Robustly Safe Compilation


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

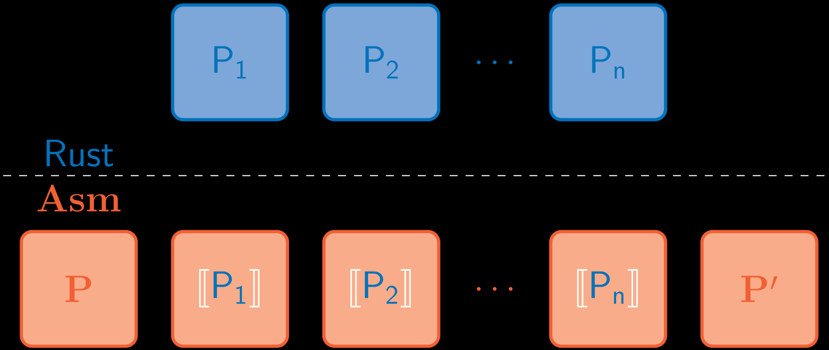

10^{th} April 2019

1 Stanford University

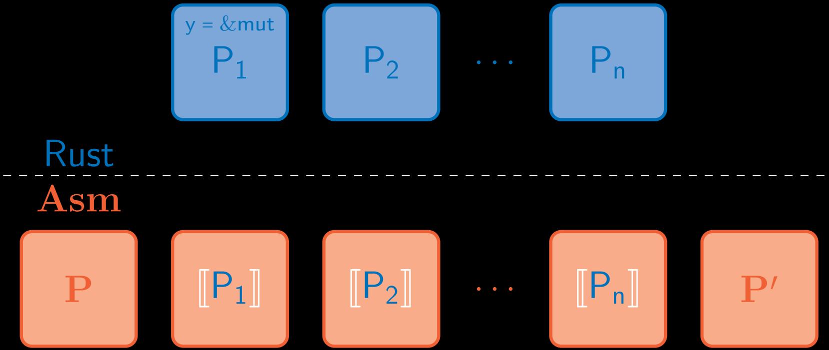
2 CISPA Helmholtz Center for Information Security

3 Max Planck Institute for Software Systems

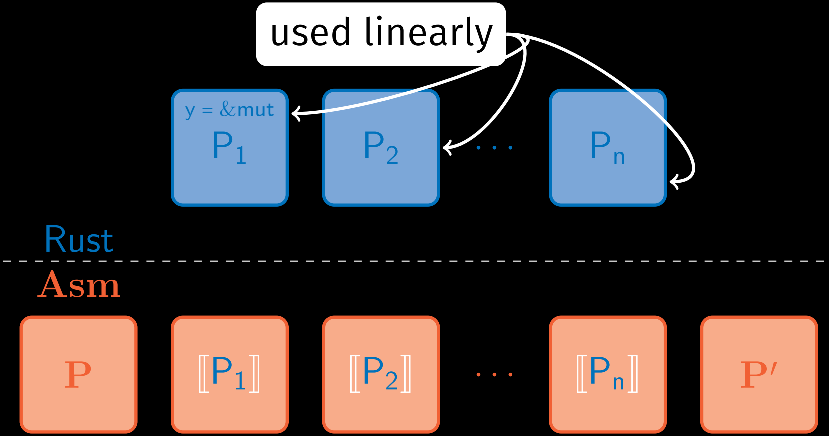
What is Secure Compilation?



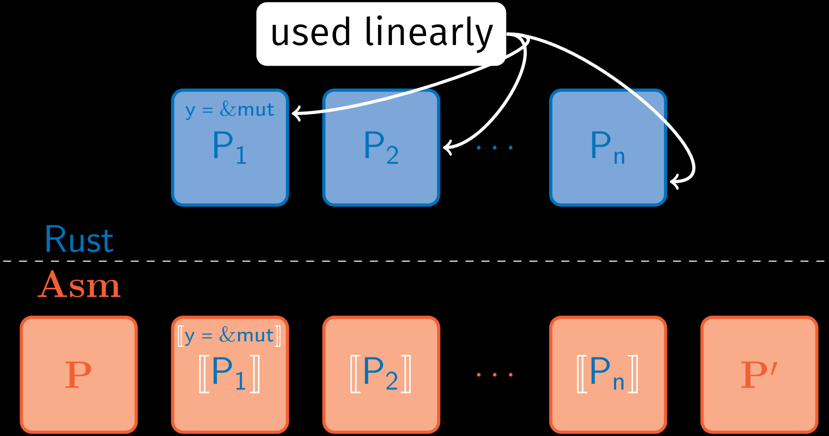
What is Secure Compilation?



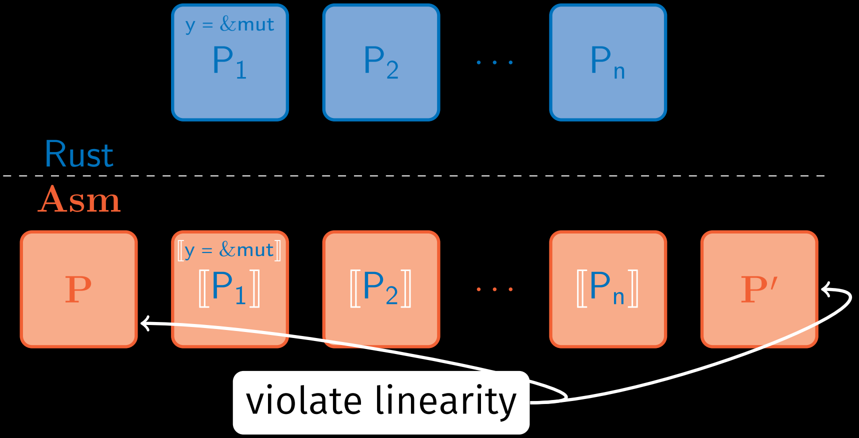
What is Secure Compilation?



