Demystifying information flow downgrading using the real–ideal paradigm

Suraaj Kanniwadi, Andrew Myers

in submission to POPL ‘27

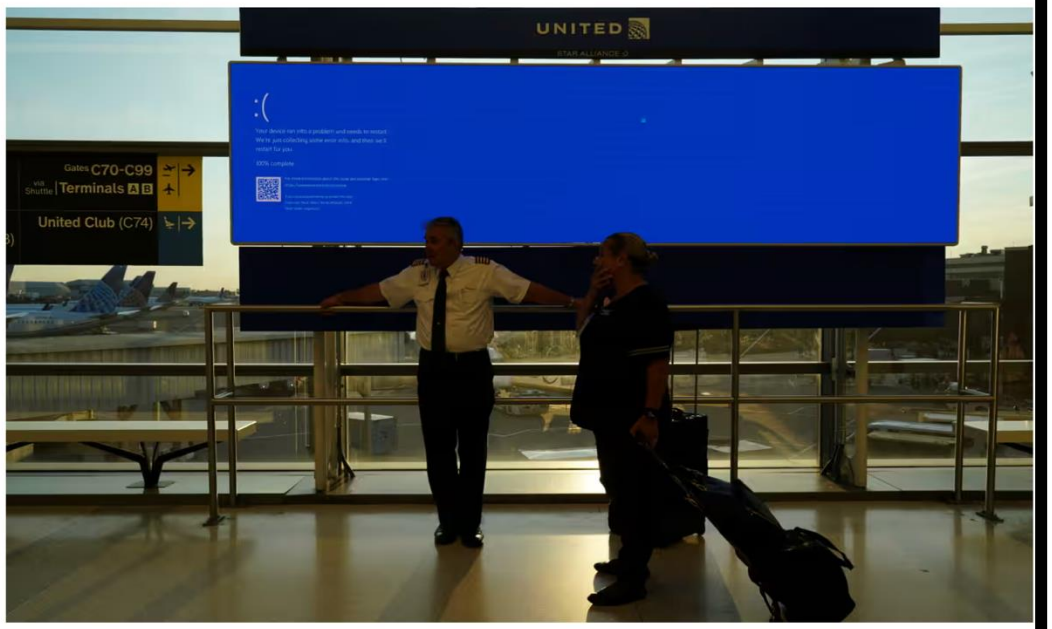
We need secure systems

We need secure systems

US transportation, police and hospital systems stricken by global CrowdStrike IT outage

US wakes up to Microsoft system collapse from software update that has crippled world IT systems

Windows global IT outage: what we know so far



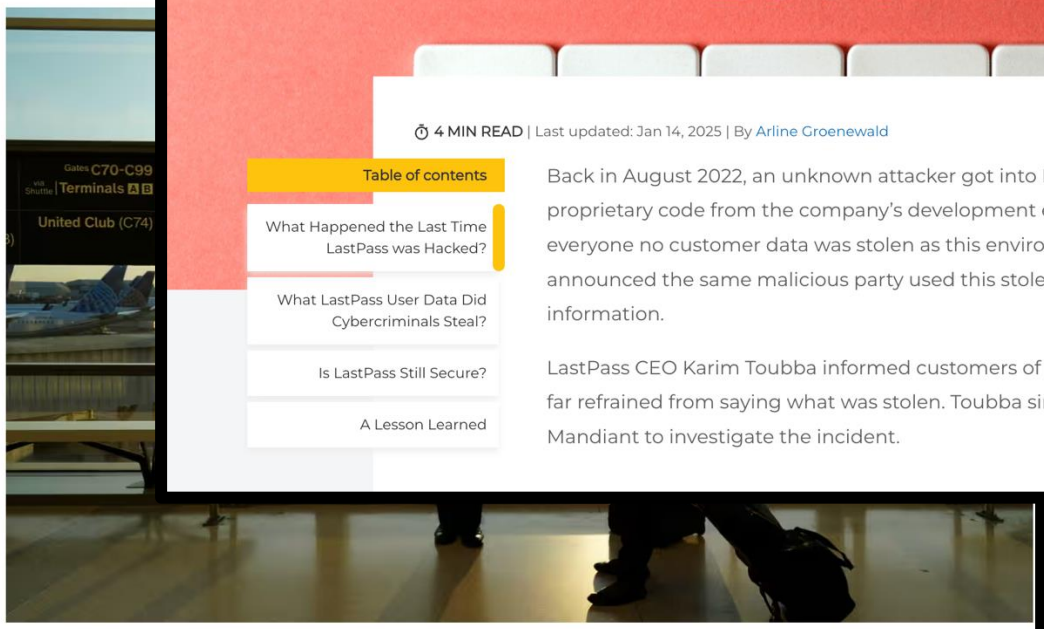
A Microsoft 'blue screen of death' us seen inside New Jersey's Newark international airport after CrowdStrike software crashed systems globally on Friday. Photograph: Bing Guan/Reuters

We need secure systems

US transportation, police and hospital systems stricken by global CrowdStrike IT outage

US wakes u
update that

Windows



A Microsoft 'blue screen of death' us seen inside New Jersey's Newark international airport after CrowdStrike software crashed systems globally on Friday. Photograph: Bing Guan/Reuters

Password Manager LastPass Suffers Data Breach After August Security Breach

4 MIN READ | Last updated: Jan 14, 2025 | By Arline Groenewald

Table of contents

- What Happened the Last Time LastPass was Hacked?
- What LastPass User Data Did Cybercriminals Steal?
- Is LastPass Still Secure?
- A Lesson Learned

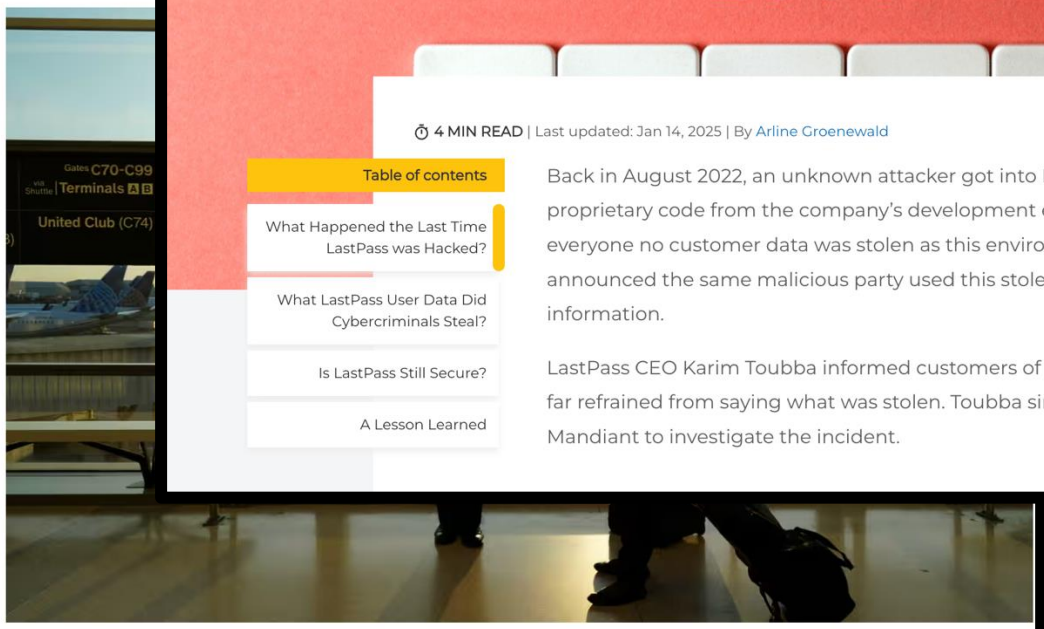
Back in August 2022, an unknown attacker got into LastPass's security systems and stole some proprietary code from the company's development environment. At the time, the company assured everyone no customer data was stolen as this environment doesn't contain any. Now, the company announced the same malicious party used this stolen data to access "certain elements" of customers' information.

LastPass CEO Karim Toubba informed customers of the breach in a November 30 blog post, but has so far refrained from saying what was stolen. Toubba simply stated LastPass is working with security firm Mandiant to investigate the incident.

We need secure systems

US transportation, police and hospital systems stricken by global CrowdStrike IT outage

US wakes u
update that
Windows



A Microsoft 'blue screen of death' us seen inside New Jersey's Newark international airport after CrowdStrike software crashed systems globally on Friday. Photograph: Bing Guan/Reuters

Password Manager LastPass Suffers Da
Security Breach

4 MIN READ | Last updated: Jan 14, 2025 | By Arline Groenewald

Table of contents

- What Happened the Last Time LastPass was Hacked?
- What LastPass User Data Did Cybercriminals Steal?
- Is LastPass Still Secure?
- A Lesson Learned

Back in August 2022, an unknown attacker got into LastPass's sec
proprietary code from the company's development environment.
everyone no customer data was stolen as this environment doesn
announced the same malicious party used this stolen data to acc
information.

LastPass CEO Karim Toubba informed customers of the breach in
far refrained from saying what was stolen. Toubba simply stated L
Mandiant to investigate the incident.

THE DAO HACK

The DAO was a decentralized autonomous organization on Ethereum, designed to manage investments through smart contracts, without traditional intermediaries.

JUNE 17, 2016

An attacker exploited a vulnerability in the DAO's smart contract.

– $60 MILLION

Drained in ETH, causing widespread panic in the Ethereum community.

HOW WAS THE DAO HACKED?

Reentrancy attack

The attacker used a "reentrancy attack," repeatedly calling the DAO's withdrawal function before the contract's balance updated. This flaw allowed funds to be siphoned without reducing the balance.

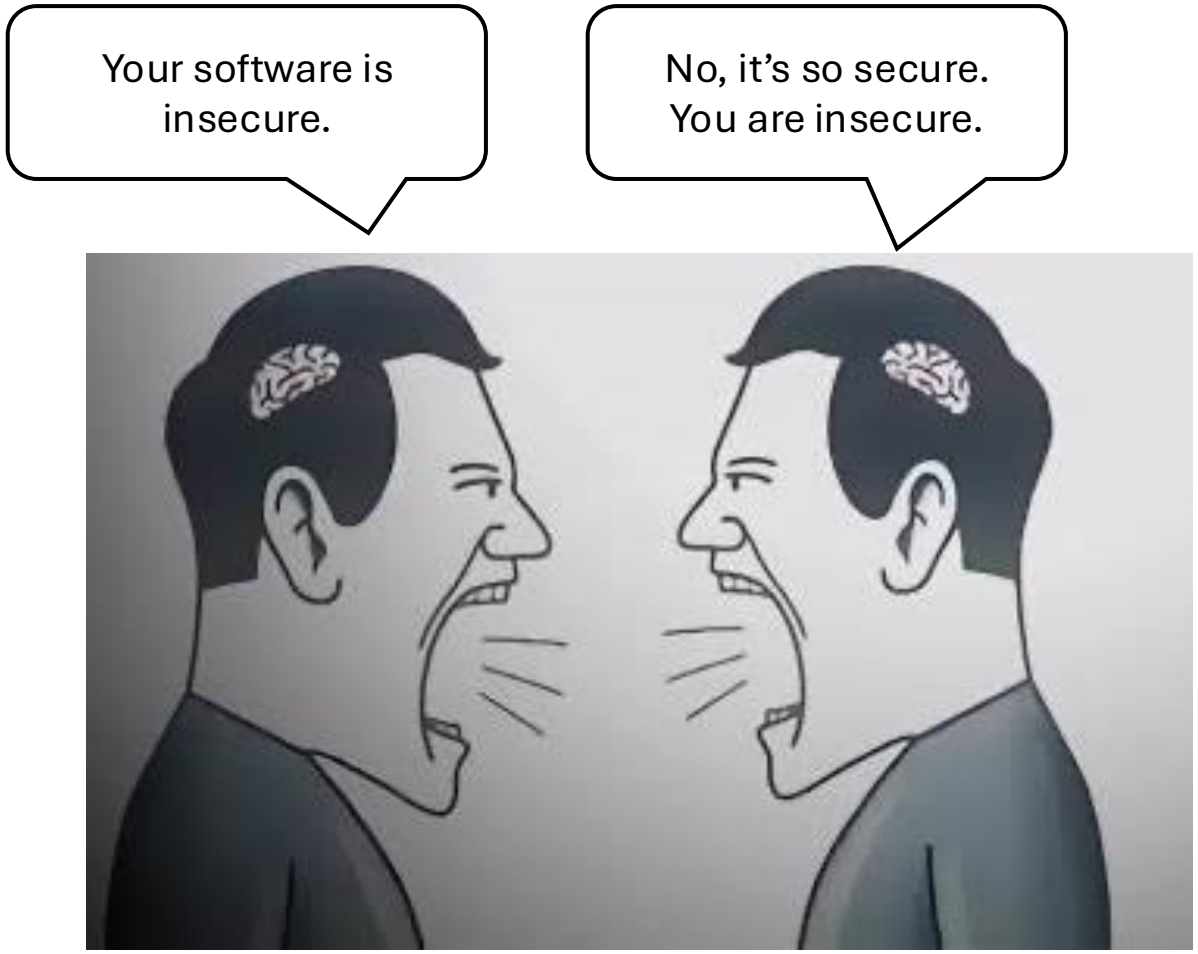
WHO HACKED THE DAO?

Unknown

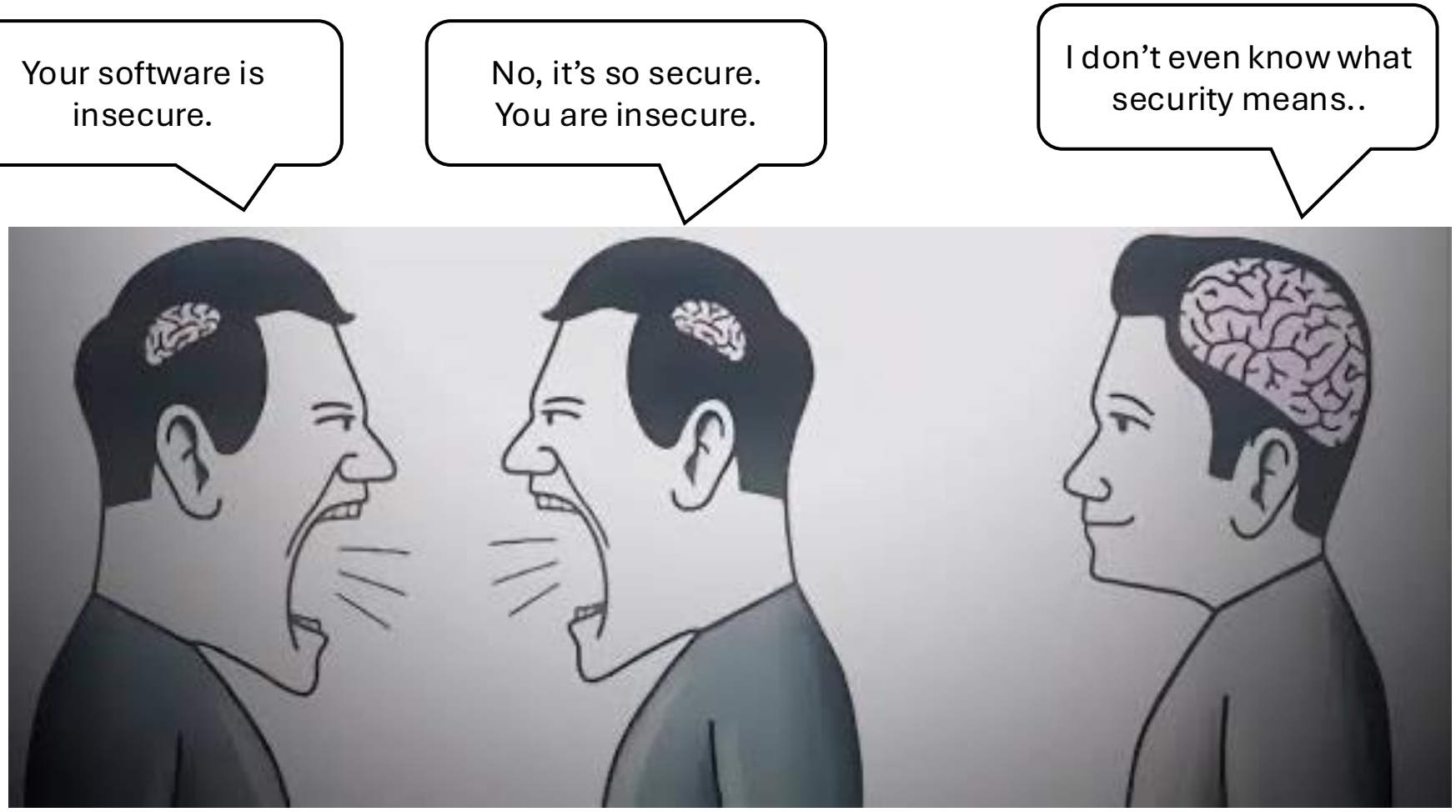
The identity of the DAO hacker remains unknown. Despite extensive investigation, the attacker has not been identified, and theories about who might be responsible are largely speculative.

What is security?

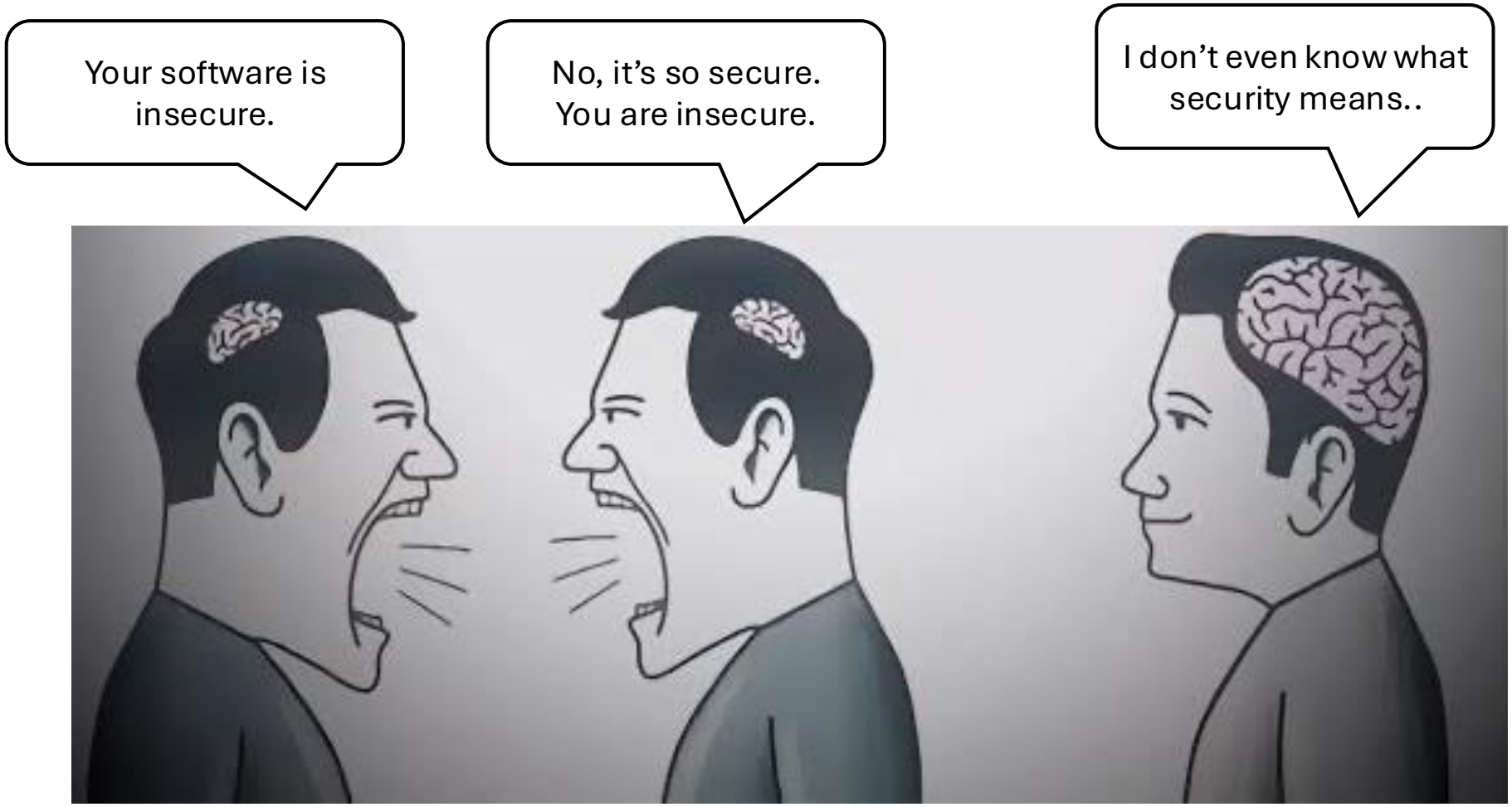
What is security?



What is security?



What is security?



The first step in building secure systems is to specify precisely what security means!

Surely, we know how to write specifications..

Surely, we know how to write specifications..

The existence of refinement mappings
Marin Abadi, Leslie Lamport

A lattice of information
Jaisook Landauer, Tmothy Redmond

Hyperproperties
Micheal R. Clarkson, Fred B. Schneider

Surely, we know how to write specifications..

The existence of refinement mappings
Marin Abadi, Leslie Lamport

The temporal logic of actions
Leslie Lamport

Temporal logics for hyperproperties
Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, César Sánchez

A lattice of information
Jaisook Landauer, Tmothy Redmond

Hyperproperties
Micheal R. Clarkson, Fred B. Schneider

Surely, we know how to write specifications..

An axiomatic basis for computer programming
C. A. R. Hoare

The existence of refinement mappings
Marin Abadi, Leslie Lamport

The temporal logic of actions
Leslie Lamport

JFlow: practical mostly-static information flow control
Andrew C. Myers

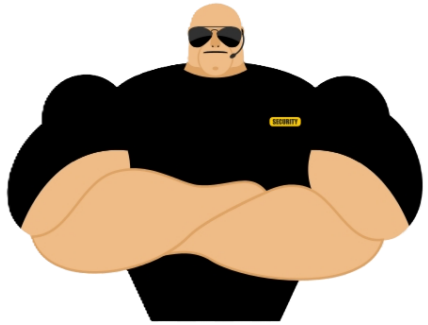
A lattice model of secure information flow
Dorothy E. Denning

A lattice of information
Jaisook Landauer, Tmothy Redmond

Temporal logics for hyperproperties
Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, César Sánchez

Hyperproperties
Micheal R. Clarkson, Fred B. Schneider

Surely, we know how to write specifications..



An axiomatic basis for computer programming
C. A. R. Hoare

The existence of refinement mappings
Marin Abadi, Leslie Lamport

The temporal logic of actions
Leslie Lamport

JFlow: practical mostly-static information flow control
Andrew C. Myers

A lattice model of secure information flow
Dorothy E. Denning

A lattice of information
Jaisook Landauer, Tmothy Redmond

Temporal logics for hyperproperties
Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, César Sánchez

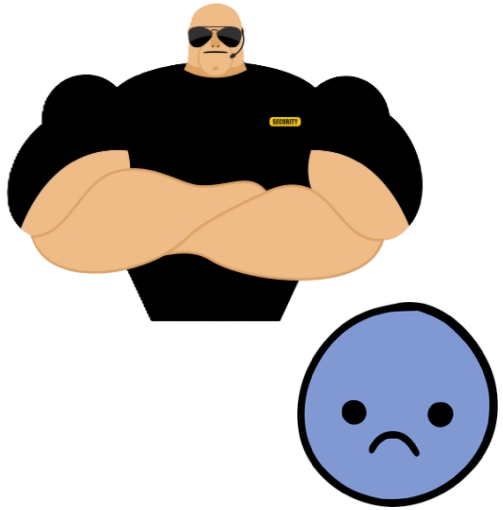
Hyperproperties
Micheal R. Clarkson, Fred B. Schneider

Surely, we know how to write specifications..

