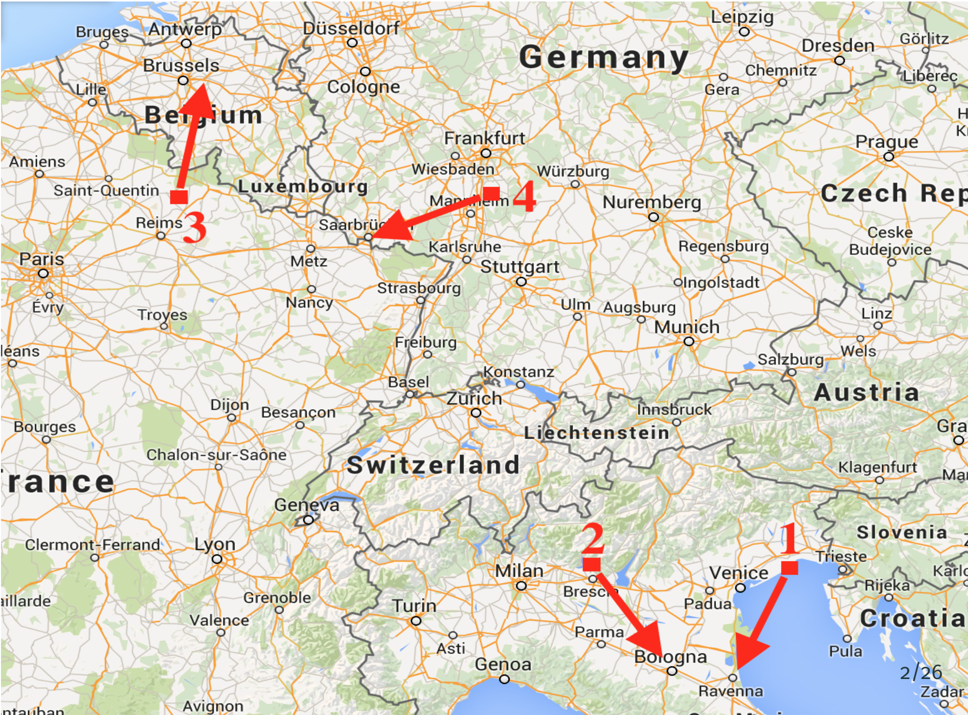
Secure Compilation

Marco Patrignani

17th October 2018

CISPA

Center for IT-Security, Privacy and Accountability


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

Bruges
Antwerp
Düsseldorf
Leipzig
Brussels
Cologne
Germany
Dresden
Görlitz
Chemnitz
Gera
Liberec
Lille
Belgium
Fran
Wiesbade
Prague
Amiens
Saint-Quentin
Luxembourg
Nuremberg
Czech Rep
Reims
3
4
Saarbrüc
Mannheim
Karlsruhe
Metz
Stu
Strasbourg
ancy
Augsburg
Munich
Linz
léans
Freiburg
Konstanz
Basel
Zurich
Dijon
Besançon
Inns
Bourges
tenstein
Gra
Chalon-sur-Saône
rance
Mar
Genev
Clermont-Ferrand
Lyon
2
1
Slovenia
Milan
Trieste
Venice
Karlo
Bresc
Rijeka
aillarde
Grenoble
Padua
Turin
Croatia
Valence
Parma
Asti
Pula
Bologna
Genoa
Ravenna
Zadar
ntauban
Avignon

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

CISPA-Stanford Center

FOR CYBERSECURITY

KU LEUVEN

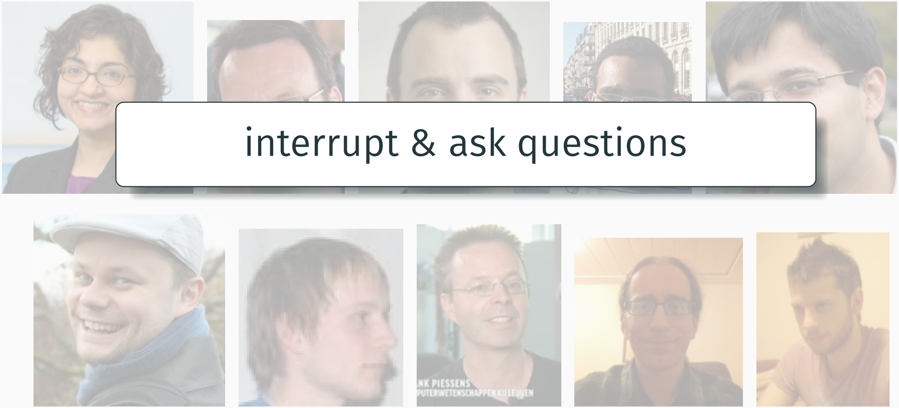
ALMA MATER STUDIORUM A.D. 1088

ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

Collaborators

and more whose image I couldn't find...

Collaborators



interrupt & ask questions

and more whose image I couldn't find...

Contents

- High-level picture

  (i.e., yes, you should pay attention)

Contents

- High-level picture

  (i.e., yes, you should pay attention)
- Low-level details of a secure compiler

  (i.e., what some published work do)

Contents

- High-level picture

  (i.e., yes, you should pay attention)
- Low-level details of a secure compiler

  (i.e., what some published work do)
- Formal definitions of criteria for secure compilation

  (i.e., why is a secure compiler secure)

Contents

- High-level picture

  (i.e., yes, you should pay attention)
- Low-level details of a secure compiler

  (i.e., what some published work do)
- Formal definitions of criteria for secure compilation

  (i.e., why is a secure compiler secure)
- Advanced proof techniques for secure compilation

  (i.e., how much greek gives me a q.e.d.)

What is Secure Compilation?