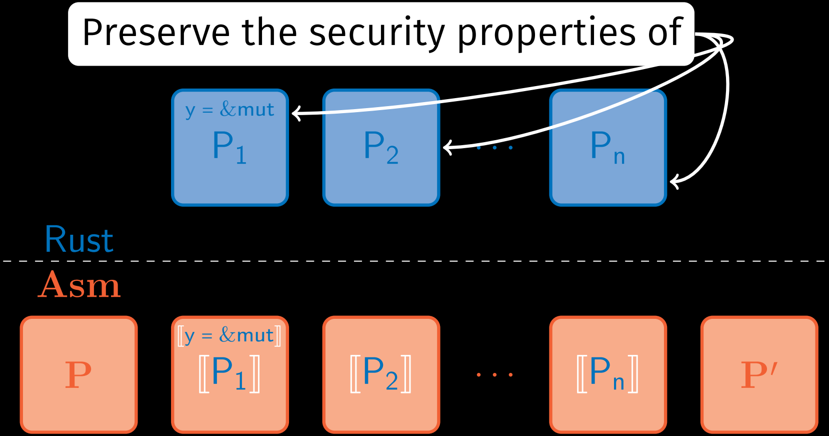
What is Secure Compilation?



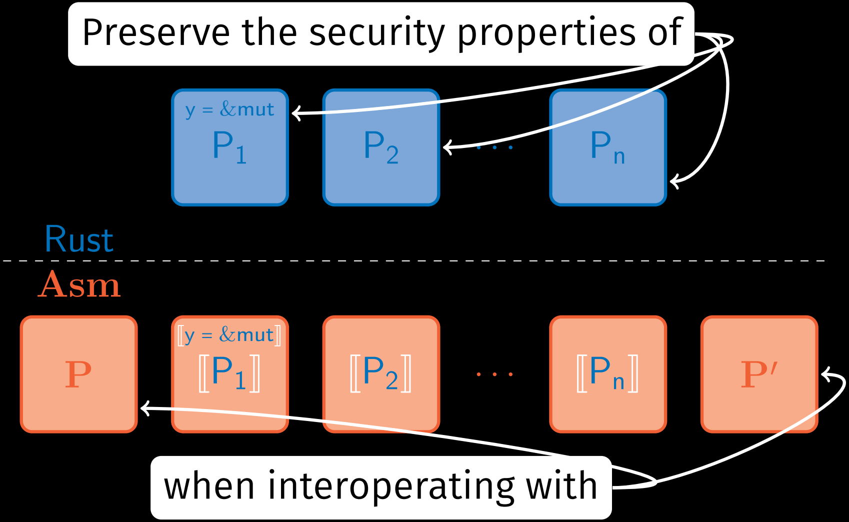
What is Secure Compilation?



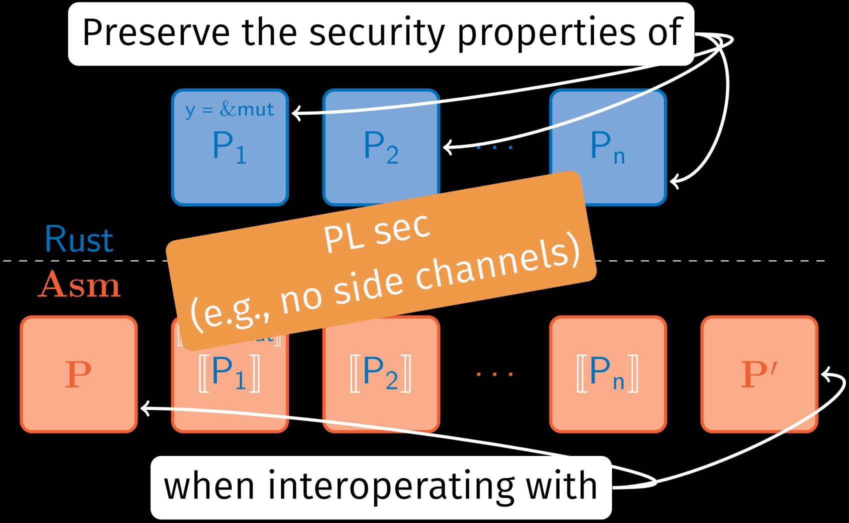
What is Secure Compilation?



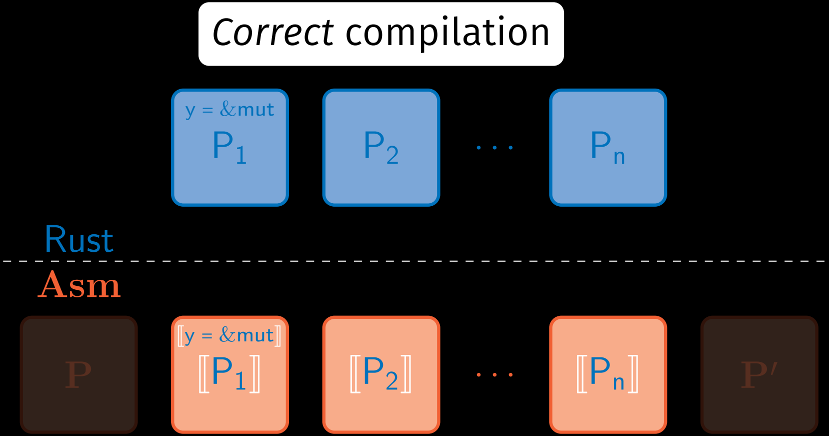
What is Secure Compilation?



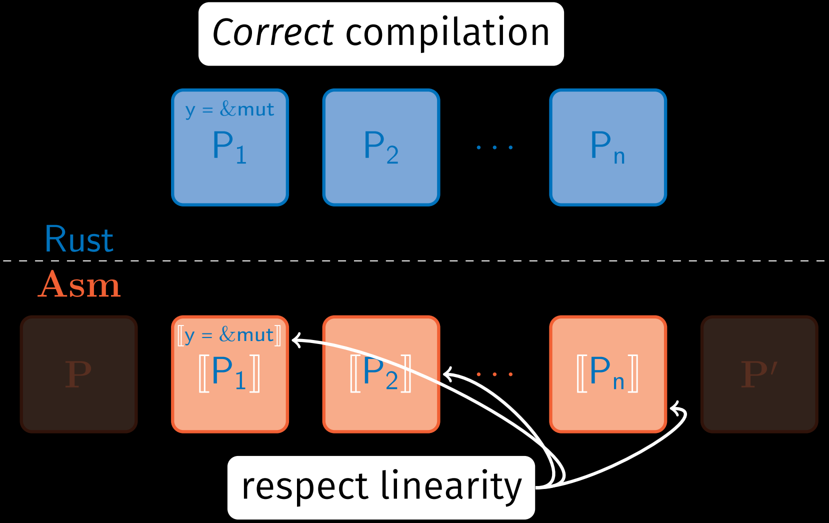
What is Secure Compilation?



