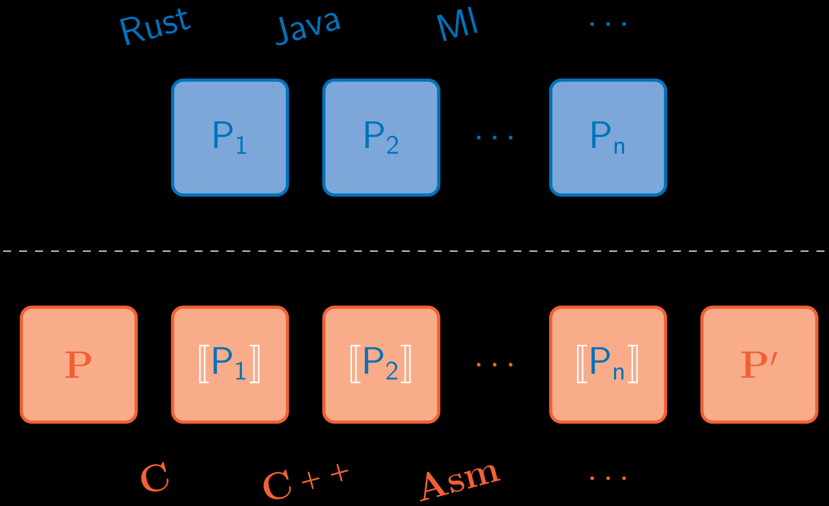
Robustly Safe Compilation

 Marco Patrignani[1,2]   Deepak Garg[3] 

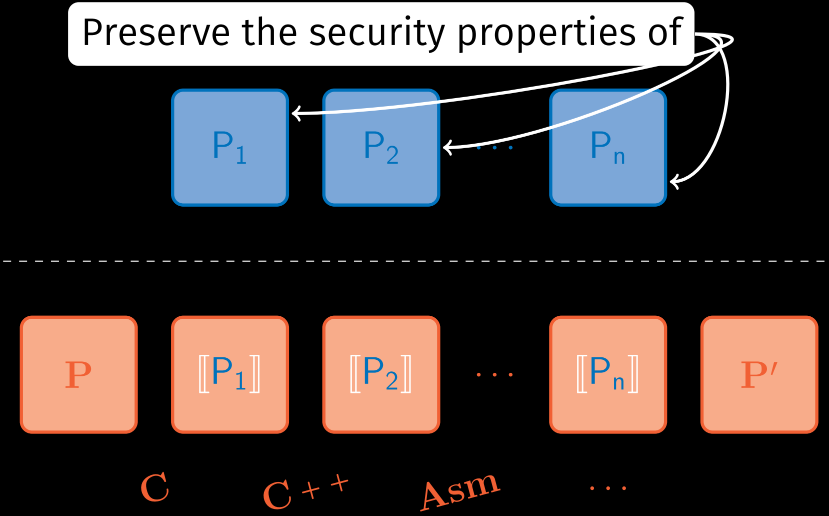
10th April 2019

1 Stanford University
2 CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY
3 MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

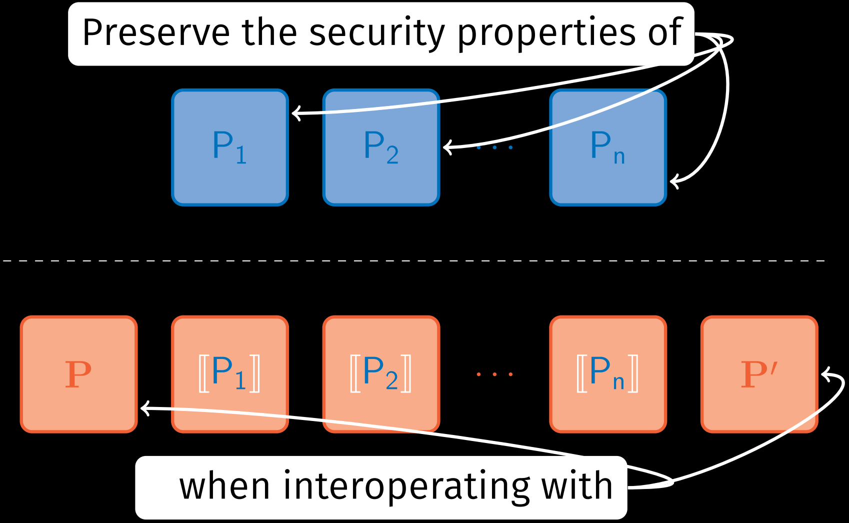
What is Secure Compilation?



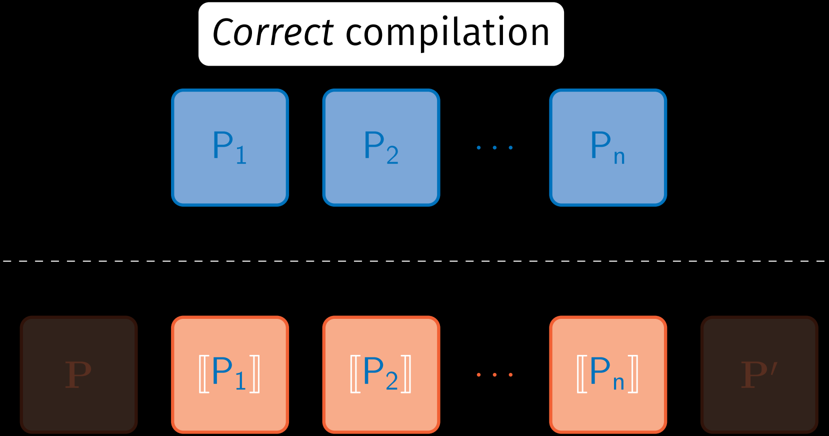
What is Secure Compilation?



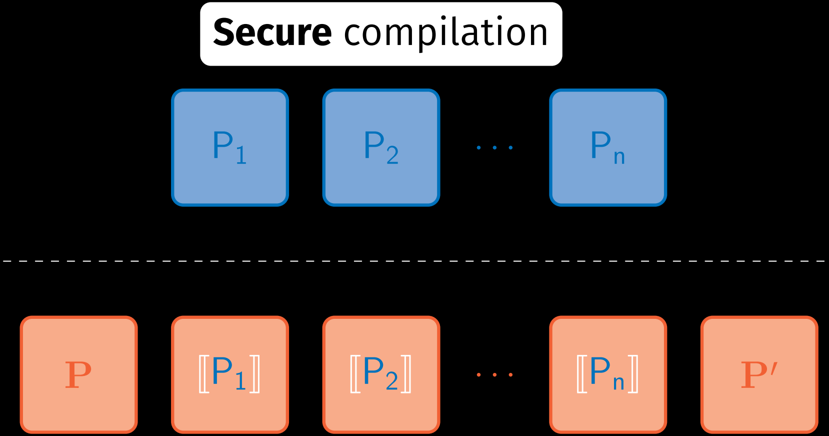
What is Secure Compilation?



What is Secure Compilation?

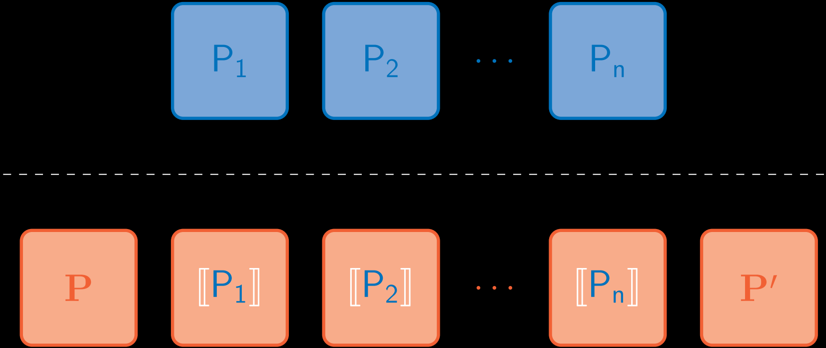


What is Secure Compilation?



What is Secure Compilation?

Enable source-level security reasoning



Do Secure Compilers Exist?

Do Secure Compilers Exist?