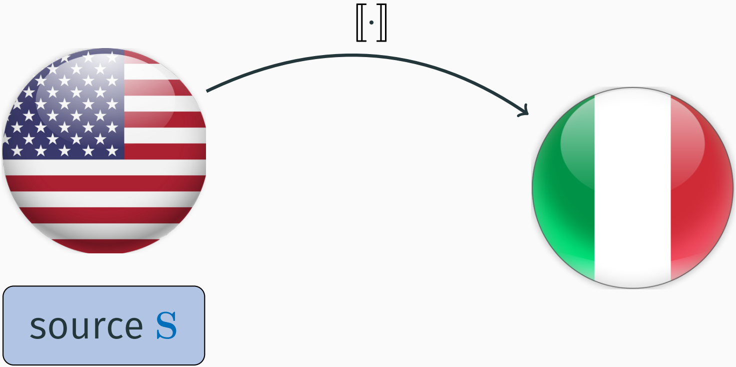
Compilation

Compilation

Compilation

source $\mathbf{S}$

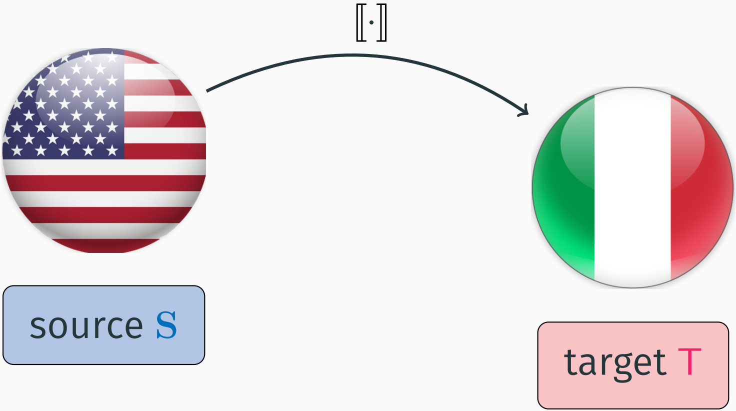
Compilation



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

source $\mathbf{S}$

Compilation

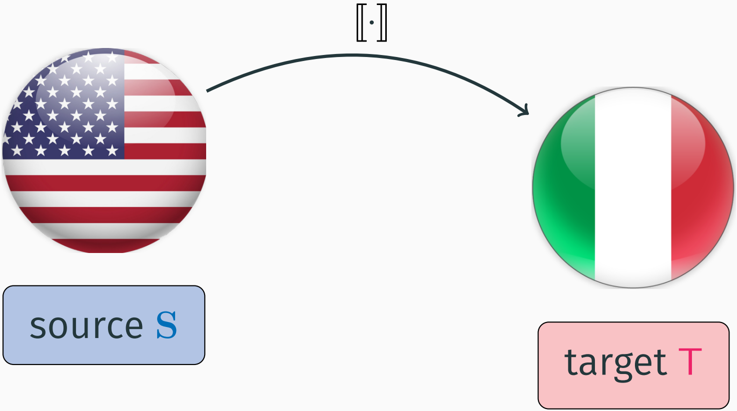


$\llbracket \cdot \rrbracket$

source S

target T

Correct Compilation

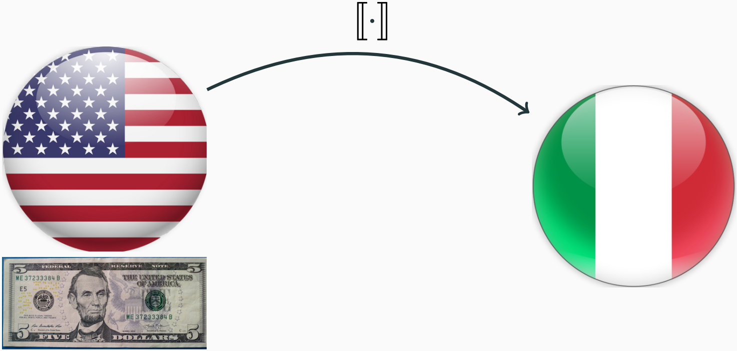


$[\![\cdot]\!]$

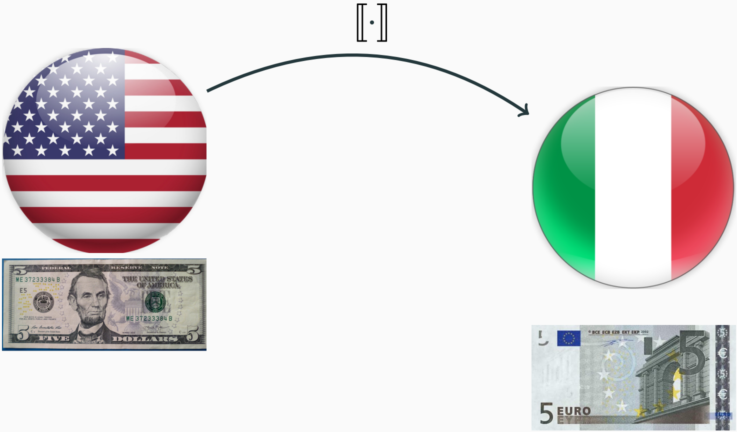
source S

target T

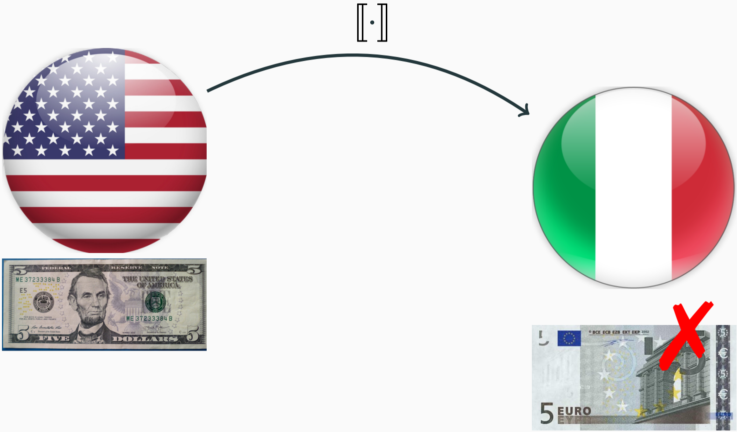
Correct Compilation



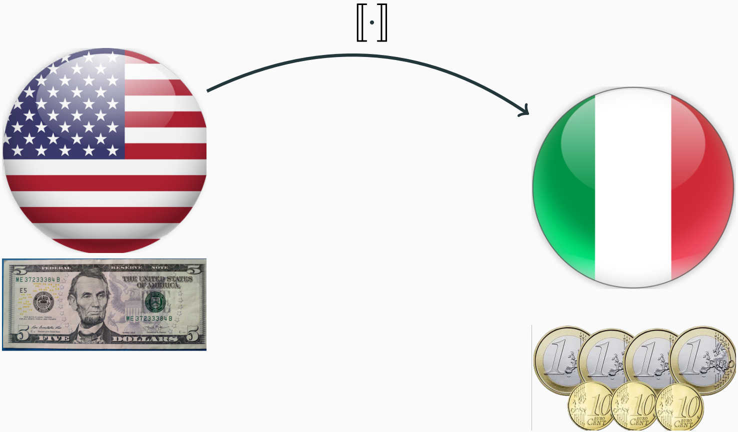
Correct Compilation



Correct Compilation

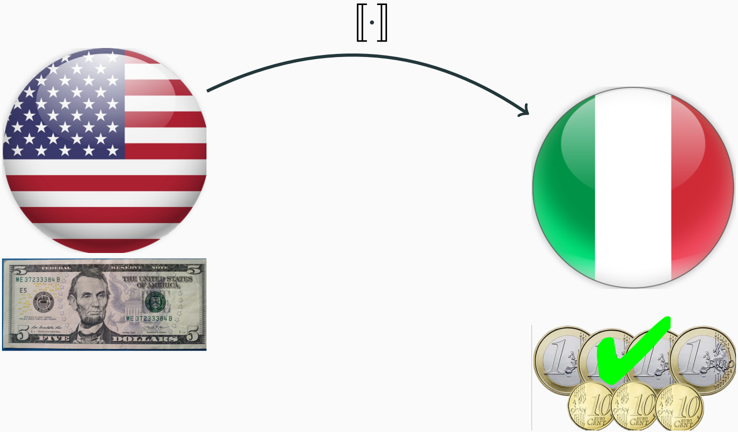


Correct Compilation

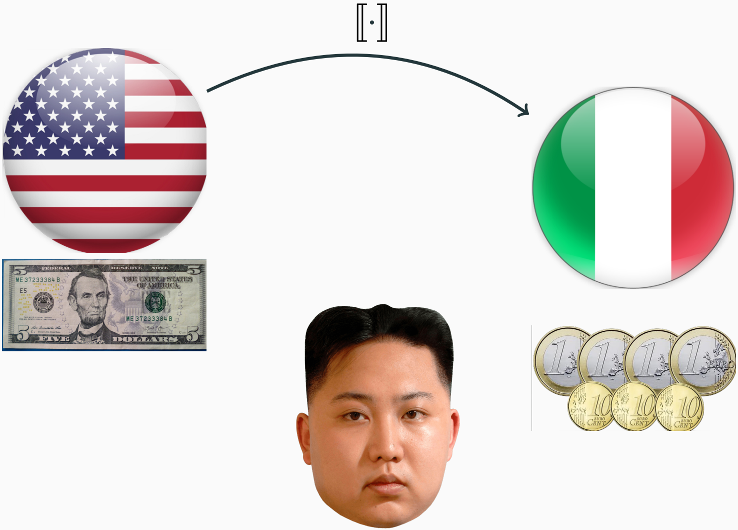


$[\![\cdot]\!]$

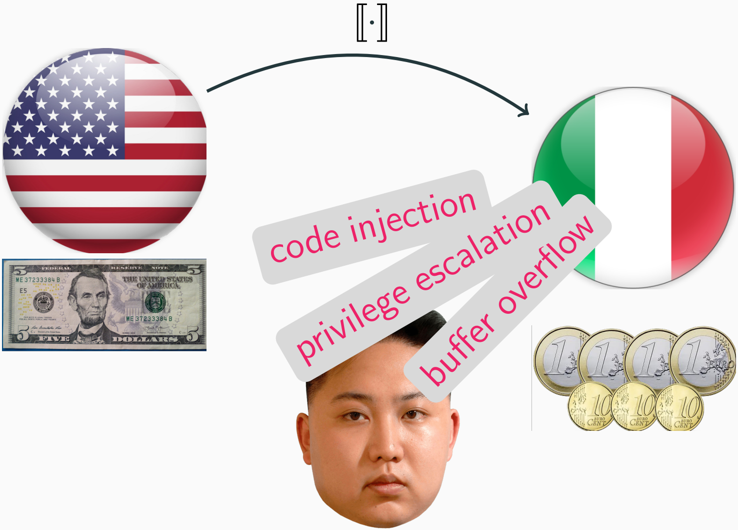
Correct Compilation



Secure Compilation



Secure Compilation



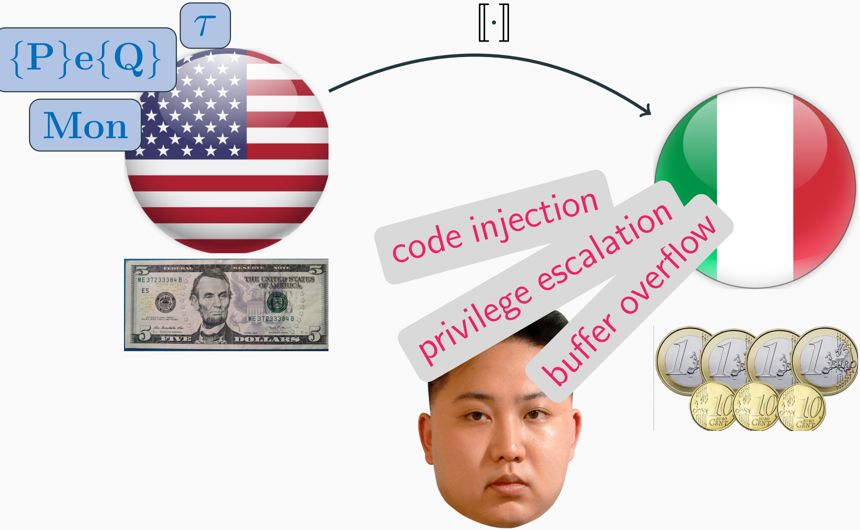
$[\![\cdot]\!]$

code injection

privilege escalation

buffer overflow

Secure Compilation



τ

$\{P\}e\{Q\}$

Mon

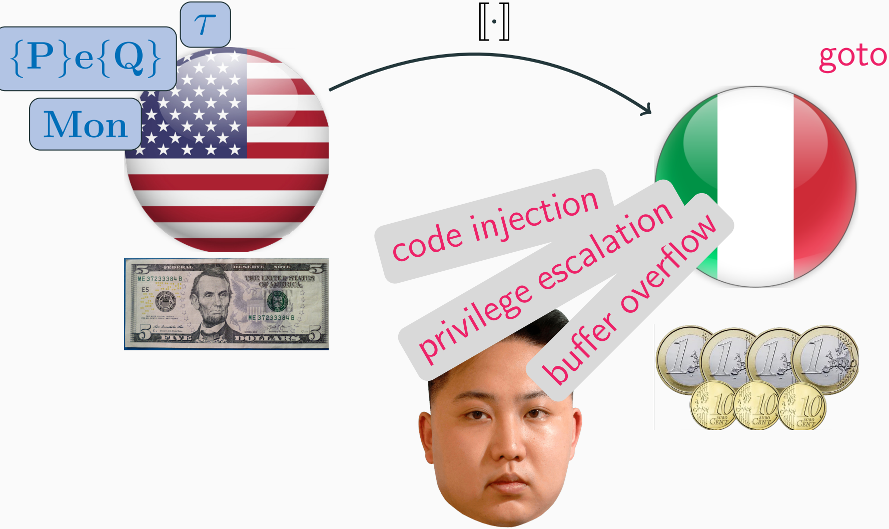
$[\![\cdot]\!]$

code injection

privilege escalation

buffer overflow

Secure Compilation



$\mathcal{T}$

$\{P\}e\{Q\}$

Mon

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

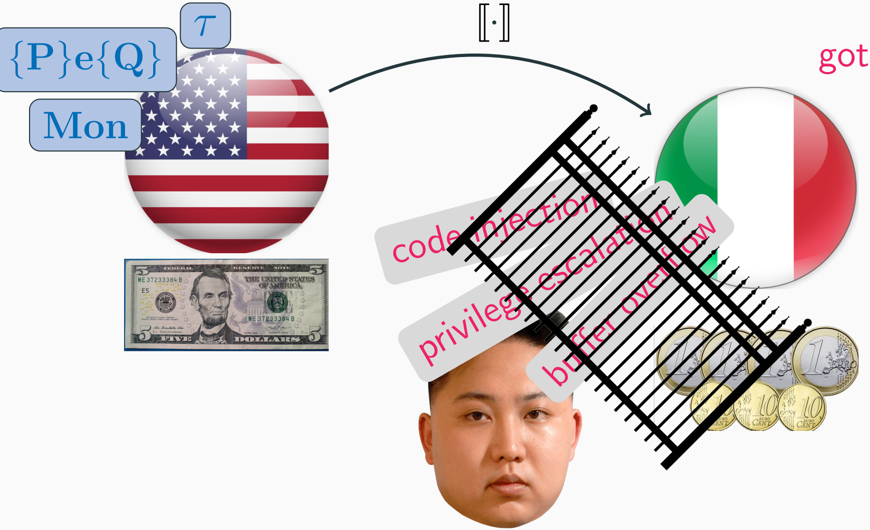
goto

code injection

privilege escalation

buffer overflow

Secure Compilation



τ

$\{P\}e\{Q\}$

Mon

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

goto

code injection

privilege escalation

buffer overflow

Secure Compilation

τ

{P}e{Q}

Mon

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

goto

code injectio

privilege esca

buffe

Enclaves

ASLR

Cap.Mach.

Tag.Mach.

Secure Compilation

- use security architectures to protect code

Secure Compilation

- use security architectures to protect code
- demonstrate that $[\![\cdot]\!]$ attains security

Secure Compilation

- use security architectures to protect code SGX-like enclaves (coming up)
- demonstrate that $[\![\cdot]\!]$ attains security

Secure Compilation

- use security architectures to protect code SGX-like enclaves (coming up)
- demonstrate that $[\![\cdot]\!]$ attains security criteria and proof techniques (later)

Secure Compilation

- use security architectures to protect code SGX-like enclaves (coming up)
- demonstrate that $[\![\cdot]\!]$ attains security criteria and proof techniques (later)

more generally

- build securely, don't fix afterwards

Secure Compilation

- use security architectures to protect code SGX-like enclaves (coming up)
- demonstrate that $[\![\cdot]\!]$ attains security criteria and proof techniques (later)

more generally

- build securely, don’t fix afterwards
- understand what ‘securely’ means

Example of a Secure Compiler

- source = **Java-like language**

Example of a Secure Compiler

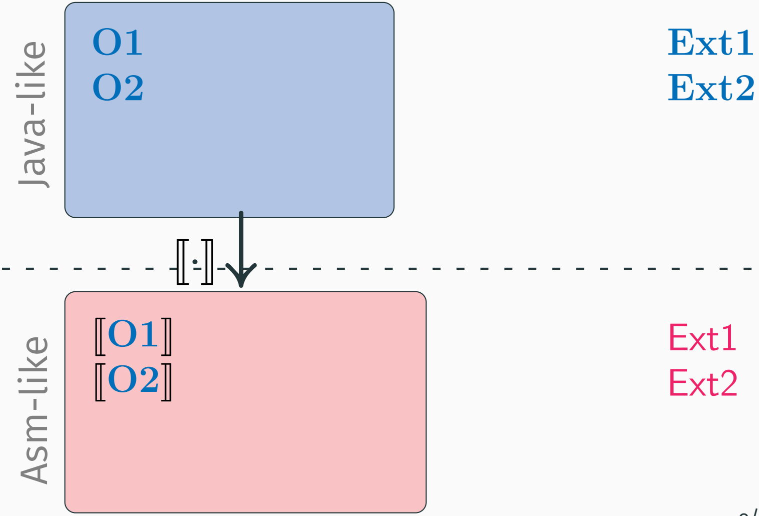
- source = **Java-like language**
- target = Assembly-like + isolation (sgx-likes)