An axiomatic basis for computer programming
C. A. R. Hoare

The existence of refinement mappings
Marin Abadi, Leslie Lamport

The temporal logic of actions
Leslie Lamport



JFlow: practical mostly-static information flow control
Andrew C. Myers

A lattice model of secure information flow
Dorothy E. Denning

A lattice of information
Jaisook Landauer, Tmothy Redmond

Temporal logics for hyperproperties
Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, César Sánchez

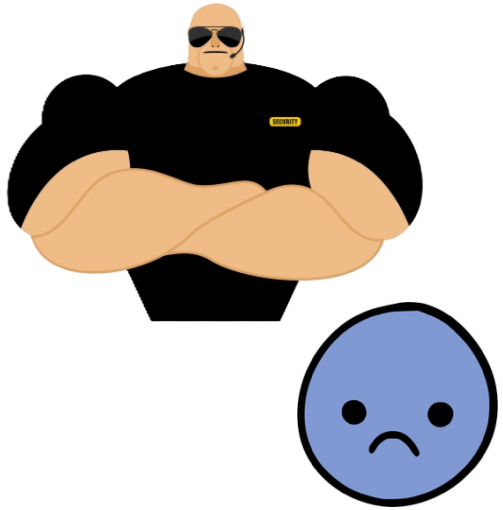
Hyperproperties
Micheal R. Clarkson, Fred B. Schneider

Surely, we know how to write specifications..

An axiomatic basis for computer programming
C. A. R. Hoare

The existence of refinement mappings
Marin Abadi, Leslie Lamport

The temporal logic of actions
Leslie Lamport



JFlow: practical mostly-static information flow control
Andrew C. Myers

A lattice model of secure information flow
Dorothy E. Denning

A lattice of information
Jaisook Landauer, Tmothy Redmond

Temporal logics for hyperproperties
Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, César Sánchez

Hyperproperties
Micheal R. Clarkson, Fred B. Schneider

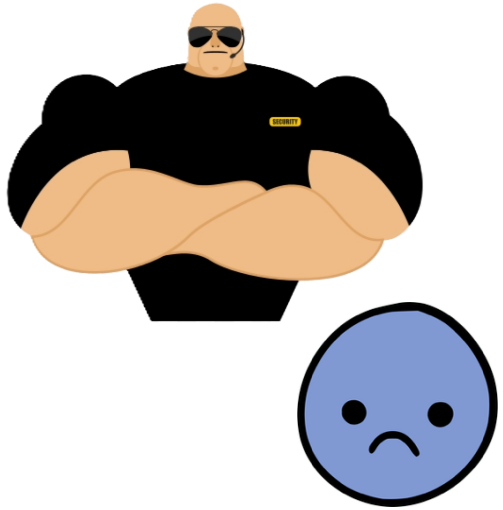
Surely, we know how to write specifications..

Trace refinement

An axiomatic basis for computer programming
C. A. R. Hoare

The existence of refinement mappings
Marin Abadi, Leslie Lamport

The temporal logic of actions
Leslie Lamport



JFlow: practical mostly-static information flow control
Andrew C. Myers

A lattice model of secure information flow
Dorothy E. Denning

A lattice of information
Jaisook Landauer, Tmothy Redmond

Temporal logics for hyperproperties
Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, César Sánchez

Hyperproperties
Micheal R. Clarkson, Fred B. Schneider

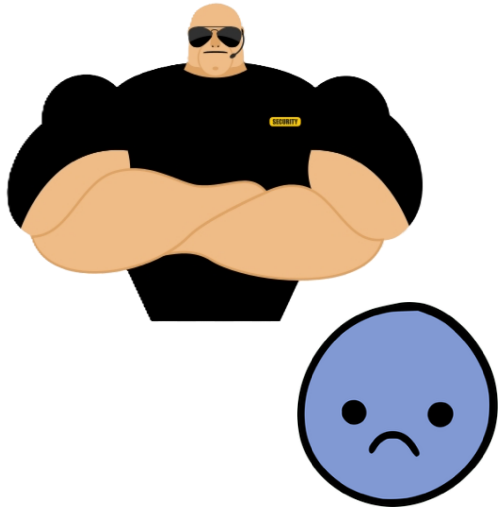
Surely, we know how to write specifications..

Trace refinement

An axiomatic basis for computer programming
C. A. R. Hoare

The existence of refinement mappings
Marin Abadi, Leslie Lamport

The temporal logic of actions
Leslie Lamport



JFlow: practical mostly-static information flow control
Andrew C. Myers

A lattice model of secure information flow
Dorothy E. Denning

A lattice of information
Jaisook Landauer, Tmothy Redmond

Noninterference

Temporal logics for hyperproperties
Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, César Sánchez

Hyperproperties
Micheal R. Clarkson, Fred B. Schneider

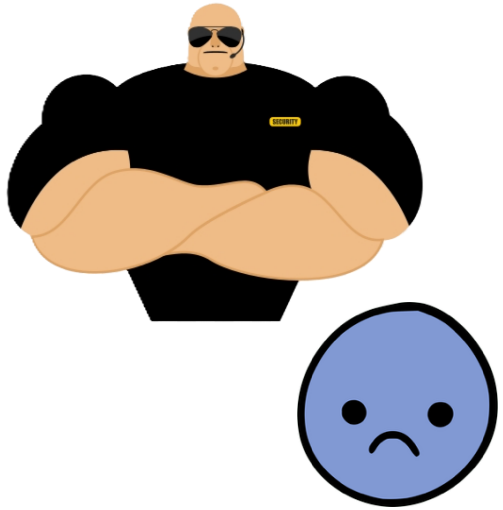
Surely, we know how to write specifications..

Trace refinement

An axiomatic basis for computer programming
C. A. R. Hoare

The existence of refinement mappings
Marin Abadi, Leslie Lamport

The temporal logic of actions
Leslie Lamport



JFlow: practical mostly-static information flow control
Andrew C. Myers

A lattice model of secure information flow
Dorothy E. Denning

A lattice of information
Jaisook Landauer, Tmothy Redmond

Noninterference

Temporal logics for hyperproperties
Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, César Sánchez

Hyperproperties
Micheal R. Clarkson, Fred B. Schneider

Hyperproperties

There are limitations

Trace Refinement

Noninterference

Hyperproperties

There are limitations

Trace Refinement

“Specifications are collections of permitted single executions”

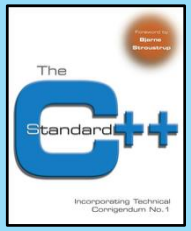
Noninterference

Hyperproperties

There are limitations

Trace Refinement

"Specifications are collections of permitted single executions"



Noninterference

Hyperproperties

There are limitations

Trace Refinement

"Specifications are collections of permitted single executions"



Noninterference

Hyperproperties

There are limitations

Trace Refinement

"Specifications are collections of permitted single executions"



Noninterference

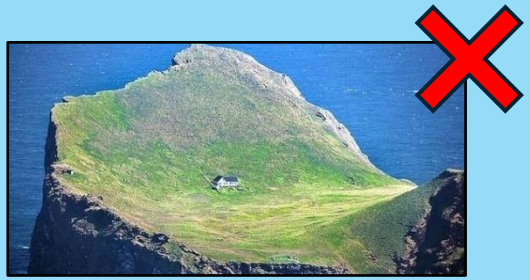
Hyperproperties

There are limitations

Trace Refinement

“Specifications are collections of permitted single executions”





Noninterference

Hyperproperties

There are limitations

Trace Refinement

“Specifications are collections of permitted single executions”





Noninterference

Hyperproperties

There are limitations

Trace Refinement

“Specifications are collections of permitted single executions”





“Secret data must not flow to public data”

Noninterference

Hyperproperties

There are limitations

Trace Refinement

“Specifications are collections of permitted single executions”





“Secret data must not flow to public data”
“Untrusted data must not flow to trusted data”

Noninterference

Hyperproperties

There are limitations

Trace Refinement

"Specifications are collections of permitted single executions"





"Secret data must not flow to public data"
"Untrusted data must not flow to trusted data"



Noninterference

Hyperproperties

There are limitations

Trace Refinement

"Specifications are collections of permitted single executions"





"Secret data must not flow to public data"
"Untrusted data must not flow to trusted data"



Noninterference

Hyperproperties

There are limitations

Trace Refinement

"Specifications are collections of permitted single executions"





"Secret data must not flow to public data"
"Untrusted data must not flow to trusted data"





Noninterference

Hyperproperties

There are limitations

Trace Refinement

"Specifications are collections of permitted single executions"

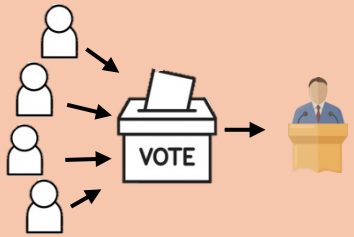




"Secret data must not flow to public data"
"Untrusted data must not flow to trusted data"







Noninterference

Hyperproperties

There are limitations

Trace Refinement

"Specifications are collections of permitted single executions"

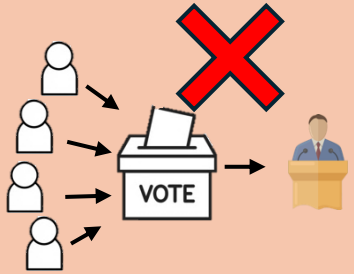




"Secret data must not flow to public data"
"Untrusted data must not flow to trusted data"







Noninterference

Hyperproperties

There are limitations

Trace Refinement

“Specifications are collections of permitted single executions”

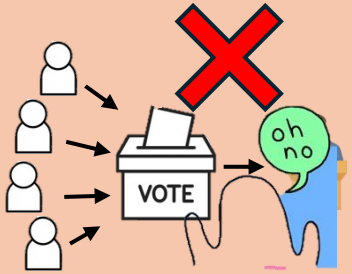




“Secret data must not flow to public data”
“Untrusted data must not flow to trusted data”







Noninterference

Hyperproperties

There are limitations

Trace Refinement

"Specifications are collections of permitted single executions"

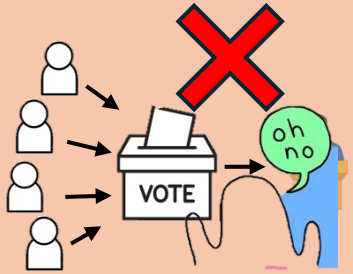




"Secret data must not flow to public data"
"Untrusted data must not flow to trusted data"







Noninterference

"Specifications are collections of permitted execution sets"

Hyperproperties

There are limitations

Trace Refinement

"Specifications are collections of permitted single executions"

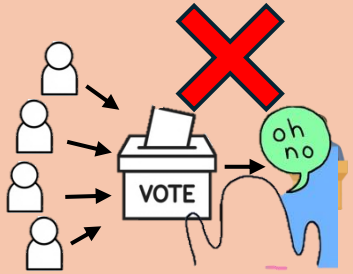




"Secret data must not flow to public data"
"Untrusted data must not flow to trusted data"

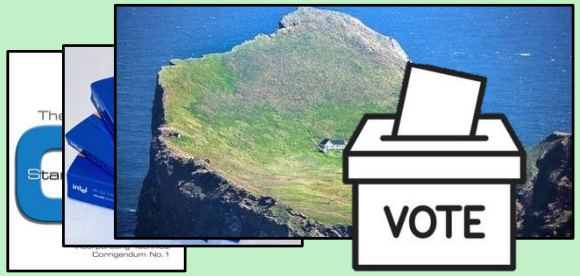






Noninterference

"Specifications are collections of permitted execution sets"



Hyperproperties

There are limitations

Trace Refinement

"Specifications are collections of permitted single executions"

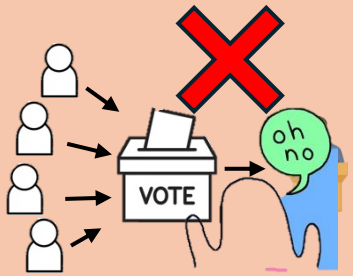




"Secret data must not flow to public data"
"Untrusted data must not flow to trusted data"

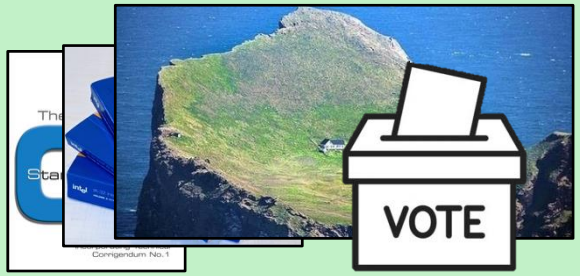






Noninterference

"Specifications are collections of permitted execution sets"



"A system is secure if it is in the set of secure systems"

Hyperproperties

There are limitations

Trace Refinement

"Specifications are collections of permitted single executions"

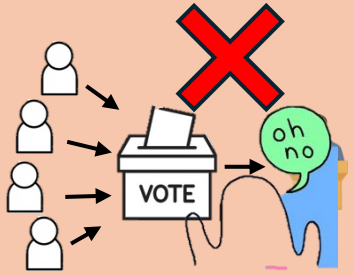




"Secret data must not flow to public data"
"Untrusted data must not flow to trusted data"

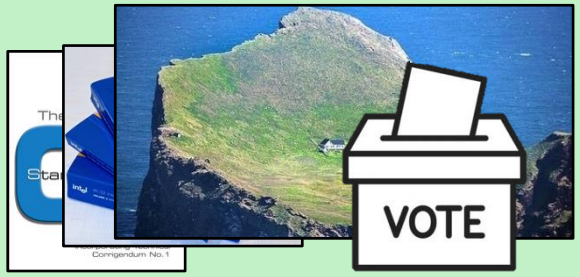






Noninterference

"Specifications are collections of permitted execution sets"



"A system is secure if it is in the set of secure systems"



Hyperproperties

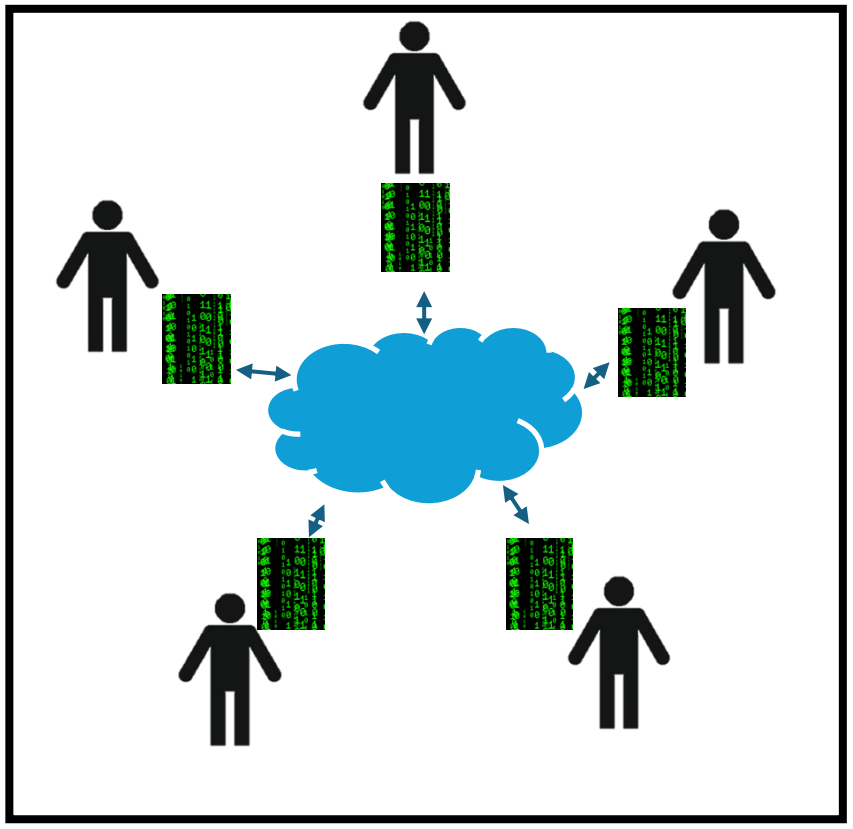
What does cryptography do?

What does cryptography do?



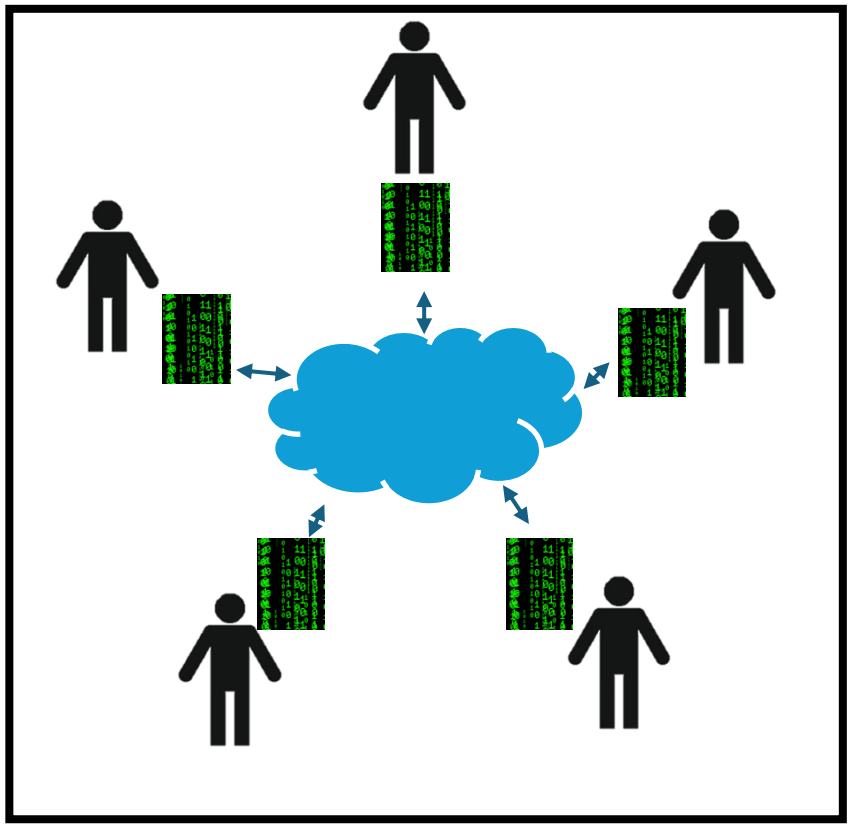
What does cryptography do?

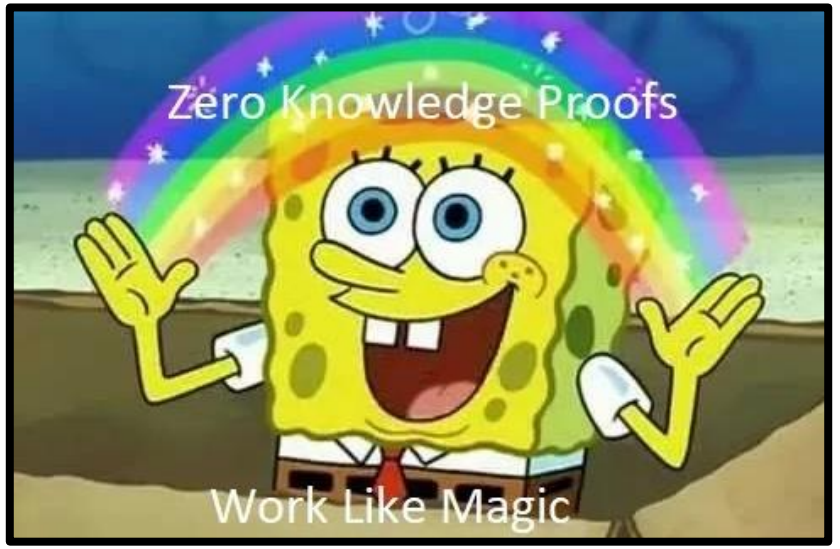




What does cryptography do?

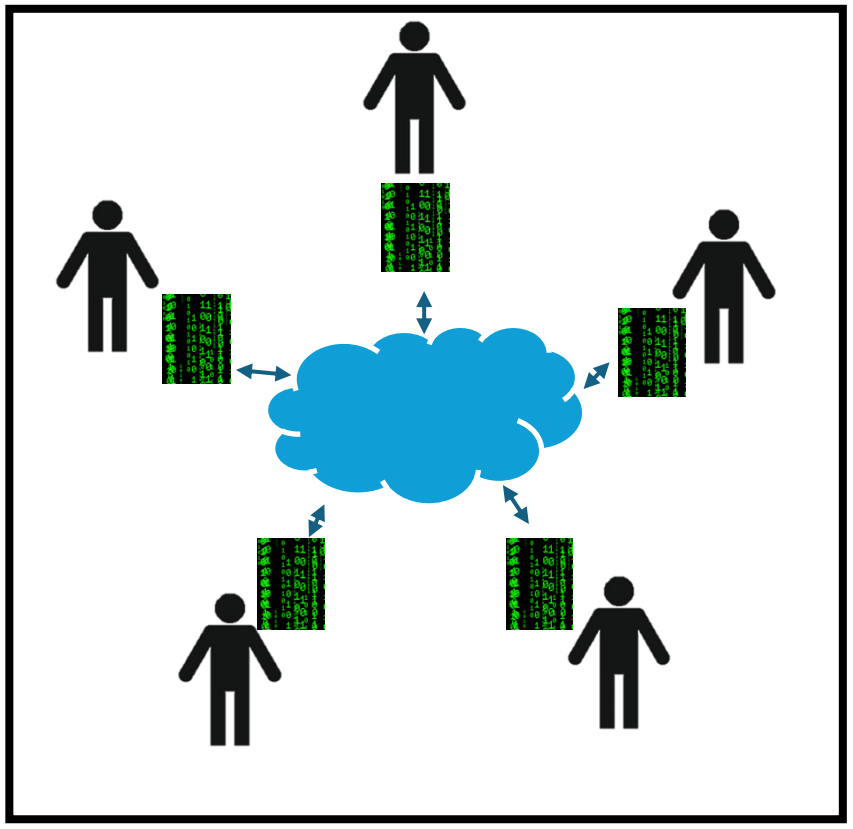


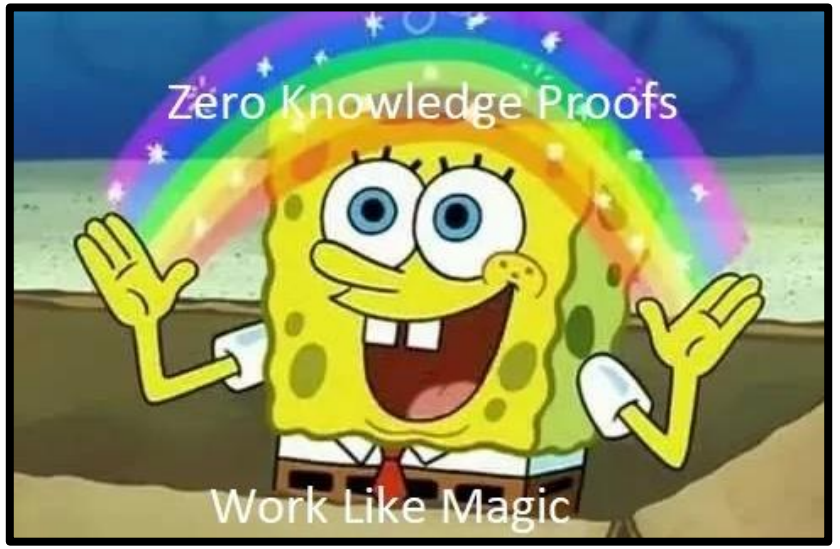




What does cryptography do?