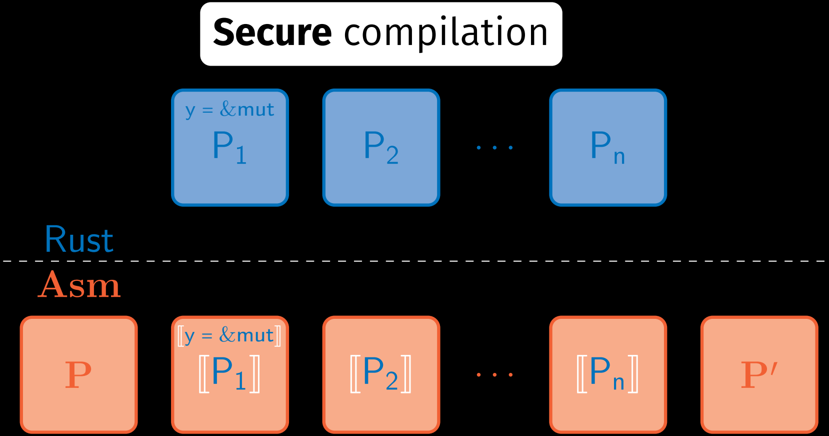
What is Secure Compilation?



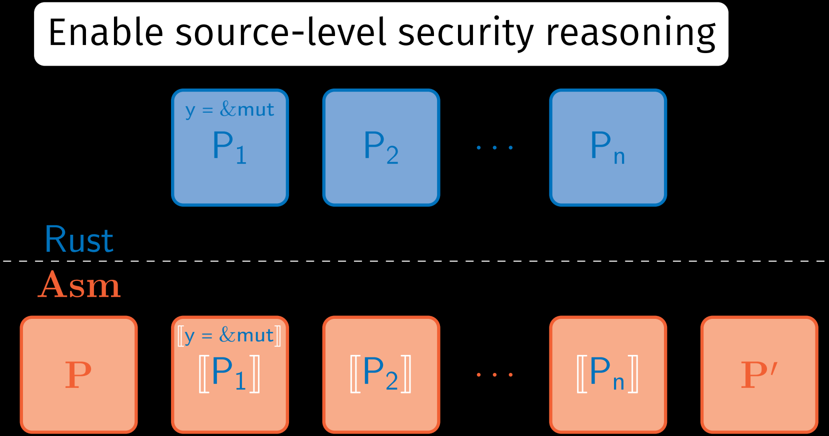
What is Secure Compilation?



What is Secure Compilation?



What is Secure Compilation?



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:

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

But...

Se secure compilers:

complex proofs

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

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: known criterion, meets our goals

Contributions

- RSC: known criterion, meets our goals
  - a compiler preserves all safety properties

Contributions

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

Contributions

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

Contributions

- RSC: known criterion, meets 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: known criterion, meets 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: known criterion, meets 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: known criterion, meets 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: known criterion, meets 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: known criterion, meets 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!

part 2

- two proof techniques for RSC
  - simplifications on existing ones
- a comparison between RSC and FAC

