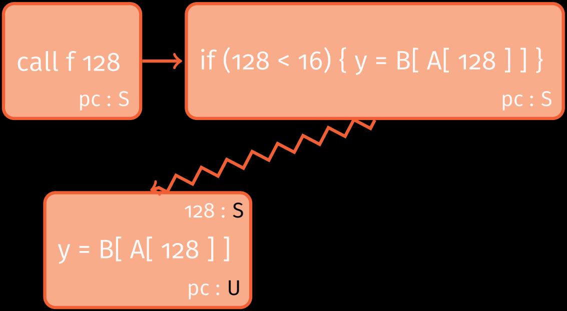
integrity lattice: $S \subset U$ $S \sqcap U = S$ U does not flow to S



Speculative Safety (SS): Taint Tracking

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

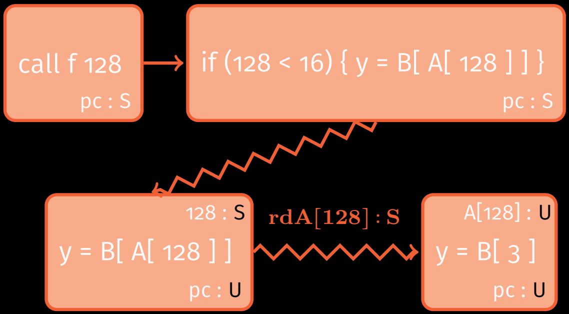
integrity lattice: $S \subset U$ $S \sqcap U = S$ U does not flow to S



Speculative Safety ($\mathcal{SS}$): Taint Tracking

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

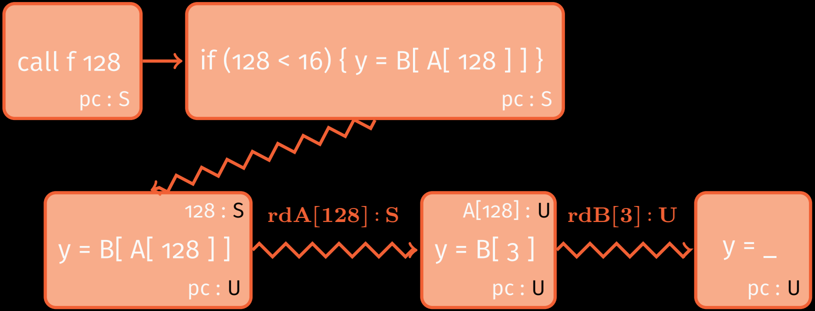
integrity lattice: $S \sqsubset U$ $S \sqcap U = S$ U does not flow to S



Speculative Safety (SS): Taint Tracking

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

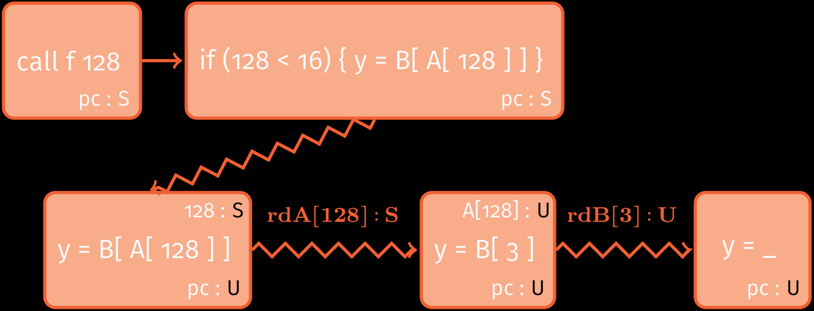
integrity lattice: $S \subset U$ $S \sqcap U = S$ U does not flow to S



Speculative Safety (SS): Taint Tracking

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

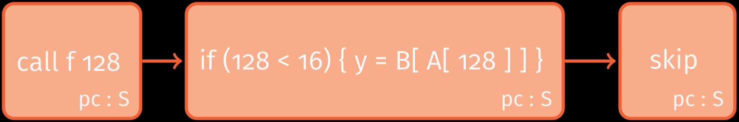
integrity lattice: $S \subset U$ $S \sqcap U = S$ U does not flow to S



Speculative Safety (SS): Taint Tracking

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

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



$\mathbf{rdA[128] : S}$ $\mathbf{rdB[3] : U}$

SS-**Preserving Compiler:** RSSC **&** RSSP

$$[\![\cdot]\!] : \text{RSSP} \overset{\text{def}}{=} \text{ if } \forall \mathsf{A}.\mathsf{A}[\mathsf{P}] : SS \text{ then } \forall \mathbf{A}.\mathbf{A}[[\![\mathsf{P}]\!]] : SS$$