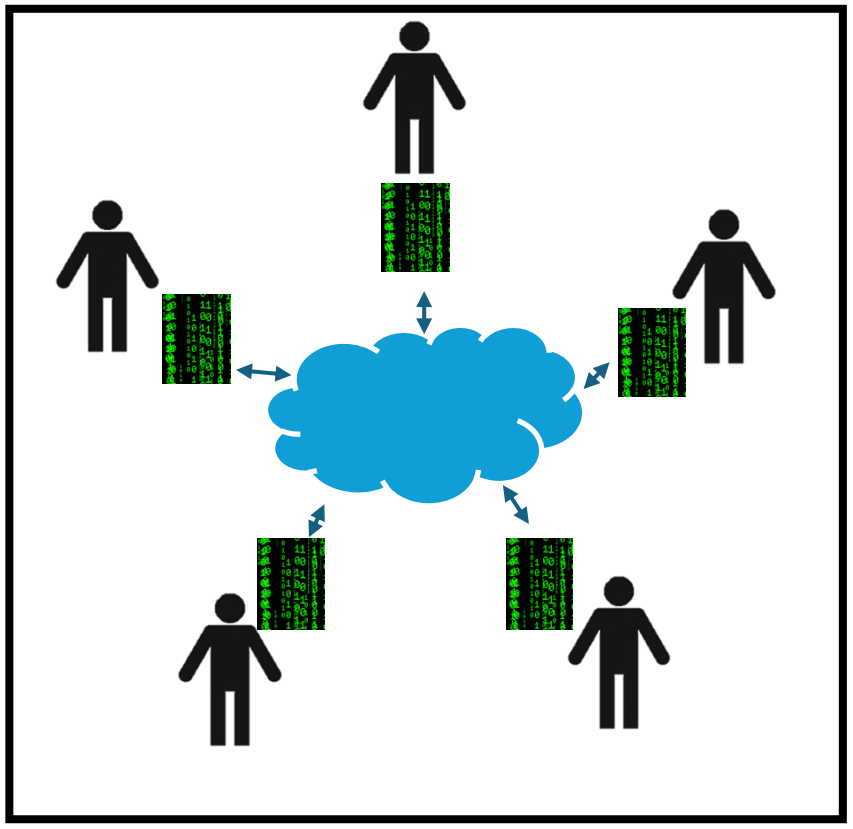


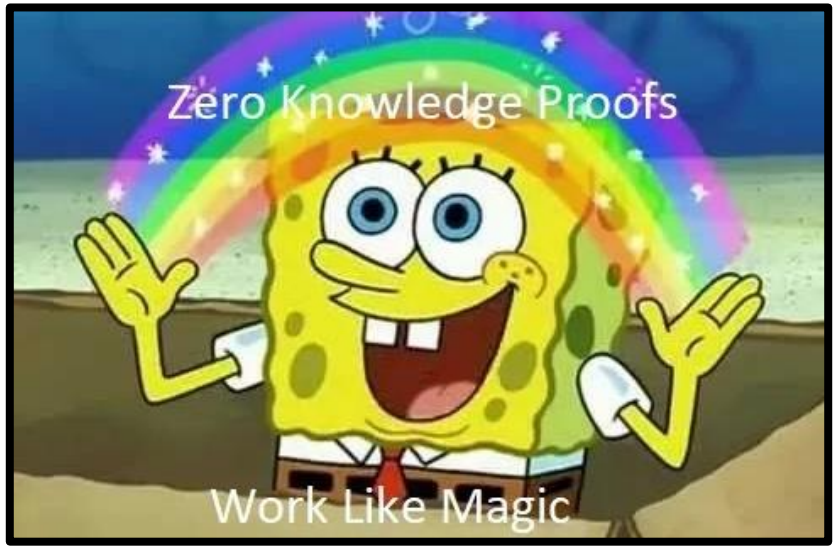


How to specify requirements?

What does cryptography do?





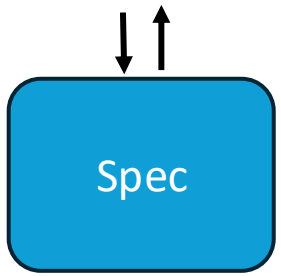


How to specify requirements?