Talk Roadmap

Robust Safety

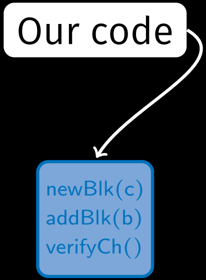
Robustly Safe Compilation

Backtranslation Proof Technique

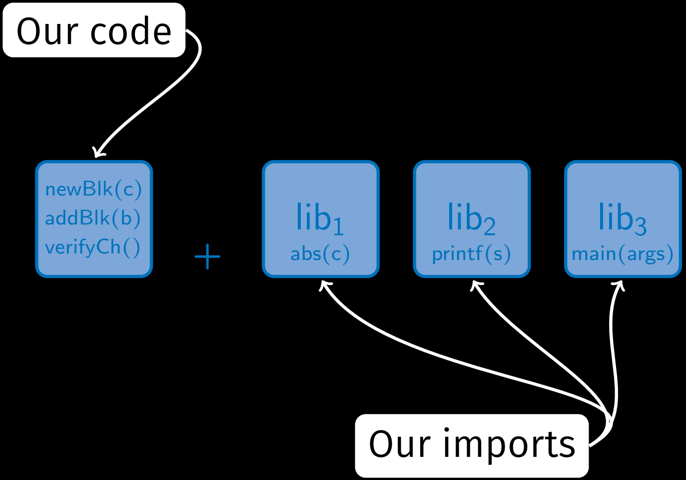
Robust Safety

- Robustness
- Behaviour
- Safety

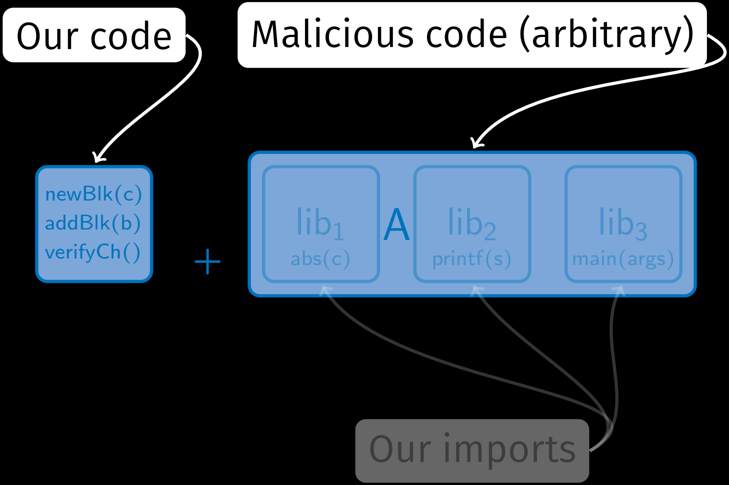
Robustness



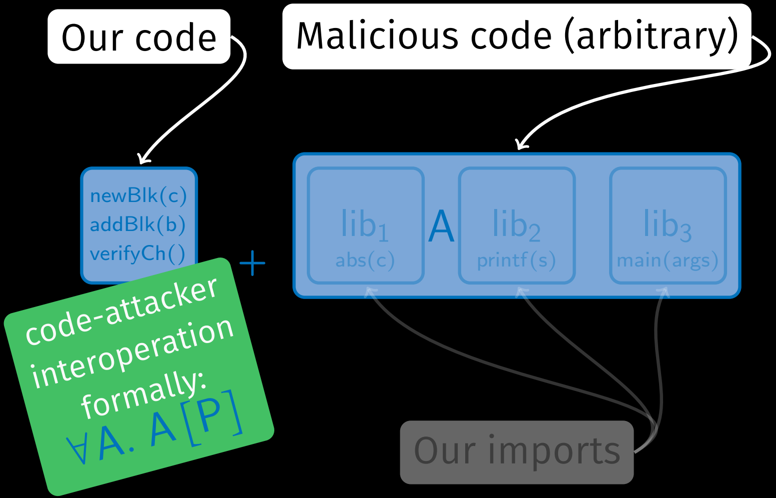
Robustness


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

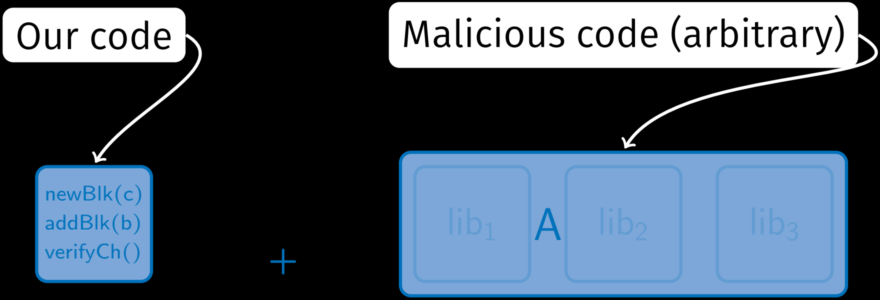
Robustness


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

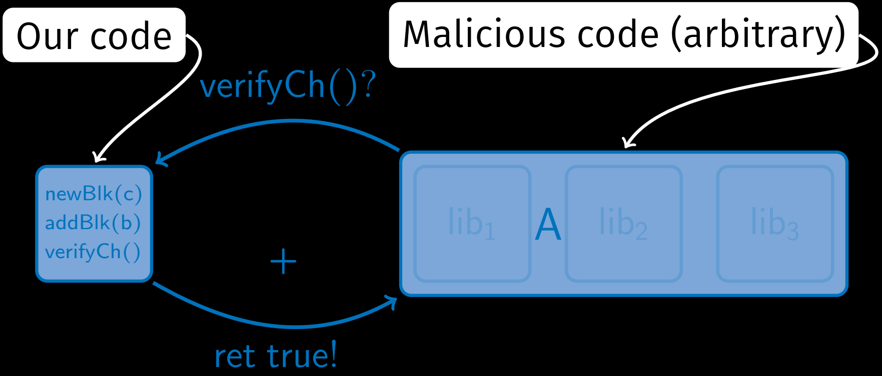
Robustness


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

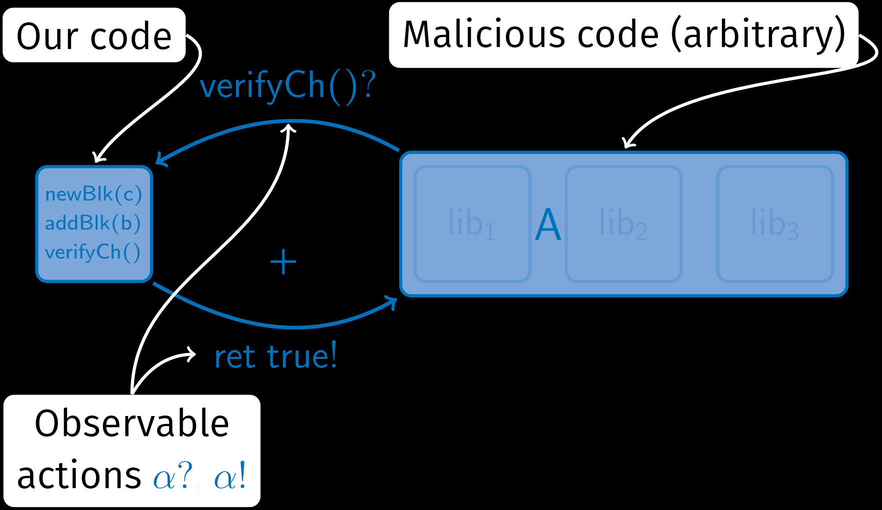
Program Behaviour



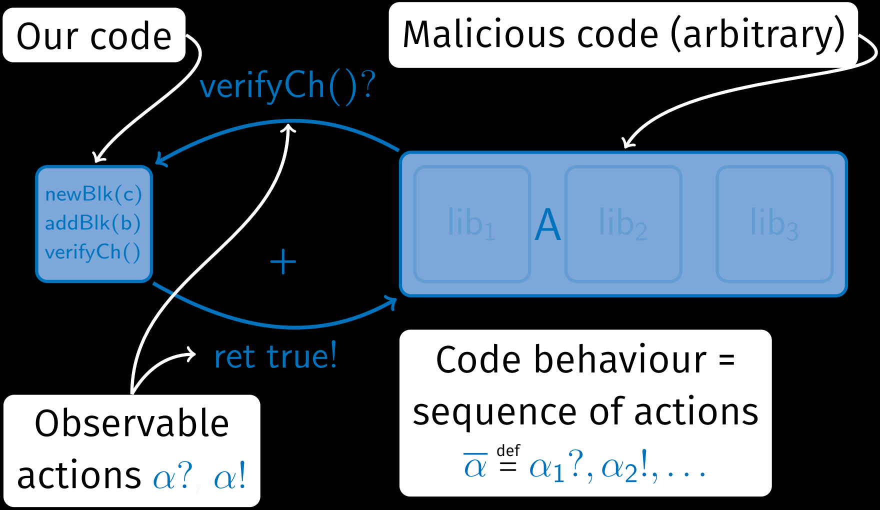
Program Behaviour



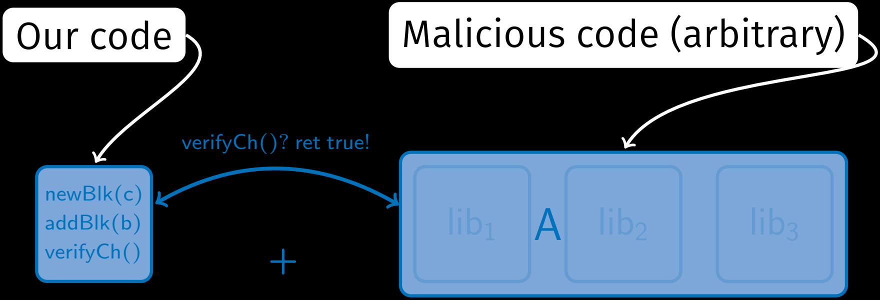
Program Behaviour



Program Behaviour



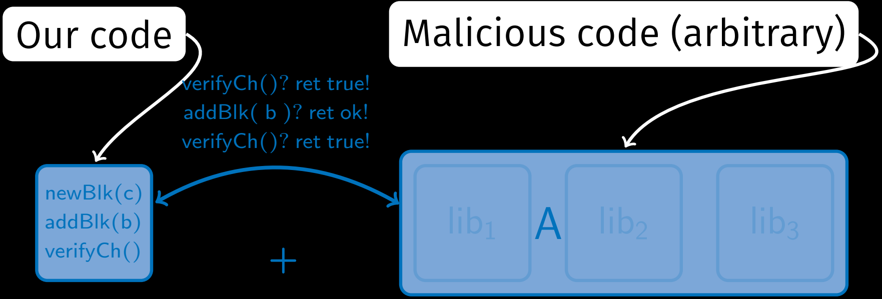
Program Behaviour



Code behaviour = sequence of actions

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

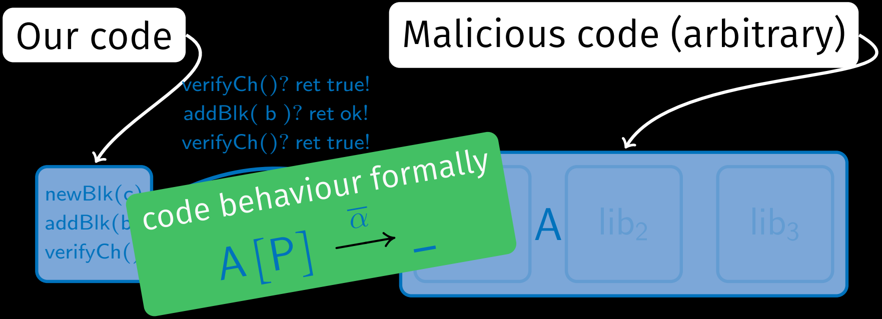
Program Behaviour



Code behaviour =