Example of a Secure Compiler

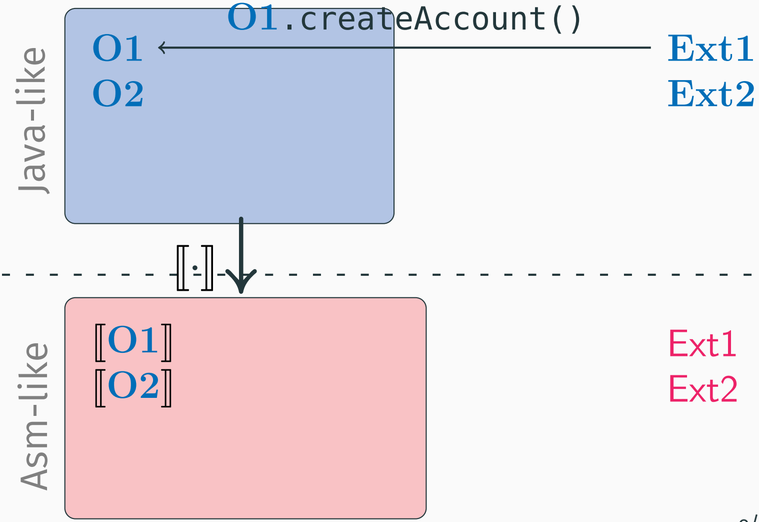
- source = **Java-like language**
- target = Assembly-like + isolation (sgx-likes)
- based on Agten *et al.*'12, Patrignani *et al.*'15'16