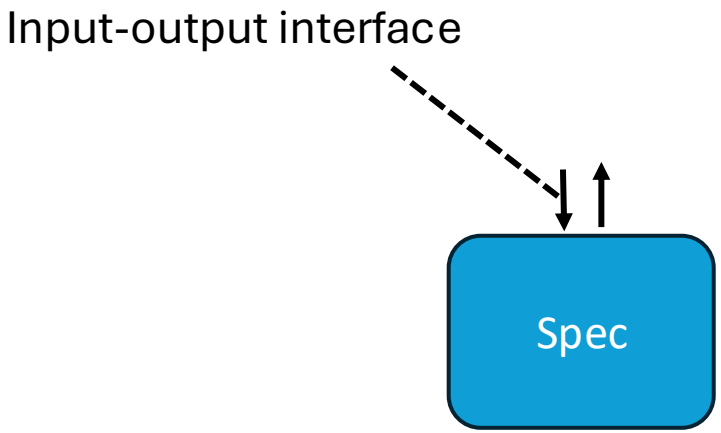
Real–ideal paradigm

A generalization of refinement

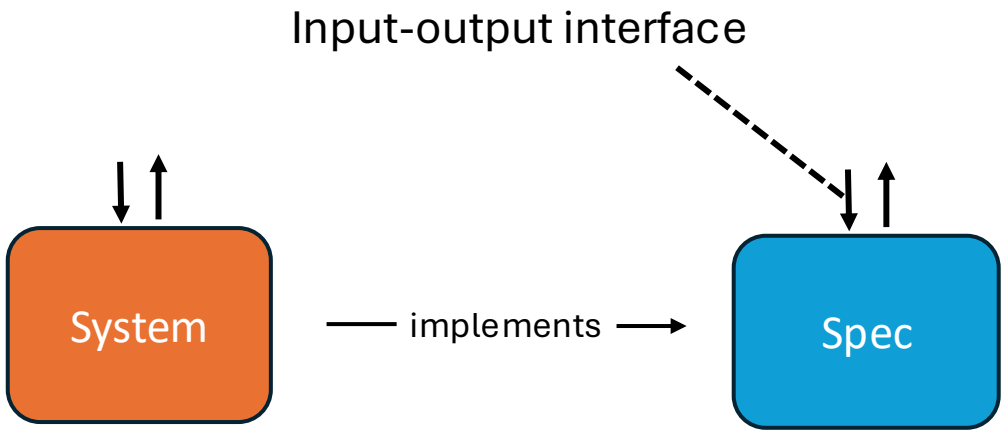
A generalization of refinement



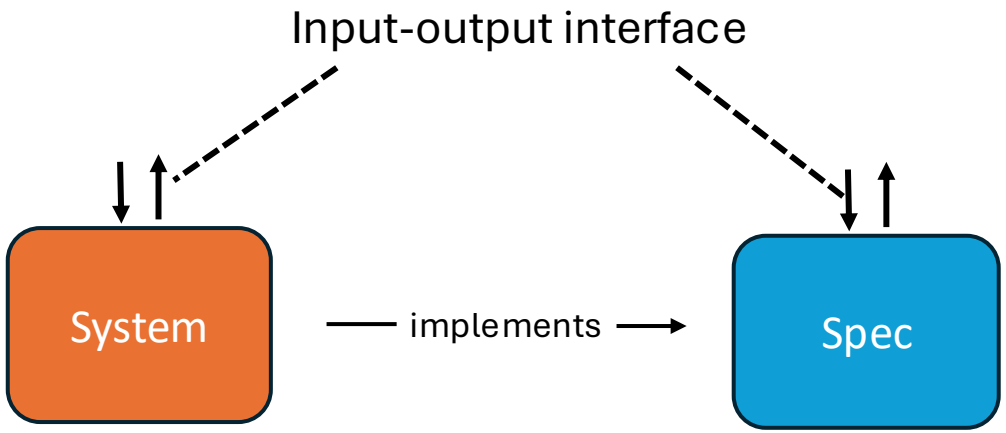
A generalization of refinement



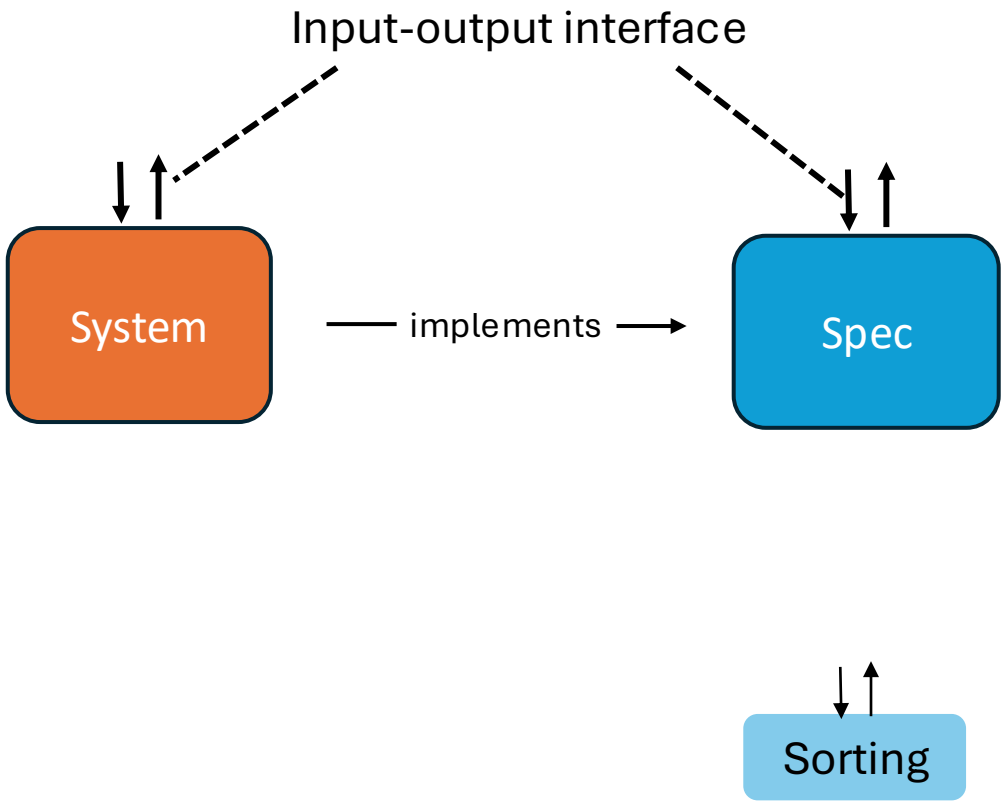
A generalization of refinement



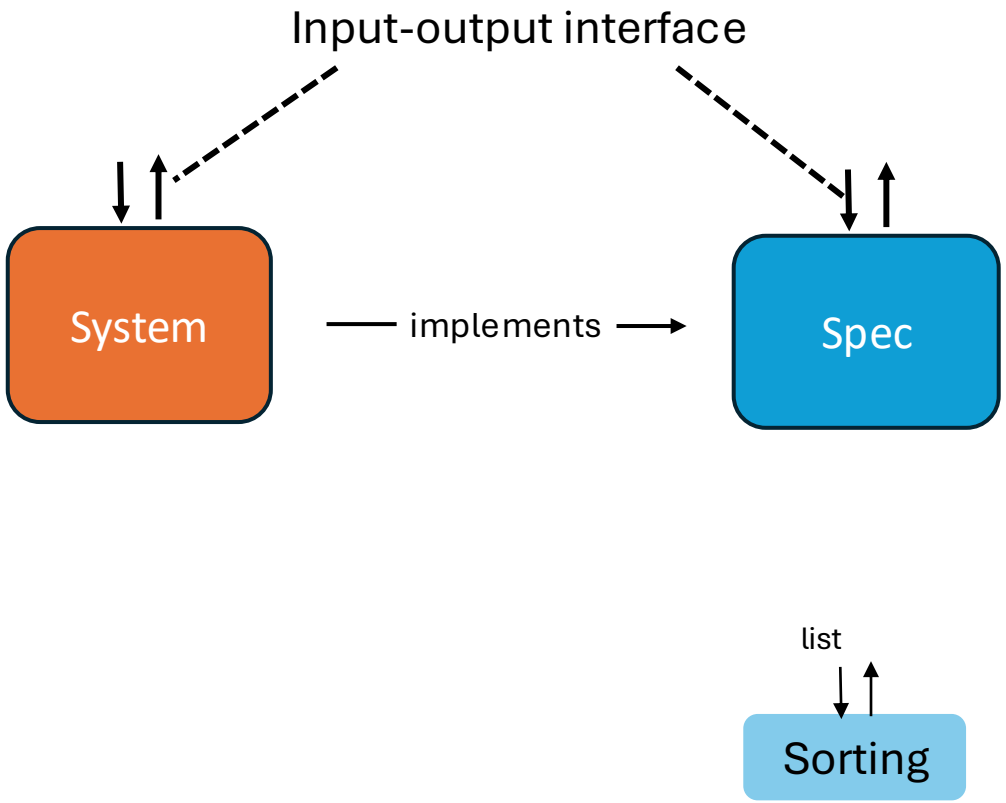
A generalization of refinement



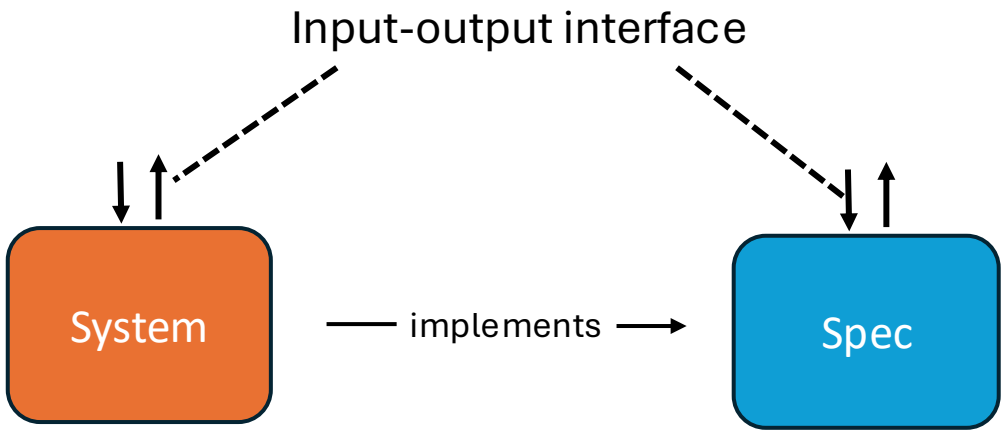
A generalization of refinement



A generalization of refinement

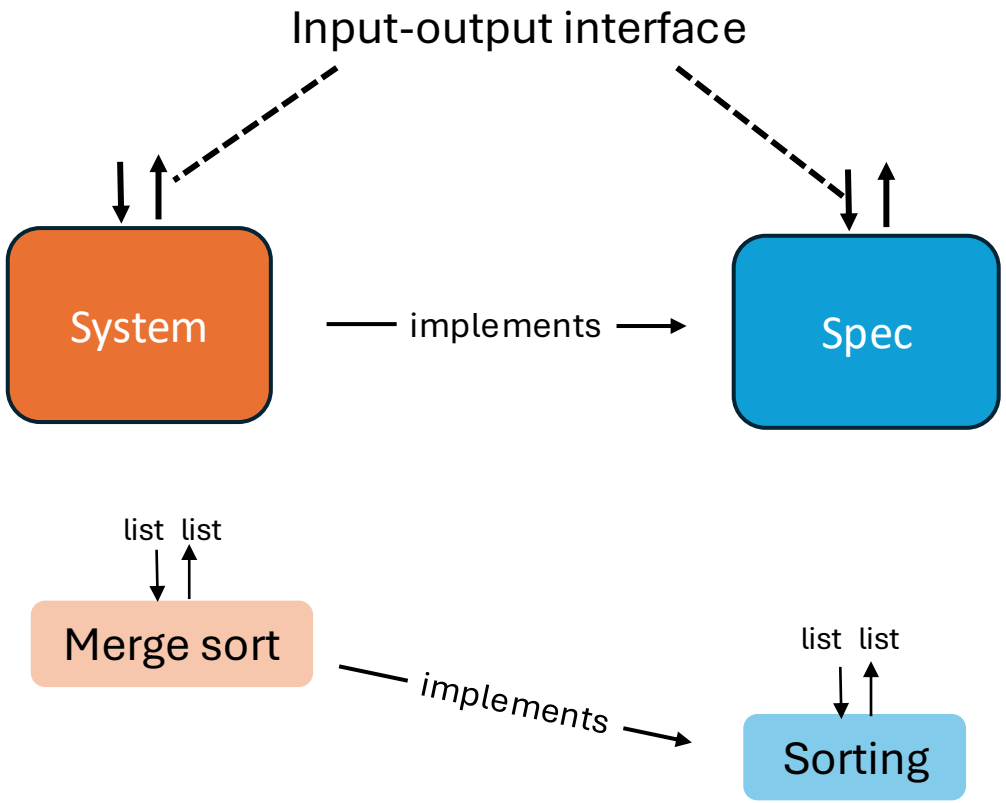


A generalization of refinement

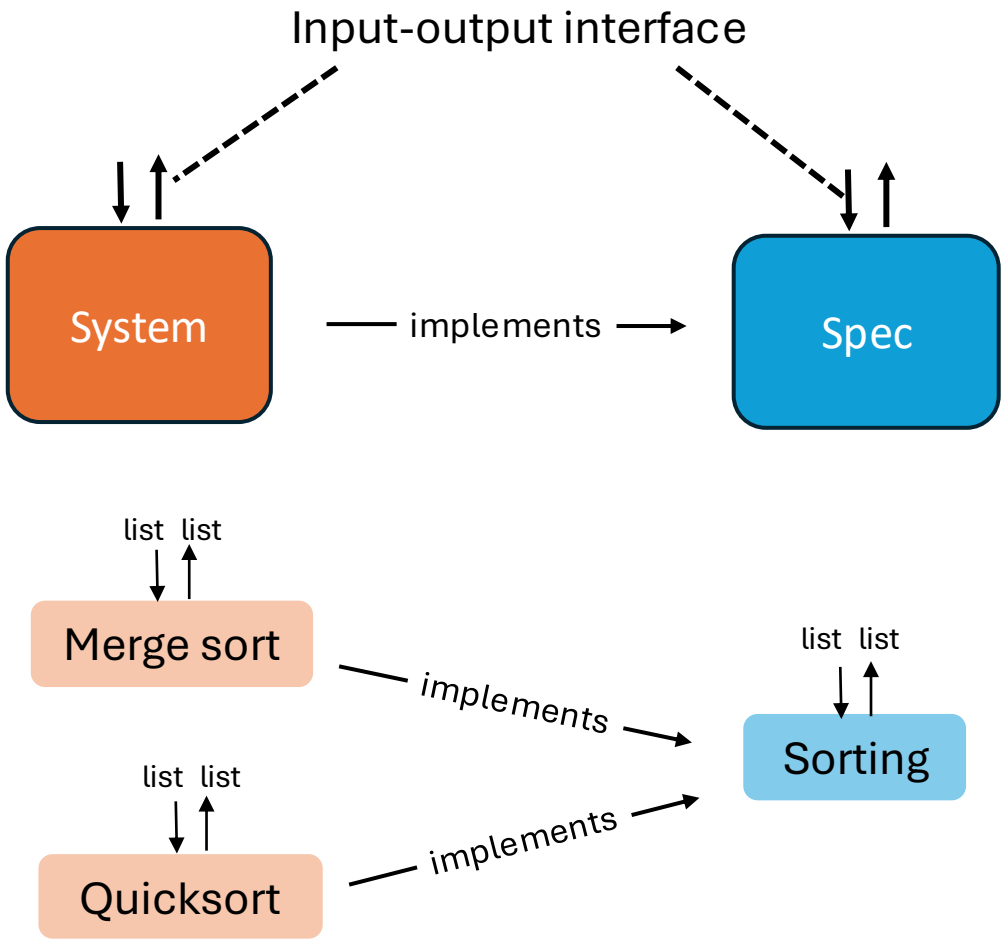




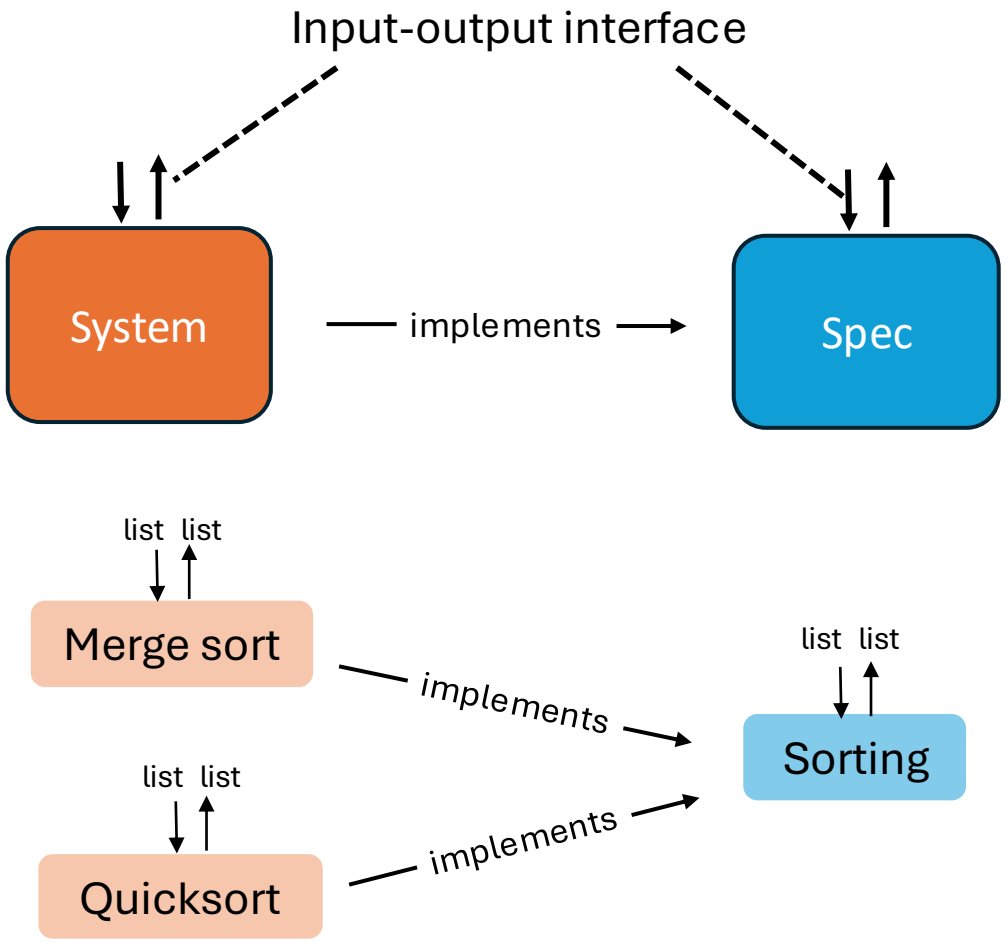
A generalization of refinement

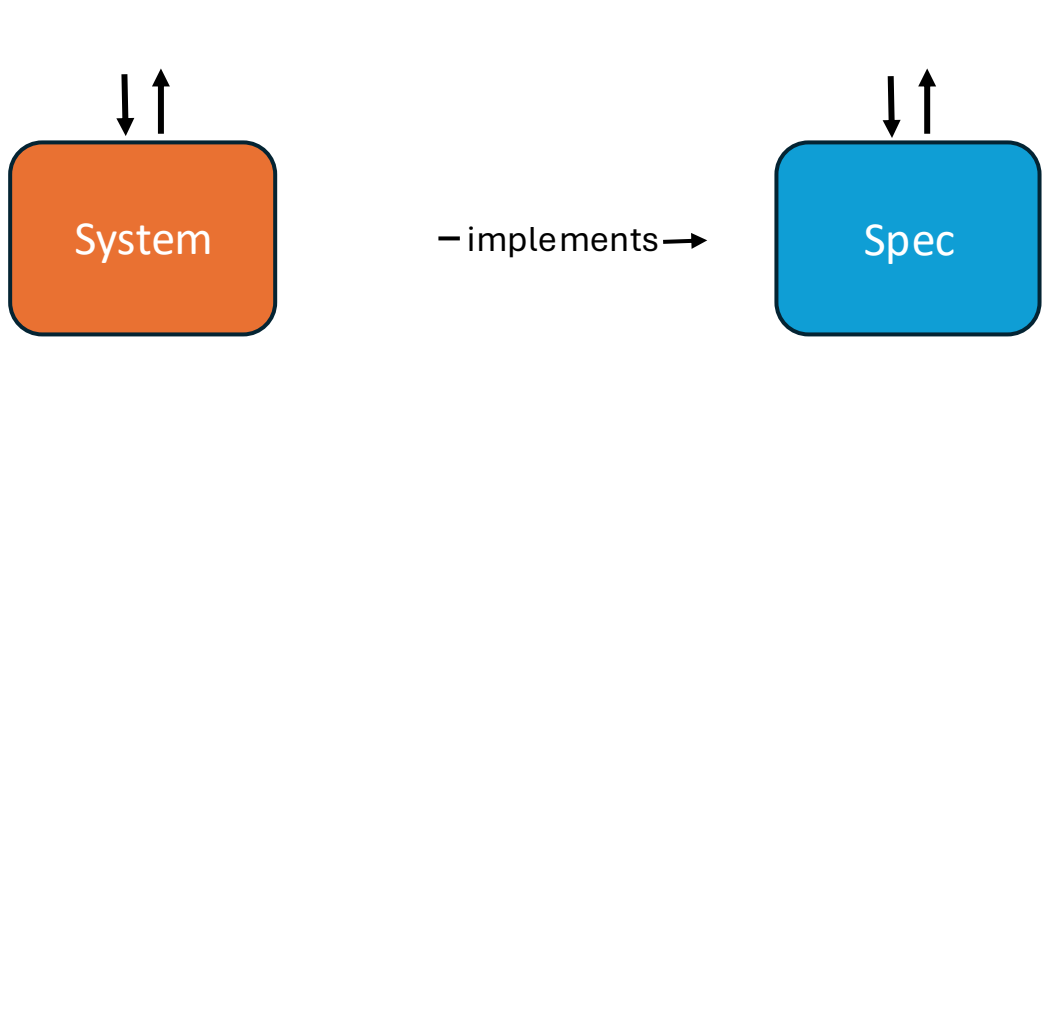


A generalization of refinement

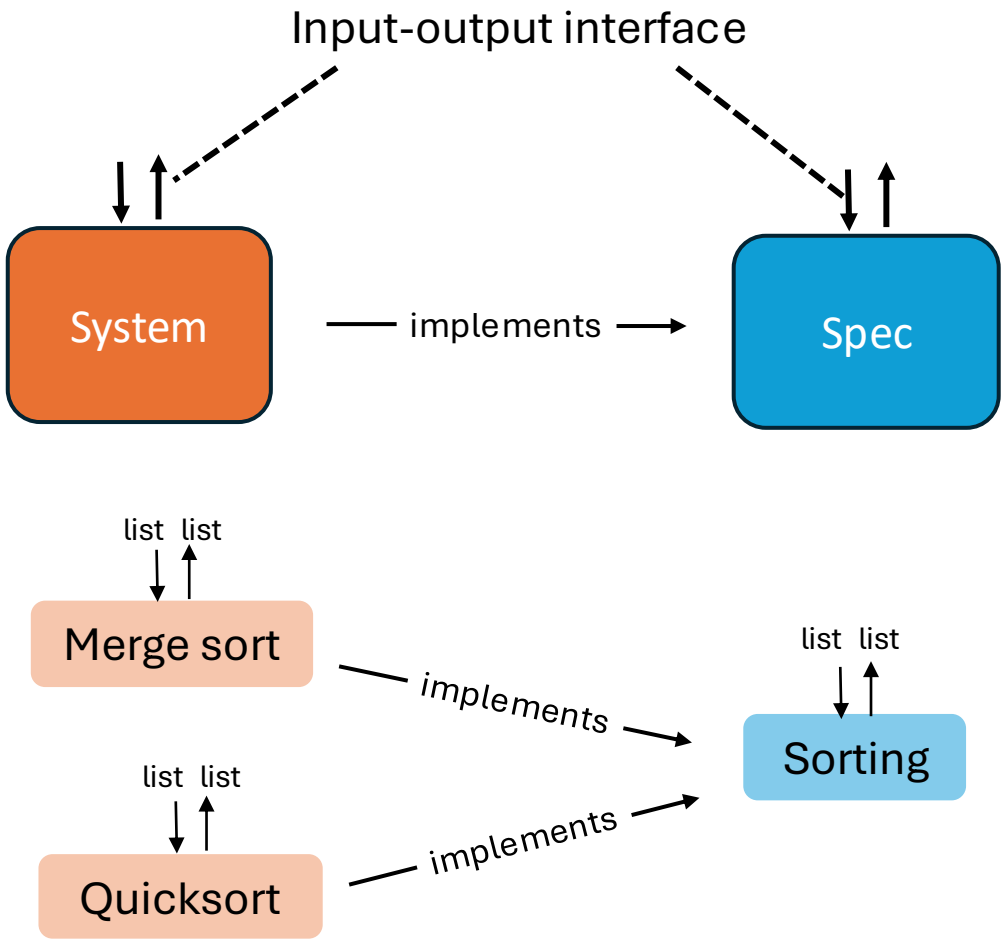


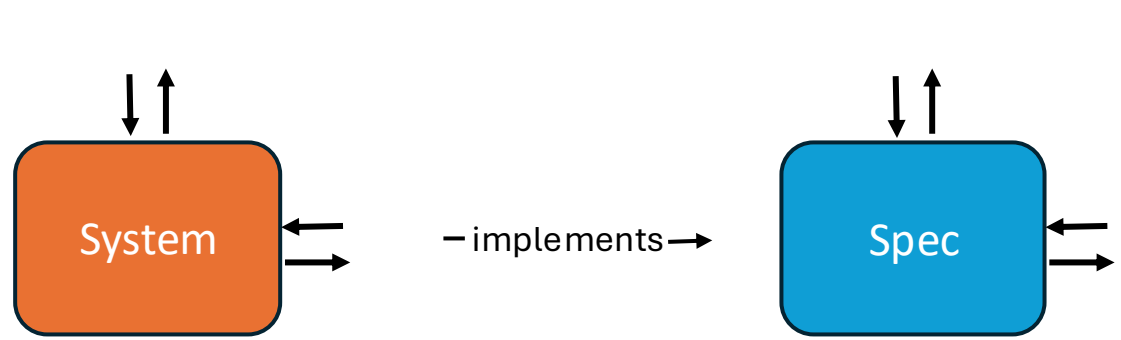
A generalization of refinement



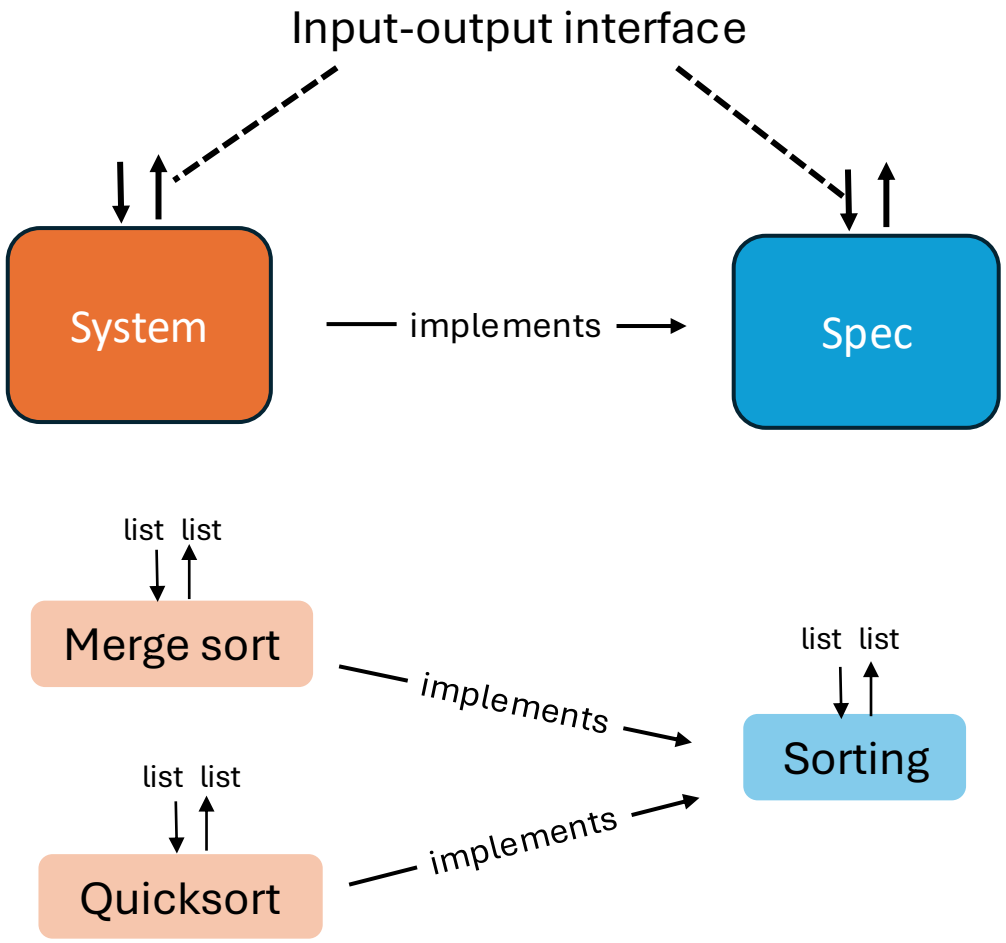


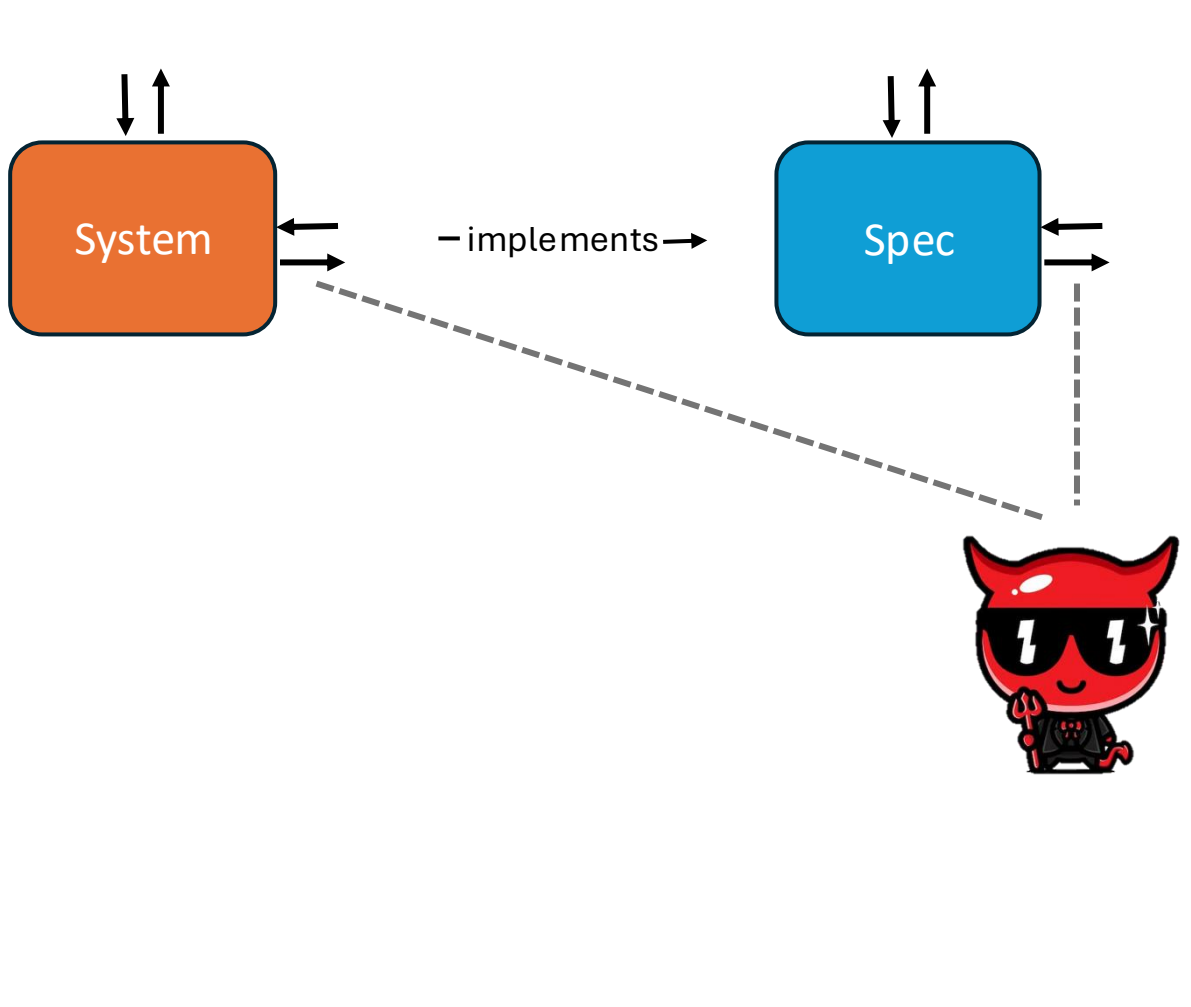
A generalization of refinement



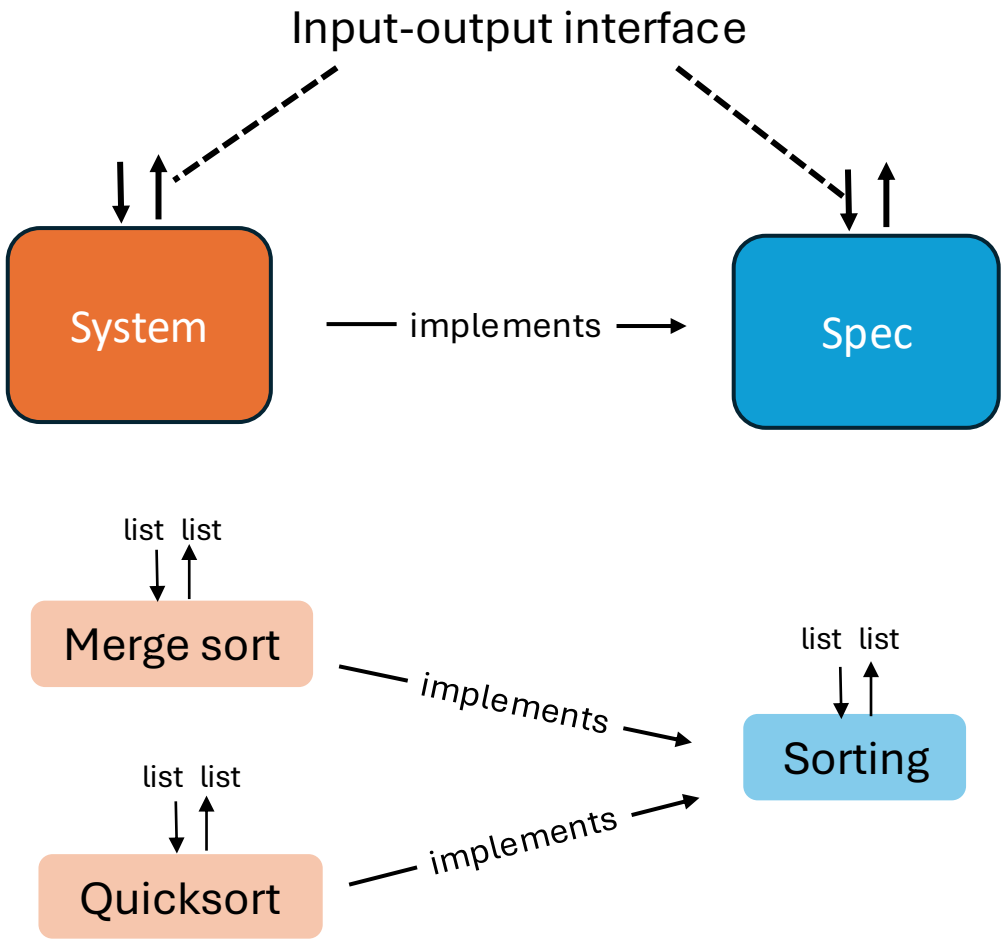


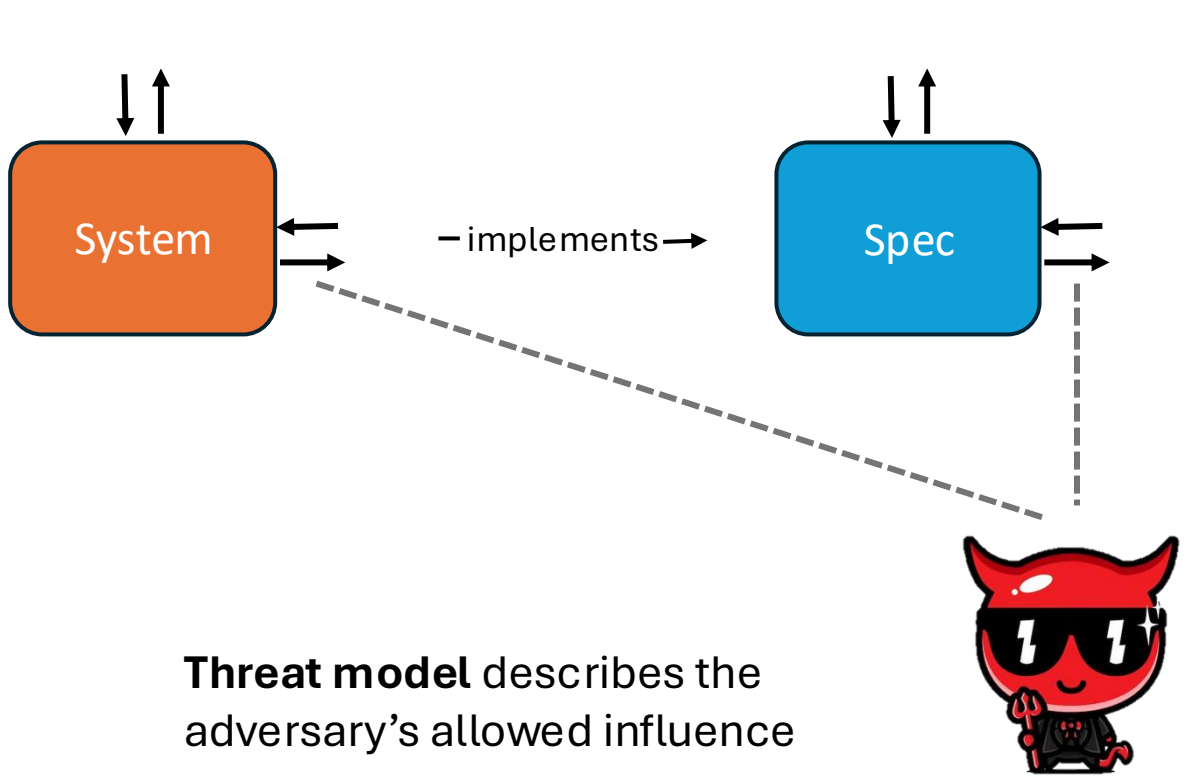
A generalization of refinement





A generalization of refinement





Threat model describes the adversary's allowed influence

Threat models?

Threat models?



Threat models?





Threat models?









Threat models?











Threat models?













Threat models?











Threat models?











Threat models?













Threat models?













Threat models?













Threat models?