Yes!

Do Secure Compilers Exist?

Yes!

They rely on security mechanisms:

- enclaves
- capabilities
- types
- tagged memory
- ASLR
- CFI, SFI
- processes
- ...

But...

Some secure compilers:

- $\mathrm{P1}$: lack formal proof of their security guarantees

But...

Some secure compilers:

- $\mathbf{P1}$: lack formal proof of their security guarantees
- $\mathbf{P2}$: prove preservation of ad-hoc security properties

But...

Some secure compilers:

- $\mathbf{P1}$: lack formal proof of their security guarantees
- $\mathbf{P2}$: prove preservation of ad-hoc security properties
- $\mathbf{P3}$: inefficient

But...

secure compilers:

- : lack formal proof of their security guarantees
- **P2** : prove preservation of ad-hoc security properties
- **P3** : inefficient

complex proofs

unclear how to generalise

dictated by existing definitions

Goal:

Define a formal criterion for secure compilation:

Goal:

Define a formal criterion for secure compilation:

- attainable
- efficient (wrt existing ones)
- ~~easy~~ not too hard to prove

Goal:

Define a formal criterion for secure compilation:

- attainable
- efficient (wrt existing ones)
- ~~easy~~ not too hard to prove
- with clear security guarantees

Contributions

- RSC: a criterion fulfilling our goals

Contributions

- RSC: a criterion fulfilling our goals
  - a compiler preserves all safety properties

Contributions

- RSC: a criterion fulfilling our goals
    - a compiler preserves all safety properties
- three compilers $[\![\cdot]\!]$ that attain RSC

Contributions

- RSC: a criterion fulfilling our goals
  - a compiler preserves all safety properties
- three compilers $[\![\cdot]\!]$ that attain RSC
  - relying on memory isolation (via capabilities or enclaves)

Contributions

- RSC: a criterion fulfilling our goals
  - a compiler preserves all safety properties
- three compilers $[\![\cdot]\!]$ that attain RSC
  - relying on memory isolation (via capabilities or enclaves)

    no runtime checks!

Contributions

- RSC: a criterion fulfilling our goals
  - a compiler preserves all safety properties
- three compilers $[\![\cdot]\!]$ that attain RSC
  - relying on memory isolation (via capabilities or enclaves)

    no runtime checks!
- two proof techniques for RSC

Contributions

- RSC: a criterion fulfilling our goals
  - a compiler preserves all safety properties
- three compilers $[\![\cdot]\!]$ that attain RSC
  - relying on memory isolation (via capabilities or enclaves)

    no runtime checks!
- two proof techniques for RSC
  - simplifications on existing ones

Contributions

- RSC: a criterion fulfilling our goals
  - a compiler preserves all safety properties
- three compilers $[\![\cdot]\!]$ that attain RSC
  - relying on memory isolation (via capabilities or enclaves)

    no runtime checks!
- two proof techniques for RSC
  - simplifications on existing ones
- a comparison between RSC and FAC

Contributions

part 1

- RSC: a criterion fulfilling our goals
  - a compiler preserves all safety properties
- three compilers $[\![\cdot]\!]$ that attain RSC
  - relying on memory isolation (via capabilities or enclaves)

    no runtime checks!
- two proof techniques for RSC
  - simplifications on existing ones
- a comparison between RSC and FAC

Contributions

- RSC: a criterion fulfilling our goals
  - a compiler preserves all safety properties
- three compilers $[\![\cdot]\!]$ that attain RSC
  - relying on memory isolation (via capabilities or enclaves)
    no runtime checks!
- two proof techniques for RSC
  - simplifications on existing ones
- a comparison between RSC and FAC

part 1

part 2

Talk Roadmap

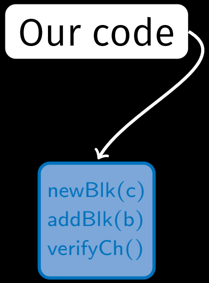
Robust Safety

Robustly Safe Compilation

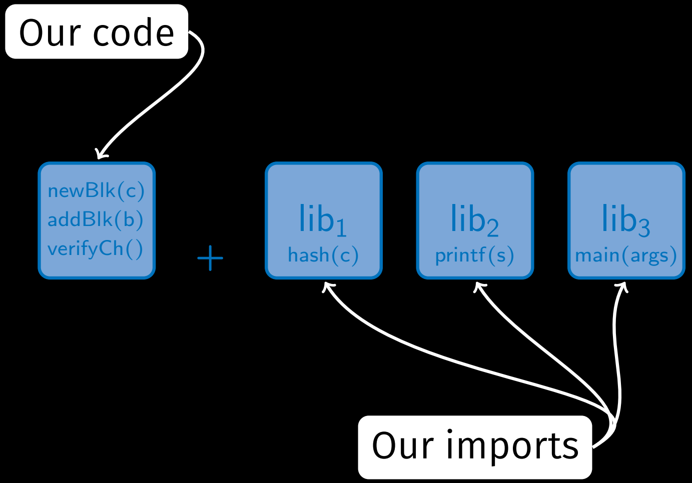
Backtranslation Proof Technique

Robust Safety

Robustness

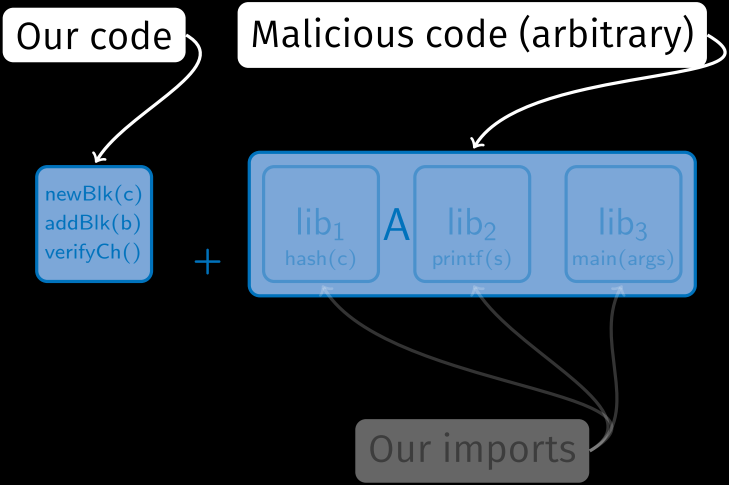


Robustness

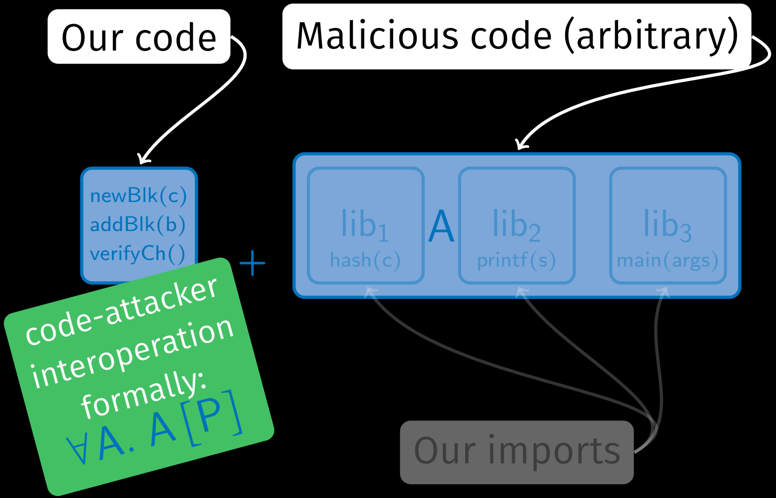


Our code
newBlk(c)
addBlk(b)
verifyCh()
+
lib1
hash(c)
lib2
printf(s)
lib3
main(args)
Our imports