Warning fairly high level

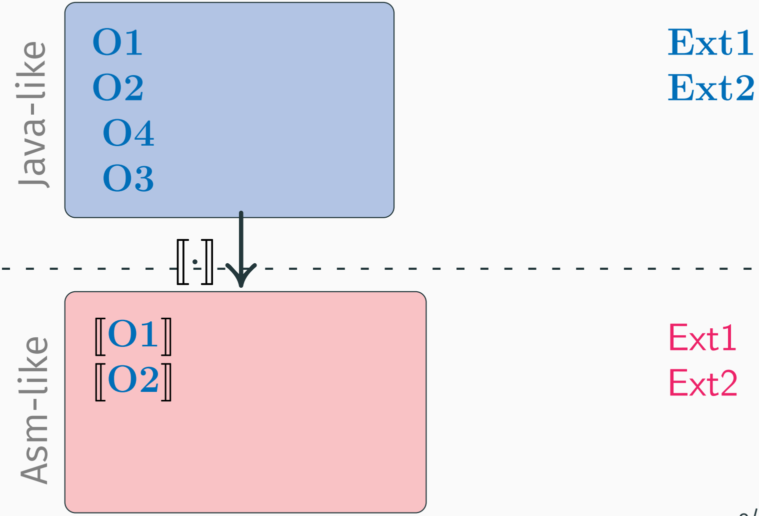
Memory Allocation Issues



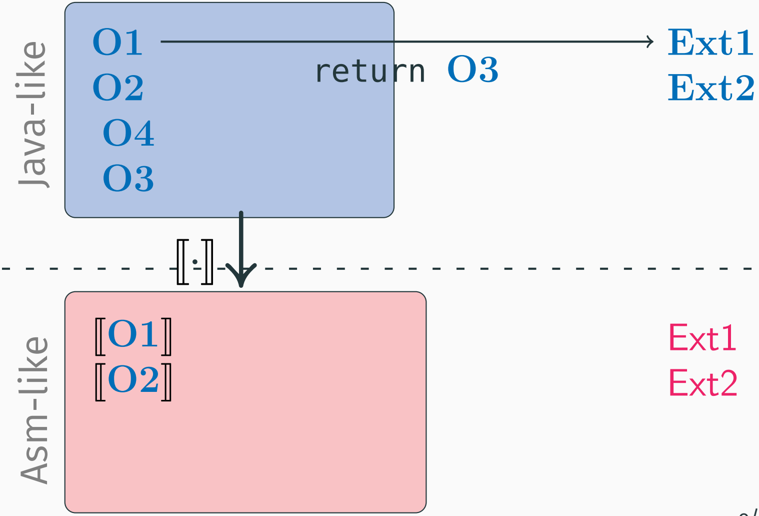
Memory Allocation Issues



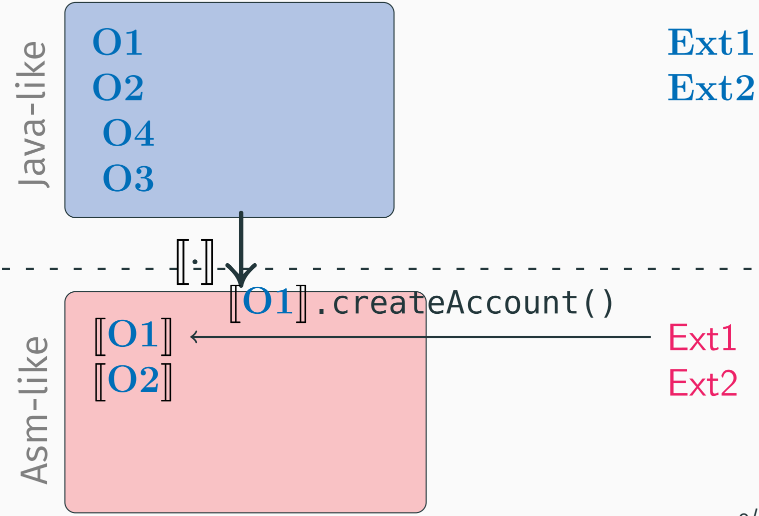
Memory Allocation Issues



Memory Allocation Issues



Memory Allocation Issues



Java-like

O1

O2

O4

O3

Ext1

Ext2

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

$[\![\mathbf{O1}]\!]$`.createAccount()`

Asm-like

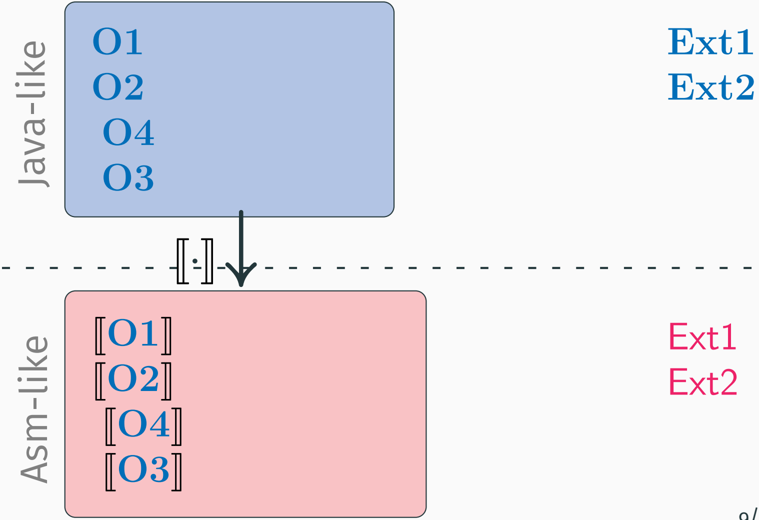
$[\![\mathbf{O1}]\!]$

$[\![\mathbf{O2}]\!]$

Ext1

Ext2

Memory Allocation Issues



Java-like

O1

O2

O4

O3

Ext1

Ext2

$\llbracket \cdot \rrbracket$

Asm-like

$\llbracket \mathbf{O1} \rrbracket$

$\llbracket \mathbf{O2} \rrbracket$

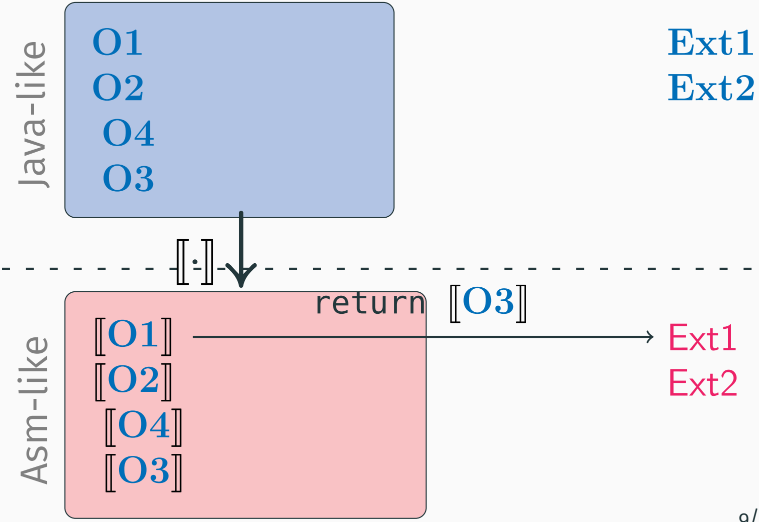
$\llbracket \mathbf{O4} \rrbracket$

$\llbracket \mathbf{O3} \rrbracket$

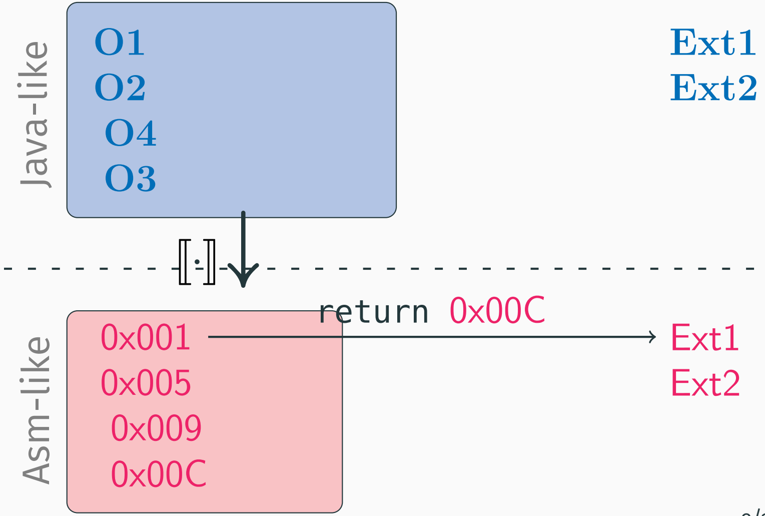
Ext1

Ext2

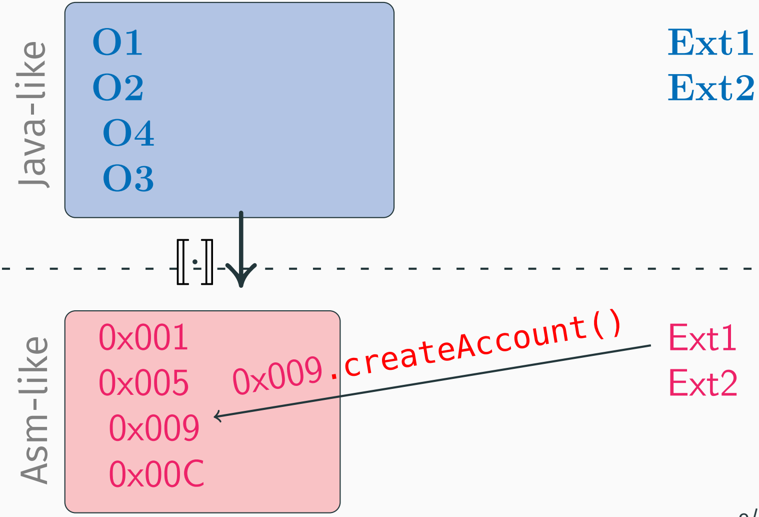
Memory Allocation Issues



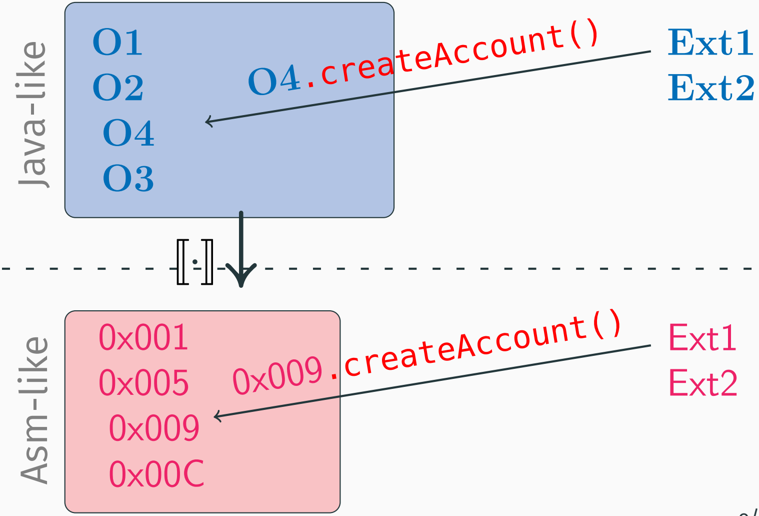
Memory Allocation Issues



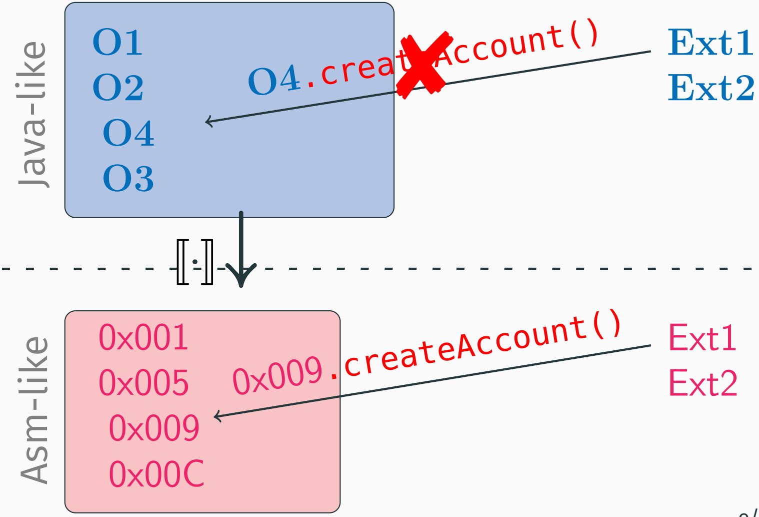
Memory Allocation Issues



Memory Allocation Issues



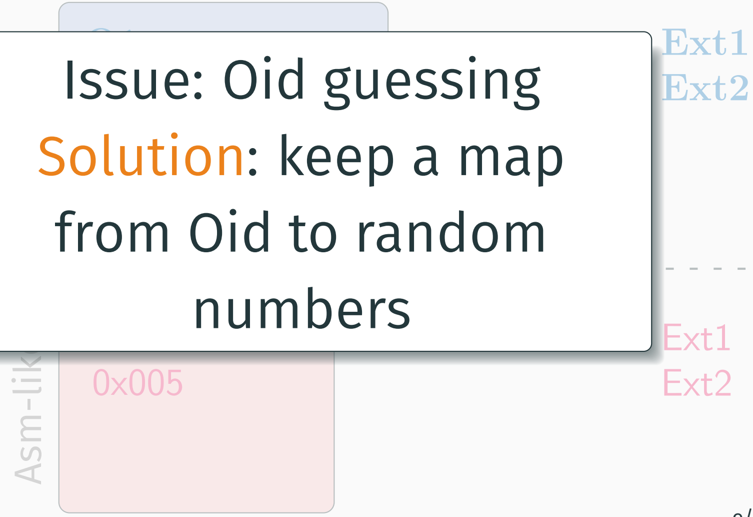
Memory Allocation Issues



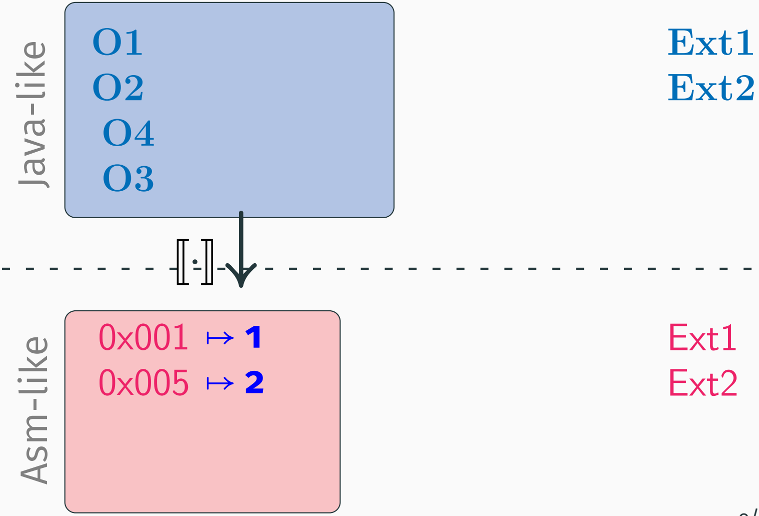
Memory Allocation Issues

Issue: Oid guessing

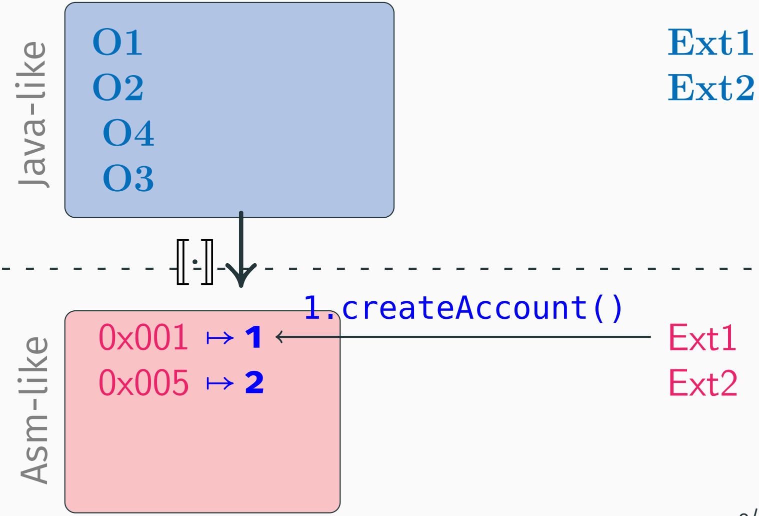
Solution: keep a map from Oid to random numbers