‘Secret’ traps

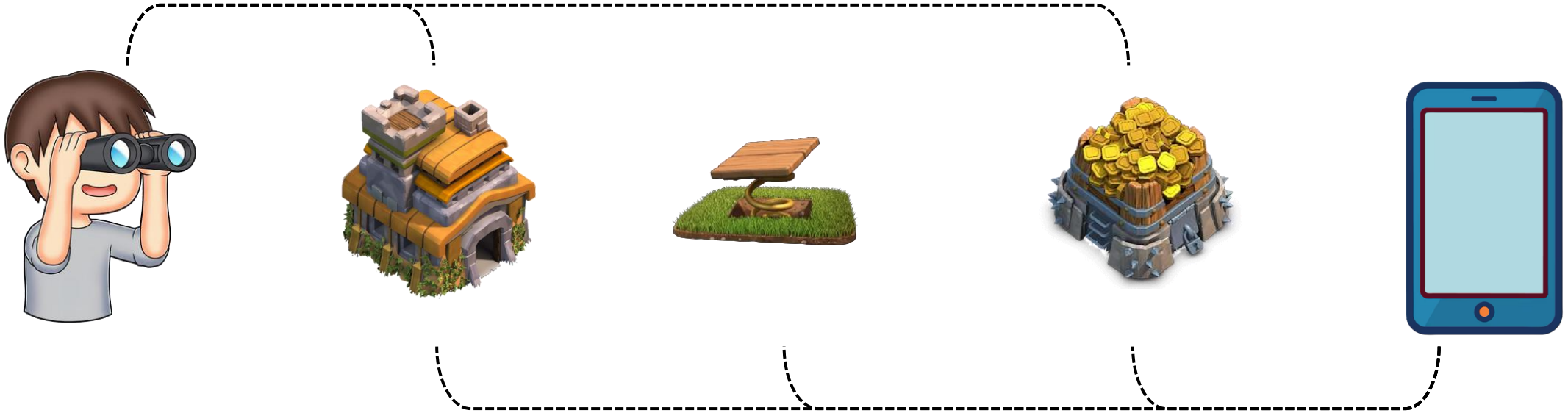
‘Secret’ traps







‘Secret’ traps



‘Secret’ traps











‘Secret’ traps





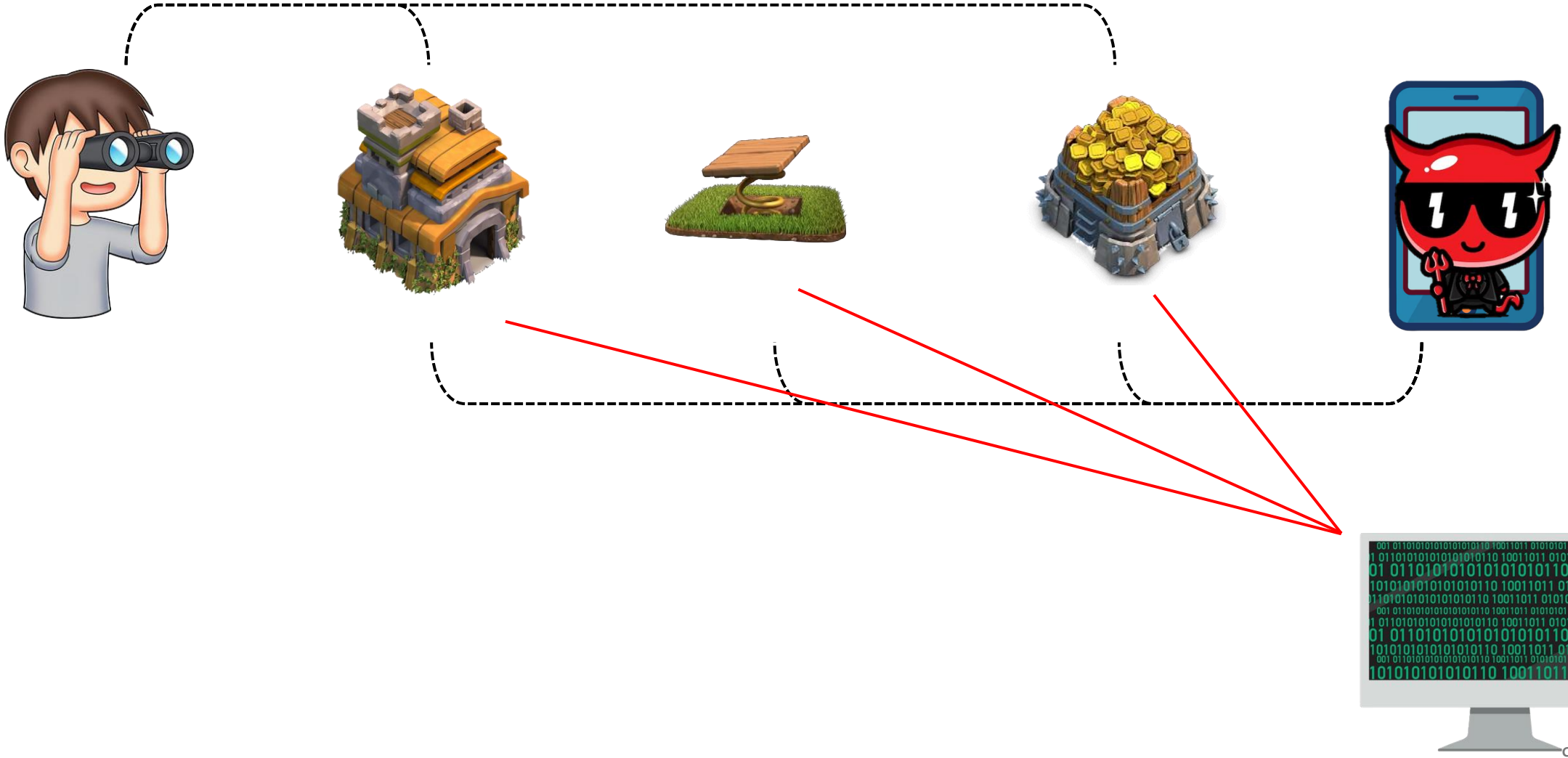




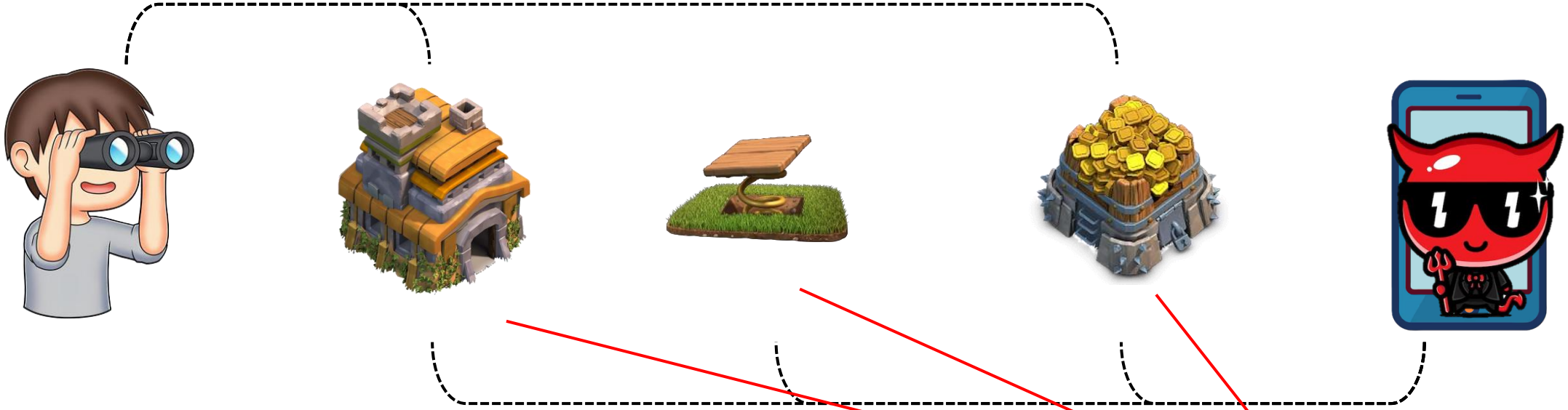




‘Secret’ traps



‘Secret’ traps





‘Secret’ traps

















‘Secret’ traps

Security analysis only makes sense with respect to a threat model!











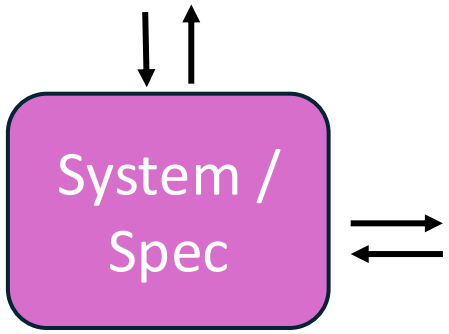




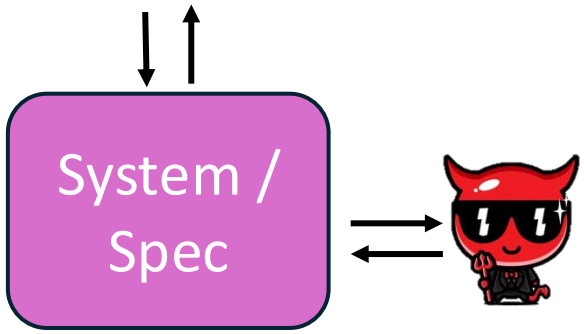


Real-ideal paradigm

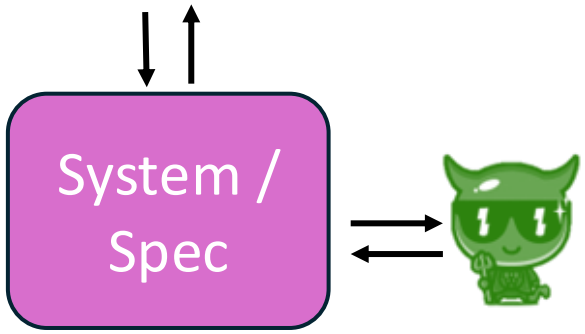
Real-ideal paradigm



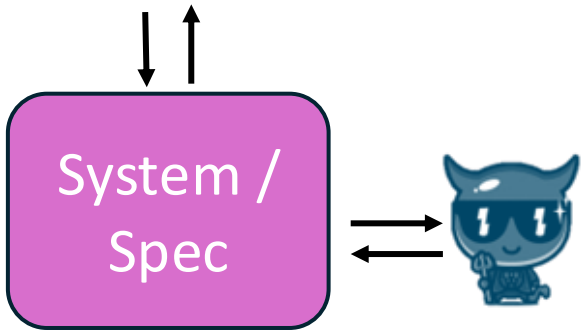
Real-ideal paradigm



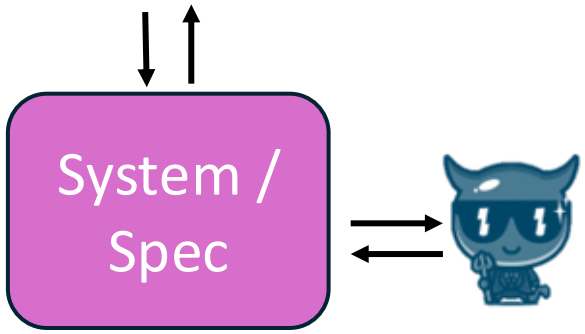
Real-ideal paradigm



Real-ideal paradigm

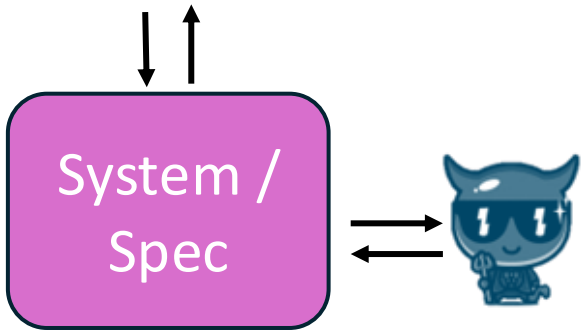


Real-ideal paradigm

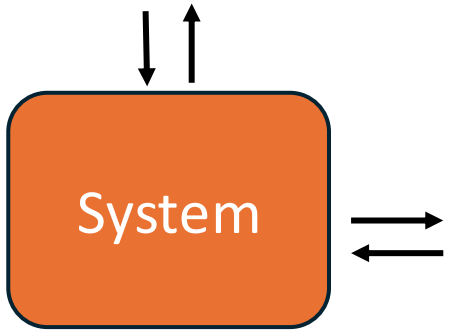


What does it mean to implement?

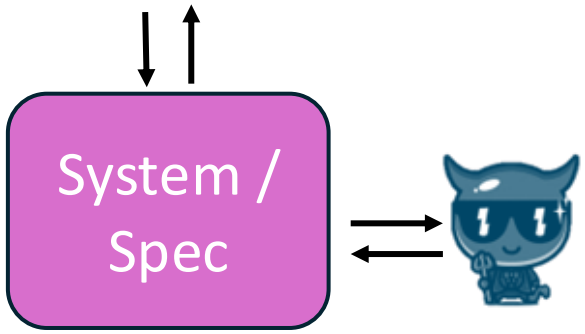
Real-ideal paradigm



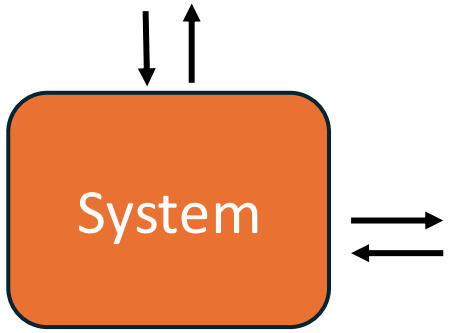
What does it mean to implement?

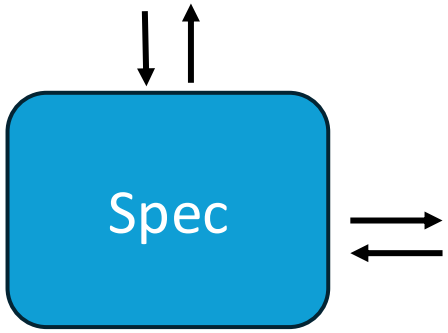


Real-ideal paradigm

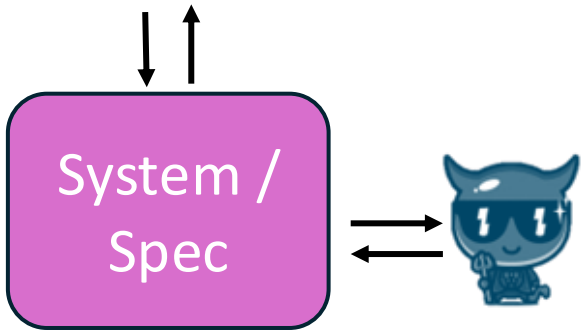


What does it mean to implement?





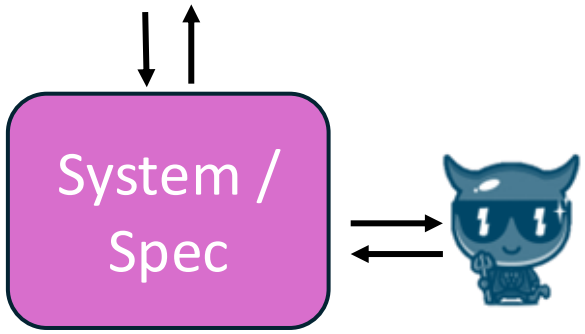
Real-ideal paradigm



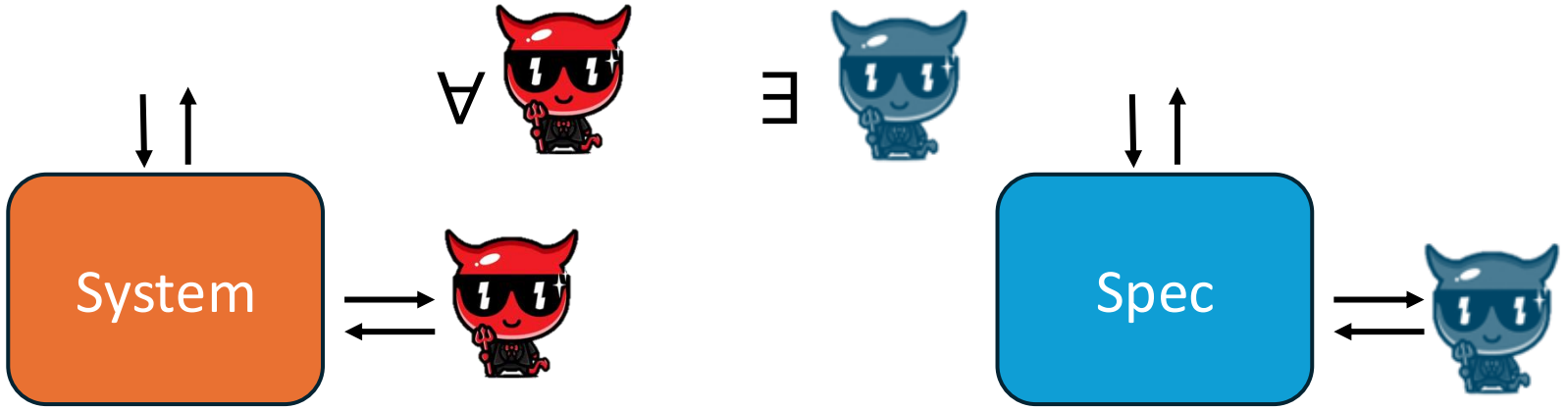
What does it mean to implement?



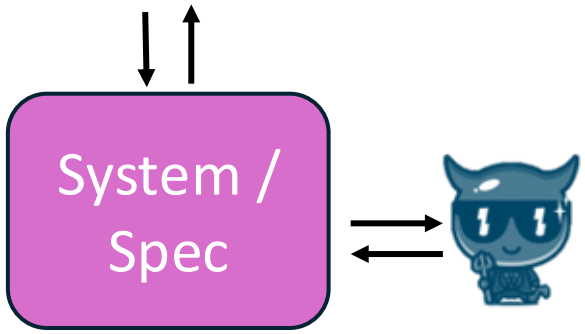
Real-ideal paradigm



What does it mean to implement?



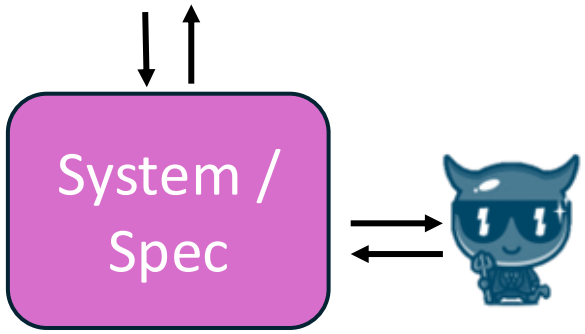
Real-ideal paradigm



What does it mean to implement?



Real-ideal paradigm

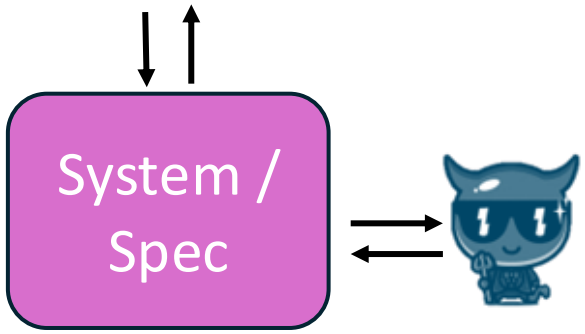


What does it mean to implement?

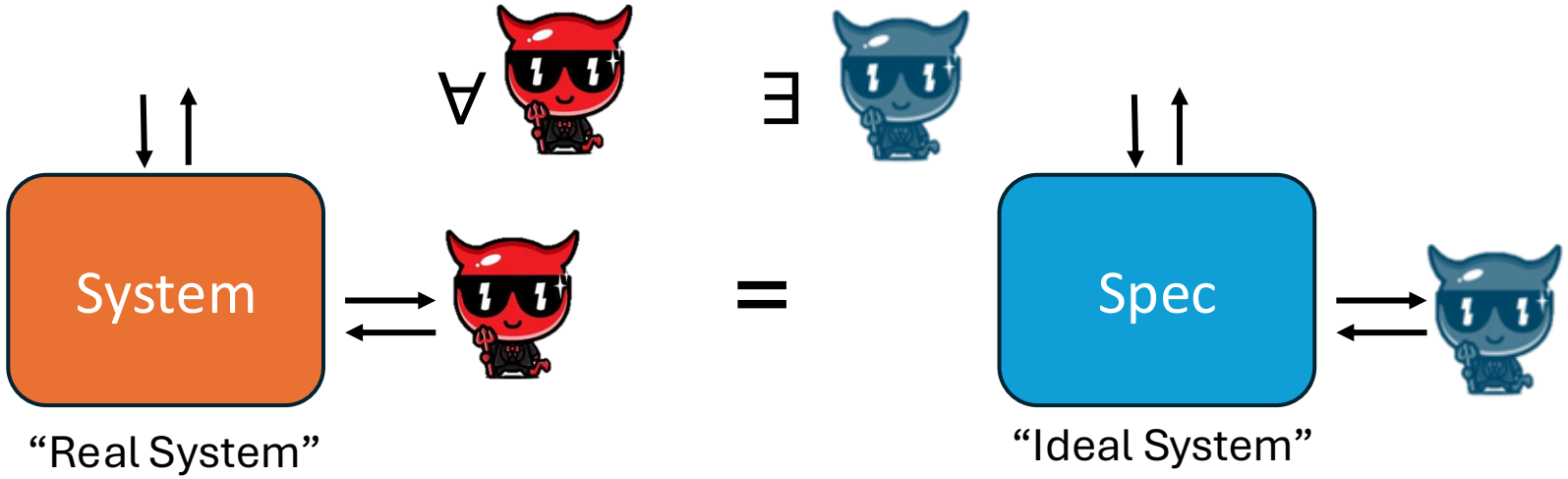


"Every system attack is allowed in the spec"

Real-ideal paradigm



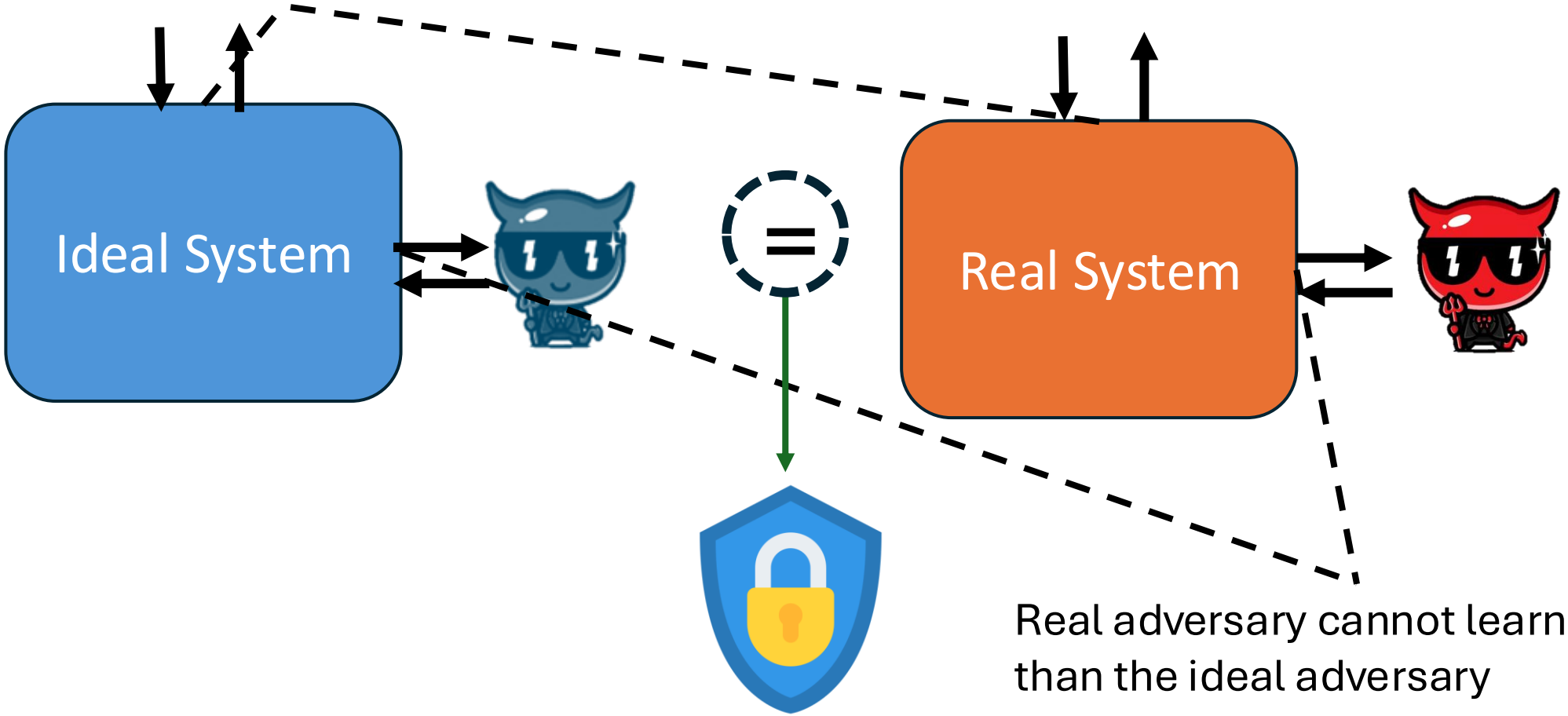
What does it mean to implement?