sequence of actions

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

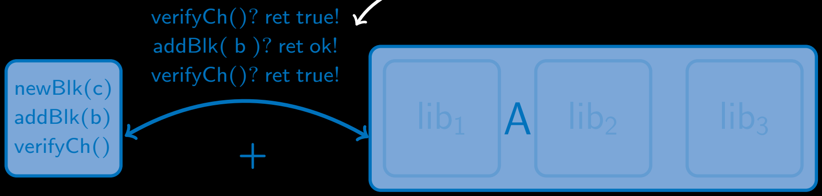
Program Behaviour



Code behaviour =
sequence of actions
$\overline{\alpha} \stackrel{\text{def}}{=} \alpha_1?, \alpha_2!, \ldots$

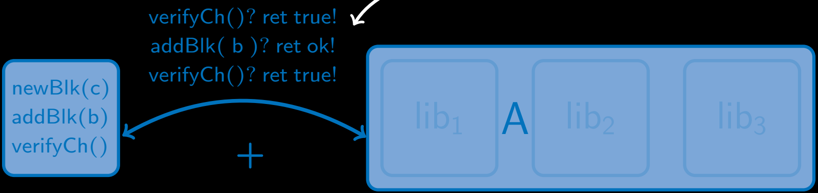
Safety Properties

no bad thing happens (finitely)



Safety Properties

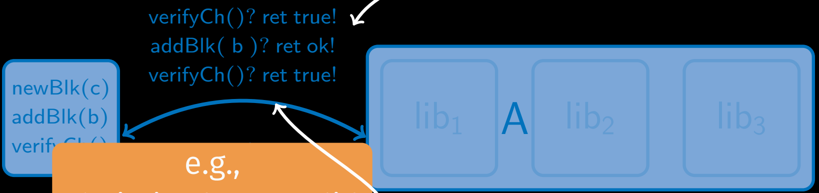
no bad thing happens (finitely)



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

Safety Properties

no bad thing happens (finitely)

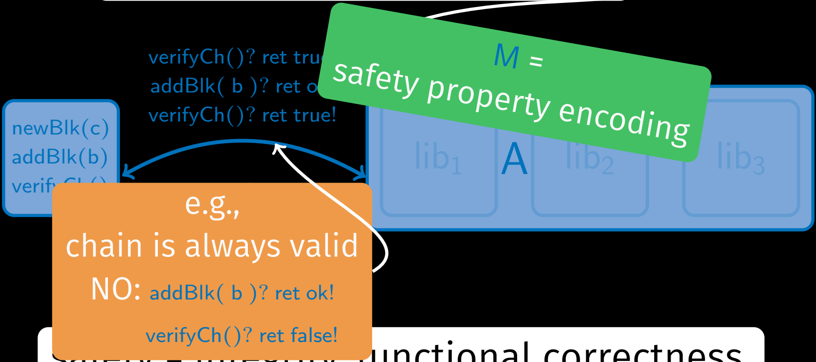


e.g.,
chain is always valid
NO: addBlk(b)? ret ok!
verifyCh()? ret false!

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

Safety Properties

no bad thing happens (finitely)



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

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 $\mathsf{A}\left[\mathsf{P}\right] \xrightarrow{\overline{\alpha}}$)
- that respects that safety property (then $\mathsf{M} \vdash \overline{\alpha}$)

