

CSC Report – Foundations of Secure Compilation



Marco Patrignani^{1,2}

23rd June 2021



CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

²



Stanford
University

Talk Outline

My Stanford Experience

Foundations of Secure Compilation

Future Outlook

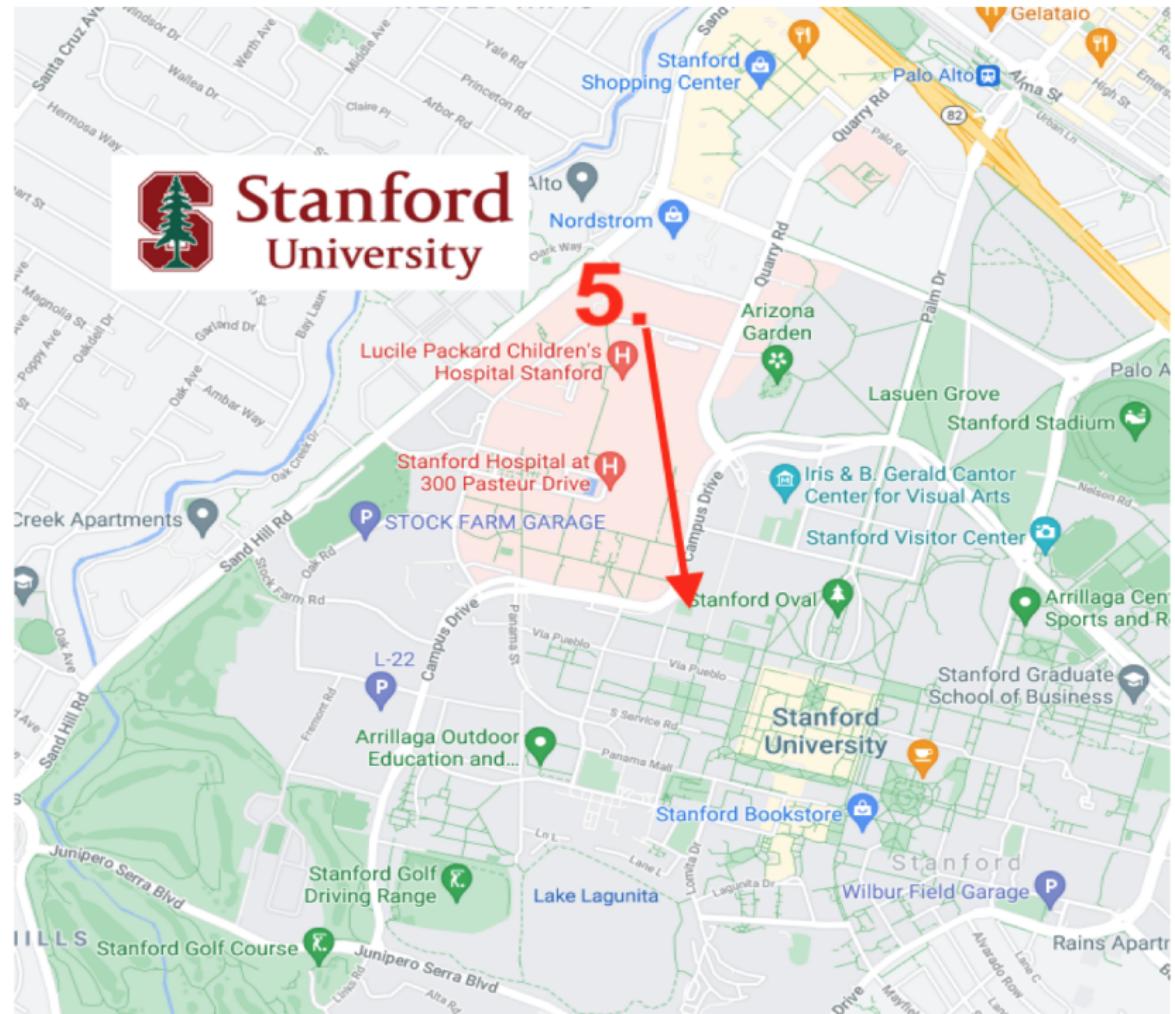
My Stanford Experience

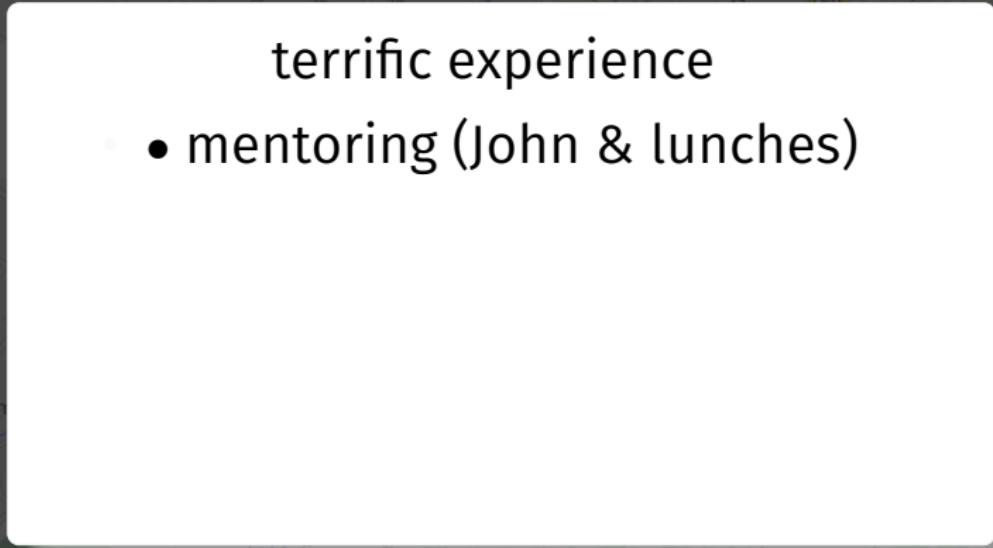




Stanford University

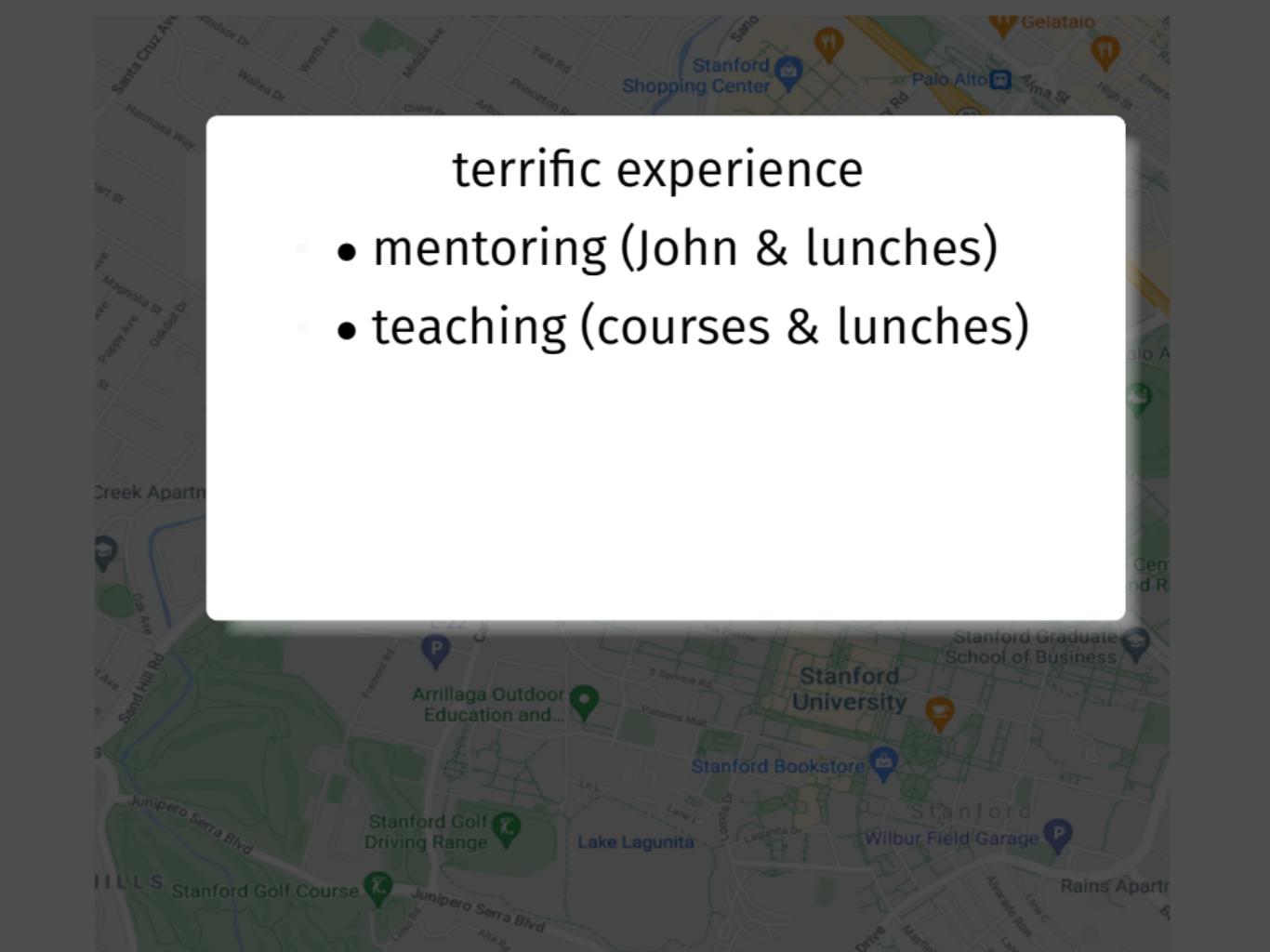
5.