SS-Preserving Compiler: RSSC & RSSP

$$[\![\cdot]\!] : \text{RSSP} \overset{\text{def}}{=} \text{ if } \forall \mathsf{A}.\mathsf{A}[\mathsf{P}] : SS \text{ then } \forall \mathbf{A}.\mathbf{A}[[\![\mathsf{P}]\!]] : SS$$

$$[\![\cdot]\!] : \text{RSSC} \overset{\text{def}}{=} \text{ if } \forall \mathbf{A}.\mathbf{A}[[\![\mathsf{P}]\!]] \rightsquigarrow \mathbf{m} \text{ then } \exists \mathsf{A}.\mathsf{A}[\mathsf{P}] \rightsquigarrow \mathsf{m} \approx \mathbf{m}$$

$\approx$= same traces, plus $\mathbf{S}$ actions in $\mathbf{m}$

SS-Preserving Compiler: RSSC & RSSP

$$[\![\cdot]\!] : \text{RSSP} \stackrel{\text{def}}{=} \text{ if } \forall \mathsf{A}.\mathsf{A}[\mathsf{P}] : SS \text{ then } \forall \mathbf{A}.\mathbf{A}[[\![\mathsf{P}]\!]] : SS$$

$$[\![\cdot]\!] : \text{RSSC} \stackrel{\text{def}}{=} \text{ if } \forall \mathbf{A}.\mathbf{A}[[\![\mathsf{P}]\!]] \rightsquigarrow \mathbf{m} \text{ then } \exists \mathsf{A}.\mathsf{A}[\mathsf{P}] \rightsquigarrow \mathsf{m} \approx \mathbf{m}$$

$\approx =$ same traces, plus $\mathbf{S}$ actions in $\mathbf{m}$

- RSSC & RSSP are equivalent

SS-Preserving Compiler: RSSC & RSSP

$$[\![\cdot]\!] : \text{RSSP} \overset{\text{def}}{=} \text{ if } \forall A. A[P] : SS \text{ then } \forall \mathbf{A}. \mathbf{A}[[\![P]\!]] : SS$$

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

$\approx$= same traces, plus $\mathbf{S}$ actions in $\mathbf{m}$

- RSSC & RSSP are equivalent
- `lfence` : RSSC because it has no speculation ($\mathbf{pc}$: $\mathbf{S}$ always)

SS-**Preserving Compiler:** RSSC **&** RSSP

$$[\![\cdot]\!] : \mathrm{RSSP} \overset{\text{def}}{=} \text{ if } \forall A. A\left[P\right] : \mathit{SS} \text{ then } \forall \mathbf{A}. \mathbf{A}\left[[\![P]\!]\right] : \mathit{SS}$$

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

$\approx =$ same traces, plus $\mathbf{S}$ actions in $\mathbf{m}$

- RSSC & RSSP are equivalent
- `lfence` : RSSC because it has no speculation ($\mathbf{pc}$: $\mathbf{S}$ always)
- `SLH` : RSSC because masking taints as $\mathbf{S}$

RSSC **for** `lfence`

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

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

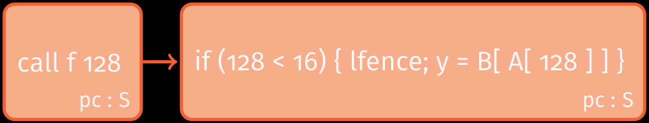
call f 128

pc : S

RSSC **for** `lfence`

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

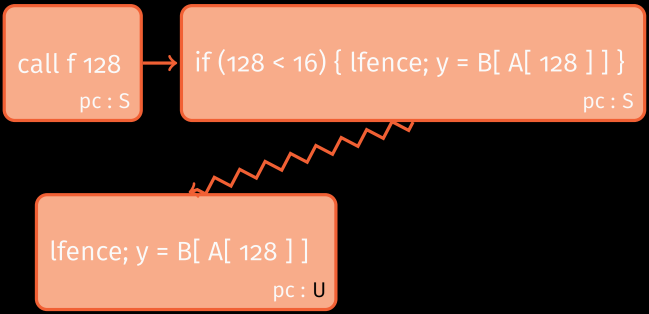
$[\![\cdot]\!] = \textbf{void f(int x)} \mapsto \textbf{if}(\mathbf{x} < \mathbf{A.size})\{\textbf{lfence}; \mathbf{y} = \mathbf{B}[\mathbf{A}[\mathbf{x}]]\}$



