

# CSC Report – Foundations of Secure Compilation

---



Marco Patrignani<sup>1,2</sup>

23<sup>rd</sup> June 2021



# Talk Outline

My Stanford Experience

Foundations of Secure Compilation

Future Outlook

# **My Stanford Experience**

---



Germany

Belgium

Luxembourg

MAX PLANCK INSTITUTE  
FOR SOFTWARE SYSTEMS

CISPA-Stanford Center

FOR CYBERSECURITY

KU LEUVEN



ALMA MATER STUDIORUM  
UNIVERSITA DI BOLOGNA



Avignon

Slovenia

Croatia





**Stanford  
University**

**5.**



Lucile Packard Children's  
Hospital Stanford

Stanford Hospital at  
300 Pasteur Drive

STOCK FARM GARAGE

L-22

Arrillaga Outdoor  
Education and...

Stanford Golf  
Driving Range

Lake Lagunita

Stanford Bookstore

Stanford  
University

Wilbur Field Garage

Stanford  
Shopping Center

Palo Alto

Alma St

Nordstrom

Arizona  
Garden

Lasuen Grove

Stanford Stadium

Iris & B. Gerald Cantor  
Center for Visual Arts

Stanford Visitor Center

Stanford Oval

Stanford Graduate  
School of Business

Arrillaga Cen  
Sports and R

Creek Apartments

Stanford Golf Course

Rains Apart



terrific experience

- mentoring (John & lunches)



## terrific experience

- mentoring (John & lunches)
- teaching (courses & lunches)

## terrific experience

- mentoring (John & lunches)
- teaching (courses & lunches)
- research (+ talks)



## terrific experience

- mentoring (John & lunches)
- teaching (courses & lunches)
- research (+ talks)
- new perspective

## terrific experience

- mentoring (John & lunches)
- teaching (courses & lunches)
- research (+ talks)
- new perspective
- skiing (who'd have thought?)

# Foundations of Secure Compilation

---

# Outline

1. Motivation behind SC
2. history of SC
3. our contributions to the foundations of SC
4. current and future applications



# Special Thanks to:

(wrt the contents of this talk)



Carmine Abate



Amal Ahmed



Roberto Blanco



Stefan Ciobaca



Dave Clarke



Dominique Devriese



Akram El-Korashy



Deepak Garg



Marco Guarnieri



Catalin Hritcu



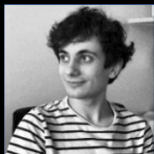
Robert Künnemann



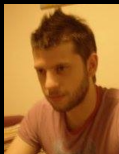
Frank Piessens



Eric Tanter



Jeremy Thibault



Stelios Tsampas



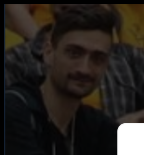
Marco Vassena



Riad Wahby

# Special Thanks to:

(wrt the contents of this talk)



Carmine Abate



riese

please interrupt & ask questions



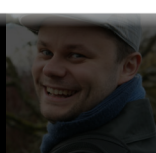
Akram El-Korashy



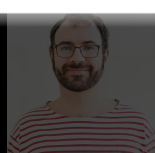
Deepak Garg



Marco Guarnieri



Catalin Hritcu



Robert Künnemann



Frank Piessens



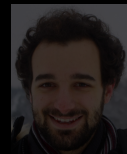
Eric Tanter



Jeremy Thibault



Stelios Tsampas



Marco Vassena



Riad Wahby

# 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 (`[.]`) and **linked** with adversarial target code

# Programming Languages: Pros and Problems

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

- helpful **abstractions** to write **secure** code

but

- when compiled (`[.]`) and **linked** with adversarial target code
- these abstractions are **NOT** enforced

# Secure Compilation: Example

ChaCha20

Poly1305

...

$F^*$

HACL\*: ... CCS'17

*Asm*

[[ChaCha20]]

[[Poly1305]]

[[...]]

# Secure Compilation: Example

ChaCha20

Poly1305

...

$F^*$

HACL\*: ... CCS'17

Asm

[[ChaCha20]]

[[Poly1305]]

[[...]]



160x C/C++ code (unsafe)

# Secure Compilation: Example

Preserve the security of

ChaCha20

Poly1305

...

$F^*$  HACL\*: ... CCS'17

Asm

[[ChaCha20]]

[[Poly1305]]

[[...]]





# Secure Compilation: Example

Preserve the security of



$F^*$  HACL\*: ... CCS'17

Asm



when interoperating with

# Secure Compilation: Example

*Correct compilation*

ChaCha20

Poly1305

...