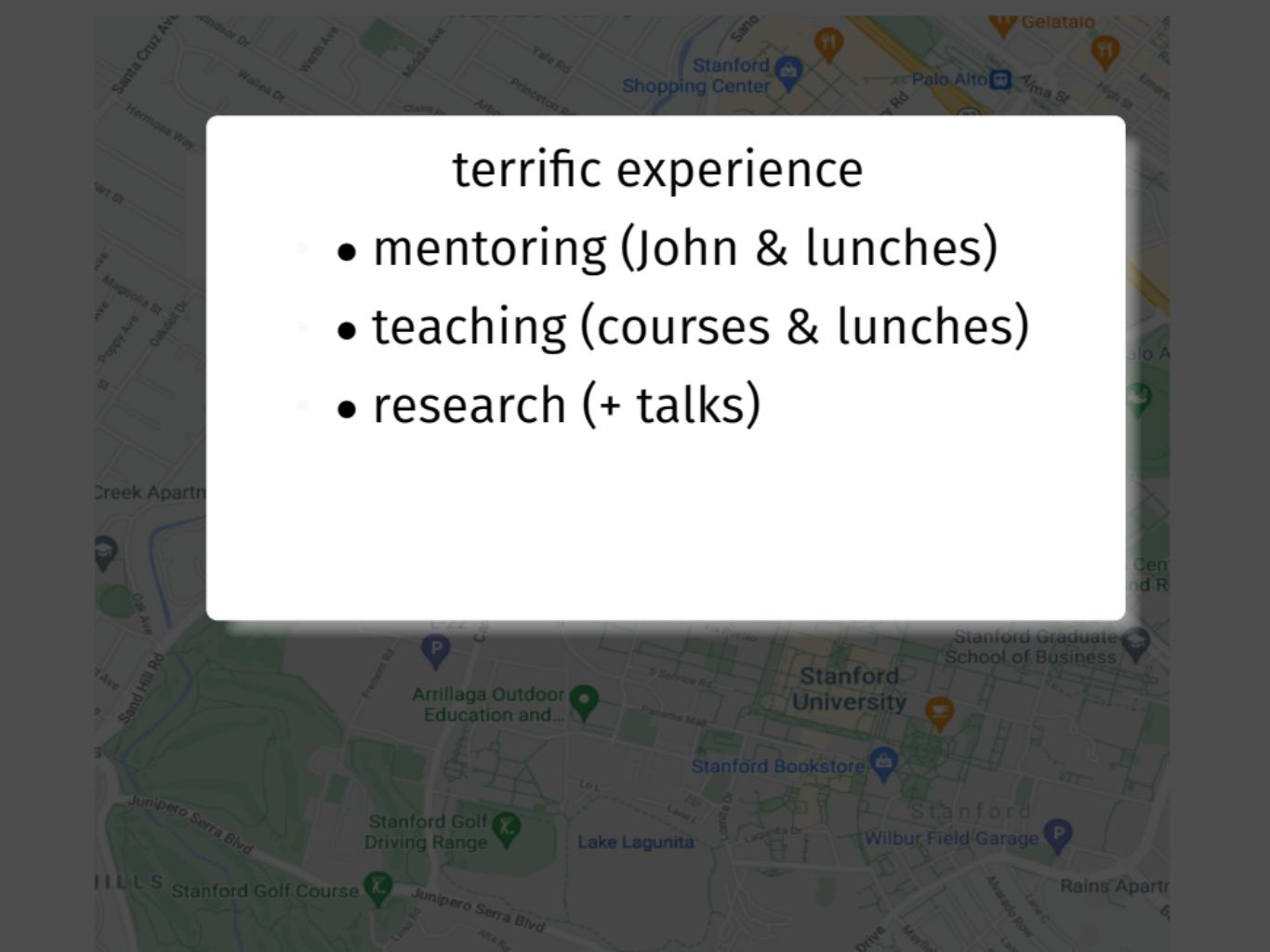
terrific experience

- 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

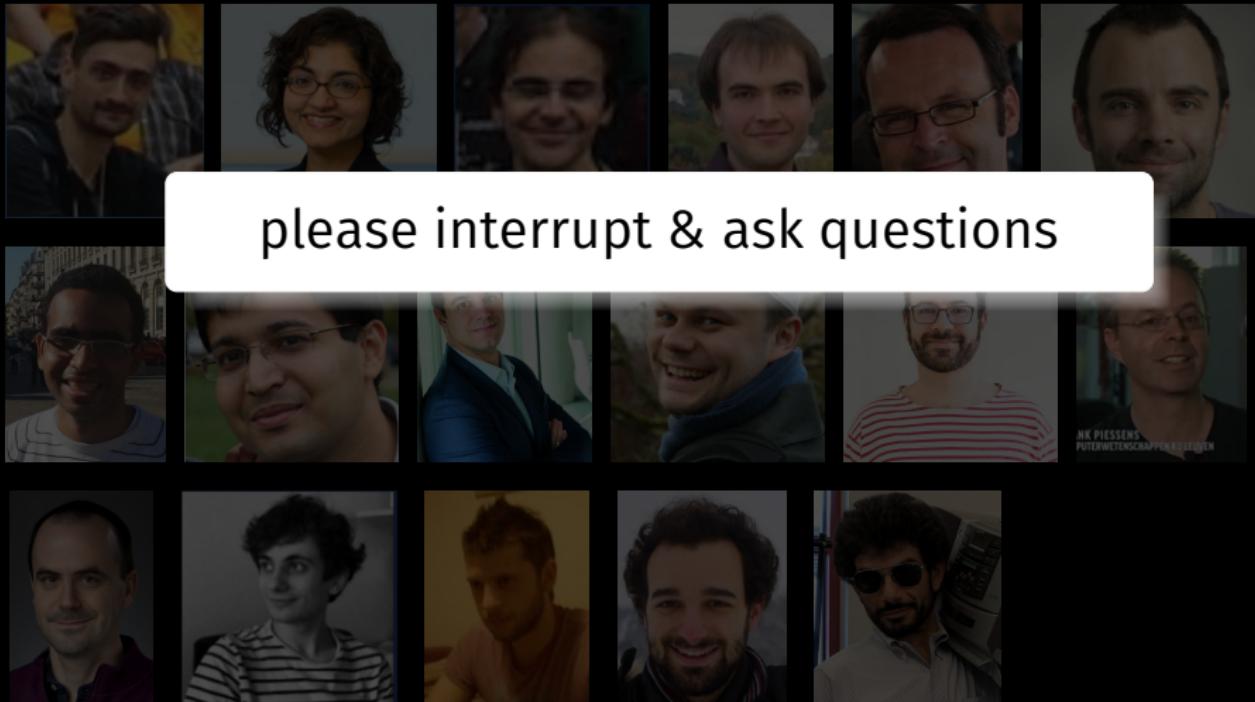
Special Thanks to:

(wrt the contents of this talk)



Special Thanks to:

(wrt the contents of this talk)



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^*
HAACL*: ...CCS'17

Asm

[[ChaCha20]]

[[Poly1305]]

[[...]]



160x C/C++ code (unsafe)

Secure Compilation: Example

Preserve the security of

ChaCha20

Poly1305

...

F* HAACL*: ...CCS'17

Asm

[[ChaCha20]]

[[Poly1305]]

[[...]]