Robustness

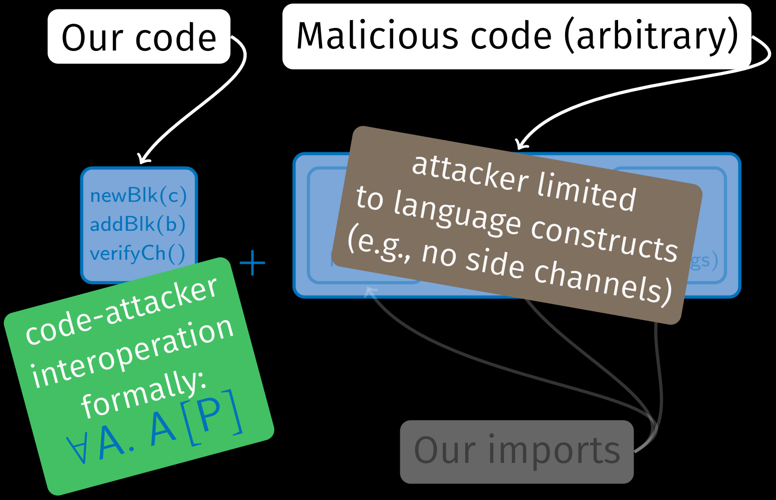

Our code
Malicious code (arbitrary)
newBlk(c)
addBlk(b)
verifyCh()
+
lib1
hash(c)
A
lib2
printf(s)
lib3
main(args)
Our imports

Robustness

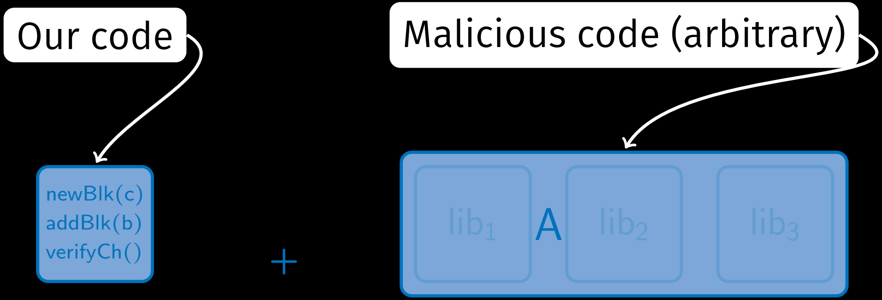


Our code
Malicious code (arbitrary)
newBlk(c)
addBlk(b)
verifyCh()
+
A
lib1
hash(c)
lib2
printf(s)
lib3
main(args)
Our imports
code-attacker
interoperation
formally:
∀A. A[P]

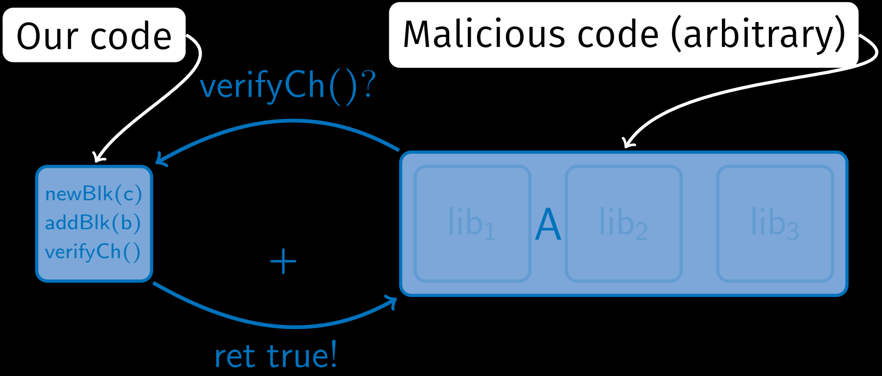
Robustness


Our code
Malicious code (arbitrary)
newBlk(c)
addBlk(b)
verifyCh()
+
attacker limited to language constructs (e.g., no side channels)
code-attacker interoperation formally:
∀A. A[P]
Our imports