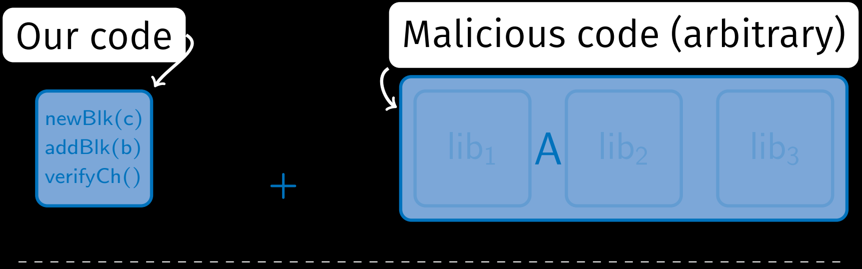
Robust Safety

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

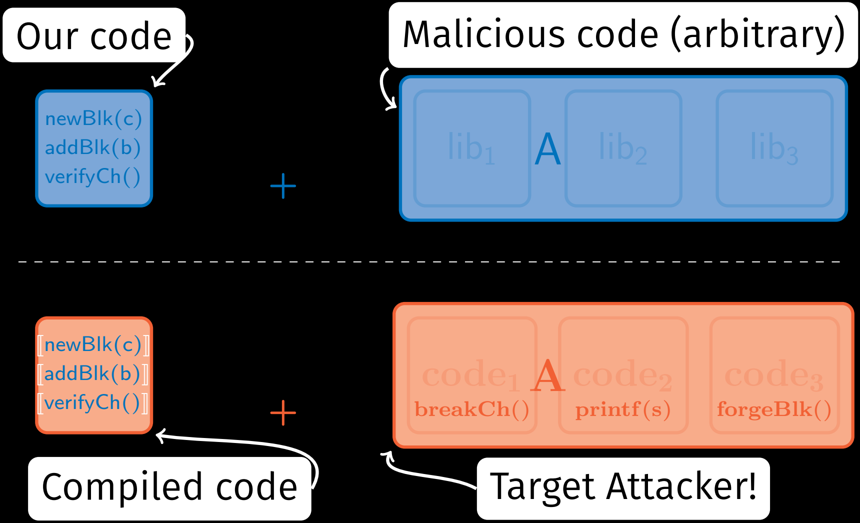
robust safety formally
$\mathsf{M} \vdash \mathsf{P}$