RSSC **for** `lfence`

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

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

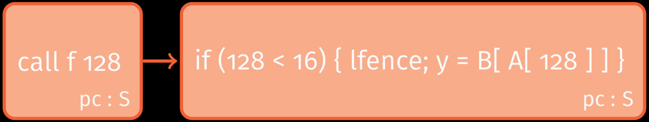


call f 128
pc : S
if (128 < 16) { lfence; y = B[A[128]] }
pc : S
lfence; y = B[A[128]]
pc : U

RSSC for `lfence`

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

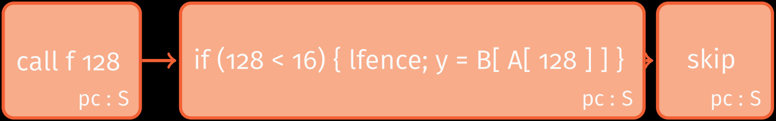
$[\![\cdot]\!] = \mathbf{void\ f(int\ x)} \mapsto \mathbf{if}(\mathbf{x} < \mathbf{A.size})\{\mathbf{lfence}; \mathbf{y} = \mathbf{B}[\mathbf{A}[\mathbf{x}]]\}$



RSSC **for** `lfence`

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

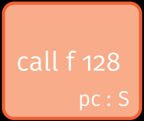
$[\![\cdot]\!] = \textbf{void f(int x)} \mapsto \textbf{if}(\mathbf{x} < \mathbf{A.size})\{\mathbf{lfence}; \mathbf{y} = \mathbf{B}[\mathbf{A}[\mathbf{x}]]\}$



RSSC for SLH

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

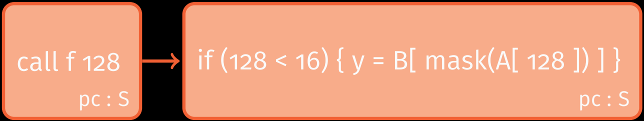
$[\![\cdot]\!] = \textbf{void f(int x)} \mapsto \textbf{if}(\textbf{x} < \textbf{A.size})\{\textbf{y} = \textbf{B}[\textbf{mask}(\textbf{A}[\textbf{x}])]\}$



RSSC **for** SLH

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

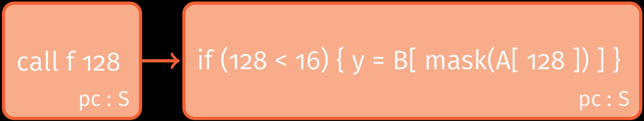
$[\![\cdot]\!] = \textbf{void f(int x)} \mapsto \textbf{if(x < A.size)\{y = B[mask(A[x])]\}}$



RSSC **for** SLH

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

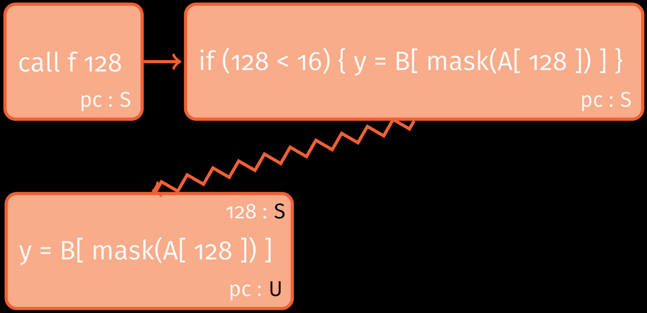
$[\![\cdot]\!] = \mathbf{void\ f(int\ x)} \mapsto \mathbf{if}(\mathbf{x} < \mathbf{A.size})\{\mathbf{y} = \mathbf{B}[\mathbf{mask}(\mathbf{A}[\mathbf{x}])]\}$



RSSC **for** SLH

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

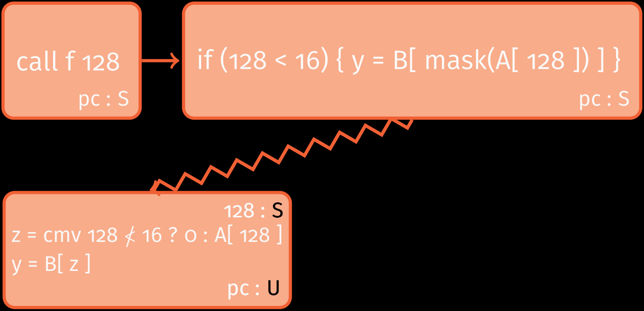
$[\![\cdot]\!] = \mathbf{void\ f(int\ x)} \mapsto \mathbf{if}(\mathbf{x} < \mathbf{A.size})\{\mathbf{y} = \mathbf{B}[\mathbf{mask}(\mathbf{A}[\mathbf{x}])]\}$