Secure Compilation: Example

Preserve the security of

ChaCha20

Poly1305

...

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

Asm

[[ChaCha20]]

[[Poly1305]]

[[...]]



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^* HAACL*: ... 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 ,*

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

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 (FAQ)

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

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

Fully Abstract Compilation Influence

Fully Abstract Compilation to JavaScript

J.-Chen¹ Pierre-Evariste Dagand² Pierre-Yves Strub¹ Benj³
Amal Ahmed⁴ Matthias Blume⁵
Toyota Technological Institute at Chicago
[amal.blume@ttic.org](mailto:{amal.blume}@ttic.org)

¹ MSR-INRIA² strath.ac.uk pierre-yves@strath.ac.uk

³ MSR-INRIA Joint Centre benjamin.leifer@inria.fr

⁴ Microsoft Research rstefan@research.microsoft.com

⁵ University of Illinois Urbana-Champaign blume@illinois.edu

Authentication primitives and their compilation

Martín Abadi*
Bell Labs Research
Lucent Technologies

Cédric Fournet
Microsoft Research

Georges G. Giacinto
INRIA Rocquencourt

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 Jugla^{1,2} Cătălin Hritcu¹ Arthur Azevedo de Amorim⁴ Boris Eng^{1,3} Benjamin C. Pierce⁴
¹Inria Paris ²Université Paris Diderot (Paris 7) ³Université Paris 8 ⁴University of Pennsylvania

Secure Compilation of Object-Oriented Components to Protected Module Architectures

Marco Patrignani, Dave Clarke, and Frank Piessens*

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

Local Memory via Layout Randomization
Corin Pitcher¹ Julian Rathke²
¹University of Southampton ²University of Cambridge

Secure Compilation to Protected Module Architectures
Amal Ahmed¹ and Raoul Strackx and Bart Jacobs,²
¹ and iMinds-DISTRICT ² MPI-SWS

Marco Patrignani
Dept. Computer Science
and Dave C. Clarke

Fully Abstract Compilation via Universal Embedding*

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 Leifer¹

¹ MSR-INRIA Joint Centre ² Microsoft Research ³ University of Illinois Urbana-Champaign

Fully-Abstract Compilation by Approximate Back-Translation

Dominique Devriese¹ Marco Patrignani² Frank Piessens³
¹iMinds-DistriNet, Computer Science dept., KU Leuven
{first.last}@cs.kuleuven.be

A Secure Compiler for ML Modules

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

Amal Ahmed

* and Dave Clarke

Matthias Blume
Google
blume@google.com

On Modular and Fully-Abstract Compilation
Dominique Devriese¹ and Marco Patrignani²

Fully Abstract Compilation Influence

How does Fully Abstract Compilation entail security?

Authentication

Martín Abadi^{*}
Bell Labs Research
Lucent Technologies

Security
of Object-C
o Protected

Marco Patrignani, Dave Clarke, and Frank Piessens*

iMinds-DistriNet, Dept. Computer Sci.
 $\{first, last\}arr$

Secure Compilation to Protected Module Architectures
Marco Patrignani and Raoul Strackx and Bart Jacobs,ⁱ
and iMinds-D

Fully Abstract Compilation via Universal Embedding*

Marco Patrignani
Dept. Computer Science
and Dave C

Local Memory via Layout Randomization
Corin Pitcher
University of Southampton

James Riedy
Cornell University

On Modular and Fully-Abstract Compilations
Amal Ahmed
MPI-Saarland

Matthias Blume
Google
blume@google.com

An Equivalence-Preserving CPS Translation
via Multi-Language Semantics^{*}
Dominique Pachet
INRIA