“Every system attack is allowed in the spec”

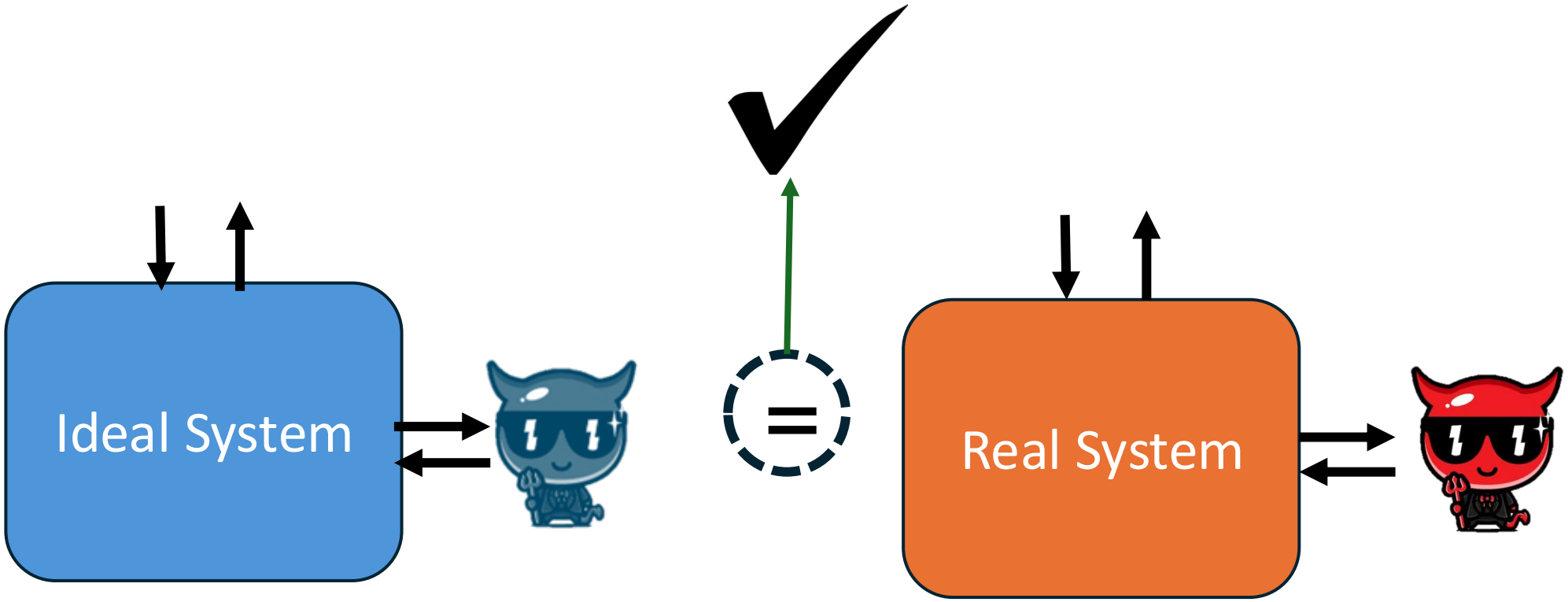
Features

Input output behavior in both cases must match



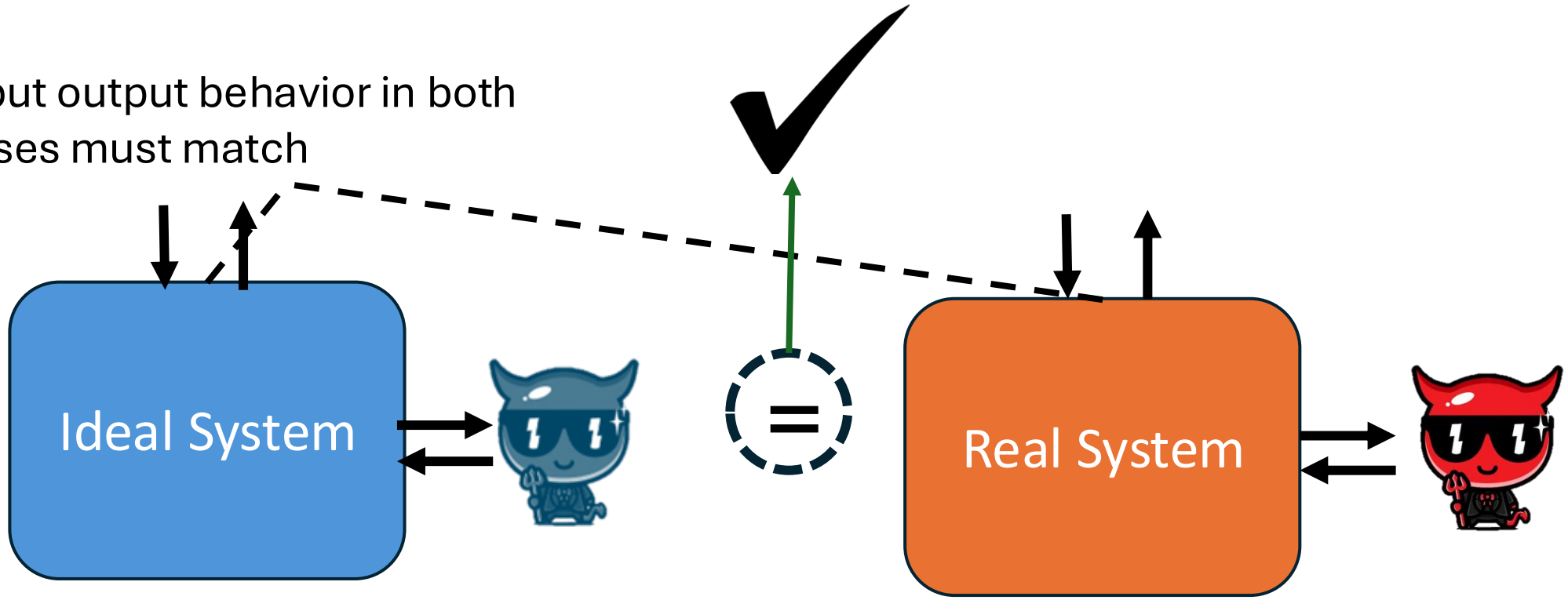
Real adversary cannot learn **more** than the ideal adversary

Features



Features

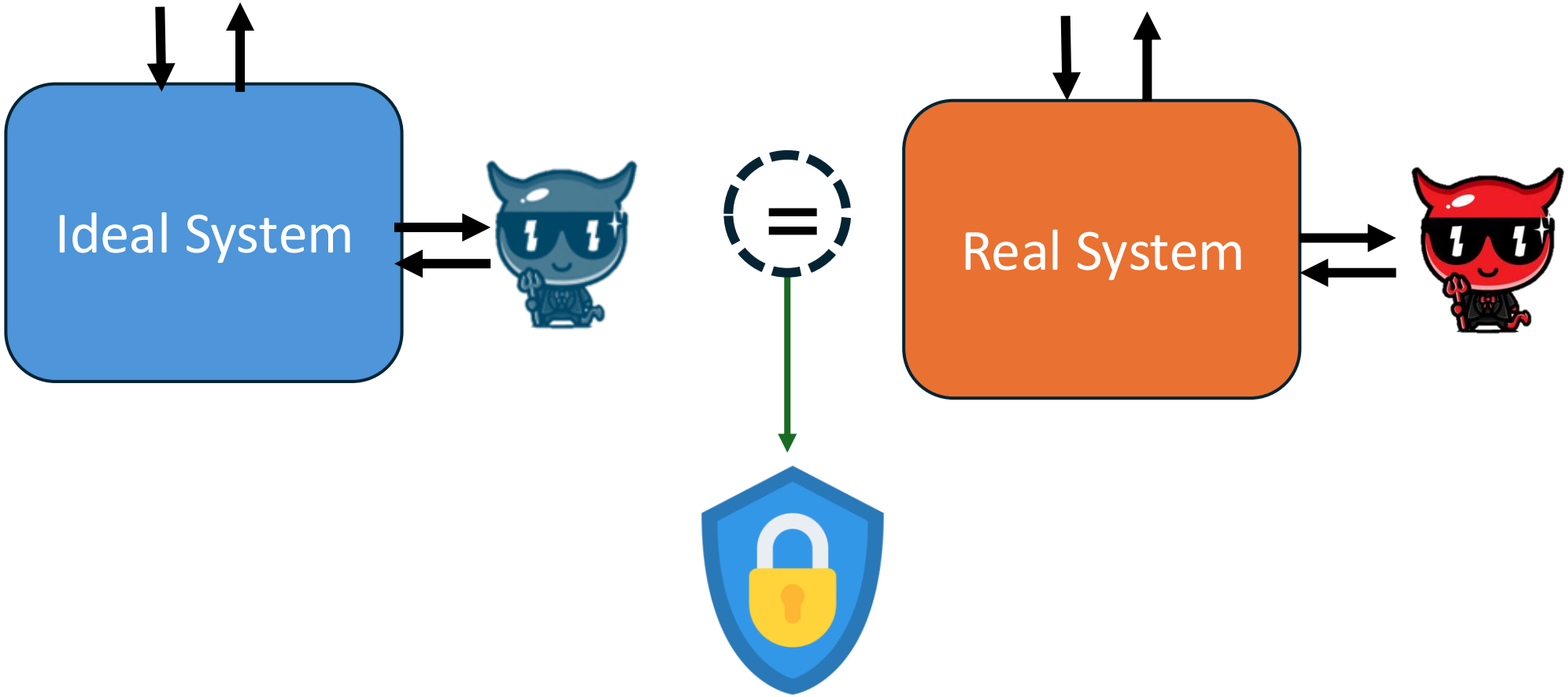
Input output behavior in both cases must match



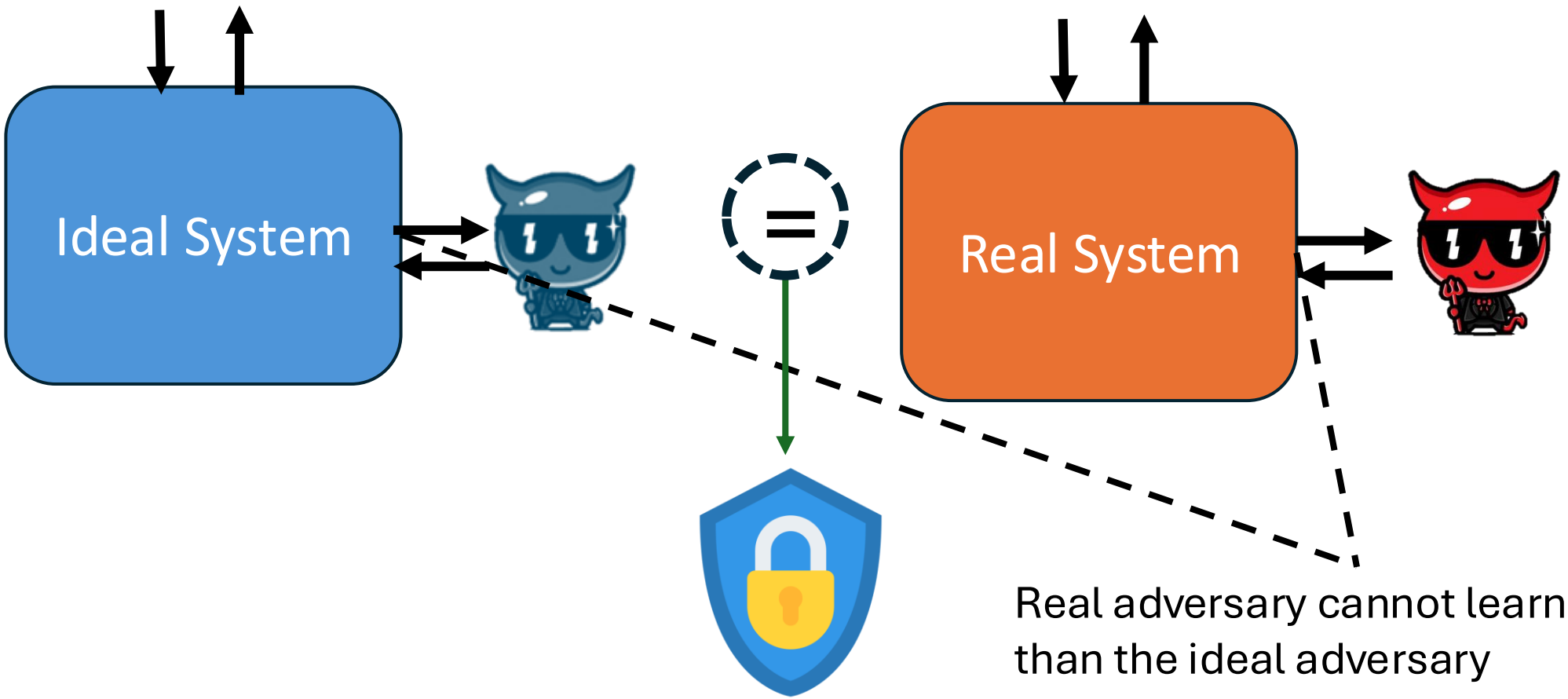
Features



Features



Features


Ideal System
Real System
Real adversary cannot learn **more** than the ideal adversary

Cryptographic formulations

Cryptographic formulations



Cryptographic formulations



Definition 3.1 (Def. 5.2.1 in [18]) *A private-key encryption scheme* (G, E, D) *is* semantically secure *(in the private-key model) if for every non-uniform probabilistic-polynomial time algorithm* $\mathcal{A}$ *there exists a non-uniform probabilistic-polynomial time algorithm* $\mathcal{A}'$ *such that for every probability ensemble* $\{X_n\}_{n\in\mathbb{N}}$ *with* $|X_n| \leq \text{poly}(n)$*, every pair of polynomially-bounded functions* $f, h : \{0,1\}^* \to \{0,1\}^*$*, every positive polynomial* $p(\cdot)$ *and all sufficiently large* n*:*

$$\Pr_{k\leftarrow G(1^n)}\left[\mathcal{A}(1^n, E_k(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\right]$$
$$< \Pr\left[\mathcal{A}'(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X_n)\right] + \frac{1}{p(n)}$$

(The probability in the above terms is taken over X_n *as well as over the internal coin tosses of the algorithms* G, E *and* $\mathcal{A}$ *or* $\mathcal{A}'$*.)*

Cryptographic formulations



Definition 3.1 (Def. 5.2.1 in [18]) *A private-key encryption*
secure *(in the private-key model) if for every non-uniform probab*
there exists a non-uniform probabilistic-polynomial time algorith
ity ensemble $\{X_n\}_{n\in\mathbb{N}}$ *with* $|X_n| \leq \text{poly}(n)$*, every pair of poly*
$\{0,1\}^* \to \{0,1\}^*$*, every positive polynomial* $p(\cdot)$ *and all sufficier*

$$\Pr_{k\leftarrow G(1^n)}\left[\mathcal{A}(1^n, E_k(X_n), 1^{|X_n|}, h(1^n, X_n)) = f(1^n, X\right.$$

$$< \Pr\left[\mathcal{A}'(1^n, 1^{|X_n|}, h(1^n, X_n)) = f(\right.$$

(The probability in the above terms is taken over X_n *as well as algorithms* G, E *and* $\mathcal{A}$ *or* $\mathcal{A}'$*.)*

Functionality $\mathcal{F}_{\text{NC}}$

1. `init` inputs are ignored, except for the inital one, Upon initial invocation (due to an `(init,`s`)` input from some party, Report s to the adversary. (Subsequent `init,`s`)` input are ignored.)
2. When receiving message `(Start,`$\hat{\mathcal{A}}$`)` from the adversary, where $\hat{\mathcal{A}}$ is the code of an ITM (representing an adversary), invoke $\hat{\mathcal{A}}$ and change state to `running`.
3. When receiving input `(Backdoor,`m`)` from party (i.e., ITI) P, verify that the state is `running`, and that P is a member of session s of π. If verification succeeds then activate $\hat{\mathcal{A}}$ with backdoor message m from P. Then:
   (a) If $\hat{\mathcal{A}}$ instructs to deliver a backdoor message m' to party P' then verify that P' is a member of session s of π. If so, then output `(Backdoor,`m'`)` to P'.
   (b) If $\hat{\mathcal{A}}$ generates an output v to the environment, then write v to the backdoor tape of the external adversary and halt.

Figure 15: The non-concurrent communication functionality, $\mathcal{F}_{\text{NC}}$.

Cryptographic formulations



Definition 3.1 (Def. 5.2.1 in [18]) *A private-key encryption*
secure *(in the private-key model) if for every non-uniform probab*
there exists a non-uniform probabilistic-polynomial time algorith
ity ensemble $\{X_n\}_{n\in\mathbb{N}}$ *with* $|X_n| \leq \text{poly}(n)$*, every pair of poly*
$\{0,1\}^* \to \{0,1\}^*$*, every positive polynomial* $p(\cdot)$ *and all sufficier*

Functionality $\mathcal{F}_{\text{NC}}$

1. `init` inputs are ignored, except for the inital one, Upon initial invocation (due to an `(init,`s`)` input from some party, Report s to the adversary. (Subsequent `init,`s`)` input are ignored.)
2. When receiving message `(Start,`$\hat{\mathcal{A}}$`)` from the adversary, where $\hat{\mathcal{A}}$ is the code of an ITM (representing an adversary), invoke $\hat{\mathcal{A}}$ and change state to `running`.

) from party (i.e., ITI) P, verify that the state is `running`,
of π. If verification succeeds then activate $\hat{\mathcal{A}}$ with backdoor

ckdoor message m' to party P' then verify that P' is a
, then output `(Backdoor,`m'`)` to P'.

he environment, then write v to the backdoor tape of the

rrent communication functionality, $\mathcal{F}_{\text{NC}}$.

B Previous versions of this work

The present framework has evolved considerably over the years; We highlight the main advances made in each version. We note that, in addition to these advances, each revision corrects inaccuracies and modeling discrepancies that were found in previous versions. We have tried to document each such correction in the right place in the text. For discrepancies that were pointed out to us by others, we also made all effort to acknowledge these colleagues. We sincerely thank these colleagues and apologize for any omissions. We feel particularily obliged to the anonymous referees who have pointed out great many inaccuracies, large and small.

The first versions of this work [C00A, Versions of 2000 and 2001] do not formulate a separate, rigorous model of distributed computation, and have different models for the execution of a protocol and for the ideal process. Also different communication and corruption models are treated as variants of the basic model.

The next revision [C00A, Version of 2005] introduces the notion of a system of ITMs and is the

Cryptographic formulations



Definition 3.1 (Def. 5.2.1 in [18]) *A private-key encryption*
secure *(in the private-key model) if for every non-uniform probab*
there exists a non-uniform probabilistic-polynomial time algorith
ity ensemble $\{X_n\}_{n\in\mathbb{N}}$ *with* $|X_n| \leq \text{poly}(n)$*, every pair of poly*
$\{0,1\}^* \to \{0,1\}^*$*, every positive polynomial* $p(\cdot)$ *and all sufficier*

Functionality $\mathcal{F}_{\text{NC}}$

1. `init` inputs are ignored, except for the inital one, Upon initial invocation (due to an `(init,`s`)` input from some party, Report s to the adversary. (Subsequent `init,`s`)` input are ignored.)
2. When receiving message `(Start,`$\hat{\mathcal{A}}$`)` from the adversary, where $\hat{\mathcal{A}}$ is the code of an ITM (representing an adversary), invoke $\hat{\mathcal{A}}$ and change state to `running`.

) from party (i.e., ITI) P, verify that the state is `running`,
of π. If verification succeeds then activate $\hat{\mathcal{A}}$ with backdoor

ckdoor message m' to party P' then verify that P' is a
, then output `(Backdoor,`m'`)` to P'.

he environment, then write v to the backdoor tape of the

rrent communication functionality $\mathcal{F}_{\text{NC}}$

B Previous versions of this work

The present framework has evolved considerably over the years; We highlight the main advances made in each version. We note that, in addition to these advances, each revision corrects inaccuracies and modeling discrepancies that were found in previous versions. We have tried to document each such correction in the right place in the text. For discrepancies that were pointed out to us by others, we also made all effort to acknowledge these colleagues. We sincerely thank these colleagues and apologize for any omissions. We feel particularily obliged t
pointed out great many inaccuracies, large and small.

The first versions of this work [C00A, Versions of 2000 and
rigorous model of distributed computation, and have different m
and for the ideal process. Also different communication and
variants of the basic model.

The next revision [C00A, Version of 2005] introduces the nc

Paper 2025/934

Diving Deep Into UC: Uncovering and Resolving Issues in Universal Composability

Céline Chevalier, CRED, Paris-Panthéon-Assas University, Paris, France, DIENS, École normale supérieure, Paris, France, PSL University, France, CNRS, France, Inria, France
Éric Sageloli, Thales, Gennevilliers, France, DIENS, École normale supérieure, Paris, France, PSL University, France, CNRS, France, Inria, France

Cryptographic formulations



Definition 3.1 (Def. 5.2.1 in [18]) *A private-key encryption*
secure *(in the private-key model) if for every non-uniform probab*
there exists a non-uniform probabilistic-polynomial time algorith
ity ensemble $\{X_n\}_{n\in\mathbb{N}}$ *with* $|X_n| \leq \text{poly}(n)$*, every pair of poly*
$\{0,1\}^* \to \{0,1\}^*$,

Functionality $\mathcal{F}_{\text{NC}}$

1. `init` inputs are ignored, except for the inital one, Upon initial invocation (due to an `(init,`s`)` input from some party, Report s to the adversary. (Subsequent `init,`s`)` input are ignored.)

sary, where $\hat{\mathcal{A}}$ is the code of an ITM
e to `running`.

TI) P, verify that the state is `running`,
succeeds then activate $\hat{\mathcal{A}}$ with backdoor

′ to party P' then verify that P' is a
`ckdoor,`m'`)` to P'.
nen write v to the backdoor tape of the

tion functionality $\mathcal{F}_{\text{NC}}$

B Previous ver

The present framework
made in each version. V
cies and modeling discr
each such correction in t
others, we also made all
and apologize for any o
pointed out great many

The first versions of
rigorous model of distributed computation, and have different m
and for the ideal process. Also different communication and
variants of the basic model.

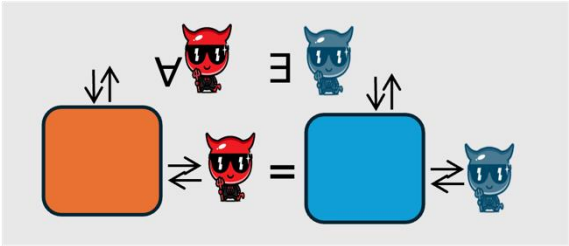
The next revision [c00a, Version of 2005] introduces the nc

I'm reading Canetti's original papers (UC with Global Setup, 2006 and UC Authentication and Key Exchange, 2014) and find it so hard to dig through the handwavy proofs and spaghetti code (written in English). I get the overall idea of simulation and encapsulating processes external to the protocol inside environment Z, but when I get to the nuances of how corruptions work and similar, there's a lot of things that are implicit and swept under the carpet. I can see that the idea is powerful, but the presentation certainly could be improved, and I was wondering if someone did improve it and any of you could direct me to such resource.