RSSC **for** SLH

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

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

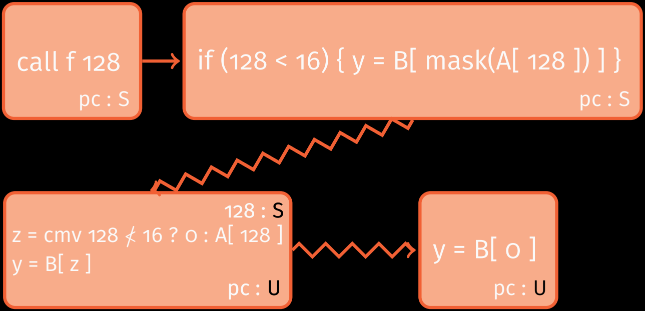


call f 128
pc : S
if (128 < 16) { y = B[mask(A[128])] }
pc : S
128 : S
z = cmv 128 ≮ 16 ? 0 : A[128]
y = B[z]
pc : U

RSSC **for** SLH

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

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

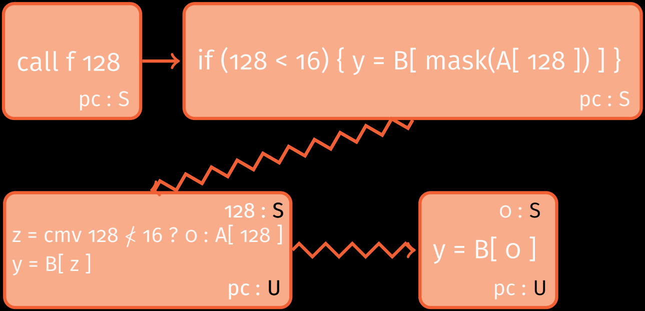


call f 128
pc : S
if (128 < 16) { y = B[mask(A[128])] }
pc : S
128 : S
z = cmv 128 ≮ 16 ? 0 : A[128]
y = B[z]
pc : U
y = B[0]
pc : U

RSSC **for** SLH

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

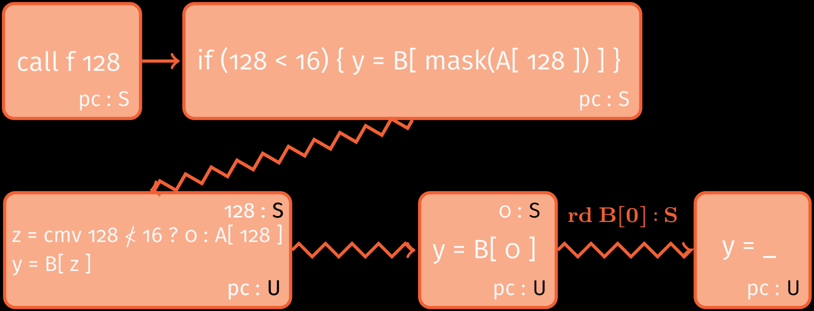
$[\![\cdot]\!]$ = **void f(int x) $\mapsto$ if(x < A.size){y = B[mask(A[x])]}**


call f 128
pc : S
if (128 < 16) { y = B[mask(A[128])] }
pc : S
128 : S
z = cmv 128 ≮ 16 ? 0 : A[128]
y = B[z]
pc : U
0 : S
y = B[0]
pc : U

RSSC **for** SLH

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

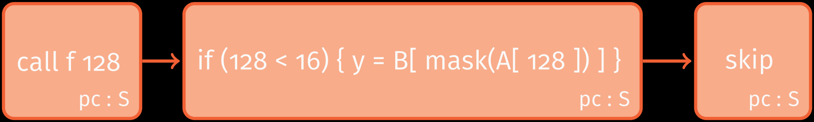
$[\![\cdot]\!]$ = **void f(int x) $\mapsto$ if(x < A.size){y = B[mask(A[x])]}**



RSSC for SLH

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

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



$\mathbf{rd\ B[0] : S}$

Future Outlook

What More?

What More?

- secure compilation to webassembly
- secure compilation is universal composability
- secure compilation and optimisations
- secure compilation for linear languages
- …

Questions?