^{1,2} and Dave Clark

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

Fully Abstract Compilation: Definition

$$P_1 \simeq_{ctx} P_2$$



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

Fully Abstract Compilation: Definition

$$P_1 \simeq_{ctx} P_2$$



$$[\![P_1]\!] \simeq_{ctx} [\![P_2]\!]$$

Fully Abstract Compilation: Definition

$$P_1 \simeq_{ctx} P_2$$



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

Fully Abstract Compilation: Definition

$$P_1 \simeq_{ctx} P_2$$



$$\forall A. A [[P_1]] \Downarrow \iff A [[P_2]] \Downarrow$$

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

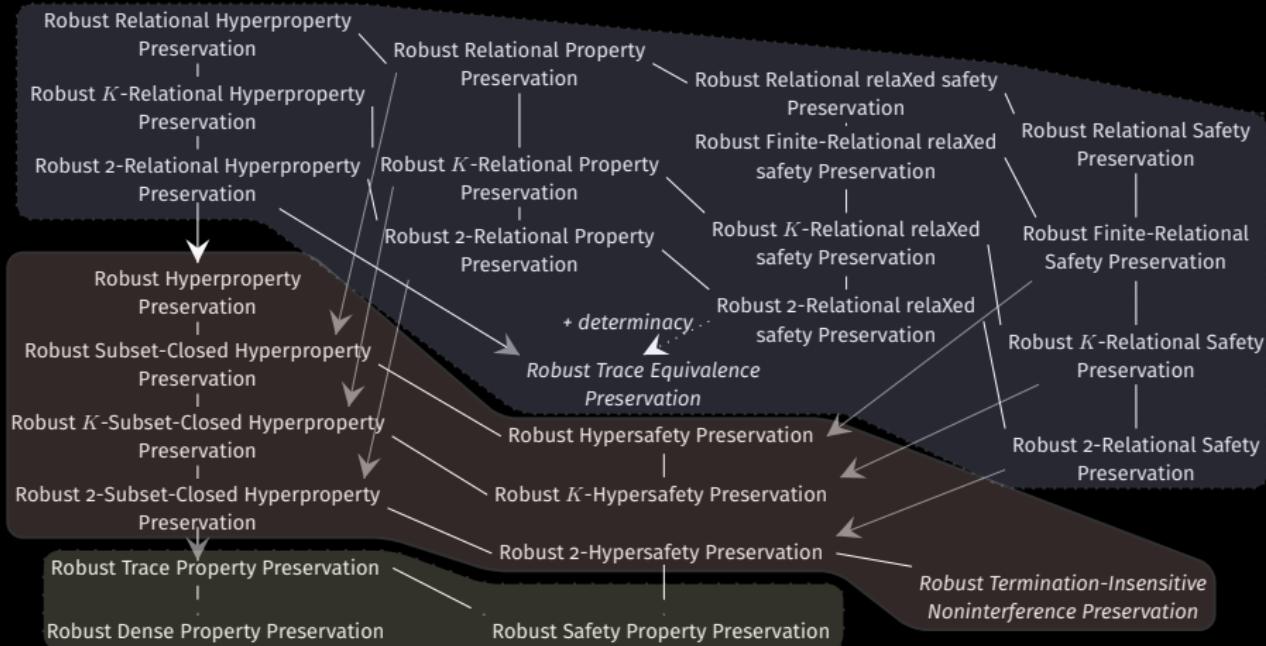
Robust Compilation Criteria