$F^*$

HACL\*: ...CCS'17

*Asm*

[[ChaCha20]]

[[Poly1305]]

[[...]]

# Secure Compilation: Example

Secure compilation

ChaCha20

Poly1305

...

$F^*$

HACL\*: ... CCS'17

Asm

[[ChaCha20]]

[[Poly1305]]

[[...]]



# Secure Compilation: Example

Enable source-level security reasoning

ChaCha20

Poly1305

...

$F^*$

HACL\*: ... 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**?

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  $Q$ ,*

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

# Once Upon a Time in Process Algebra

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



# Once Upon a Time in Process Algebra

## Fully Abstract Compilation (FAC)

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

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

# Fully Abstract Compilation Influence

Fully Abstract Compilation to JavaScript

Secure Implementations for Typed Session Abstraction

*Typed Closure Conversion Preserves Observational Equivalence*

Chen Pierre-Evariste Dagand Pierre-Yves Strub<sup>1</sup> Benj  
MSR-INRIA<sup>1</sup>  
math.ac.uk pierre-yves@stru

Ricardo Corin<sup>1,2,3</sup> Pierre-Malo Deniérou<sup>1,2</sup> Cédric Fournet<sup>1,2</sup>  
Karthikeyan Bhargavan<sup>1,2</sup> James Leifer<sup>1</sup>  
<sup>1</sup> MSR-INRIA Joint Centre <sup>2</sup> Microsoft Research <sup>3</sup> University of T

Amal Ahmed Matthias Blume  
Toyota Technological Institute at Chicago  
(amal,blume)@ti-c.org

Fully-Abstract Compilation by Approximate Back-Translation

Dominique Devriese Marco Patrignani Frank Piessens  
iMinds-DistriNet, Computer Science dept., KU Leuven  
frank.last@cs.kuleuven.ac.be

Authentication primitives and their compilation

Martín Abadi\*  
Bell Labs Research  
Lucent Technologies

Cédric Fournet  
Microsoft Research

Georges G  
INRIA Rocq

On Protection by Layout Randomization

MARTÍN ABADI, Microsoft Research, Silicon Valley  
Santa Cruz, Collège de France  
GORDON D. PLOTKIN, University of Edinburgh

*Beyond Good and Evil*

*Formalizing the Security Guarantees of Compartmentalizing Compilation*

Yannis Juglaret<sup>1,2</sup> Cătălin Hrișcu<sup>1</sup> Arthur Azevedo de Amorim<sup>1</sup> Boris Eng<sup>1,3</sup> Benjamin C. Pierce<sup>4</sup>  
<sup>1</sup>Inria Paris <sup>2</sup>Université Paris Diderot (Paris 7) <sup>3</sup>Université Paris 8 <sup>4</sup>University of Pennsylvania

A Secure Compiler for ML Modules

Marco Patrignani, Dave Clarke, and Frank Piessens

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

*Local Memory via Layout Randomization*

Corin Pitecher

Julian Rathke  
University of Southampton

James Riely  
University of Cambridge

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

Amal Ahmed

Matthias Blume  
Google  
blume@google.com

*Secure Compilation to Protected Module Architectures*

Marco Patrignani  
Dept. Computer Science  
and Dave Clarke

Fully Abstract Compilation via Universal Embedding\*

Marco Patrignani  
MPI-SWS

Dominique Devriese

# Fully Abstract Compilation Influence

## How

does Fully Abstract Compilation entail  
security?

Typed Closure

Authentication

Martín Abadi\*  
Bell Labs Research  
Lucent Technologies

Security  
of Object-C  
Protected

Marco Patrignani, Dave Clarke, and Frank Piessens

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

Local Memory via Layout Randomization

Corin Pitcher

Julian Rathke  
University of Southampton

Secure Compilation to Protected Module Architectures

Marco Patrignani  
Dept. Comput  
and Dave C

Fully Abstract Compilation via Universal Embedding\*

James Riely  
University

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

Amal Ahmed

On Modular and Fully-Abstract Compil

Matthias Blume  
Google  
blume@google.com

Dominique D

ion Abstraction

Cédric Fournet<sup>1,2</sup>  
nes Leifer<sup>1</sup>

<sup>2</sup> University of T

-Translation

Pierce<sup>a</sup>  
sylvania

L Module

and Dave Clar

# Fully Abstract Compilation Influence

## How

does Fully Abstract Compilation entail  
security?