Resolving Issues in Universal Composability

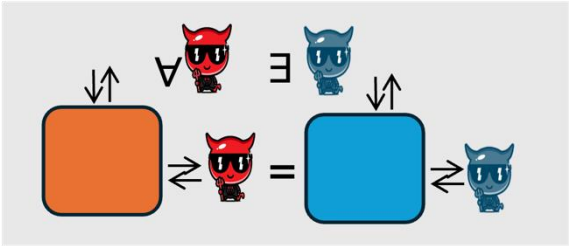
Céline Chevalier, CRED, Paris-Panthéon-Assas University, Paris, France, DIENS, École normale supérieure, Paris, France, PSL University, France, CNRS, France, Inria, France
Éric Sageloli, Thales, Gennevilliers, France, DIENS, École normale supérieure, Paris, France, PSL University, France, CNRS, France, Inria, France

Our Work

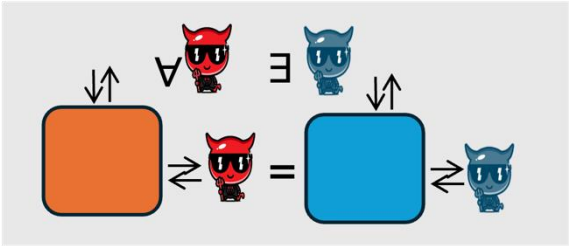


Our Work





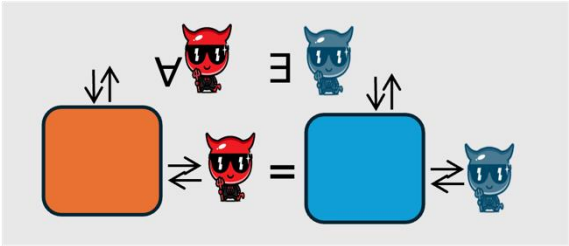
Our Work







Our Work

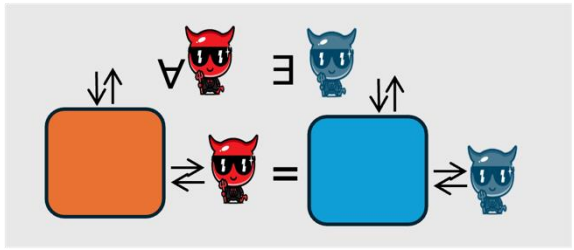








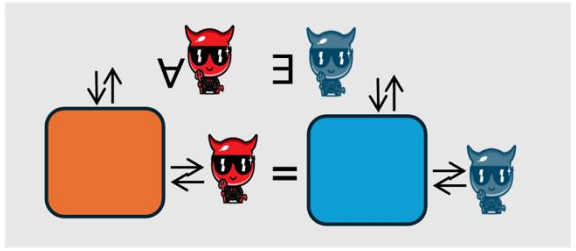
Our Work



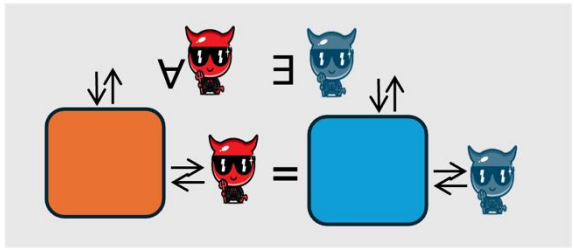








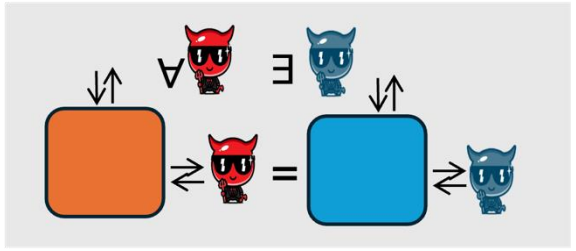
Our Work

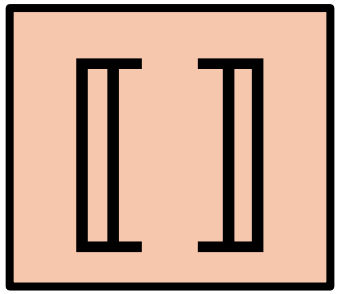




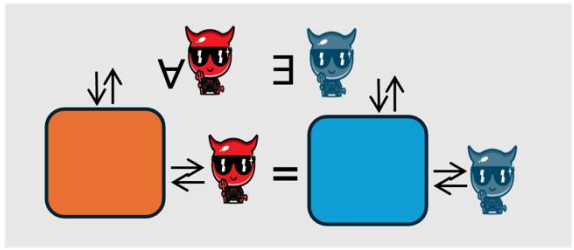








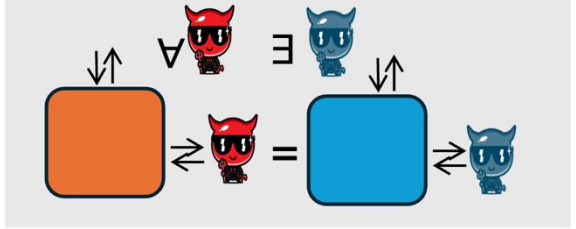
Our Work







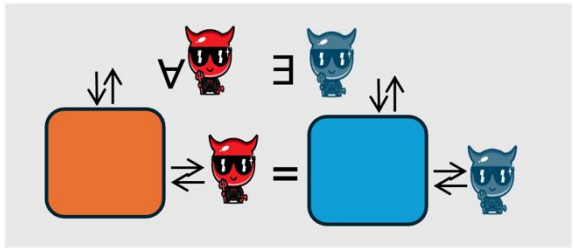




$[\![\ ]\!]$

$$\Gamma \vdash t : \tau$$

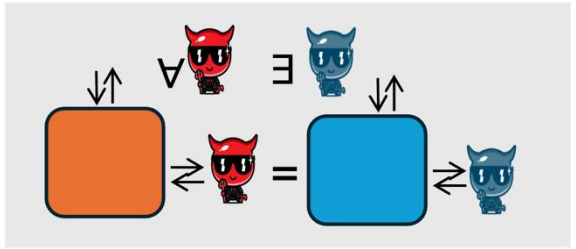
Our Work









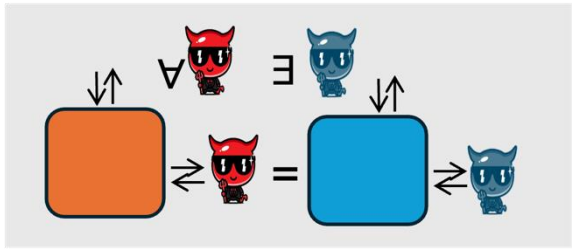


$[\![\ ]\!]$

$$\Gamma \vdash t : \tau$$



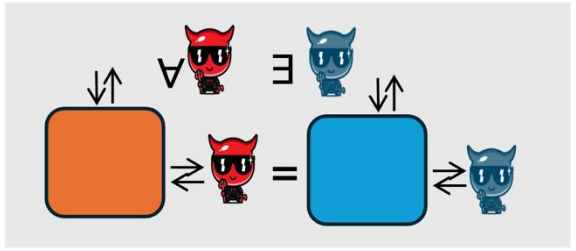
Our Work











$[\![\ ]\!]$

$$\Gamma \vdash t : \tau$$

Understanding nonmalleable information flow

Understanding nonmalleable information flow

Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS ‘17)

Understanding nonmalleable information flow

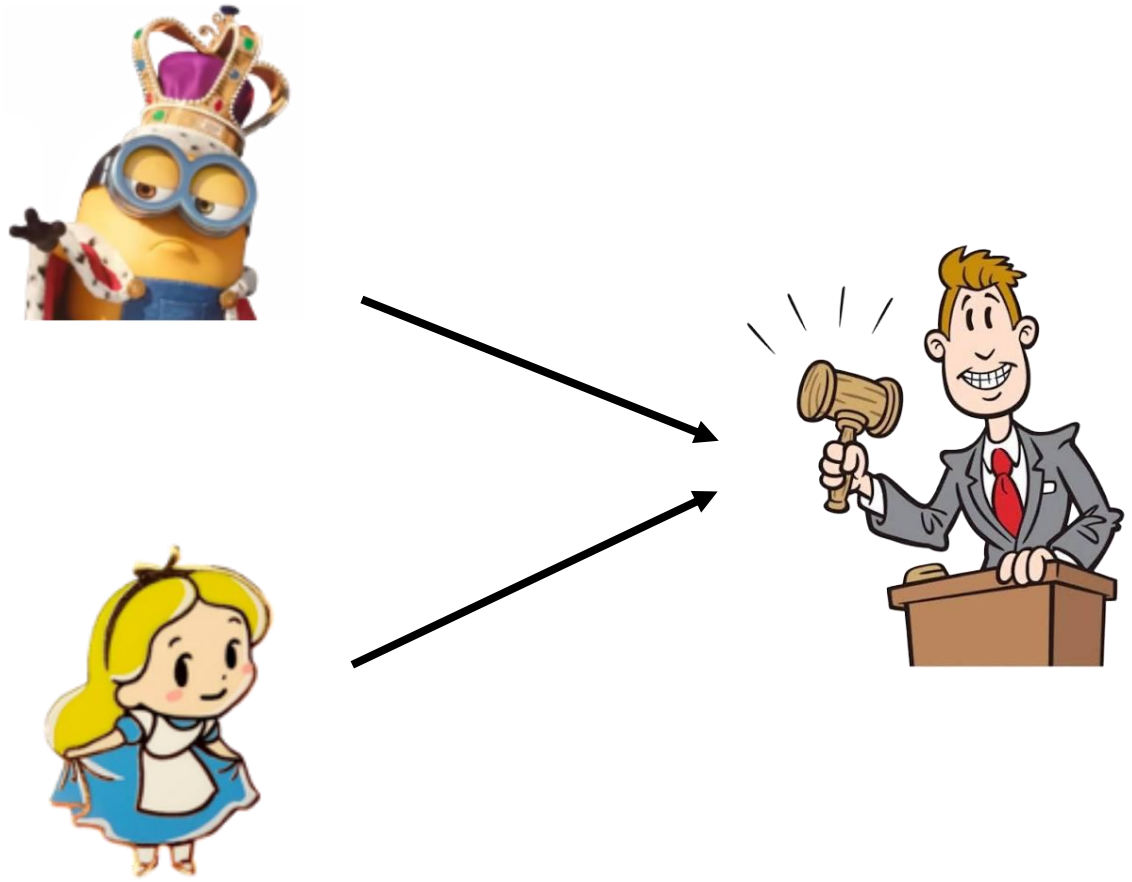
Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS ‘17)





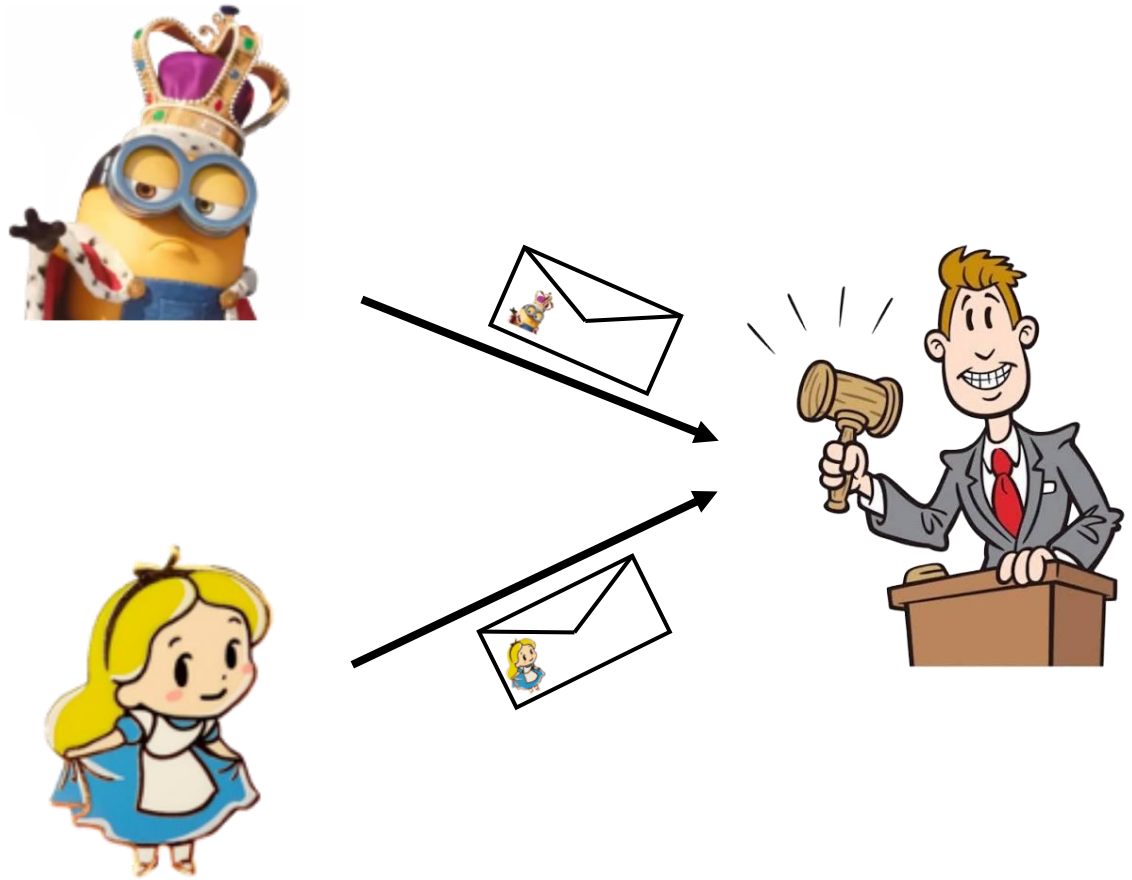
Understanding nonmalleable information flow

Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)



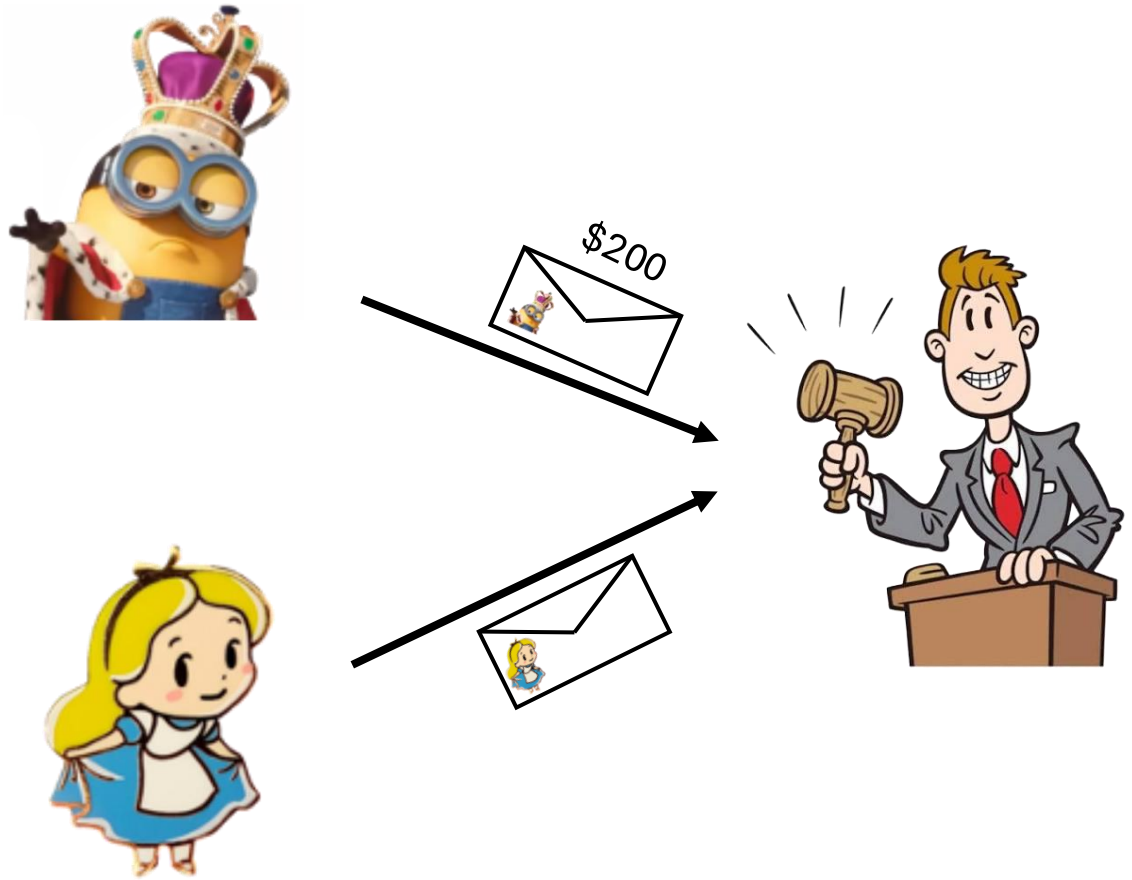
Understanding nonmalleable information flow

Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)



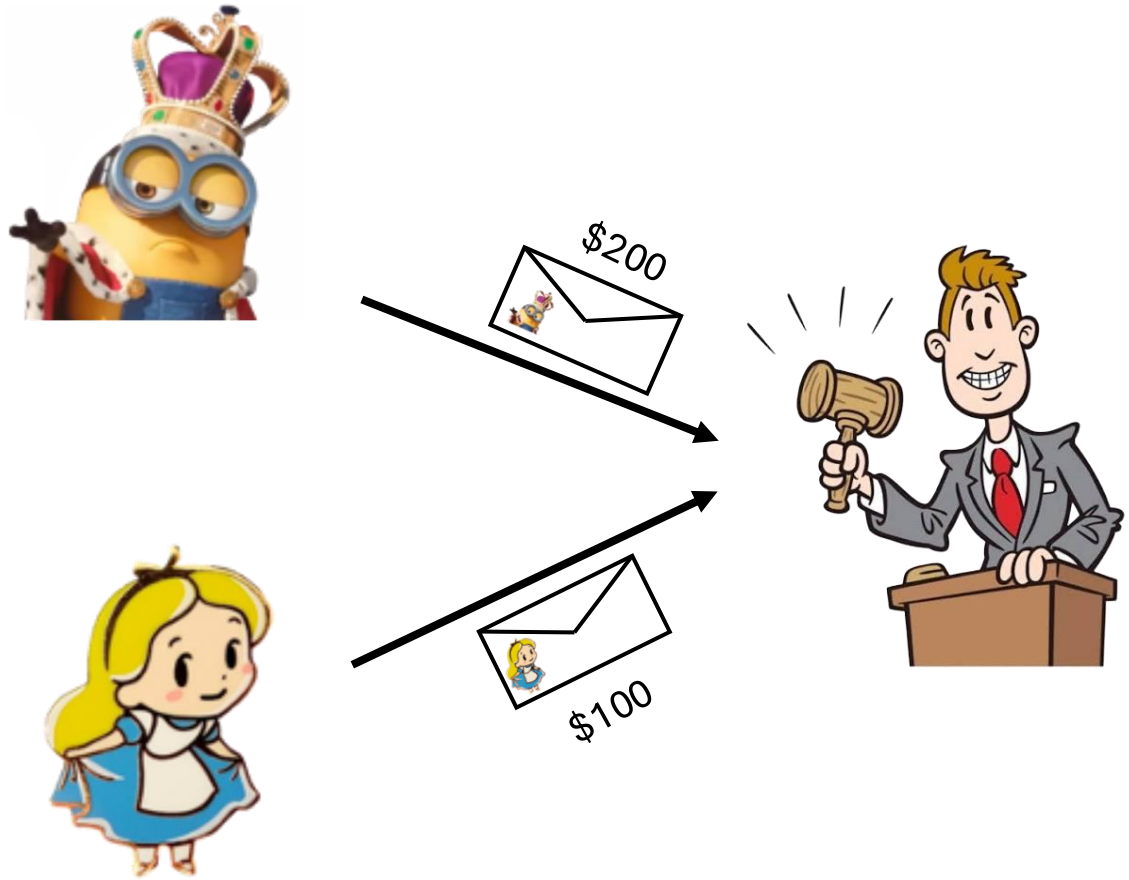
Understanding nonmalleable information flow

Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)



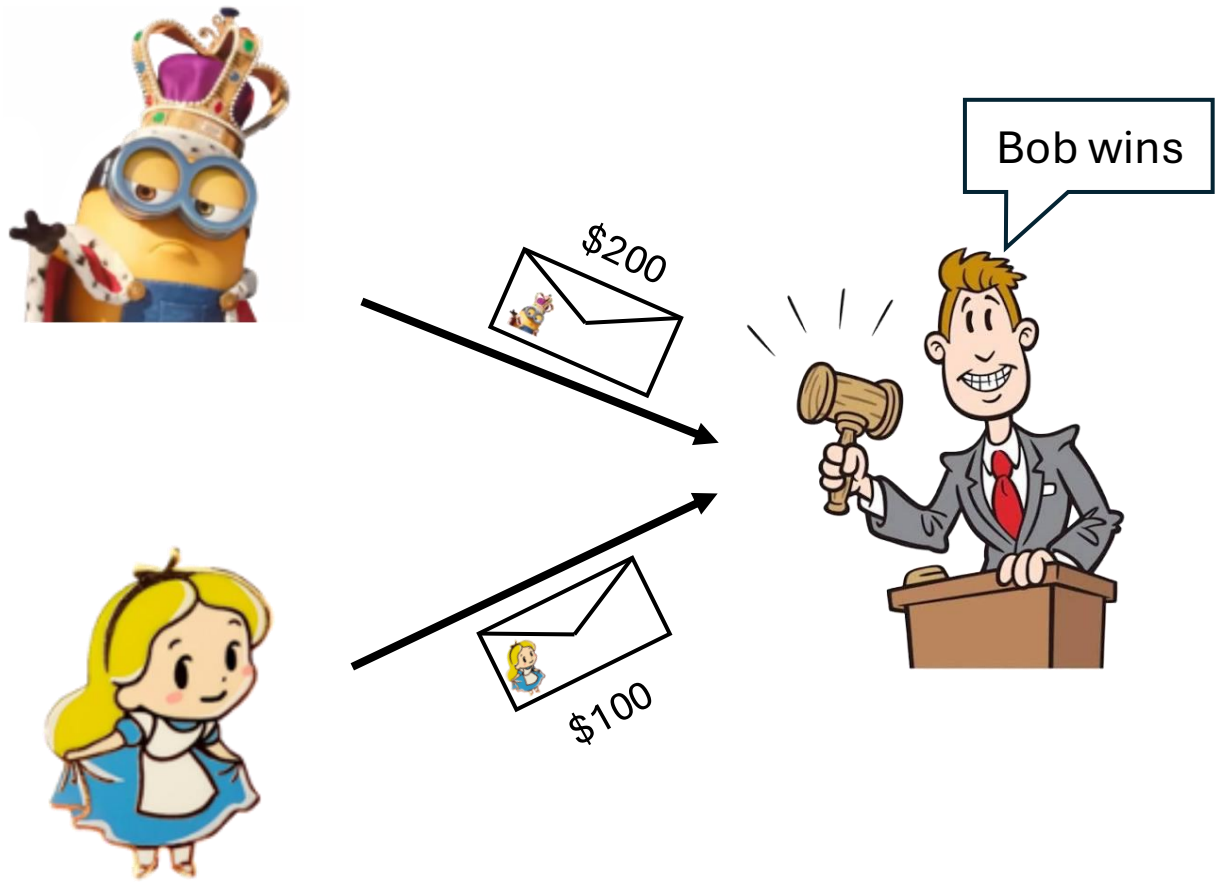
Understanding nonmalleable information flow

Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)