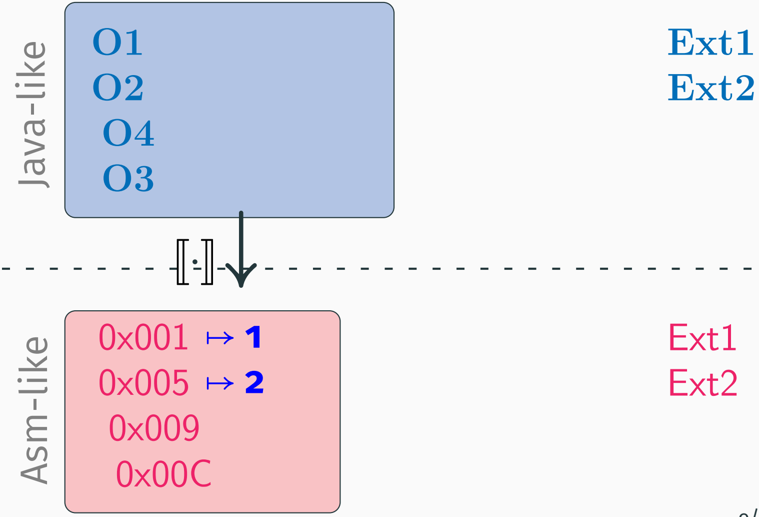
Memory Allocation Issues



Memory Allocation Issues



Memory Allocation Issues



Java-like

O1
O2
O4
O3

Ext1
Ext2

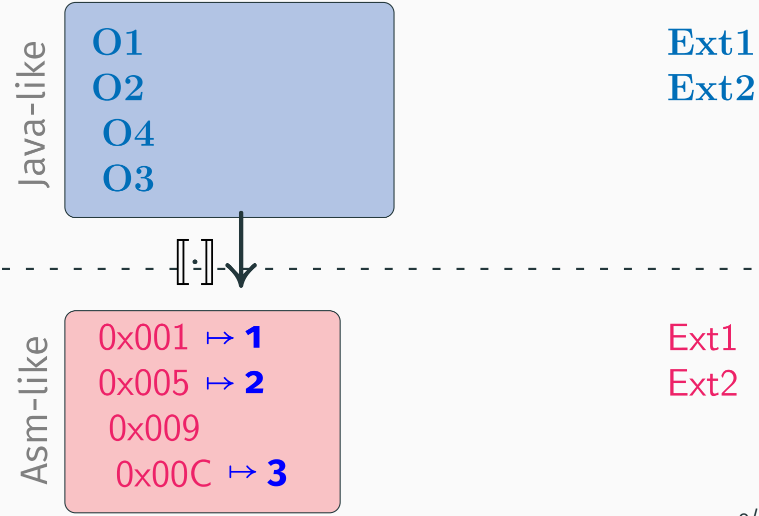
$[\![\cdot]\!]$

Asm-like

0x001 $\mapsto$ **1**
0x005 $\mapsto$ **2**
0x009
0x00C

Ext1
Ext2

Memory Allocation Issues



Java-like

O1

O2

O4

O3

Ext1

Ext2

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

Asm-like

0x001 $\mapsto$ **1**

0x005 $\mapsto$ **2**

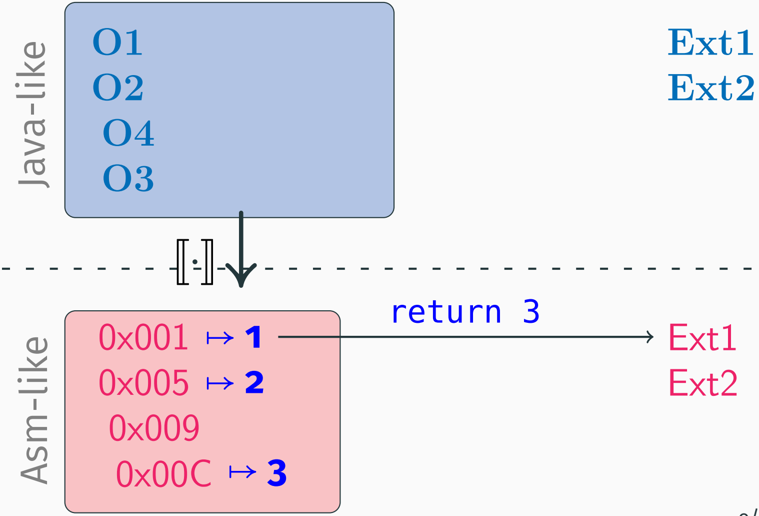
0x009

0x00C $\mapsto$ **3**

Ext1

Ext2

Memory Allocation Issues



Java-like

O1

O2

O4

O3

Ext1

Ext2

⟦·⟧

Asm-like

0x001 $\mapsto$ **1**

0x005 $\mapsto$ **2**

0x009

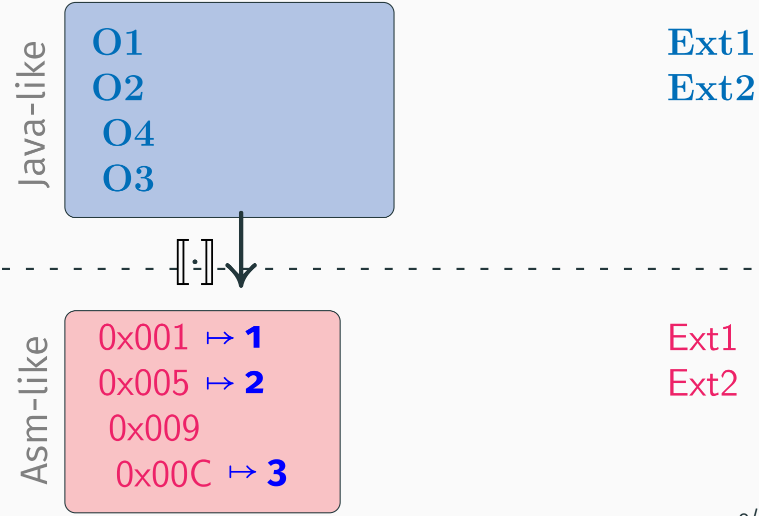
0x00C $\mapsto$ **3**

`return 3`

Ext1

Ext2

Memory Allocation Issues



Java-like

O1

O2

O4

O3

Ext1

Ext2

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

Asm-like

0x001 $\mapsto$ **1**

0x005 $\mapsto$ **2**

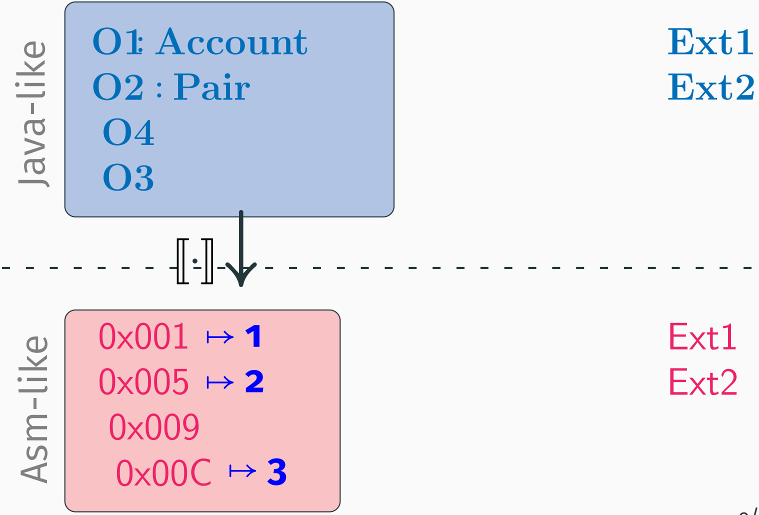
0x009

0x00C $\mapsto$ **3**

Ext1

Ext2

Memory Allocation Issues



Java-like

O1 : Account

O2 : Pair

O4

O3

Ext1

Ext2

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

Asm-like

0x001 ↦ **1**

0x005 ↦ **2**

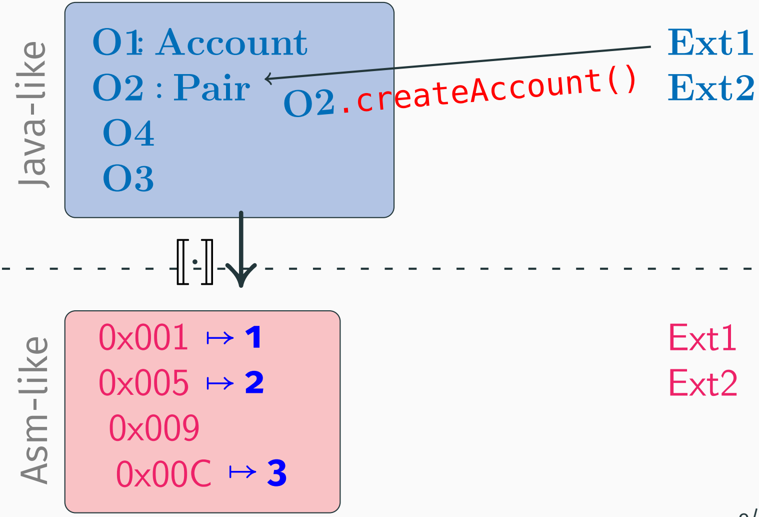
0x009

0x00C ↦ **3**

Ext1

Ext2

Memory Allocation Issues



Java-like

O1: Account

O2 : Pair

O4

O3

O2.`createAccount()`

Ext1

Ext2

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

Asm-like

0x001 $\mapsto$ **1**

0x005 $\mapsto$ **2**

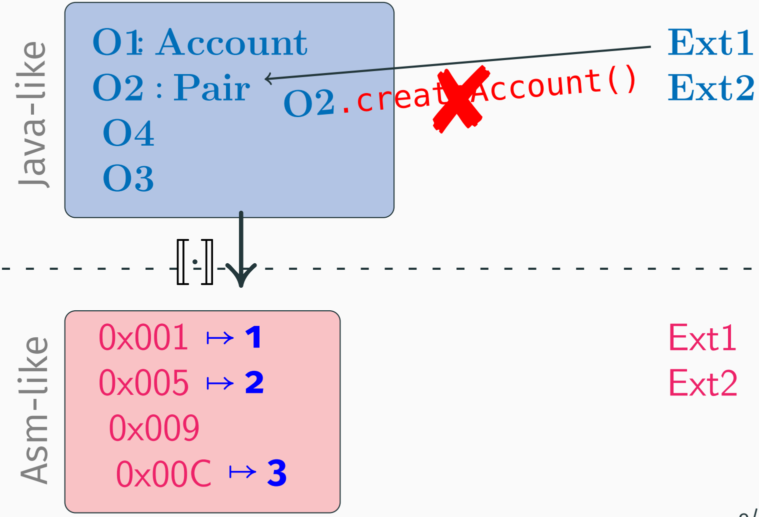
0x009

0x00C $\mapsto$ **3**

Ext1

Ext2

Memory Allocation Issues



Java-like

O1: Account

O2 : Pair

O4

O3

O2.creat✘Account()

Ext1

Ext2

$\llbracket \cdot \rrbracket$

Asm-like

0x001 ↦ **1**

0x005 ↦ **2**

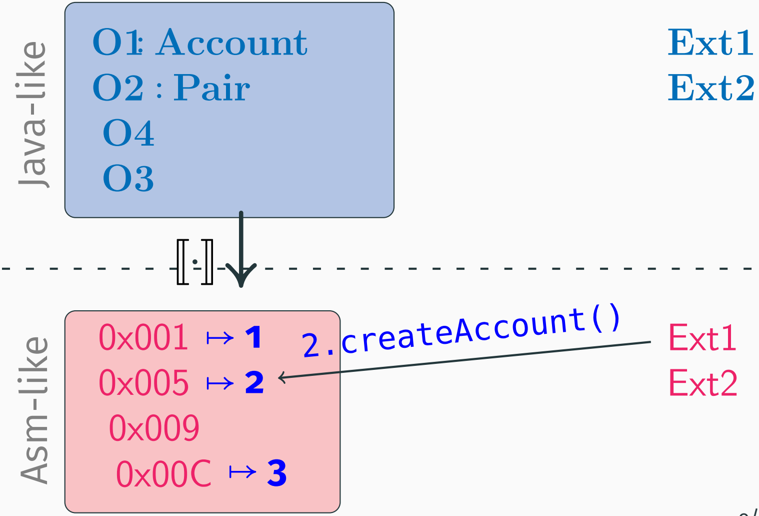
0x009

0x00C ↦ **3**

Ext1

Ext2

Memory Allocation Issues



Java-like

O1: Account

O2 : Pair

O4

O3

Ext1

Ext2

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

Asm-like

0x001 $\mapsto$ **1**

0x005 $\mapsto$ **2**

0x009

0x00C $\mapsto$ **3**

2.createAccount()

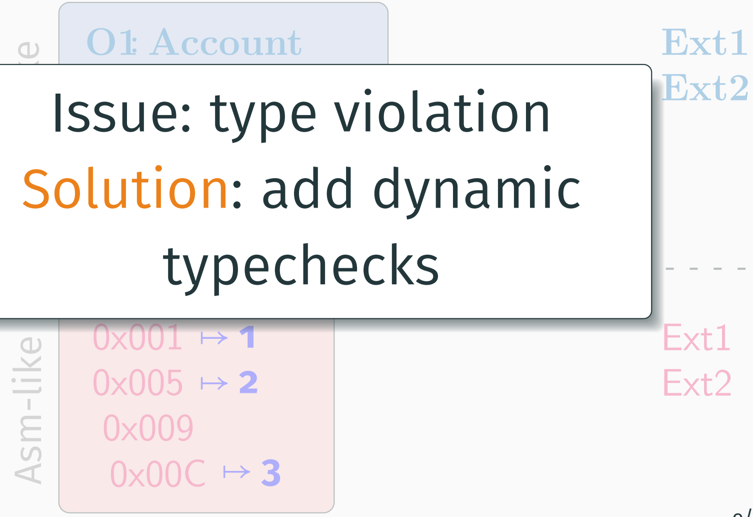
Ext1

Ext2

Memory Allocation Issues

Issue: type violation

Solution: add dynamic typechecks



O1: Account

Ext1

Ext2

Asm-like

0x001 ↦ 1

0x005 ↦ 2

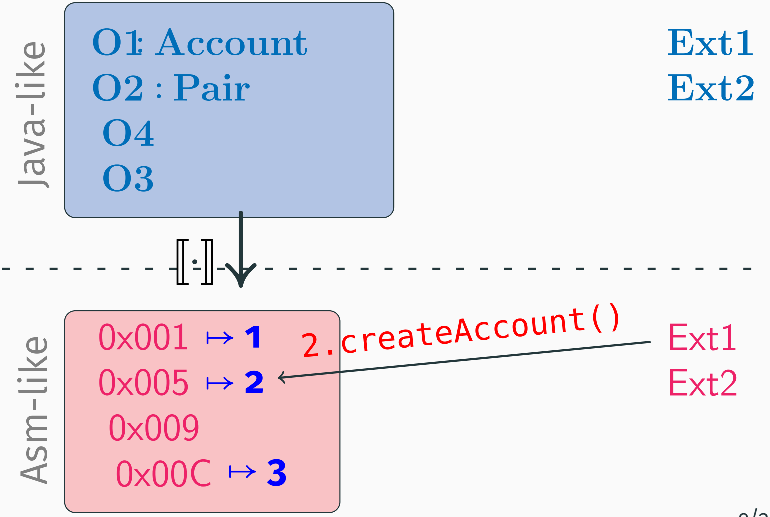
0x009

0x00C ↦ 3

Ext1

Ext2

Memory Allocation Issues



Java-like

O1: Account

O2 : Pair

O4

O3

Ext1

Ext2

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

Asm-like

0x001 $\mapsto$ **1**

0x005 $\mapsto$ **2**

0x009

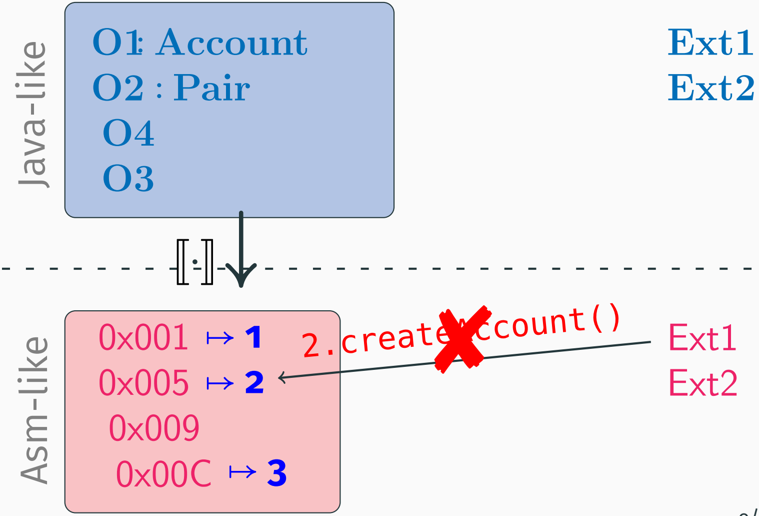
0x00C $\mapsto$ **3**

`2.createAccount()`

Ext1

Ext2

Memory Allocation Issues



Java-like

O1: Account

O2 : Pair

O4

O3

Ext1

Ext2

$⟦\cdot⟧$

Asm-like

0x001 ↦ **1**

0x005 ↦ **2**

0x009

0x00C ↦ **3**

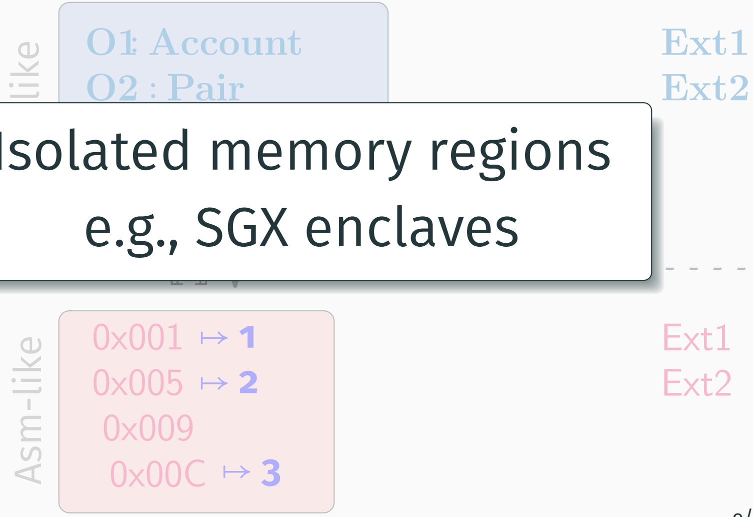
`2.createAccount()`

Ext1

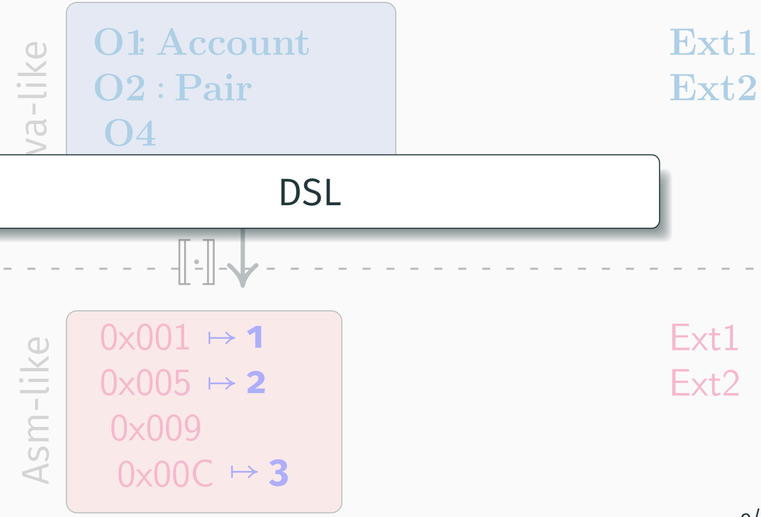
Ext2

Memory Allocation Issues

Isolated memory regions
e.g., SGX enclaves



Memory Allocation Issues



va-like

O1: Account

O2 : Pair

O4

Ext1

Ext2

DSL

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

Asm-like

0x001 ↦ **1**

0x005 ↦ **2**

0x009

0x00C ↦ **3**

Ext1

Ext2

Memory Allocation Issues

- design
- implement
- fund a startup in the Bay Area
-

Ext1
Ext2

Ext1
Ext2

Asm-

0x009
0x00C ↦ **3**

Concerns

1. Is this actually secure?

Concerns

1. Is this actually secure?
2. How efficient is this?

Concerns

1. Is this actually secure?
2. How efficient is this?

1. Yes!

Concerns

1. Is this actually secure?
2. How efficient is this?

1. Yes! So prove it!

Concerns

1. Is this actually secure?
2. How efficient is this?

1. Yes! So prove it!
2. Not bad, but we can aim for better.

Concerns

We need criteria for secure compilation, they:

- tell us what to prove about the compiler (e.g., compiler correctness, or type soundness criteria)

Concerns

1

2

1

2

We need criteria for secure compilation, they:

- tell us what to prove about the compiler (e.g., compiler correctness, or type soundness criteria)
- impact efficiency

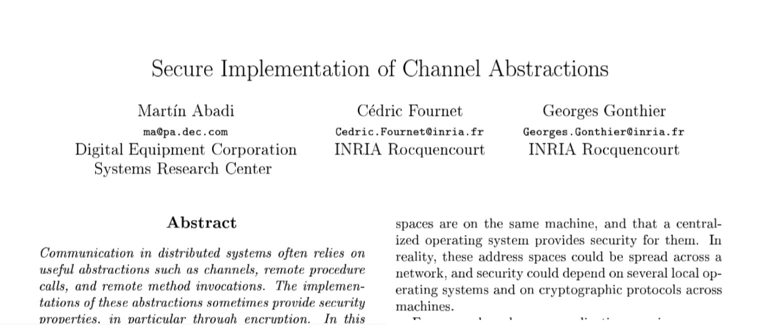
Concerns

We need criteria for secure compilation, they:

- tell us what to prove about the compiler (e.g., compiler correctness, or type soundness criteria)
- impact efficiency
- define security guarantees (what security properties they preserve)

Secure Compilation Criteria

The Origins of the Secure Compiler



Secure Implementation of Channel Abstractions

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

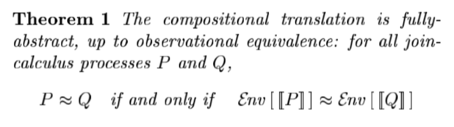
Cédric Fournet
Cedric.Fournet@inria.fr
INRIA Rocquencourt

Georges Gonthier
Georges.Gonthier@inria.fr
INRIA Rocquencourt

Abstract

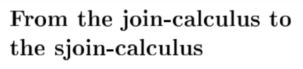
Communication in distributed systems often relies on useful abstractions such as channels, remote procedure calls, and remote method invocations. The implementations of these abstractions sometimes provide security properties, in particular through encryption. In this

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.



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

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



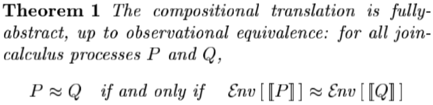
From the join-calculus to the sjoin-calculus

The Origins of the Secure Compiler

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

The Origins of the Secure Compiler

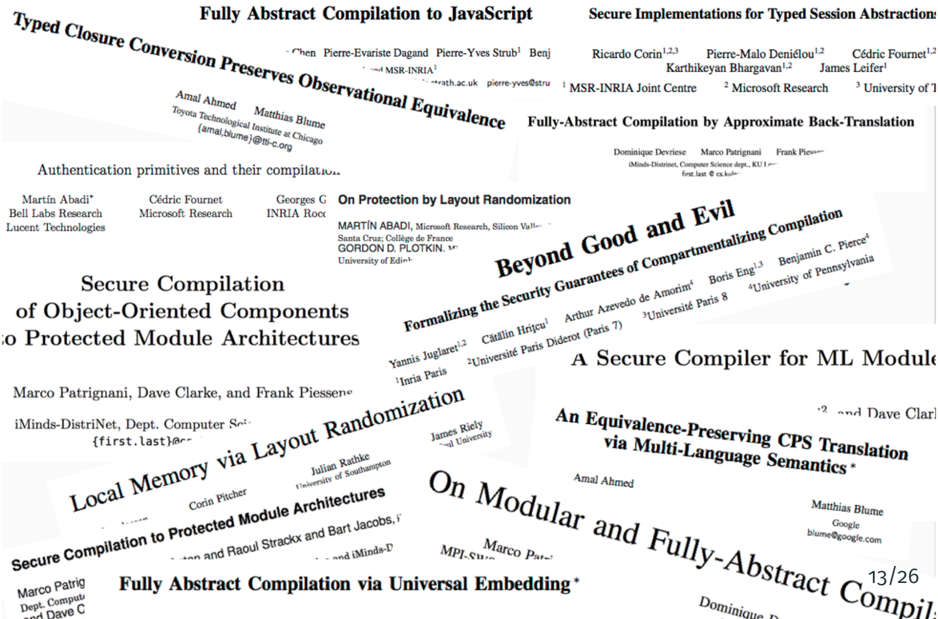
Fully Abstract Compilation (FAC)



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

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

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

Secure Implementations for Typed Session Abstractions

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

Fully-Abstract Compilation by Approximate Back-Translation

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

Authentication primitives and their compilation

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

On Protection by Layout Randomization

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

Beyond Good and Evil

Formalizing the Security Guarantees of Compartmentalizing Compilation

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

Secure Compilation
of Object-Oriented Components
to Protected Module Architectures

Marco Patrignani, Dave Clarke, and Frank Piessens
iMinds-DistriNet, Dept. Computer Sci
{first.last}@c

A Secure Compiler for ML Module

2 and Dave Clar

Local Memory via Layout Randomization

Corin Pitcher Julian Rathke James Riely
University of Southampton ul University

Secure Compilation to Protected Module Architectures

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

Marco Patrig
Dept. Comput
and Dave C

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

Amal Ahmed Matthias Blume
Google
blume@google.com

On Modular and Fully-Abstract Compil

Marco Patr
MPI-SWS

Dominique D

Fully Abstract Compilation via Universal Embedding *

Fully Abstract Compilation Influence

Typed Closure Conversion Preserv

Fully Abstract Compilation to JavaScript

Chen Pierre-Evariste Dagand Pierre-Yves Strub[1] Benj

MSR-INRIA[1]

Secure Implementations for Typed Session Abstractions

Ricardo Corin[1,2,3] Pierre-Malo Deniélou[1,2] Cédric Fournet[1,2]

Karthikeyan Bhargavan[1,2] James Leifer[1]

[3] University of T

-Translation

Authentication

Martín Abadi*
Bell Labs Research
Lucent Technologies

Pierce[4]

sylvania

Secure Compilation
of Object-Oriented Components
o Protected Module Architectures

Marco Patrignani, Dave Clarke, and Frank Piessens

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

Formalizing the Security Guarantees

Yannis Juglaret[1,2] Cătălin Hriţcu[1] Arthur Azevedo de Amorim[4] Boris

[1]Inria Paris [2]Université Paris Diderot (Paris 7) [3]Université Paris 8 [4]University of

A Secure Compiler for ML Module

and Dave Clar

Local Memory via Layout Randomization

Corin Pitcher Julian Rathke
University of Southampton

James Riely
Paul University

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

Amal Ahmed

Matthias Blume
Google
blume@google.com

Secure Compilation to Protected Module Architectures

and Raoul Strackx and Bart Jacobs,

and iMinds-D

On Modular and Fully-Abstract Compil

Marco Patr

MPI-SW

Marco Patrig
Dept. Comput
and Dave C

Fully Abstract Compilation via Universal Embedding *

Dominique D

How

does Fully Abstract Compilation entail

security?

FAC and Security

FAC ensures that a target-level attacker has the same power of a source-level one

Compiler Full Abstraction

$$\begin{array}{ll} \mathbf{x} = \mathbf{1}; & \mathbf{x} = \mathbf{0}; \\ \mathbf{x}{+}{+}; & \mathbf{x}{+}{=}\ \mathbf{2}; \\ \mathbf{x} & \mathbf{x} \end{array}$$

Compiler Full Abstraction

$$\begin{array}{l} \mathbf{x} = \mathbf{1}; \\ \mathbf{x}\ ++; \\ \mathbf{x} \end{array} \quad = \quad \begin{array}{l} \mathbf{x} = \mathbf{0}; \\ \mathbf{x} += \mathbf{2}; \\ \mathbf{x} \end{array}$$

Compiler Full Abstraction

$$\begin{array}{lcl} \mathbf{x} = \mathbf{1}; & & \mathbf{x} = \mathbf{0}; \\ \mathbf{x}{+}{+}; & = & \mathbf{x}{+}{=}\ \mathbf{2}; \\ \mathbf{x} & & \mathbf{x} \end{array}$$

$$\Updownarrow$$

$$\begin{array}{ll} \mathsf{loadi}\ \mathsf{r}_0\ 1 & \mathsf{loadi}\ \mathsf{r}_0\ 0 \\ \mathsf{inc}\ \mathsf{r}_0 & \mathsf{addi}\ \mathsf{r}_0\ 2 \\ \mathsf{ret}\ \mathsf{r}_0 & \mathsf{ret}\ \mathsf{r}_0 \end{array}$$

Compiler Full Abstraction

$$\begin{array}{lcl} \mathbf{x} = \mathbf{1}; & & \mathbf{x} = \mathbf{0}; \\ \mathbf{x}\ {+}{+}; & = & \mathbf{x} {+}{=}\ \mathbf{2}; \\ \mathbf{x} & & \mathbf{x} \end{array}$$

$$\Updownarrow$$

$$\begin{array}{lcl} \mathsf{loadi}\ \mathsf{r}_0\ 1 & & \mathsf{loadi}\ \mathsf{r}_0\ 0 \\ \mathsf{inc}\ \mathsf{r}_0 & = & \mathsf{addi}\ \mathsf{r}_0\ 2 \\ \mathsf{ret}\ \mathsf{r}_0 & & \mathsf{ret}\ \mathsf{r}_0 \end{array}$$

Compiler Full Abstraction

$x = 1;$ $\qquad\qquad$ $x = 0;$

$x++;$ $\qquad = \qquad$ $x += 2;$

and have different powers!

inc r_0 $\qquad = \qquad$ addi r_0 2

ret r_0 $\qquad\qquad$ ret r_0

Why is FAC Secure?

- is an attacker linking or injecting target code

Why is FAC Secure?

- is an attacker linking or injecting target code

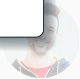
Why is FAC Secure?

- is an attacker linking or injecting target code
- is not constrained by **source** constructs

Why is FAC Secure?

- is an attacker linking or injecting target code
- is not constrained by **source** constructs
- the co-implied equalities reduce to

Why is FAC Secure?

- is an attacker linking or injecting target code
- ts
- the co-implied equalities reduce to 

- FAC protects against target attacks

Why is FAC Secure?