FAC ensures that a **target – level**  
attacker has the **same power** of a  
**source – level** one  
as captured by the semantics

Marco Patrignani, Dave Clarke, and Frank Piessens

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

Local Memory via Layout Randomization

Corin Pitcher

Julian Rathke

University of Southampton

James Riely

University of Cambridge

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

Amal Ahmed

Secure Compilation to Protected Module Architectures

Marco Patrignani  
Dept. Computer Science  
and Dave Clarke

Fully Abstract Compilation via Universal Embedding\*

Marco Patrignani  
MPI-SW

Matthias Blume  
Google  
blume@google.com

# Fully Abstract Compilation: Definition

$$P_1 \approx_{ctx} P_2$$

$$\llbracket P_1 \rrbracket \approx_{ctx} \llbracket P_2 \rrbracket$$

# Fully Abstract Compilation: Definition

$$P_1 \approx_{ctx} P_2$$



$$\llbracket P_1 \rrbracket \approx_{ctx} \llbracket P_2 \rrbracket$$

# Fully Abstract Compilation: Definition

$$P_1 \approx_{ctx} P_2$$

$$\llbracket P_1 \rrbracket \approx_{ctx} \llbracket P_2 \rrbracket$$

# Fully Abstract Compilation: Definition

$$P_1 \simeq_{ctx} P_2$$



$$\forall A. A \llbracket P_1 \rrbracket \Downarrow \iff A \llbracket P_2 \rrbracket \Downarrow$$



# 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

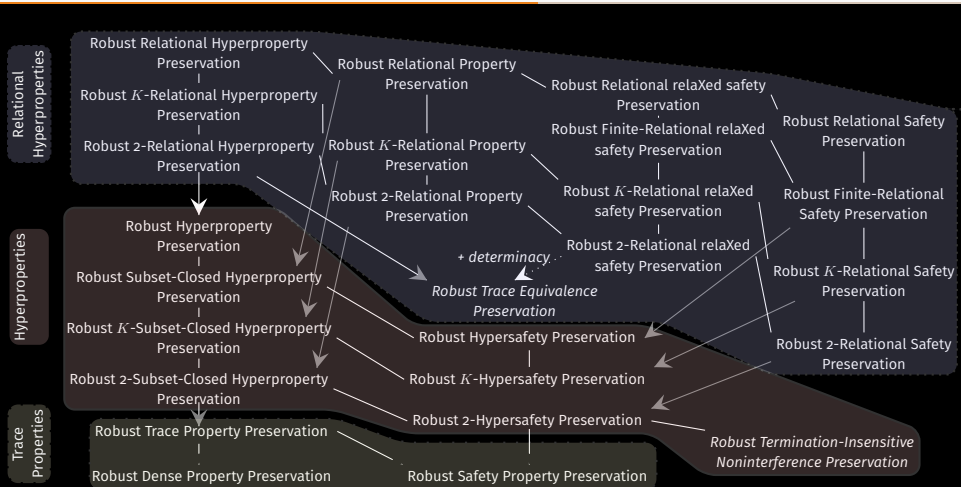
# Are there Alternatives to FAC?