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

Relational
Hyperproperties

Hyperproperties

Trace
Properties



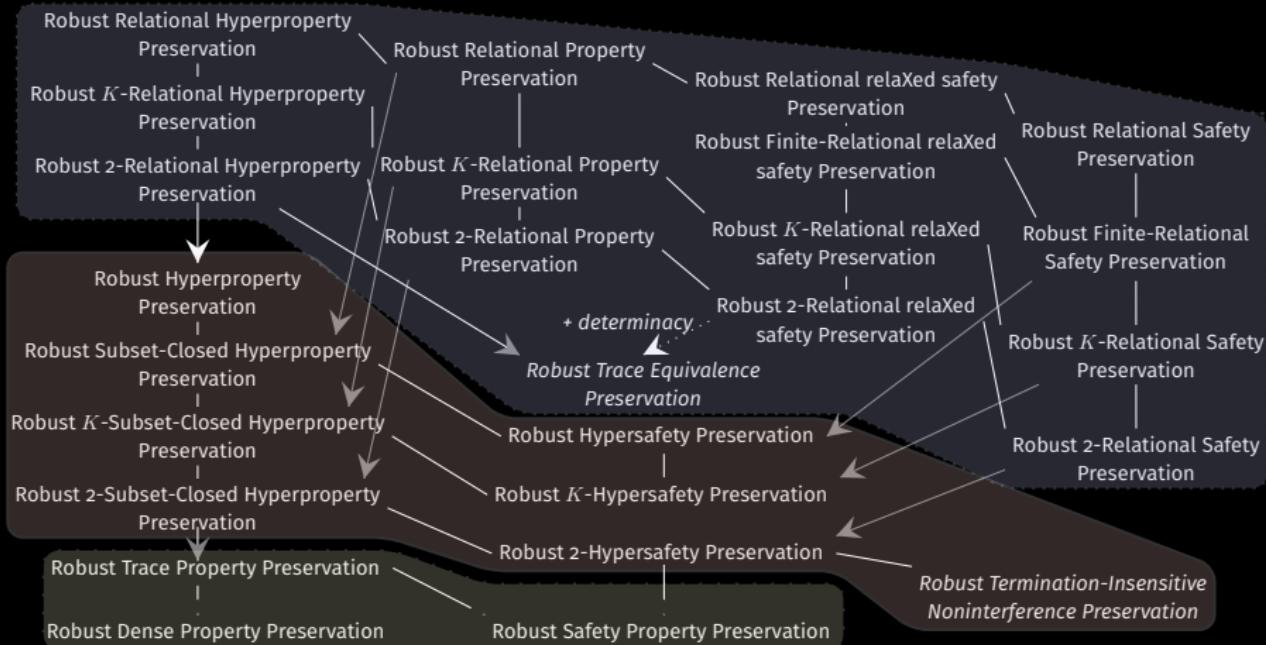
Robust Compilation Criteria

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

Relational
Hyperproperties

Hyperproperties

Trace
Properties



Tradeoffs for code efficiency, security guarantees, proof complexity

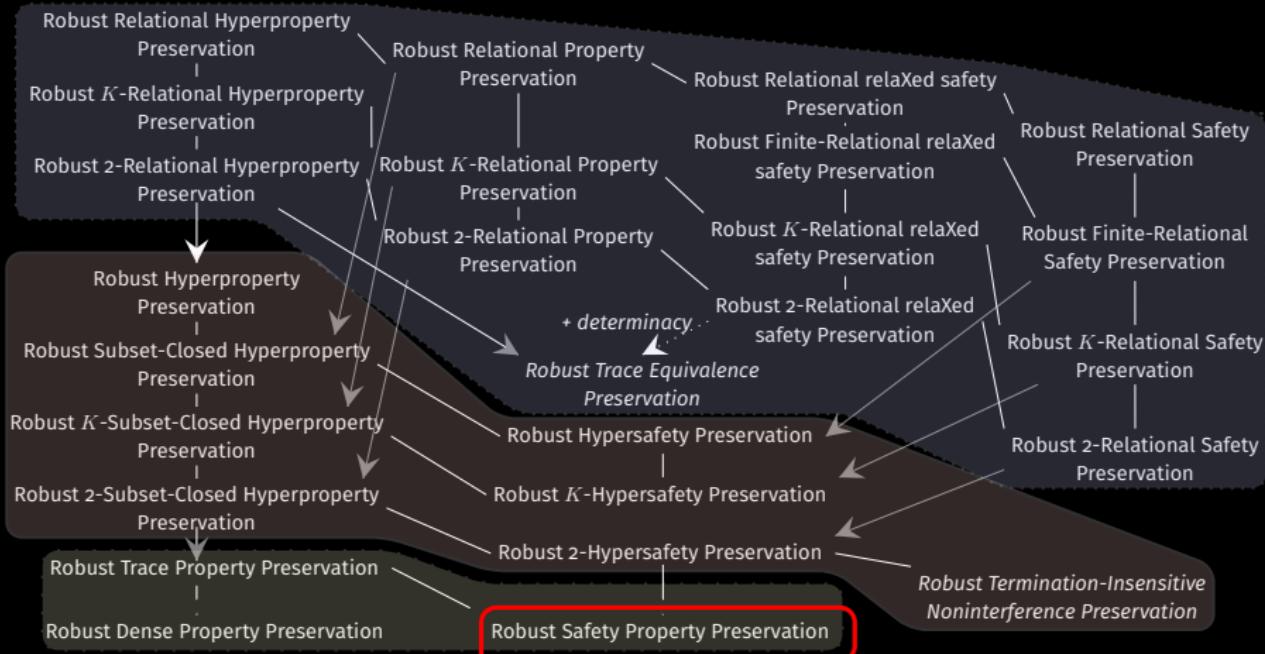
Robust Compilation Criteria

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

Relational
Hyperproperties

Hyperproperties

Trace
Properties



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

$\llbracket \cdot \rrbracket = \text{compiler}$ $\llbracket \cdot \rrbracket : \text{RSP} \stackrel{\text{def}}{=}$