1. confidentiality
2. integrity
3. invariant definition
4. memory allocation
5. well-bracketed control flow

Survey by Patrignani *et al.*'19, based on Agten *et al.*'12, Abadi and Plotkin '10, Jagadeesan *et al.*'11, Patrignani *et al.*'15'16

Why is

confidentiality:

$$\mathbf{P1} = \mathbf{P2} \iff [\![\mathbf{P1}]\!] = [\![\mathbf{P2}]\!]$$

- $\mathbf{P1}$ and $\mathbf{P2}$ have different secrets
- but they are equivalent

1. c
2. i
3. i
4. r
5. w

Survey by Patrignani *et al.*’19, based on Agten *et al.*’12, Abadi and Plotkin ’10, Jagadeesan *et al.*’11, Patrignani *et al.*’15’16

Why i

1. c
2. i
3. i
4. r
5. w

confidentiality:

$$\mathbf{P1} = \mathbf{P2} \iff [\![\mathbf{P1}]\!] = [\![\mathbf{P2}]\!]$$

- $\mathbf{P1}$ and $\mathbf{P2}$ have different secrets
- but they are equivalent
- $[\![\mathbf{P1}]\!]$ and $[\![\mathbf{P2}]\!]$ also have different secrets
- but they are equivalent

Survey by Patrignani *et al.*'19, based on Agten *et al.*'12, Abadi and Plotkin '10, Jagadeesan *et al.*'11, Patrignani *et al.*'15'16

Why is

1. c
2. i
3. i
4.
5. w

confidentiality:

$$\mathbf{P1} = \mathbf{P2} \iff \llbracket \mathbf{P1} \rrbracket = \llbracket \mathbf{P2} \rrbracket$$

- $\mathbf{P1}$ and $\mathbf{P2}$ have different secrets
- but they are equivalent
- $\llbracket \mathbf{P1} \rrbracket$ and $\llbracket \mathbf{P2} \rrbracket$ also have different secrets
- but they are equivalent
- so the secret does not leak

Survey by Patrignani *et al.*'19, based on Agten *et al.*'12, Abadi and Plotkin '10, Jagadeesan *et al.*'11, Patrignani *et al.*'15'16

Why is FAC Secure?

1. confidentiality
2. integrity
3. invariant definition
4. memory allocation
5. well-bracketed control flow

If the source has it.

Survey by Patrignani *et al.*'19, based on Agten *et al.*'12, Abadi and Plotkin '10, Jagadeesan *et al.*'11, Patrignani *et al.*'15'16

Why is FAC Secure?

1. confidentiality
2. integrity
3. i
4. memory allocation
5. well-bracketed control flow

- FAC preserves these properties

If the source has it.

Survey by Patrignani *et al.*’19, based on Agten *et al.*’12, Abadi and Plotkin ’10, Jagadeesan *et al.*’11, Patrignani *et al.*’15’16

Not All That Glitters is Gold

- No support for separate compilation

  [Patrignani *et al.*'16, Juglaret *et al.*'16]

Not All That Glitters is Gold

- No support for separate compilation

  [Patrignani *et al.*'16, Juglaret *et al.*'16]

Not All That Glitters is Gold

- No support for separate compilation [Patrignani *et al.*'16, Juglaret *et al.*'16]
- No support for undefined behaviour [Juglaret *et al.*'16]

Not All That Glitters is Gold

- No support for separate compilation [Patrignani *et al.*'16, Juglaret *et al.*'16]
- No support for undefined behaviour [Juglaret *et al.*'16]
- Costly to enforce [Patrignani and Garg '19]

Not All That Glitters is Gold

- No support for separate compilation [Patrignani *et al.*'16, Juglaret *et al.*'16]
- No support for undefined behaviour [Juglaret *et al.*'16]
- Costly to enforce [Patrignani and Garg '19]
- Preserves hypersafety under certain conditions [Patrignani and Garg '17]

Alternatives

- FAC is not precise about security

Alternatives

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

Alternatives

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

Alternatives

- 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

Preserving Safety Patrignani and Garg ’19, Abate *et al.*’18

- we have a **source program** with a safety property

Preserving Safety Patrignani and Garg ’19, Abate *et al.*’18

- we have a **source program** with a safety property against any **source attacker**

Preserving Safety Patrignani and Garg ’19, Abate *et al.*’18

- we have a **source program** with a safety property against any **source attacker**
- safety = integrity / weak secrecy / correctness

Preserving Safety Patrignani and Garg ’19, Abate *et al.*’18

- we have a **source program** with a safety property against any **source attacker**
- safety = integrity / weak secrecy / correctness
- we want its compiled counterpart to have the same safety property

Preserving Safety Patrignani and Garg ’19, Abate *et al.*’18

- we have a **source program** with a safety property against any **source attacker**
- safety = integrity / weak secrecy / correctness
- we want its compiled counterpart to have the same safety property against any target attacker

Preserving Safety Patrignani and Garg '19, Abate *et al.*'18

- property (π) = set of traces $\{t_1, t_2, \cdots\}$
- traces (t) = infinite sequences of observables
- prefixes (m) = finite sequences of observables
- $P \rightsquigarrow t$ = program P generates trace t

Preserving Safety Patrignani and Garg '19, Abate *et al.*'18

$$
\begin{aligned}
RSC: \quad & \forall \pi, \pi \in \mathit{Safety}.\ \pi \approx \pi.\ \forall \mathbf{P}. \\
& (\forall \mathbb{C}_{\mathbf{S}}, \mathbf{t}.\ \mathbb{C}_{\mathbf{S}}\left[\mathbf{P}\right] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \pi) \\
& \ \Rightarrow (\forall \mathbb{C}_{\mathsf{T}}, \mathsf{t}.\ \mathbb{C}_{\mathsf{T}}\left[[\![\mathbf{P}]\!]\right] \rightsquigarrow \mathsf{t} \Rightarrow \mathsf{t} \in \pi)
\end{aligned}
$$

Preserving Safety Patrignani and Garg '19, Abate *et al.*'18

$$\begin{aligned}
\mathit{RSC}: \quad & \forall \pi, \pi \in \mathit{Safety}.\ \pi \approx \pi.\ \forall \mathbf{P}. \\
& (\forall \mathbb{C}_{\mathbf{S}}, \mathbf{t}.\ \mathbb{C}_{\mathbf{S}}\left[\mathbf{P}\right] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \pi) \\
& \ \Rightarrow (\forall \mathbb{C}_{\mathsf{T}}, \mathsf{t}.\ \mathbb{C}_{\mathsf{T}}\left[\llbracket \mathbf{P} \rrbracket\right] \rightsquigarrow \mathsf{t} \Rightarrow \mathsf{t} \in \pi)
\end{aligned}$$

$$\begin{aligned}
\mathit{PF}\text{-}\mathit{RSC}: \quad & \forall \mathbf{P}.\ \forall \mathbb{C}_{\mathsf{T}}.\ \forall \mathbf{m}, \mathsf{m}.\ \mathbf{m} \approx \mathsf{m}. \\
& \mathbb{C}_{\mathsf{T}}\left[\llbracket \mathbf{P} \rrbracket\right] \rightsquigarrow \mathsf{m} \Rightarrow \exists \mathbb{C}_{\mathbf{S}}.\ \mathbb{C}_{\mathbf{S}}\left[\mathbf{P}\right] \rightsquigarrow \mathbf{m}
\end{aligned}$$

Preserving Safety Patrignani and Garg '19, Abate *et al.*'18

$$\begin{aligned}
\mathit{RSC}: \quad & \forall \pi, \pi \in \mathit{Safety}.\ \pi \approx \pi.\ \forall \mathbf{P}. \\
& (\forall \mathbb{C}_{\mathbf{S}}, \mathbf{t}.\ \mathbb{C}_{\mathbf{S}}\left[\mathbf{P}\right] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \pi) \\
& \ \Rightarrow (\forall \mathbb{C}_{\mathsf{T}}, \mathsf{t}.\ \mathbb{C}_{\mathsf{T}}\left[\llbracket \mathbf{P} \rrbracket\right] \rightsquigarrow \mathsf{t} \Rightarrow \mathsf{t} \in \pi)
\end{aligned}$$

$$\Updownarrow$$

$$\begin{aligned}
\mathit{PF}\text{-}\mathit{RSC}: \quad & \forall \mathbf{P}.\ \forall \mathbb{C}_{\mathsf{T}}.\ \forall \mathbf{m}, \mathsf{m}.\ \mathbf{m} \approx \mathsf{m}. \\
& \mathbb{C}_{\mathsf{T}}\left[\llbracket \mathbf{P} \rrbracket\right] \rightsquigarrow \mathsf{m} \Rightarrow \exists \mathbb{C}_{\mathbf{S}}.\ \mathbb{C}_{\mathbf{S}}\left[\mathbf{P}\right] \rightsquigarrow \mathbf{m}
\end{aligned}$$

Not All That Glitters is Gold #2

- RSC leads to more efficient compiled code

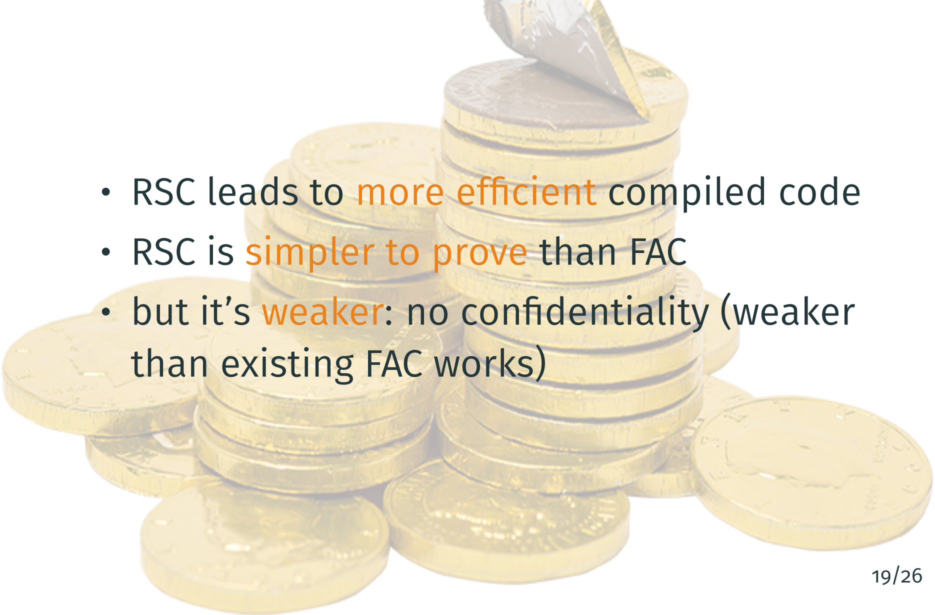
Not All That Glitters is Gold #2

- RSC leads to more efficient compiled code
- RSC is simpler to prove than FAC

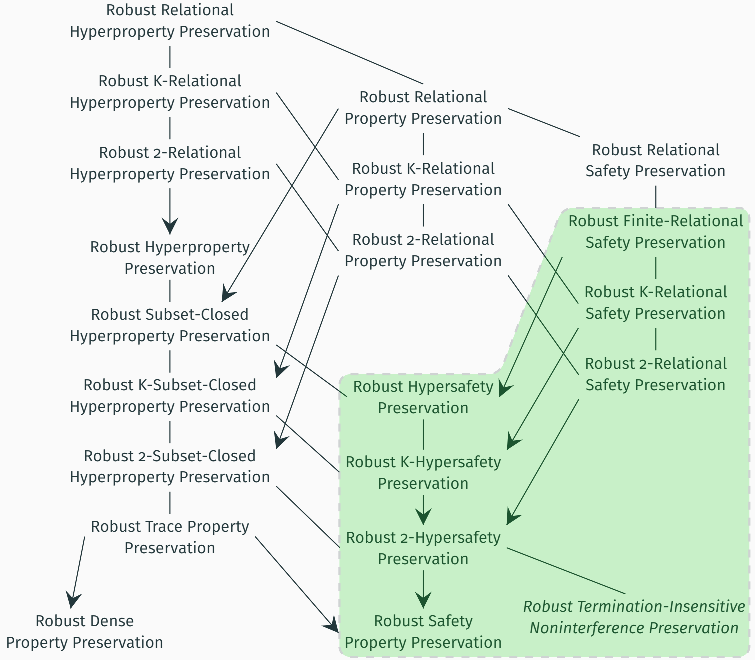
Not All That Glitters is Gold #2

- RSC leads to more efficient compiled code
- RSC is simpler to prove than FAC
- but it’s weaker: no confidentiality

Not All That Glitters is Gold #2



- RSC leads to more efficient compiled code
- RSC is simpler to prove than FAC
- but it’s weaker: no confidentiality (weaker than existing FAC works)