- 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**

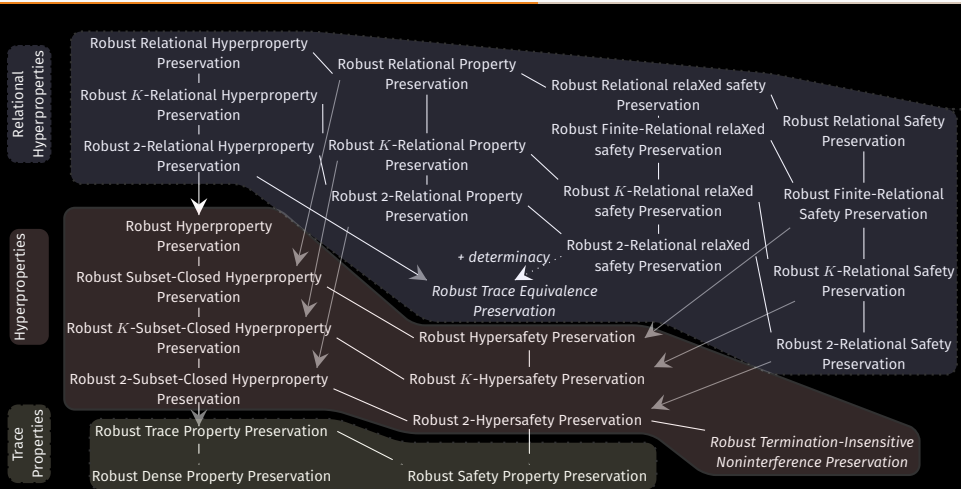
# Robust Compilation Criteria

CSF'19, ESOP'20, ACM Toplas'21



# Robust Compilation Criteria

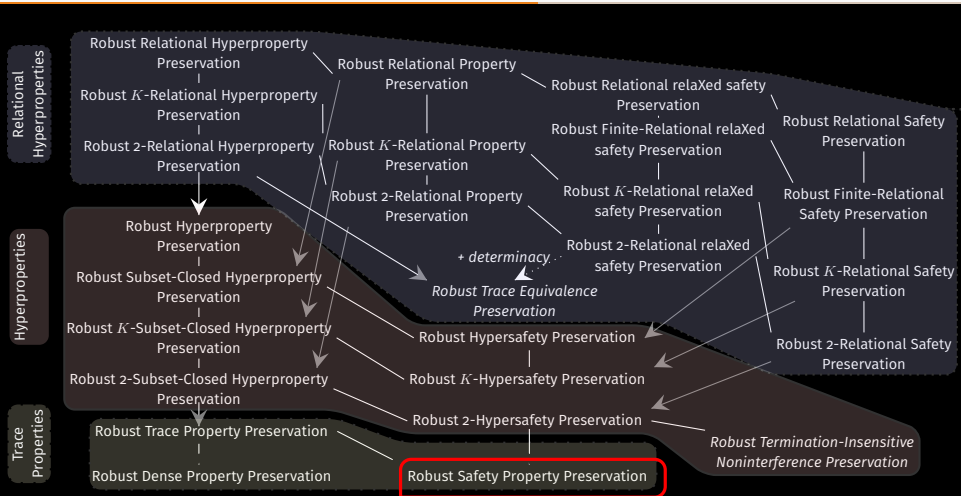
CSF'19, ESOP'20, ACM Toplas'21



Tradeoffs for code efficiency, security guarantees, proof complexity

# Robust Compilation Criteria

CSF'19, ESOP'20, ACM Toplas'21



Tradeoffs for code efficiency, security guarantees, proof complexity

# Robust Criteria: Intuition

Each point has two **equivalent** criteria:

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



# Robust Criteria: Intuition

Each point has two **equivalent** criteria:

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

# 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:

- **Property – ful** :
  - + clearly tells what **class** it preserves
  - harder to prove
- **Property – free** :
  - + **easier** to prove
  - unclear what security classes are preserved

# Robust Criteria: Intuition

Each point has two **equivalent** criteria:

- **Property – ful** :
  - + clearly tells what **class** it preserves
  - harder to prove
- **Property – free** :
  - + **easier** to prove
  - unclear what security classes are preserved
  - = akin to some crypto statements (**UC**)

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[[\cdot]] = \text{compiler}$      $[[\cdot]] : \text{RSP} \stackrel{\text{def}}{=} \text{RSC}$

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[\cdot]$  = compiler  
 $\pi / \pi$  = set of traces

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

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

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

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

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

if  $(\forall A, t.$



# In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

$\pi / \pi$  = set of traces

$P$  = partial program

$A / \mathbf{A}$  = attacker

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

$[\cdot]$  = linking

$\rightsquigarrow / \rightsquigarrow$  = trace semantics

if  $(\forall A, \mathbf{t}. A [P] \rightsquigarrow \mathbf{t})$

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[\![\cdot]\!] = \text{compiler}$

$\pi / \pi = \text{set of traces}$

$P = \text{partial program}$

$A / \mathbf{A} = \text{attacker}$

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

$[\cdot] = \text{linking}$

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

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

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

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[\![\cdot]\!] = \text{compiler}$