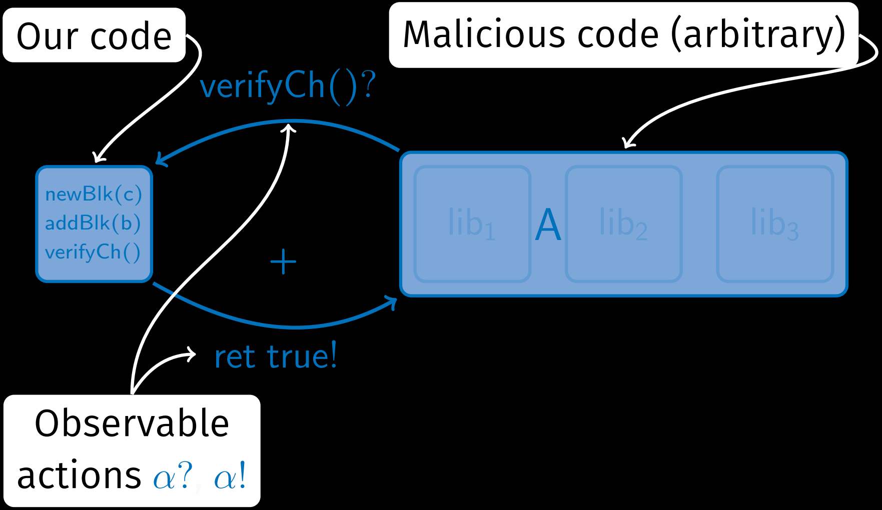
Program Behaviour



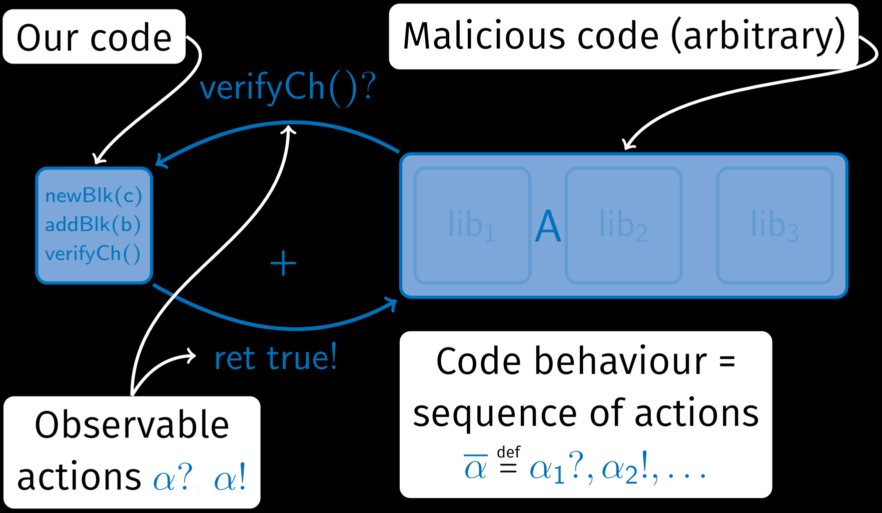
Program Behaviour



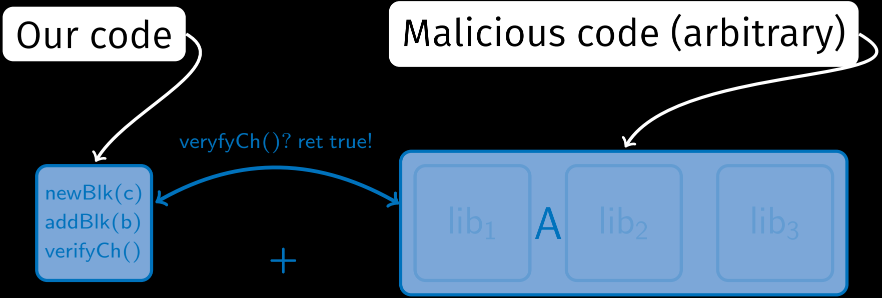
Program Behaviour



Program Behaviour



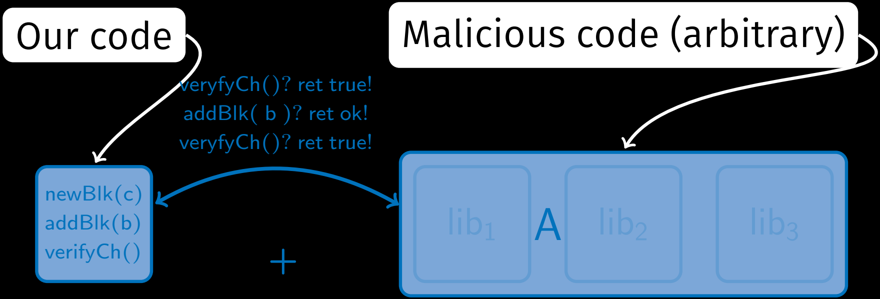
Program Behaviour



Code behaviour = sequence of actions

$\overline{\alpha} \stackrel{\text{def}}{=} \alpha_1?, \alpha_2!, \ldots$

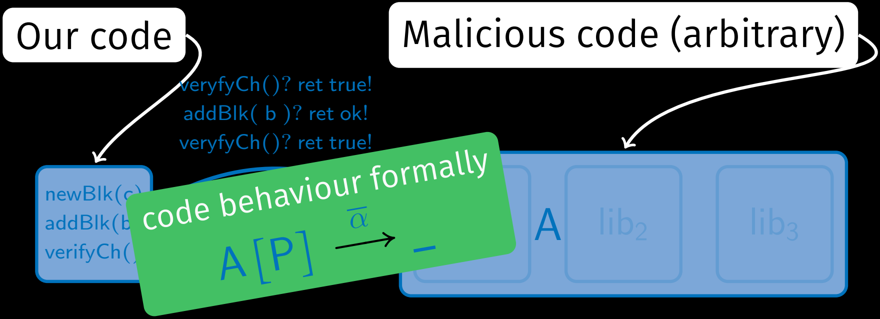
Program Behaviour



Code behaviour = sequence of actions

$$\overline{\alpha} \stackrel{\text{def}}{=} \alpha_1?, \alpha_2!, \ldots$$

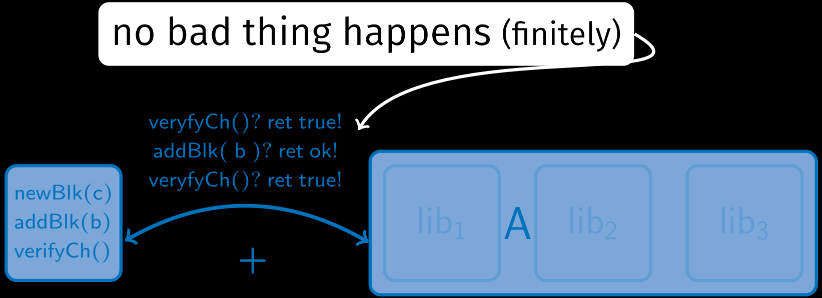
Program Behaviour



Code behaviour =
sequence of actions

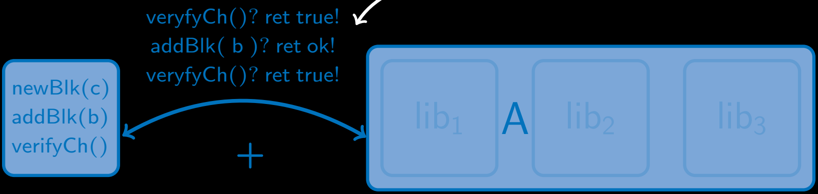
$\overline{\alpha} \stackrel{\text{def}}{=} \alpha_1?, \alpha_2!, \ldots$

Safety Properties



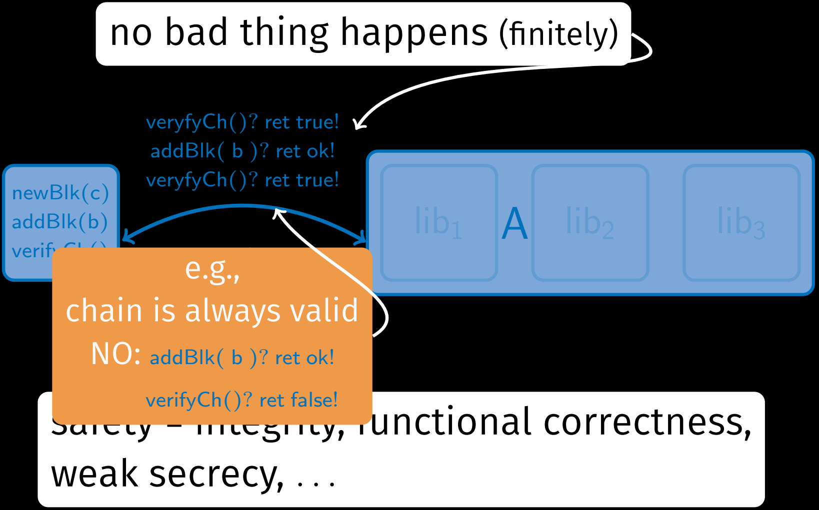
Safety Properties

no bad thing happens (finitely)

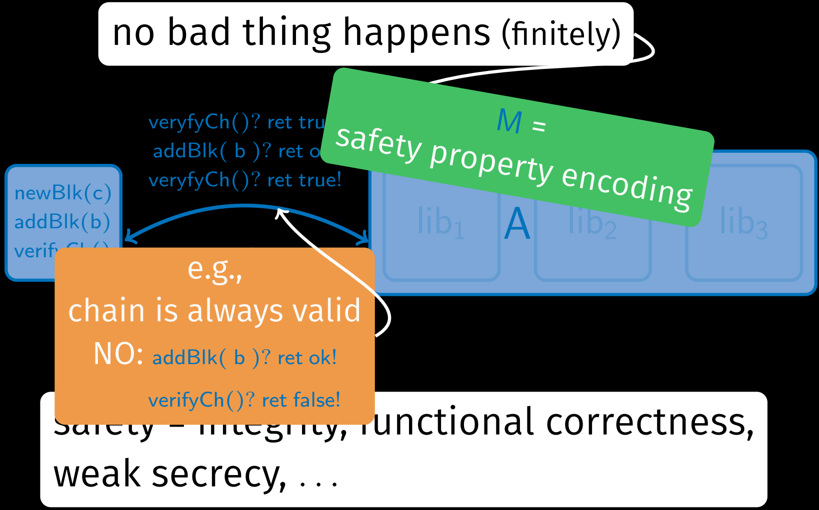


safety = integrity, functional correctness, weak secrecy, ...

Safety Properties



Safety Properties



Robust Safety

- for a safety property

Robust Safety

- for a safety property
- no matter what we link against

Robust Safety

- for a safety property
- no matter what we link against
- our program behaves in a way

Robust Safety

- for a safety property
- no matter what we link against
- our program behaves in a way
- that respects that safety property

Robust Safety

- for a safety property (M)
- no matter what we link against ($\forall \mathsf{A}, \overline{\alpha}$)
- our program behaves in a way (if $\mathsf{A}\,[\mathsf{P}] \xrightarrow{\overline{\alpha}}$)
- that respects that safety property (then $\mathsf{M} \vdash \overline{\alpha}$)

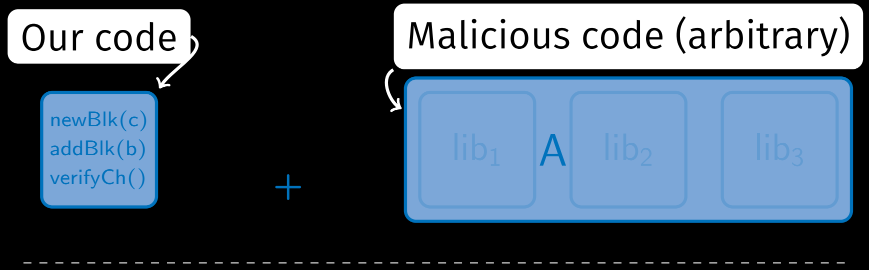
Robust Safety

- for a safety property (M)
- no matter what we link against ($\forall A, \overline{\alpha}$)
- our program behaves in a way (if $A\,[P] \xrightarrow{\overline{\alpha}}$)
- that respects that safety property (then $M \vdash \overline{\alpha}$)