Robustly Safe Compilation

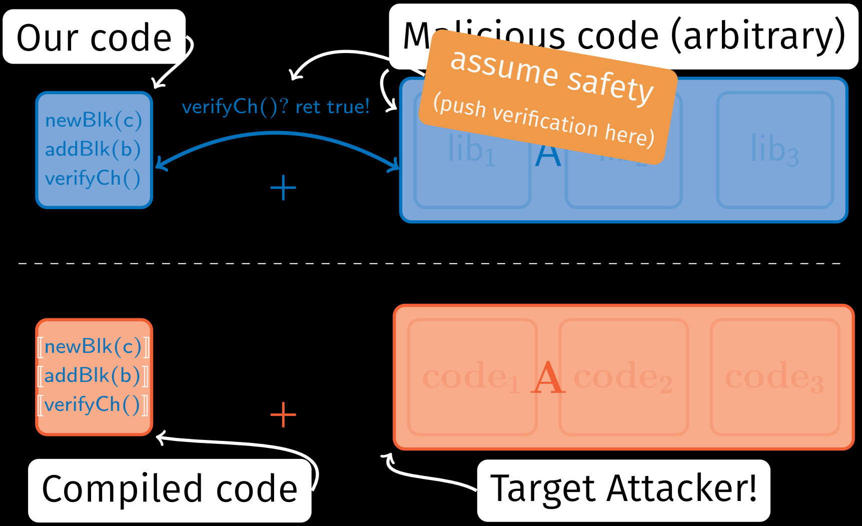
Robust Safety Across Compilation



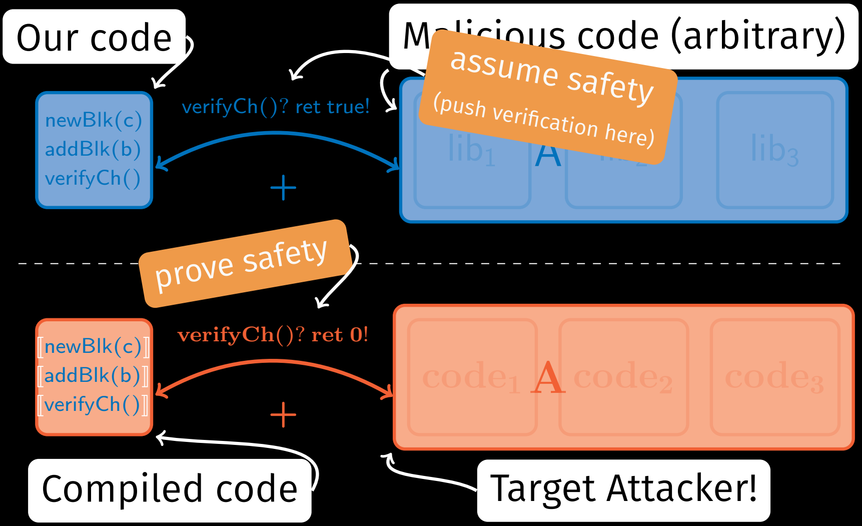
Robust Safety Across Compilation



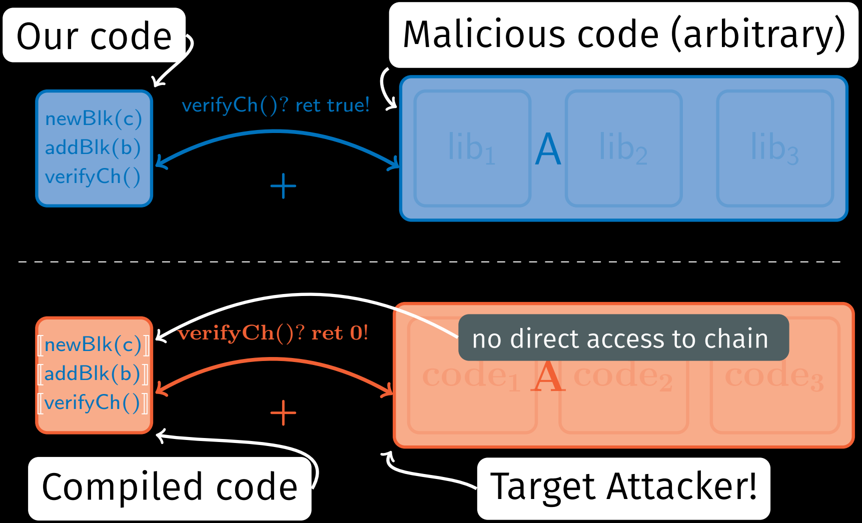
Robust Safety Across Compilation



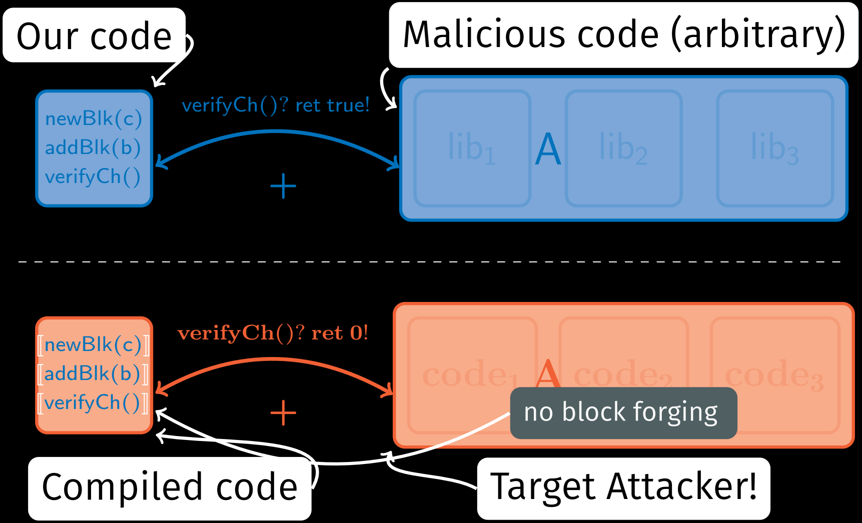
Robust Safety Across Compilation



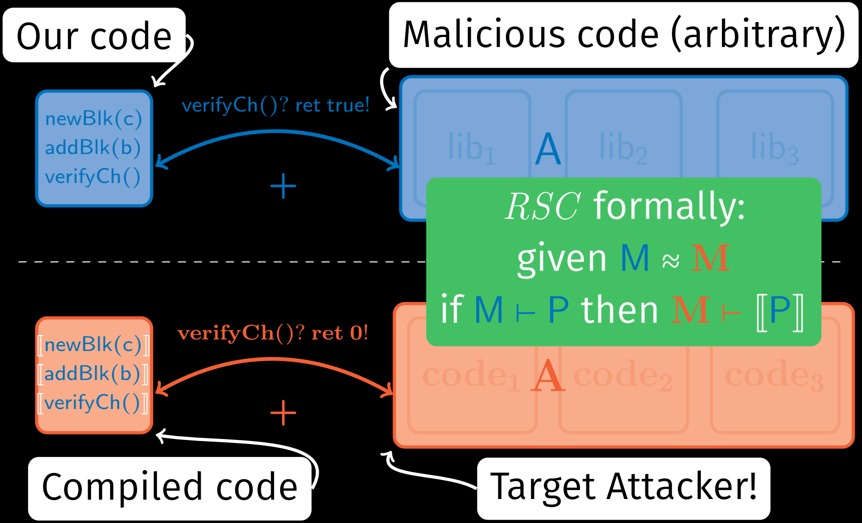
Robust Safety Across Compilation



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

$PF\text{-}RSC$: equivalent definition easier to prove than RSC

Concerns

RSC so far:

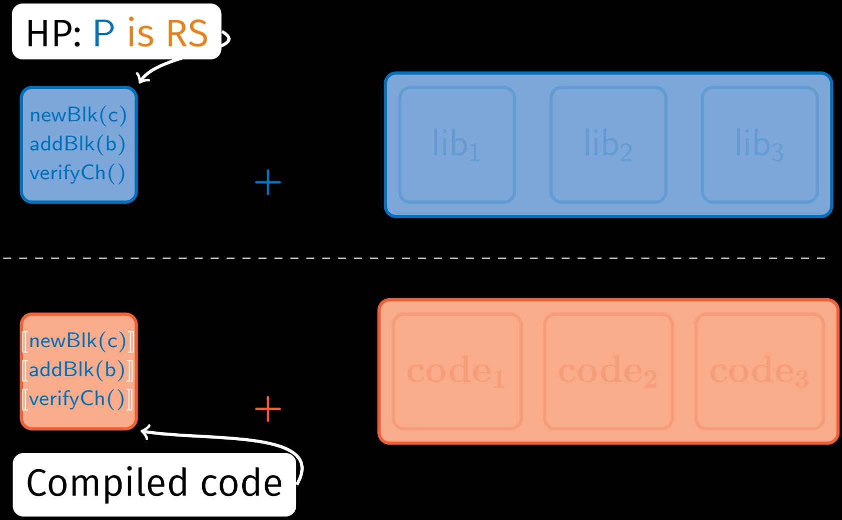
- attainable
- efficient
- possibly tricky to prove

PF-RSC: equivalent definition easier to prove than RSC

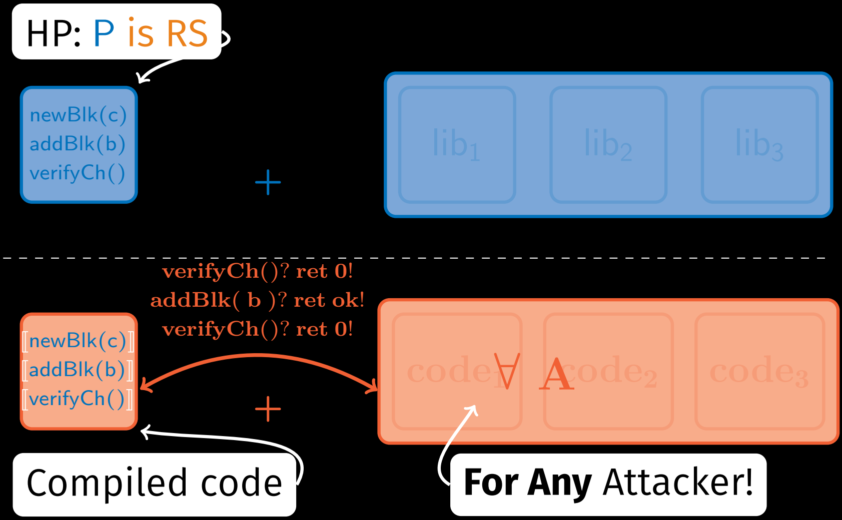
(equivalence to be proven, generally true)