Robust Relational Hyperproperty Preservation
Robust K-Relational Hyperproperty Preservation
Robust 2-Relational Hyperproperty Preservation
Robust Hyperproperty Preservation
Robust Subset-Closed Hyperproperty Preservation
Robust K-Subset-Closed Hyperproperty Preservation
Robust 2-Subset-Closed Hyperproperty Preservation
Robust Trace Property Preservation
Robust Dense Property Preservation
Robust Relational Property Preservation
Robust K-Relational Property Preservation
Robust 2-Relational Property Preservation
Robust Relational Safety Preservation
Robust Finite-Relational Safety Preservation
Robust K-Relational Safety Preservation
Robust 2-Relational Safety Preservation
Robust Hypersafety Preservation
Robust K-Hypersafety Preservation
Robust 2-Hypersafety Preservation
Robust Safety Property Preservation
Robust Termination-Insensitive Noninterference Preservation

Abate *et al.*'19

Proof Techniques for Secure Compilation

Proving FAC

$$\mathbf{P1} \;\simeq_{ctx}\; \mathbf{P2}$$

$$\Updownarrow$$

$$[\![\mathbf{P1}]\!] \simeq_{ctx} [\![\mathbf{P2}]\!]$$

Proving FAC

$$\mathbf{P1} \quad \simeq_{ctx} \quad \mathbf{P2}$$

$$\Uparrow$$

$$[\![\mathbf{P1}]\!] \simeq_{ctx} [\![\mathbf{P2}]\!]$$

Proving FAC

$$\mathbf{P1} \;\; \simeq_{ctx} \;\; \mathbf{P2}$$

$$\Downarrow$$

$$[\![\mathbf{P1}]\!] \simeq_{ctx} [\![\mathbf{P2}]\!]$$

Proving FAC

$$\mathbf{P1} \ \simeq_{ctx} \ \mathbf{P2}$$

$$\Downarrow$$

$$\forall \mathbb{C}. \mathbb{C}[\llbracket \mathbf{P1} \rrbracket] \Downarrow \iff \mathbb{C}[\llbracket \mathbf{P2} \rrbracket] \Downarrow$$

Proving FAC (History)

$$\mathbf{P1} \;\simeq_{ctx}\; \mathbf{P2}$$

$$\Downarrow$$

$$[\![\mathbf{P1}]\!] \;\overset{\top}{=}\; [\![\mathbf{P2}]\!]$$

Proving FAC (History)

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

Jagadeesan *et al.*’11,
Agten *et al.*’12,
Patrignani *et al.*’15’16,
Juglaret *et al.*’16

$$\llbracket \mathbf{P1} \rrbracket \simeq \llbracket \mathbf{P2} \rrbracket$$

Proving FAC (History)

$$\mathbf{P1} \quad \simeq_{ctx} \quad \mathbf{P2}$$

$$\Downarrow$$

$$[\![\mathbf{P1}]\!] \quad \approx \quad [\![\mathbf{P2}]\!]$$

Proving FAC (History)

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

Abadi *et al.*'00'01'02'

Bugliesi *et al.*'07

Adao *et al.*'06

Fournet *et al.*'13

$$[\![\mathbf{P1}]\!] \approx [\![\mathbf{P2}]\!]$$

Proving FAC (History)

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

$$\Downarrow$$

$$[\![\mathbf{P1}]\!] \sim_{\mathsf{n}} [\![\mathbf{P2}]\!]$$

Proving FAC (History)

$$\mathbf{P1} \quad \simeq_{ctx} \quad \mathbf{P2}$$

Ahmed *et al.*'8'11'14'15'16'17,
Devriese *et al.*'16'17

$$[\![\mathbf{P1}]\!] \quad \sim_{\mathsf{n}} \quad [\![\mathbf{P2}]\!]$$

Proving FAC with Logical Relations

$$\begin{tikzcd}
\mathbf{P1} \simeq_{ctx} \mathbf{P2} \arrow[d, Rightarrow, "\text{approx. compiler security}"] \\
[\![\mathbf{P1}]\!] \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]
\end{tikzcd}$$

Proving FAC with Logical Relations

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

approx. compiler security

$$\mathbb{C}\Big[\llbracket\mathbf{P1}\rrbracket\Big]\Downarrow_{\mathsf{n}} \quad \overset{?}{\Rightarrow} \quad \mathbb{C}\Big[\llbracket\mathbf{P2}\rrbracket\Big]\Downarrow_{\_}$$

$$\llbracket\mathbf{P1}\rrbracket \overset{?}{\simeq}_{ctx} \llbracket\mathbf{P2}\rrbracket$$

Proving FAC with Logical Relations

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim_n \mathbb{C}$

$\mathbf{P1} \sim_{\_} [\![\mathbf{P1}]\!]$

$\Uparrow$ (1)

$$\mathbb{C}\left[[\![\mathbf{P1}]\!] \right]\Downarrow_{\mathsf{n}} \overset{?}{\Rightarrow} \mathbb{C}\left[[\![\mathbf{P2}]\!] \right]\Downarrow_{\_}$$

$$[\![\mathbf{P1}]\!] \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]$$

$\Downarrow$ approx. compiler security

Proving FAC with Logical Relations

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

$$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}}\left[\mathbf{P1}\right] \Downarrow_{\_}$$

$$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim_{n} \mathbb{C}$$

$$\mathbf{P1} \sim_{\_} [\![\mathbf{P1}]\!]$$

$$\begin{tikzcd} \langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}}\left[\mathbf{P1}\right] \Downarrow_{\_} \\ \mathbb{C}\left[[\![\mathbf{P1}]\!]\right] \Downarrow_{n} \arrow[u, Rightarrow, "(1)"'] \end{tikzcd}$$

$$\mathbb{C}\left[[\![\mathbf{P1}]\!]\right] \Downarrow_{n} \quad \overset{?}{\Rightarrow} \quad \mathbb{C}\left[[\![\mathbf{P2}]\!]\right] \Downarrow_{\_}$$

$$[\![\mathbf{P1}]\!] \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]$$

$\Downarrow$ approx. compiler security

Proving FAC with Logical Relations

$$\begin{tikzcd}
 & \mathbf{P1} \simeq_{ctx} \mathbf{P2} & & \\
 & \langle\langle\mathbb{C}\rangle\rangle_{\mathbf{n}}\left[\mathbf{P1}\right]\Downarrow_{\_} \arrow[r, Rightarrow, "(2)"'] & \langle\langle\mathbb{C}\rangle\rangle_{\mathbf{n}}\left[\mathbf{P2}\right]\Downarrow_{\_} & {} \arrow[dd, Rightarrow, "\text{approx. compiler security}"] \\
 \begin{array}{l} \langle\langle\mathbb{C}\rangle\rangle_{\mathbf{n}} \sim_{n} \mathbb{C} \\ \mathbf{P1} \sim_{\_} \llbracket\mathbf{P1}\rrbracket \end{array} & & & \\
 & \mathbb{C}\left[\llbracket\mathbf{P1}\rrbracket\right]\Downarrow_{\mathsf{n}} \arrow[uu, Rightarrow, "(1)"'] \arrow[r, Rightarrow, "?"] & \mathbb{C}\left[\llbracket\mathbf{P2}\rrbracket\right]\Downarrow_{\_} & {} \\
 & \llbracket\mathbf{P1}\rrbracket \overset{?}{\simeq}_{ctx} \llbracket\mathbf{P2}\rrbracket & &
\end{tikzcd}$$

Proving FAC with Logical Relations

$$\begin{tikzcd}
\mathbf{P1} \simeq_{ctx} \mathbf{P2} & & {} \arrow[ddd, Rightarrow, "\text{approx. compiler security}"] \\
\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}}[\mathbf{P1}]\Downarrow_{\_} \arrow[r, Rightarrow, "(2)"'] & \langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}}[\mathbf{P2}]\Downarrow_{\_} \arrow[d, Rightarrow, bend left, "(3)"'] & \\
\mathbb{C}\left[[\![\mathbf{P1}]\!]\right]\Downarrow_{\mathsf{n}} \arrow[u, Rightarrow, bend left, "(1)"] \arrow[r, Rightarrow, "?"] & \mathbb{C}\left[[\![\mathbf{P2}]\!]\right]\Downarrow_{\_} & \\
[\![\mathbf{P1}]\!] \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!] & & {}
\end{tikzcd}$$

$$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim_n \mathbb{C}$$
$$\mathbf{P1} \sim_{\_} [\![\mathbf{P1}]\!]$$

$$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim_{\_} \mathbb{C}$$
$$\mathbf{P2} \sim_{\_} [\![\mathbf{P2}]\!]$$

Proving FAC with Logical Relations

$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$

$\langle\!\langle \mathbb{C} \rangle\!\rangle \; [\![\mathbf{P1}]\!]\Downarrow$ $\quad\langle\!\langle \mathbb{C} \rangle\!\rangle \; [\![\mathbf{P2}]\!]\Downarrow\_$

$\mathbf{P1} \sim [\![\mathbf{P1}]\!]$ is obtained with standard techniques

Benton *et al.*’09’10

Hur *et al.*’11

Neis *et al.*’15

$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim_n$ $\quad\sim\_ \; \mathbb{C}$

$\mathbf{P1} \sim\_ \; [\![\mathbf{P}$ $\quad[\![\mathbf{P2}]\!]$

$\mathbb{C}\left[[\![\mathbf{P1}]\!] \right] \Downarrow_{\mathsf{n}} \quad \overset{?}{\Rightarrow} \quad \mathbb{C}\left[[\![\mathbf{P2}]\!] \right] \Downarrow\_$

$[\![\mathbf{P1}]\!] \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]$

approx. compiler security

Proving FAC with Logical Relations

$\langle\langle \mathbb{C} \rangle\rangle_{\mathbf{n}} \sim \mathbb{C}$ requires

- back-translation of terms
- reasoning at the type of back-translated terms

$$[\![\mathbf{P1}]\!] \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]$$

approx. compiler security

Proving FAC with Logical Relations

$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim \mathbb{C}$ requires

- back-translation of terms
- reasoning at the type of back-translated terms
- needed for all kinds of back-translation

$$[\![\mathbf{P1}]\!] \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]$$

approx. compiler security

Proving FAC with Logical Relations

$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim \mathbb{C}$ requires

- back-translation of terms
- reasoning at the type of back-translated terms
- needed for all kinds of back-translation
- needed for RSC too

$[\![\mathbf{P1}]\!] \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]$

approx. compiler security

Proving FAC with Logical Relations

$\langle\langle \mathbb{C} \rangle\rangle_{\mathbf{n}} \sim \mathbb{C}$ requires

- back-translation of terms
- reasoning at the type of back-translated terms
- needed for all kinds of back-translation
- needed for RSC too

$[\![\mathbf{P1}]\!] \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]$

approx. compiler security

Proving FAC with Logical Relations

$\langle\langle \mathbb{C} \rangle\rangle_{\mathbf{n}} \sim \mathbb{C}$ requires

- back-translation of terms
- reasoning at the type of back-translated terms
- needed for all kinds of back-translation
- needed for RSC too

$[\![\mathbf{P1}]\!] \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]$

approx. compiler security

Proving FAC with Logical Relations

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

$$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathrm{n}}[\mathbf{P1}]\Downarrow\_ \quad \overset{(?)}{\Rightarrow} \quad \langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathrm{n}}[\mathbf{P2}]\Downarrow\_$$

COURSE

coming up next semester

(and next year)

$$\mathbb{C}\Big[[\![\mathbf{P1}]\!]\Big]\Downarrow_{\mathrm{n}} \quad \overset{?}{\Rightarrow} \quad \mathbb{C}\Big[[\![\mathbf{P2}]\!]\Big]\Downarrow\_$$

$$[\![\mathbf{P1}]\!] \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]$$

approx. compiler security

Conclusion

- motivations for secure compilation

Conclusion

- motivations for secure compilation
- secure compilation criterion: fully abstract compilation

Conclusion

- motivations for secure compilation
- secure compilation criterion: fully abstract compilation
- secure compilation criterion: robustly-safe compilation

Conclusion

- motivations for secure compilation
- secure compilation criterion: fully abstract compilation
- secure compilation criterion: robustly-safe compilation
- proof techniques for secure compilation

Research Field Prospect

- secure compilation workshop: PrISC 3rd ed. (co-located with POPL) https://popl19.sigplan.org/track/prisc-2019
- secure compilation classes: Winter quarter ’18-19, Spring quarter ’19-20 (?)
- introductory survey: *Patrignani, Ahmed, Clarke. ACM CSUR ’19*
- lots of challenging open problems to work on (talk to me!)

Questions?