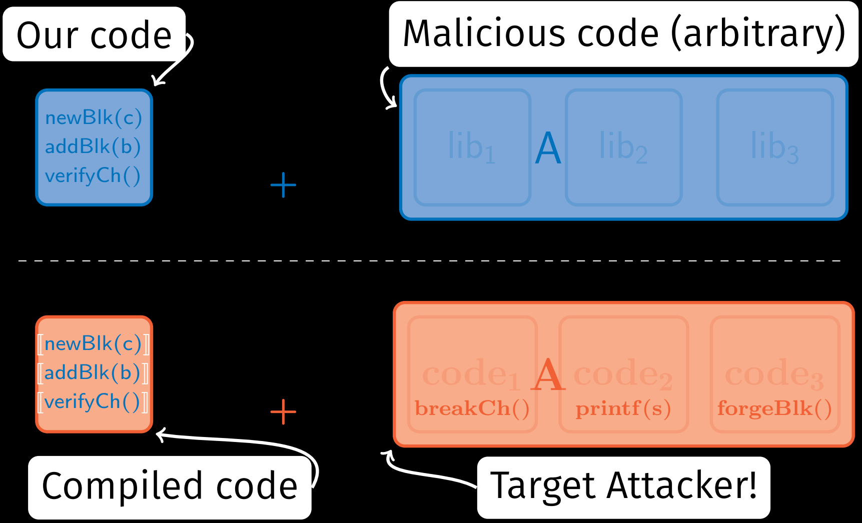
robust safety formally
$M \vdash P$

Robustly Safe Compilation

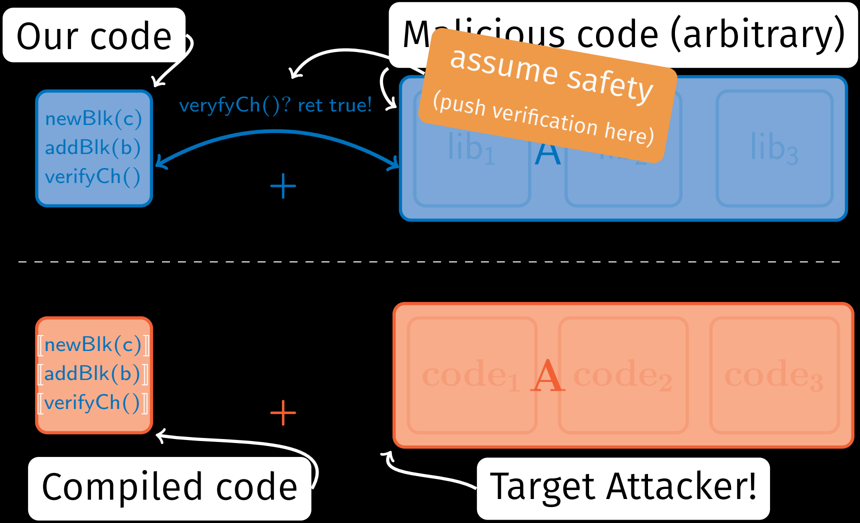
Robust Safety Across Compilation



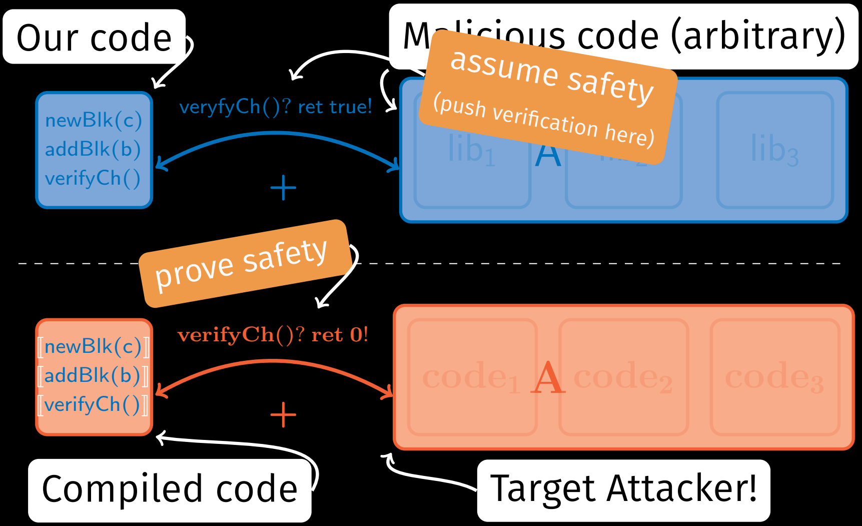
Robust Safety Across Compilation



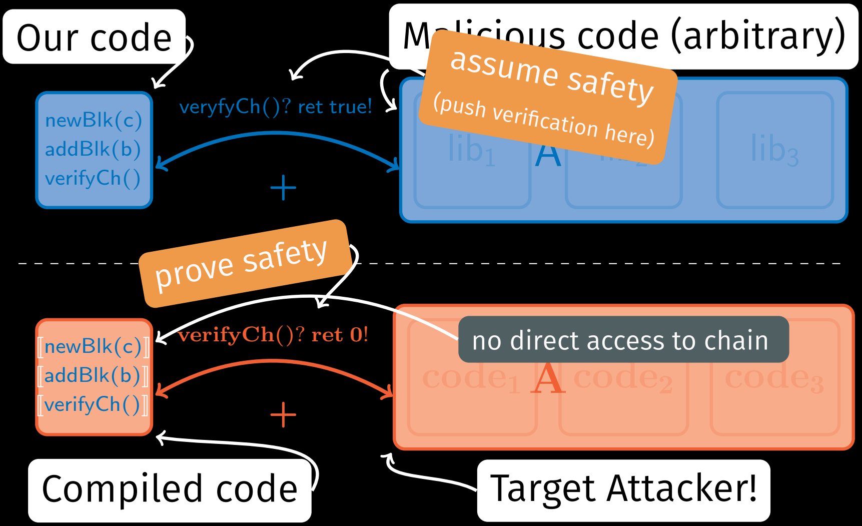
Robust Safety Across Compilation



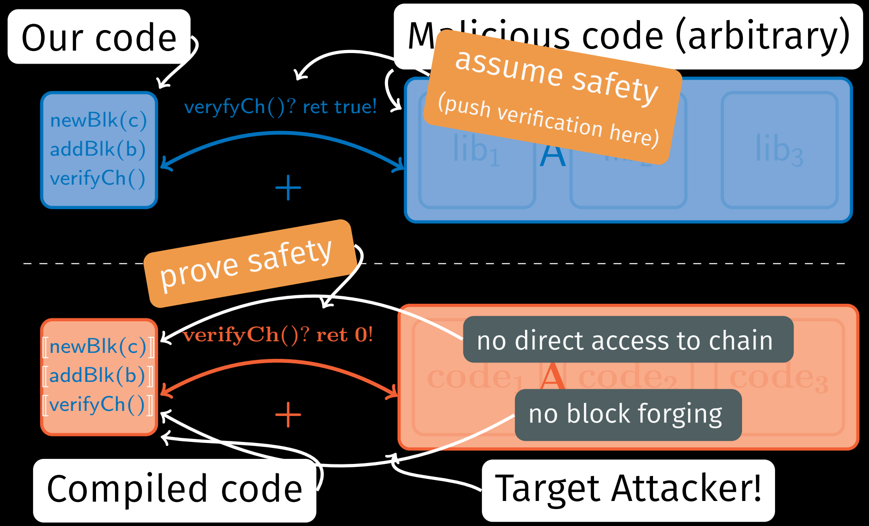
Robust Safety Across Compilation



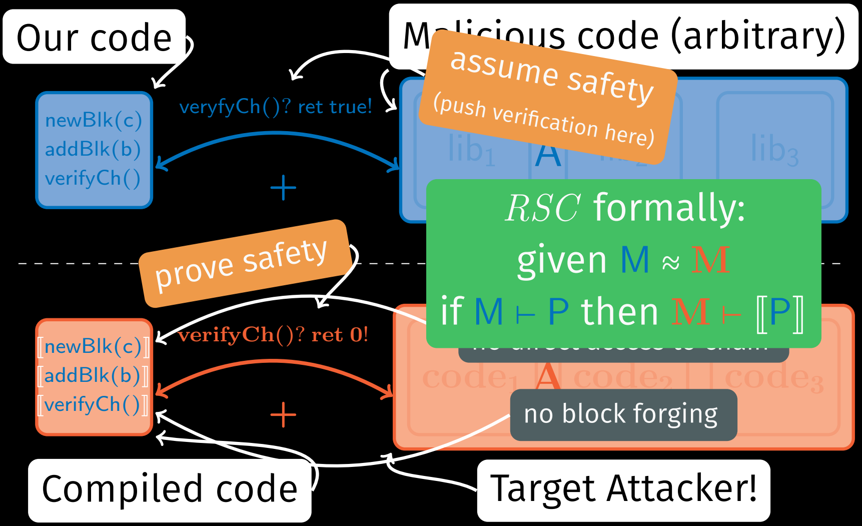
Robust Safety Across Compilation



Robust Safety Across Compilation



Robust Safety Across Compilation



Concerns

RSC so far:

- attainable
- efficient

Concerns

RSC so far:

- attainable
- efficient
- possibly tricky to prove

Concerns

RSC so far:

- attainable
- efficient
- possibly tricky to prove

PF-RSC: equivalent definition

Concerns

RSC so far:

- attainable
- efficient
- possibly tricky to prove

PF-RSC: equivalent definition easier to prove than RSC

Concerns

RSC so far:

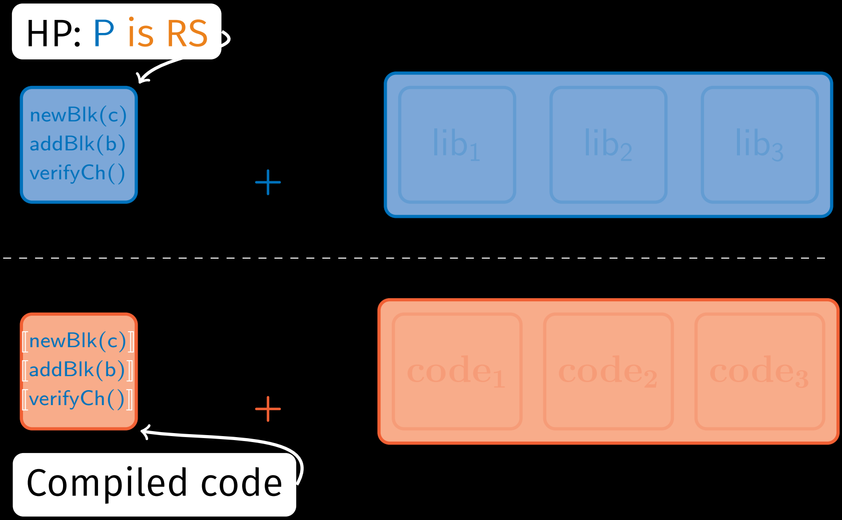
- attainable
- efficient
- possibly tricky to prove

PF-RSC: equivalent definition easier to prove than RSC

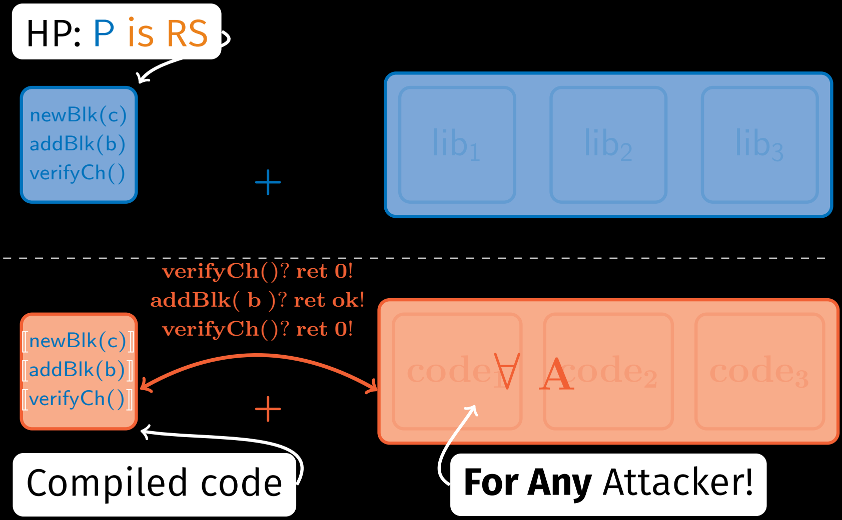
(equivalence to be proven, generally true)

Backtranslation Proof Technique

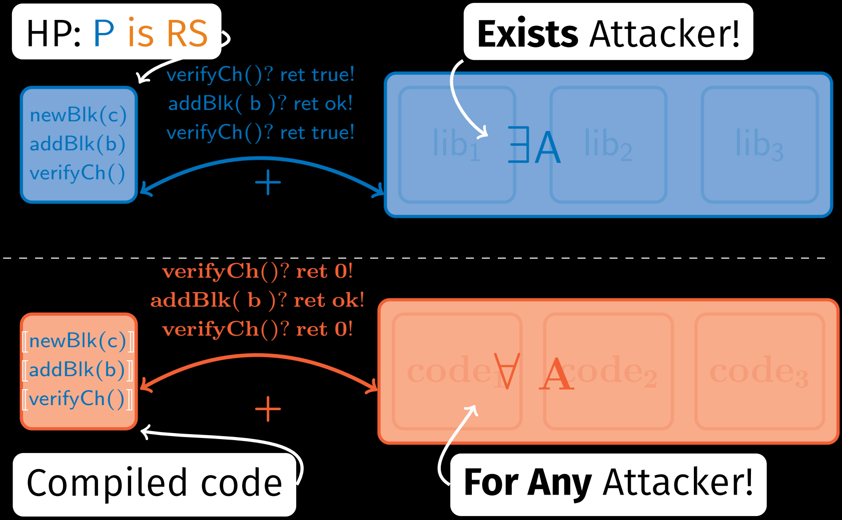
Backtranslation: Build A From A or $\overline{\alpha}$



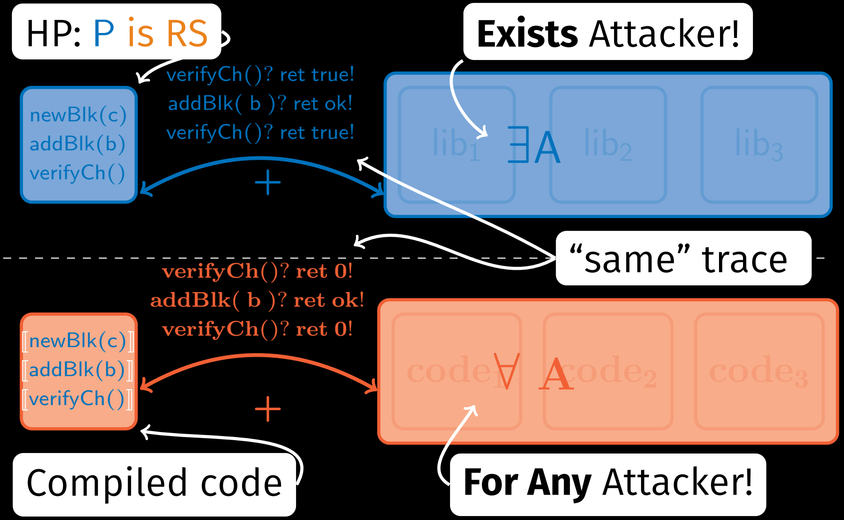
Backtranslation: Build A From A or $\overline{\alpha}$