Backtranslation Proof Technique

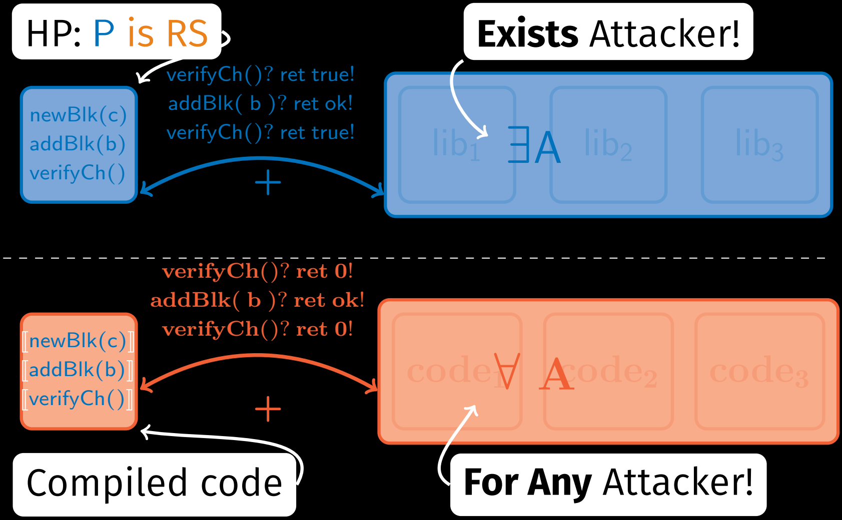
Backtranslation: Build A From A or $\overline{\alpha}$



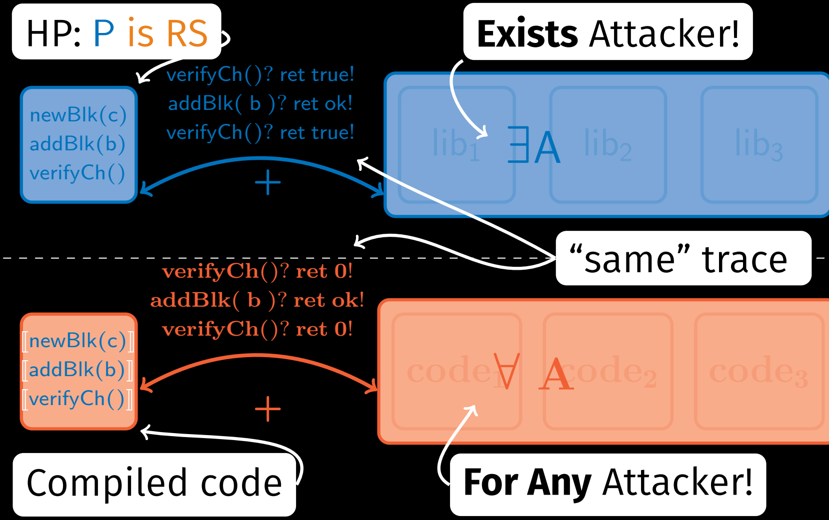
Backtranslation: Build A From A or $\overline{\alpha}$



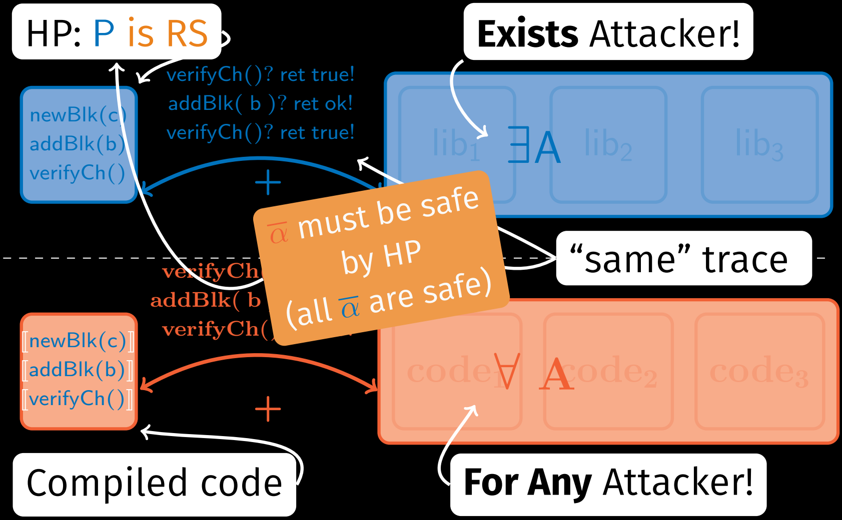
Backtranslation: Build A From $\mathbf{A}$ or $\overline{\alpha}$



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 were to happen it would finitely

Safety as a Dual and PF-RSC

- Safety = nothing bad happens
  so if it were to happen it would finitely

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

Safety as a Dual and PF-RSC

- Safety = nothing bad happens
  so if it were to happen it would finitely
- given any **behaviour** ($\mathbf{A}\left[[\![\mathsf{P}]\!]\right] \xrightarrow{\overline{\alpha}}$)
- if we can replicate that ($\exists \mathsf{A}.\mathsf{A}\left[\mathsf{P}\right] \xrightarrow{\overline{\alpha}}$)

Safety as a Dual and PF-RSC

- Safety = nothing bad happens
  so if it were to happen it would finitely
- given any **behaviour** ($\mathbf{A}\,[\llbracket \mathsf{P} \rrbracket] \xrightarrow{\overline{\alpha}}$)
- if we can replicate that ($\exists \mathsf{A}.\mathsf{A}\,[\mathsf{P}] \xrightarrow{\overline{\alpha}}$)
- then $\overline{\alpha}$ is not bad

Safety as a Dual and PF-RSC

- Safety = nothing bad happens
  so if it were to happen it would 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) (for $\overline{\alpha} \approx \overline{\alpha}$)

Safety as a Dual and PF-RSC

- Safety = nothing bad happens
  so if it were to happen it would finitely
- given any $\mathbf{behav}$
- if we can replica
- then $\overline{\alpha}$ is not ba
  because $\overline{\alpha}$ does not violate safety
  (by RS of P) (for $\overline{\alpha} \approx \overline{\alpha}$)

PF-RSC formally:
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 $\forall \mathbf{A}.\mathbf{A}\left[[\![\mathsf{P}]\!]\right] \xrightarrow{\overline{\alpha}}$
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 $\mathbf{M} \vdash [\![\mathsf{P}]\!]$ $\iff$ 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)

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

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
- 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 the systems

Take Home Message

What to make of this result?

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

Take Home Message

What to make of this result?

- encode safety properties in the systems
- ensure the 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?

Backtranslation Example

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

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

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

main(z) $\mapsto$

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

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

let x = new L(1) in x := 55; (3)
let x = new L(3) in x := 15;
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