In Depth Example: RSC

ESOP'19, ACM Toplas'21

$\llbracket \cdot \rrbracket = \text{compiler}$ $\llbracket \cdot \rrbracket : \text{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \pi \in \text{Safety}.$
 $\pi / \pi = \text{set of traces}$

In Depth Example: RSC

ESOP'19, ACM Toplas'21

$$\llbracket \cdot \rrbracket : \text{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \tau \in \text{Safety}. \ \forall P.$$

$\llbracket \cdot \rrbracket$ = compiler

π / τ = set of traces

P = partial program

In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

$\llbracket \cdot \rrbracket = \text{compiler}$

π / π = set of traces

P = partial program

A / A = attacker

t / t = trace of events

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

In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

π / π = compiler

π / π = set of traces

P = partial program

A / A = attacker

t / t = trace of events

$[\cdot]$ = linking

$\rightsquigarrow / \rightsquigarrow$ = trace semantics

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

In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

π/ϖ = compiler

π/ϖ = set of traces

P = partial program

A/A = attacker

t/t = trace of events

$[\cdot]$ = linking

$\rightsquigarrow/\rightsquigarrow$ = trace semantics

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

In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

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

then $(\forall A, t.$

$\llbracket \cdot \rrbracket$ = compiler

π / π = set of traces

P = partial program

A / A = attacker

t / t = trace of events

$\llbracket \cdot \rrbracket$ = linking

$\rightsquigarrow / \rightsquigarrow$ = trace semantics

In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

π / π = compiler

π / π = set of traces

P = partial program

A/A = attacker

t/t = trace of events

$[\cdot]$ = linking

$\rightsquigarrow / \rightsquigarrow$ = trace semantics

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

then $(\forall A, t. A[\llbracket P \rrbracket] \rightsquigarrow t \Rightarrow$

In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

π / π = compiler

π / π = set of traces

P = partial program

A/A = attacker

t/t = trace of events

$[\cdot]$ = linking

$\rightsquigarrow / \rightsquigarrow$ = trace semantics

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

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

In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

π / π = compiler

π / π = set of traces

P = partial program

A/A = attacker

t/t = trace of events

$[\cdot]$ = linking

$\rightsquigarrow / \rightsquigarrow$ = trace semantics

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

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

$\llbracket \cdot \rrbracket : \text{RSC} \stackrel{\text{def}}{=}$

In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

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

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

$\llbracket \cdot \rrbracket = \text{compiler}$

π / π = 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

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

In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

$\llbracket \cdot \rrbracket = \text{compiler}$
 $\pi / \pi = \text{set of traces}$
 $P = \text{partial program}$
 $A / A = \text{attacker}$

$t / t = \text{trace of events}$
 $[\cdot] = \text{linking}$
 $\rightsquigarrow / \rightsquigarrow = \text{trace semantics}$
 $m / m = \text{prefix of a trace}$

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

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

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

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

In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

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

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

$\llbracket \cdot \rrbracket = \text{compiler}$

π / π = 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

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

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

then $\exists A, m.$

In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

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

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

$\llbracket \cdot \rrbracket = \text{compiler}$

π / π = 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

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

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

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

In Depth Example: RSC

ESOP'19, ACM Toplas'21

$\llbracket \cdot \rrbracket : \text{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \varpi \in \text{Safety}. \ \forall P.$

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

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

$\llbracket \cdot \rrbracket = \text{compiler}$

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

$P = \text{partial program}$

$A / \textcolor{red}{A} = \text{attacker}$

$t / \textcolor{brown}{t} = \text{trace of events}$

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

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

$m / \textcolor{brown}{m} = \text{prefix of a trace}$

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

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

then $\exists A, m. A[P] \rightsquigarrow m \text{ and } \textcolor{blue}{m} \approx \textcolor{brown}{m}$

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**

Both:

Both:

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

Both:

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

Both:

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

FAC:

Both:

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

FAC:

- yields a language result

POPL'21

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, Cispa)

More Secure Compilation

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

More Secure Compilation

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

More Secure Compilation

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

More Secure Compilation

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

Questions?