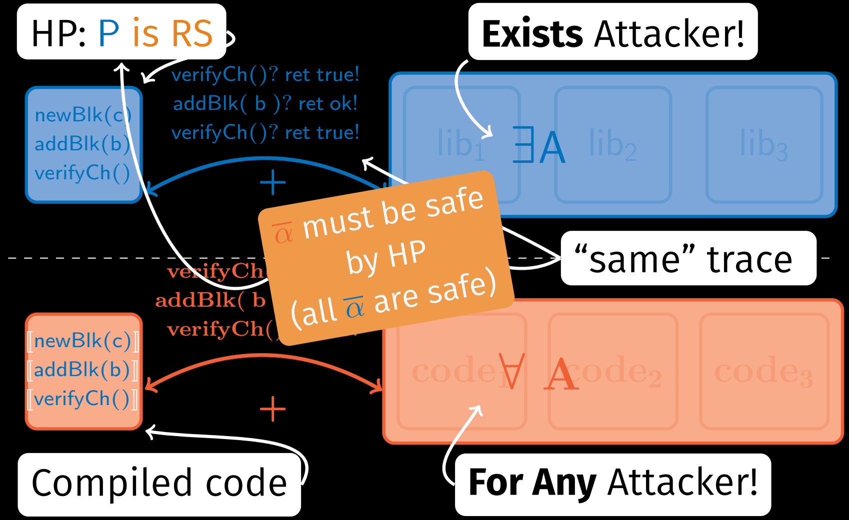
Backtranslation: Build A From A or $\overline{\alpha}$



Backtranslation: Build A From A or $\overline{\alpha}$



Backtranslation: Build A From A or $\overline{\alpha}$



Safety as a Dual and PF-RSC

- Safety = nothing bad happens

Safety as a Dual and PF-RSC

- Safety = nothing bad happens
  so if it happens it does finitely

Safety as a Dual and PF-RSC

- Safety = nothing bad happens
  so if it happens it does finitely

- given any **behaviour** ($\mathbf{A}\,[\,[\![\mathsf{P}]\!]\,] \xrightarrow{\overline{\alpha}}$)

Safety as a Dual and PF-RSC

- Safety = nothing bad happens
  so if it happens it does finitely

- given any **behaviour** ($\mathbf{A}\left[[\![P]\!] \right] \xrightarrow{\overline{\alpha}}$)
- if we can replicate that ($\exists A.A\left[P\right] \xrightarrow{\overline{\alpha}}$)

Safety as a Dual and PF-RSC

- Safety = nothing bad happens
  so if it happens it does finitely

- given any **behaviour** ($\mathbf{A}\left[\llbracket \mathsf{P} \rrbracket\right] \xrightarrow{\overline{\alpha}}$)
- if we can replicate that ($\exists \mathsf{A}.\mathsf{A}\left[\mathsf{P}\right] \xrightarrow{\overline{\alpha}}$)
- then $\overline{\alpha}$ is not bad

Safety as a Dual and PF-RSC

- Safety = nothing bad happens
  so if it happens it does finitely
- given any **behaviour** ($\mathbf{A}\left[[\![\mathsf{P}]\!]\right] \xrightarrow{\overline{\alpha}}$)
- if we can replicate that ($\exists \mathsf{A}.\mathsf{A}\left[\mathsf{P}\right] \xrightarrow{\overline{\alpha}}$)
- then $\overline{\alpha}$ is not bad
  because $\overline{\alpha}$ does not violate safety
  (by RS of P)

Safety as a Dual and PF-RSC

- Safety = nothing bad happens
  so if it happens it does finitely
- given any **behaviour** ($\mathbf{A}\left[\llbracket \mathsf{P} \rrbracket\right] \xrightarrow{\overline{\alpha}}$)
- if we can replicate that ($\exists \mathsf{A}.\mathsf{A}\left[\mathsf{P}\right] \xrightarrow{\overline{\alpha}}$)
- then $\overline{\alpha}$ is not bad
  because $\overline{\alpha}$ does not violate safety
  (by RS of P)
- so necessarily neither does $\overline{\alpha}$ (for $\overline{\alpha} \approx \overline{\alpha}$)

Safety as a Dual and PF-RSC

- Safety = nothing bad happens
  so if it happens it does finitely
- given any **behaviour** ($\mathbf{A}$ [⟦P⟧] $\xrightarrow{\overline{\alpha}}$)
- if we can replica
- then $\overline{\alpha}$ is not ba
  because $\overline{\alpha}$ does
  (by RS of P)
- so necessarily neither does $\overline{\alpha}$ (for $\overline{\alpha} \approx \overline{\alpha}$)

PF-RSC formally:
if $\forall \mathbf{A}. \mathbf{A}\,[\llbracket P \rrbracket] \xrightarrow{\overline{\alpha}}$
then $\exists A.A\,[P] \xrightarrow{\overline{\alpha}}$ and $\overline{\alpha} \approx \overline{\alpha}$

RSC and PF-RSC

RSC: given $\mathsf{M} \approx \mathbf{M}$
if $\mathsf{M} \vdash \mathsf{P}$ then $\mathbf{M} \vdash [\![\mathsf{P}]\!]$

$\Longleftrightarrow$

PF-RSC: if $\forall \mathbf{A}.\mathbf{A}\left[[\![\mathsf{P}]\!]\right] \xrightarrow{\overline{\alpha}}$
then $\exists \mathsf{A}.\mathsf{A}\left[\mathsf{P}\right] \xrightarrow{\overline{\alpha}}$ and $\overline{\alpha} \approx \overline{\alpha}$

RSC and PF-RSC

RSC: given $\mathsf{M} \approx \mathbf{M}$
if $\mathsf{M} \vdash \mathsf{P}$ then $\mathbf{M} \vdash [\![\mathsf{P}]\!]$

$\Longleftrightarrow$

PF-RSC: if $\forall \mathbf{A}.\mathbf{A}\,[[\![\mathsf{P}]\!]] \xrightarrow{\overline{\alpha}}$
then $\exists \mathsf{A}.\mathsf{A}\,[\mathsf{P}] \xrightarrow{\overline{\alpha}}$ and $\overline{\alpha} \approx \overline{\alpha}$

- $\Longleftrightarrow$ must be proven (when needed)

RSC and PF-RSC

RSC: given $\mathsf{M} \approx \mathbf{M}$
if $\mathsf{M} \vdash \mathsf{P}$ then $\mathbf{M} \vdash [\![\mathsf{P}]\!]$

$\iff$

PF-RSC: if $\forall \mathbf{A}.\mathbf{A}\,[[\![\mathsf{P}]\!]] \xrightarrow{\overline{\alpha}}$
then $\exists \mathsf{A}.\mathsf{A}\,[\mathsf{P}] \xrightarrow{\overline{\alpha}}$ and $\overline{\alpha} \approx \overline{\alpha}$

- $\iff$ must be proven (when needed)
- proof is (generally) trivial

RSC and PF-RSC

RSC: given $\mathsf{M} \approx \mathbf{M}$ if $\mathsf{M} \vdash \mathsf{P}$ then $\mathbf{M} \vdash [\![\mathsf{P}]\!]$

$\iff$

PF-RSC: if $\forall \mathbf{A}.\mathbf{A}\left[[\![\mathsf{P}]\!]\right] \xrightarrow{\overline{\alpha}}$ then $\exists \mathsf{A}.\mathsf{A}\left[\mathsf{P}\right] \xrightarrow{\overline{\alpha}}$ and $\overline{\alpha} \approx \overline{\alpha}$

- $\iff$ must be proven (when needed)
- proof is (generally) trivial
- sanity-check for cross-language safety encoding ($\mathsf{M} \approx \mathbf{M}$)

Take Home Message

What to make of this result?

Take Home Message

What to make of this result?

- encode safety properties in your systems

Take Home Message

What to make of this result?

- encode safety properties in your systems
- ensure your desired property follows from the encoding

Take Home Message

What to make of this result?

- encode safety properties in your systems
- ensure your desired property follows from the encoding
- use our proof techniques to prove safety is preserved

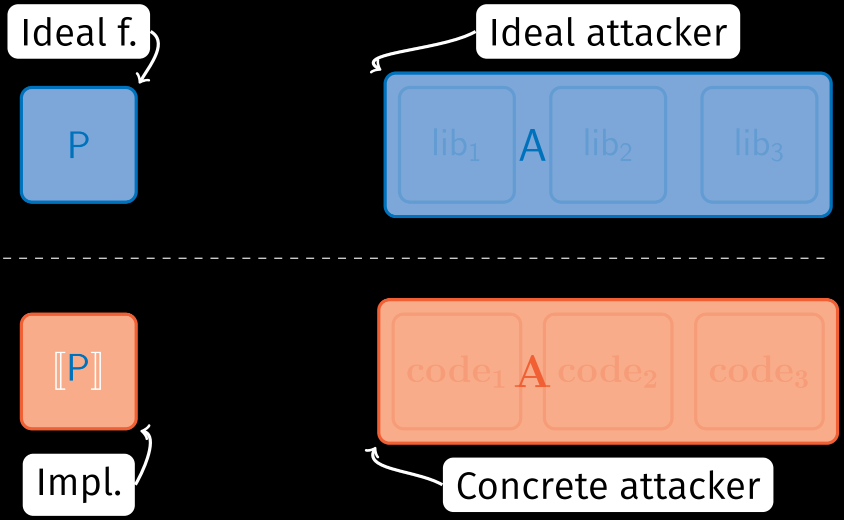
What Else?