$\pi / \pi = \text{set of traces}$

$P = \text{partial program}$

$A / \mathbf{A} = \text{attacker}$

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

$[\cdot] = \text{linking}$

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

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

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

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

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[[\cdot]]$  = compiler

$\pi / \pi$  = set of traces

$P$  = partial program

$A / \mathbf{A}$  = attacker

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

$[\cdot]$  = linking

$\rightsquigarrow / \rightsquigarrow$  = trace semantics

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

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

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

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[[\cdot]]$  = compiler

$\pi / \pi$  = set of traces

$P$  = partial program

$A / \mathbf{A}$  = attacker

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

$[\cdot]$  = linking

$\sim / \rightsquigarrow$  = trace semantics

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

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

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

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[[\cdot]]$  = compiler

$\pi / \pi$  = set of traces

$P$  = partial program

$A / \mathbf{A}$  = attacker

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

$[\cdot]$  = linking

$\rightsquigarrow / \rightsquigarrow$  = trace semantics

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

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

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

$[[\cdot]] : \text{RSC} \stackrel{\text{def}}{=}$

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[[\cdot]]$  = compiler

$\pi/\pi$  = 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

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

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

then  $(\forall A, t. A [[P]] \rightsquigarrow t \Rightarrow t \in \pi)$

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

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[\cdot]$  = compiler

$\pi/\pi$  = 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

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

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

then  $(\forall A, t. A [[P]] \rightsquigarrow t \Rightarrow t \in \pi)$

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

if  $A [[P]] \rightsquigarrow m$



# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[\cdot]$  = compiler

$\pi/\pi$  = set of traces

$P$  = partial program

$A/\mathbf{A}$  = attacker

$t/t$  = trace of events

$[\cdot]$  = linking

$\rightsquigarrow/\rightsquigarrow$  = trace semantics

$m/m$  = prefix of a trace

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

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

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

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

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

then  $\exists A, m.$

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[\cdot]$  = compiler

$\pi/\pi$  = set of traces

$P$  = partial program

$A/\mathbf{A}$  = attacker

$t/t$  = trace of events

$[\cdot]$  = linking

$\rightsquigarrow/\rightsquigarrow$  = trace semantics

$m/m$  = prefix of a trace

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

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

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

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

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

then  $\exists A, m. A [P] \rightsquigarrow m$

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[\cdot]$  = compiler

$\pi/\pi$  = set of traces

$P$  = partial program

$A/\mathbf{A}$  = attacker

$t/t$  = trace of events

$[\cdot]$  = linking

$\rightsquigarrow/\rightsquigarrow$  = trace semantics

$m/m$  = prefix of a trace

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

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

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

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

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

then  $\exists A, m. A [P] \rightsquigarrow m$  and  $m \approx m$

# 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

# RSC - FAC

Both:

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

FAC:

# RSC - FAC

Both:

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

FAC:

- yields a **language** result

POPL'21

# RSC - FAC

Both:

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

FAC:

- yields a **language** result

POPL'21

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)



# Robust(ly Safe) Compilation at Work

Instantiate RSC to specific properties

- absence of **speculation leaks**

CCS'21

# Robust(ly Safe) Compilation at Work

Instantiate RSC to specific properties

- absence of **speculation leaks**
- **memory safety** preservation (spatial, temporal)

CCS'21

# Robust(ly Safe) Compilation at Work

Instantiate RSC to specific properties

- absence of **speculation leaks**
- **memory safety** preservation (spatial, temporal)
- **constant-time** preservation

CCS'21

# Robust(ly Safe) Compilation at Work

Instantiate RSC to specific properties

- absence of **speculation leaks**
- **memory safety** preservation (spatial, temporal)
- **constant-time** preservation
- ...

CCS'21

# Future Outlook

---

# More Secure Compilation

# More Secure Compilation

- secure compilation for Spectre V2+  
(w. Imdea, Cisca)

# More Secure Compilation

- secure compilation for Spectre V2+  
(w. Imdea, Cisca)
- secure compilation to webassembly  
(w. UCSD, Harvard, Cisca)



# More Secure Compilation

- secure compilation for Spectre V2+  
(w. Imdea, Cisca)
- secure compilation to webassembly  
(w. UCSD, Harvard, Cisca)
- secure compilation & universal  
composability  
(w. Stanford, Cisca)

# More Secure Compilation

- secure compilation for Spectre V2+  
(w. Imdea, Cisca)
- secure compilation to webassembly  
(w. UCSD, Harvard, Cisca)
- secure compilation & universal  
composability  
(w. Stanford, Cisca)
- secure compilation for linear languages  
(w. Novi / FB)

# More Secure Compilation

- secure compilation for Spectre V2+  
(w. Imdea, Cisca)
- secure compilation to webassembly  
(w. UCSD, Harvard, Cisca)
- secure compilation & universal  
composability  
(w. Stanford, Cisca)
- secure compilation for linear languages  
(w. Novi / FB)
- ... (some PL too, w. Stanford, KU Leuven)

# Questions?