The paper (or the techreport) contains more:

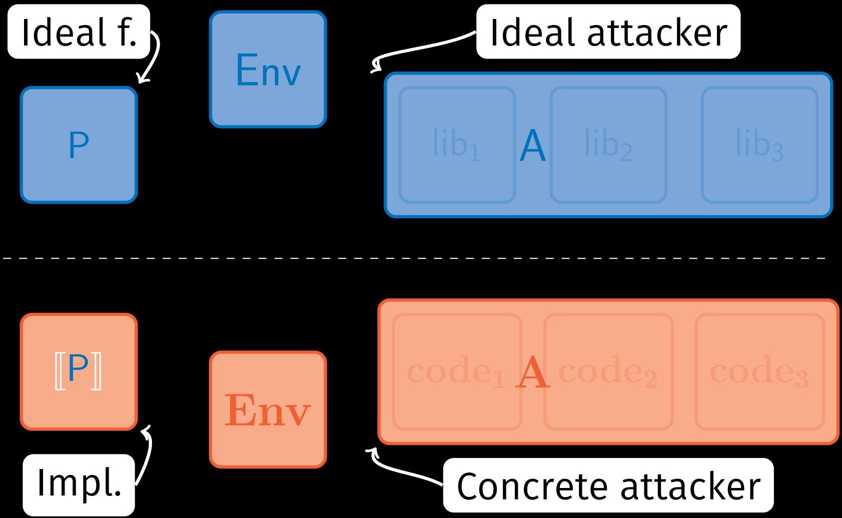
- one RSC $[\![\cdot]\!]^{\mathsf{L}^{\mathsf{U}}}_{\mathbf{L}^{\mathbf{P}}}$ from untyped while to **capabilities**
- one RSC $[\![\cdot]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L}^{\pi}}$ from typed, concurrent while to **capabilities**
- one RSC $[\![\cdot]\!]^{\mathsf{L}^{\tau}}_{L^{I}}$ from typed, concurrent while to *enclaves*
- a backtranslation-based RSC proof (for $[\![\cdot]\!]^{\mathsf{L}^{\mathsf{U}}}_{\mathbf{L}^{\mathbf{P}}}$)
- two simulation-based RSC proofs (for $[\![\cdot]\!]^{\mathsf{L}^{\tau}}_{\mathbf{L}^{\pi}}$ and $[\![\cdot]\!]^{\mathsf{L}^{\tau}}_{L^{I}}$)
- a FAC $[\![\boxed{\cdot}]\!]^{\mathsf{L}^{\mathsf{U}}}_{\mathbf{L}^{\mathbf{P}}}$ from untyped while to **capabilities**
- a backtranslation-based FAC proof sketch (for $[\![\boxed{\cdot}]\!]^{\mathsf{L}^{\mathsf{U}}}_{\mathbf{L}^{\mathbf{P}}}$)
- a comparison of efficiency and proof complexity between $[\![\cdot]\!]^{\mathsf{L}^{\mathsf{U}}}_{\mathbf{L}^{\mathbf{P}}}$ and $[\![\boxed{\cdot}]\!]^{\mathsf{L}^{\mathsf{U}}}_{\mathbf{L}^{\mathbf{P}}}$

Questions?

RSC and UC

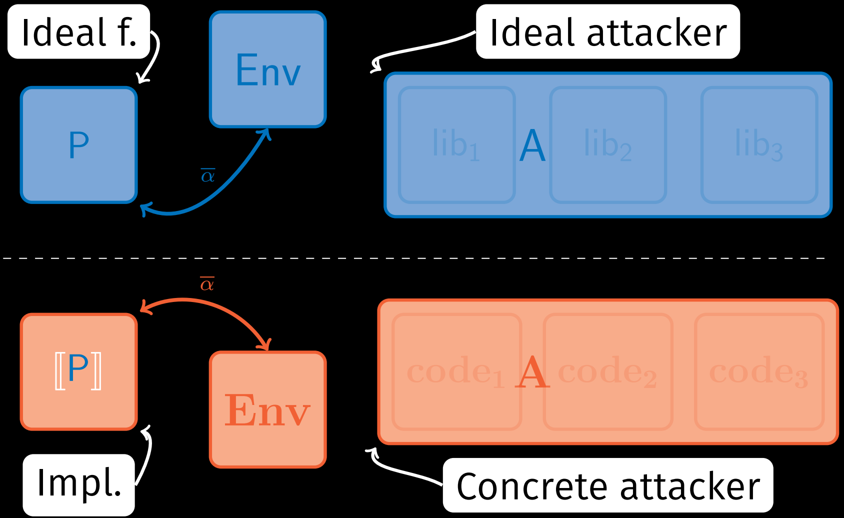

Ideal f.
P
Ideal attacker
lib1
A
lib2
lib3
⟦P⟧
Impl.
code1
A
code2
code3
Concrete attacker

RSC and UC



Ideal f.
Env
Ideal attacker
P
lib₁
A
lib₂
lib₃
⟦P⟧
Env
code₁
A
code₂
code₃
Impl.
Concrete attacker

RSC and UC



Ideal f.
Env
Ideal attacker
P
lib$_1$
A
lib$_2$
lib$_3$
$\overline{\alpha}$
$\overline{\alpha}$
⟦P⟧
Env
code$_1$
A
code$_2$
code$_3$
Impl.
Concrete attacker

Backtranslation Example

(1) $\texttt{call f 0}\ (\mathbf{1} \mapsto \mathbf{4} : \bot, \mathbf{2} \mapsto \mathbf{3} : \bot)?$

(2) $\texttt{ret}\ (\mathbf{1} \mapsto \mathbf{4} : \bot, \mathbf{2} \mapsto \langle \mathbf{3}, \mathbf{k} \rangle : \bot, \mathbf{3} \mapsto \mathbf{11} : \mathbf{k})!$

(3) $\texttt{call f 2}\ (\mathbf{1} \mapsto \mathbf{55} : \bot, \mathbf{2} \mapsto \langle \mathbf{3}, \mathbf{k} \rangle : \bot, \mathbf{3} \mapsto \mathbf{15} : \mathbf{k})?$

$\mathsf{main}(z) \mapsto$

(1)
$\mathsf{let}\ x = \mathsf{new}\ 4\ \mathsf{in}\ L :: \langle x, 1 \rangle\ ;$
$\mathsf{let}\ x = \mathsf{new}\ 3\ \mathsf{in}\ L :: \langle x, 2 \rangle\ ;$
$\mathsf{call}\ f\ 0;$

(2)
$\mathsf{let}\ x = !L(2)\ \mathsf{in}\ L :: \langle x, 3 \rangle\ ;$

(3)
$\mathsf{let}\ x = \mathsf{new}\ L(1)\ \mathsf{in}\ x := 55;$
$\mathsf{let}\ x = \mathsf{new}\ L(3)\ \mathsf{in}\ x := 15;$
$\mathsf{call}\ f\ 2;$

Simulation-Based Proof

Set up cross-language relation $\approx_\beta$ that:

- knows trusted locations: $\tau \nvdash \circ$.
- splits heaps (source and **target**) into trusted and untrusted;
- constitutes trusted heap by trusted locations ($\tau \nvdash \circ$);
- relates trusted **heap** to trusted heap
- protects every trusted **location** by a capability;
- capability protecting a trusted **location** is not in attacker code, nor in the untrusted heap