Understanding nonmalleable information flow

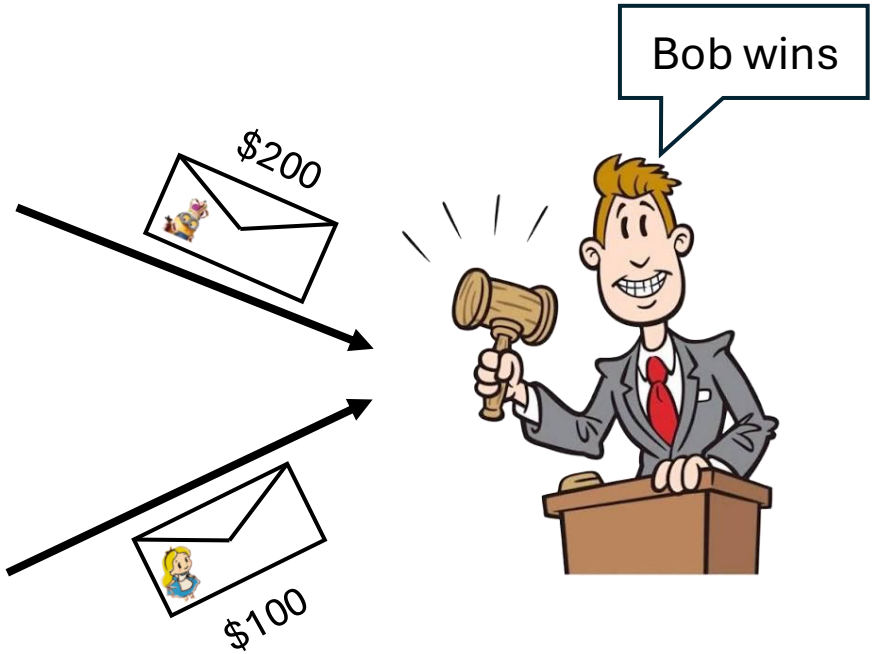
Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)



Understanding nonmalleable information flow

Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)







```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b)
with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
    end
```

An IFC implementation

Understanding nonmalleable information flow

Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)

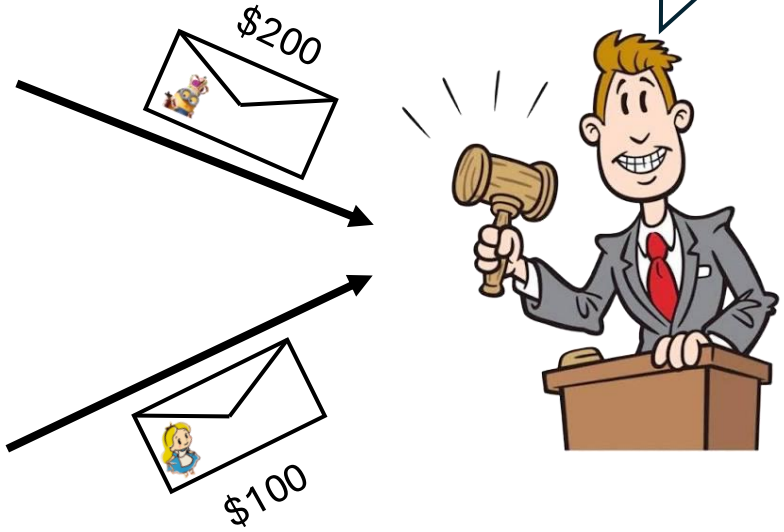


```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b)
with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
    end
```

An IFC implementation







Understanding nonmalleable information flow

Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)

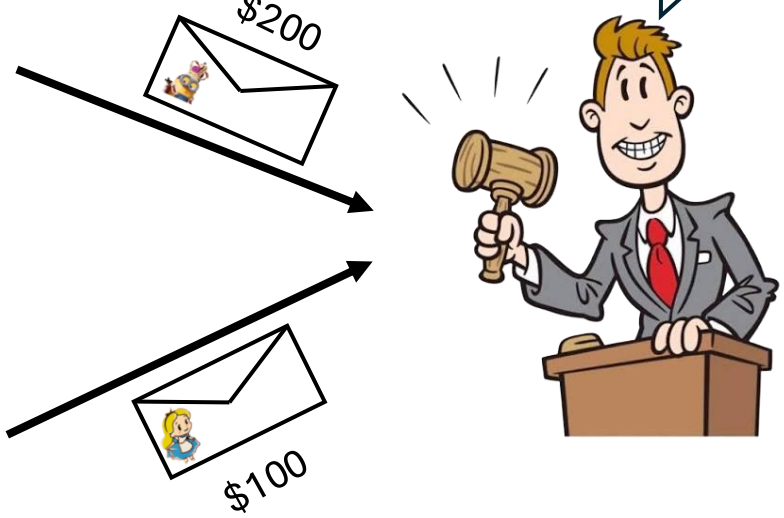
Label$_{int}$: `Alice`

```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b)
with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
    end
```

An IFC implementation







Understanding nonmalleable information flow

Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)

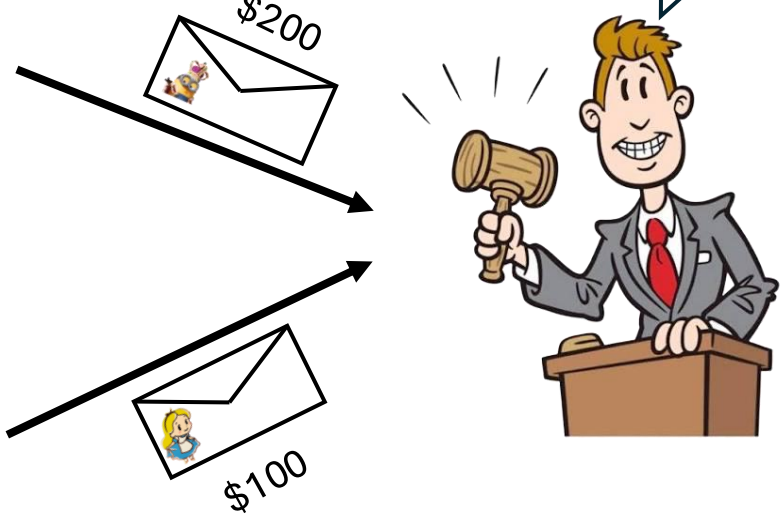
Label$_{int}$: Alice
Label$_{conf}$: Alice

```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b)
with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
    end
```

An IFC implementation







Understanding nonmalleable information flow

Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)

Label$_{int}$: Alice
Label$_{conf}$: Alice

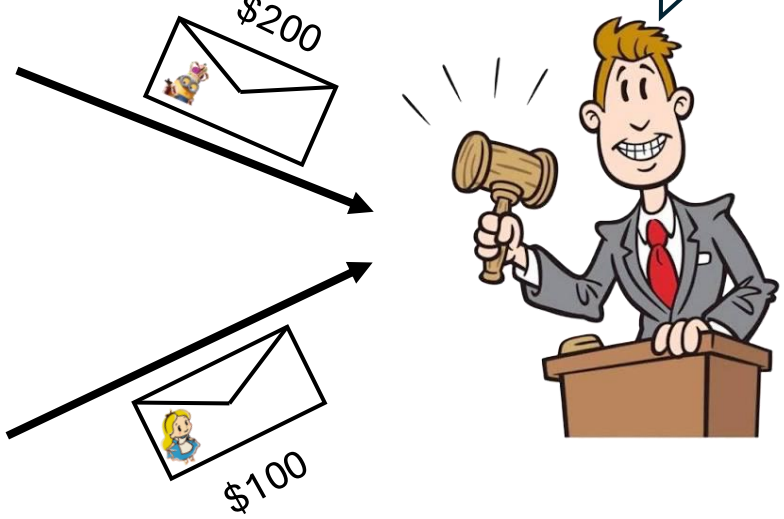
Label$_{int}$: Bob
Label$_{conf}$: Bob

```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b)
with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
    end
```

An IFC implementation







Understanding nonmalleable information flow

Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)

Label$_{int}$: Alice
Label$_{conf}$: Alice

Label$_{int}$: Bob
Label$_{conf}$: Bob

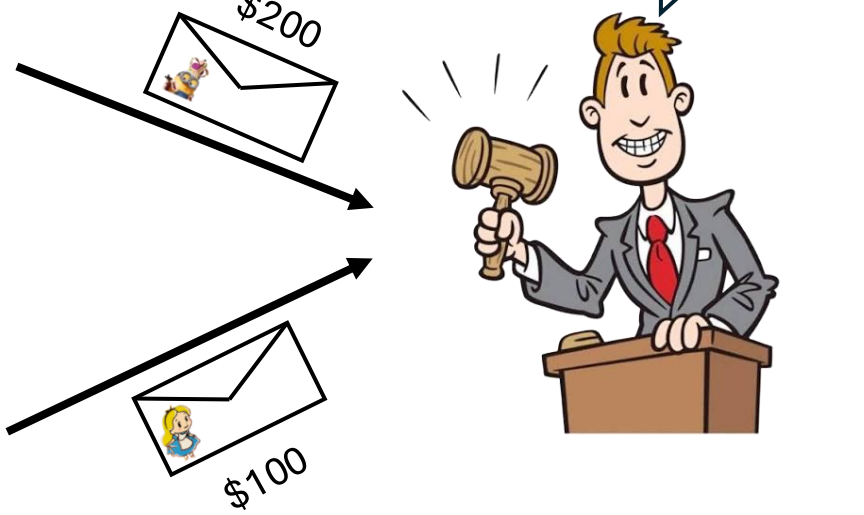
```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b)
with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
end
```

An IFC implementation

Endorsements annotate an integrity (trust) increase







An insecure implementation

Label$_{int}$: Alice
Label$_{conf}$: Alice

Label$_{int}$: Bob
Label$_{conf}$: Bob

```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b) with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
    end
```







An insecure implementation

Label$_{int}$: Bob
Label$_{conf}$: Alice ∧ Bob

Label$_{int}$: Alice
Label$_{conf}$: Alice

~~**Label$_{int}$:** Bob~~
~~**Label$_{conf}$:** Bob~~

```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b)
with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
    end
```







An insecure implementation

bob_bid might contain Alice's secrets

Label$_{int}$: Bob
Label$_{conf}$: Alice ∧ Bob

Label$_{int}$: Alice
Label$_{conf}$: Alice

~~**Label$_{int}$:** Bob~~
~~**Label$_{conf}$:** Bob~~







```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b) with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
    end
```

An insecure implementation

Label$_{int}$: Bob
Label$_{conf}$: Alice ∧ Bob

`bob_bid` might contain `Alice`'s secrets

Label$_{int}$: Alice
Label$_{conf}$: Alice

Label$_{int}$: Bob
Label$_{conf}$: Bob



Compiler may accept inputs encrypted using a **malleable encryption scheme**





```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b)
with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
    end
```

An insecure implementation

Label$_{int}$: Bob
Label$_{conf}$: Alice ∧ Bob

`bob_bid` might contain `Alice`'s secrets

Label$_{int}$: Alice
Label$_{conf}$: Alice

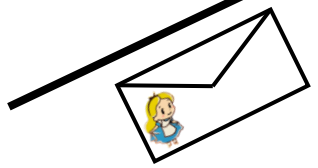
Label$_{int}$: Bob
Label$_{conf}$: Bob



Compiler may accept inputs encrypted using a **malleable encryption scheme**







```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b)
with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
    end
```

An insecure implementation

Label$_{int}$: Bob
Label$_{conf}$: Alice ∧ Bob

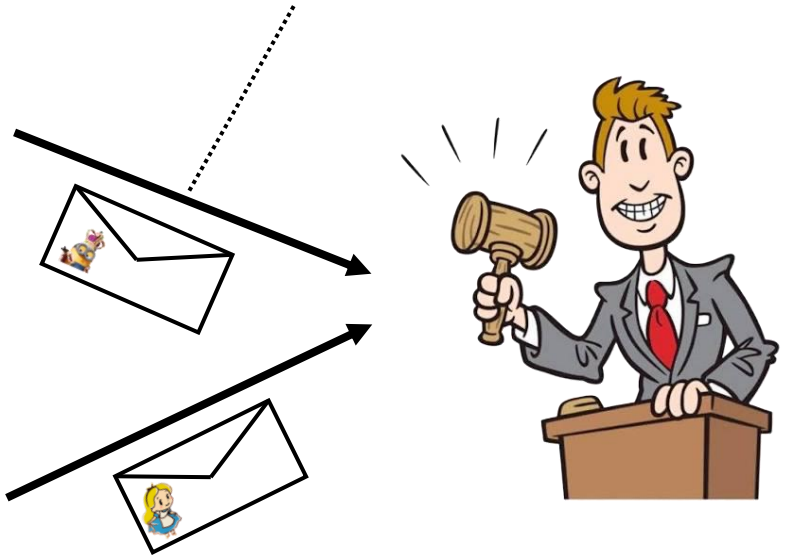
`bob_bid` might contain `Alice`'s secrets

Label$_{int}$: Alice
Label$_{conf}$: Alice

Label$_{int}$: Bob
Label$_{conf}$: Bob



Compiler may accept inputs encrypted using a **malleable encryption scheme**





```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b)
with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
    end
```

An insecure implementation

Label$_{int}$: Bob
Label$_{conf}$: Alice ∧ Bob

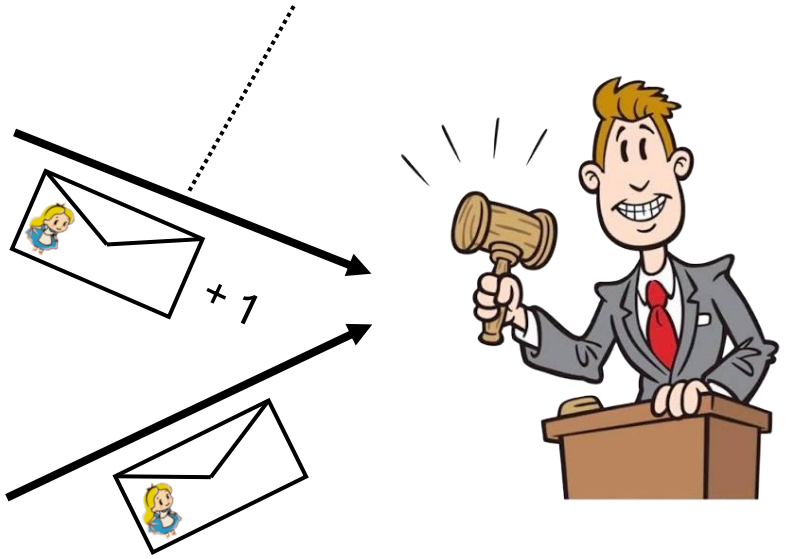
`bob_bid` might contain `Alice`'s secrets

Label$_{int}$: Alice
Label$_{conf}$: Alice

Label$_{int}$: Bob
Label$_{conf}$: Bob



Compiler may accept inputs encrypted using a **malleable encryption scheme**





```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b)
with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
    end
```

An insecure implementation

Label$_{int}$: Bob
Label$_{conf}$: Alice ∧ Bob

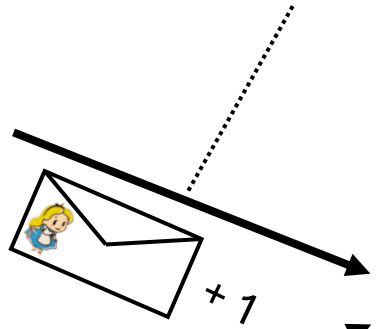
`bob_bid` might contain `Alice`'s secrets

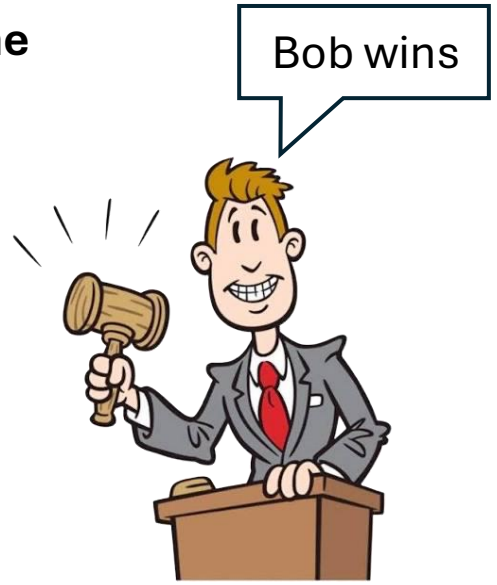
Label$_{int}$: Alice
Label$_{conf}$: Alice

Label$_{int}$: Bob
Label$_{conf}$: Bob



Compiler may accept inputs encrypted using a **malleable encryption scheme**







```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b)
with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
    end
```

An insecure implementation

Bob does not even know the value of this bid!

bob_bid might contain Alice's secrets

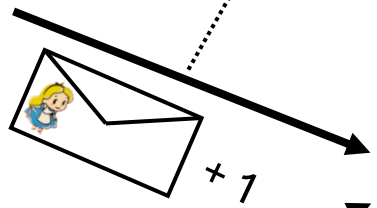
Label$_{int}$: Bob
Label$_{conf}$: Alice ∧ Bob

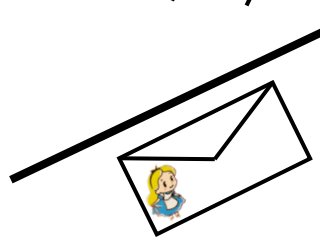
Label$_{int}$: Alice
Label$_{conf}$: Alice

~~**Label$_{int}$:** Bob~~
~~**Label$_{conf}$:** Bob~~



Compiler may accept inputs encrypted using a **malleable encryption scheme**







Bob wins



```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b) with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
    end
```

An insecure implementation

Bob does not even know the value of this bid!

bob_bid might contain Alice's secrets

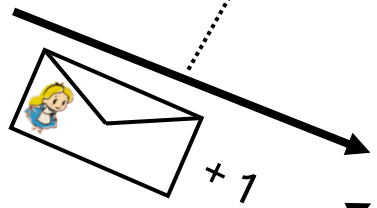
Label$_{int}$: Bob
Label$_{conf}$: Alice ∧ Bob

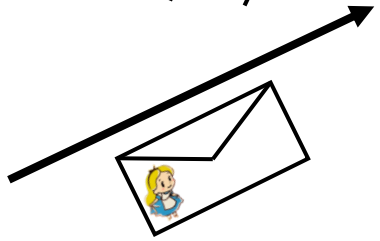
Label$_{int}$: Alice
Label$_{conf}$: Alice

~~**Label$_{int}$:** Bob~~
~~**Label$_{conf}$:** Bob~~



Compiler may accept inputs encrypted using a **malleable encryption scheme**







Bob wins



```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b) with
        | Gt => "Alice wins"
        | Lt => "Bob wins"
    end
```

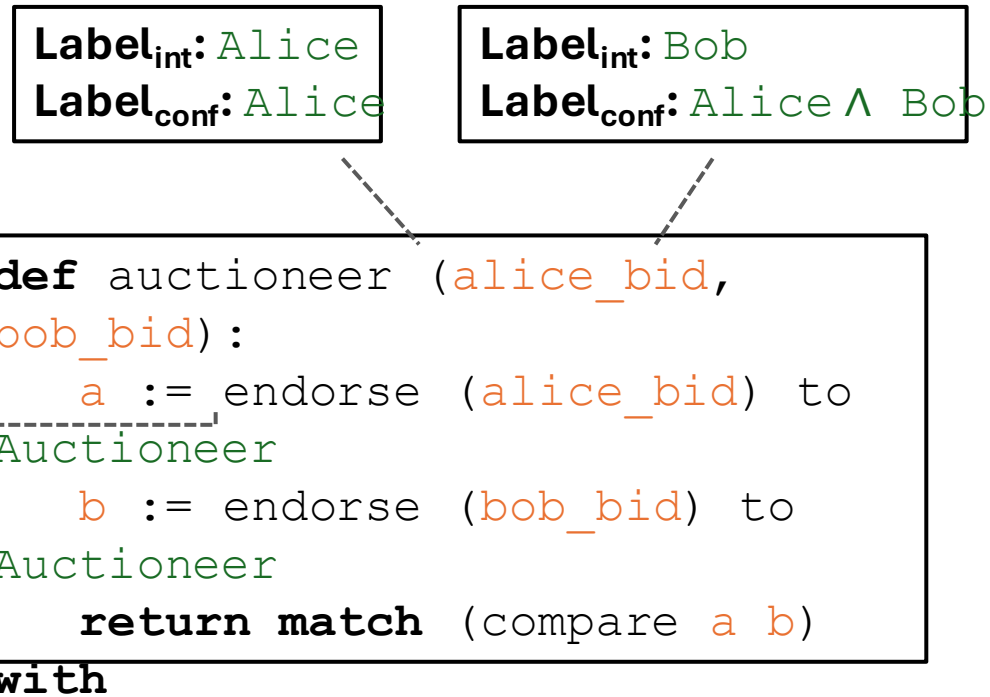
Nonmalleable information flow disallows such insecurities

CCS'17 definition is intensional



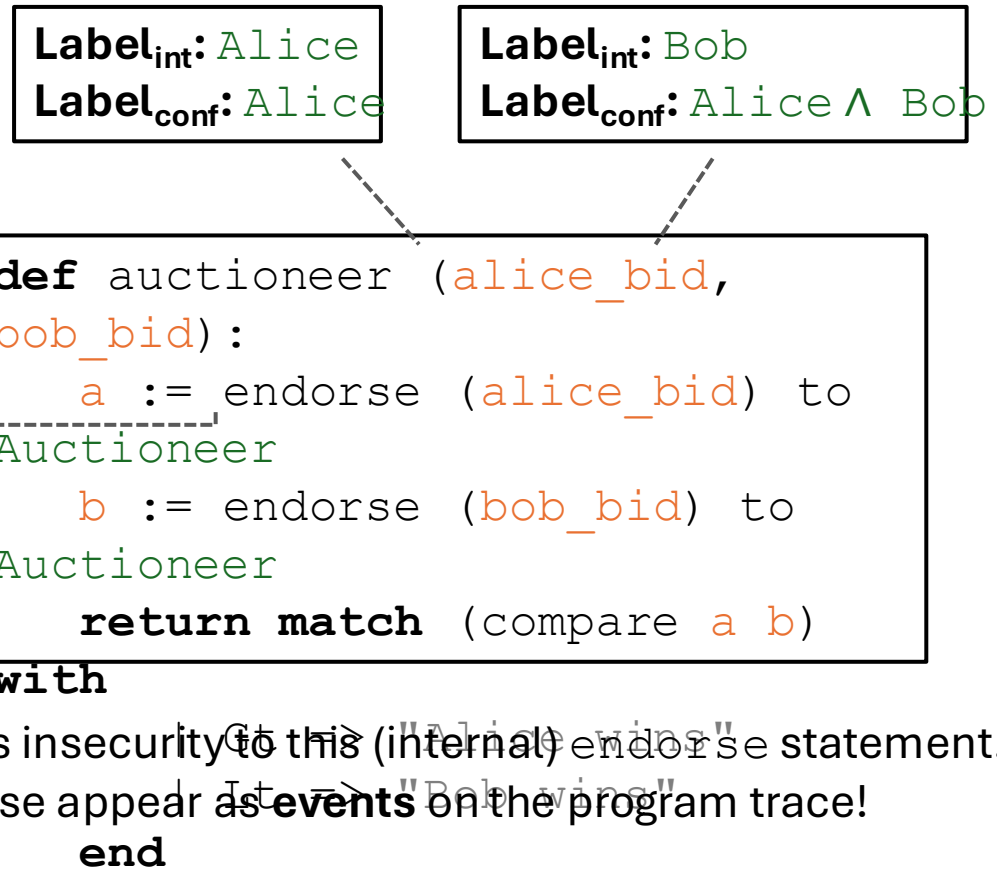

```
def auctioneer (alice_bid,
bob_bid):
   a := endorse (alice_bid) to
Auctioneer
   b := endorse (bob_bid) to
Auctioneer
   return match (compare a b)
with
      | Gt => "Alice wins"
      | Lt => "Bob wins"
   end
```

CCS'17 definition is intensional



Pins insecurity to this (internal) `endorse` statement.

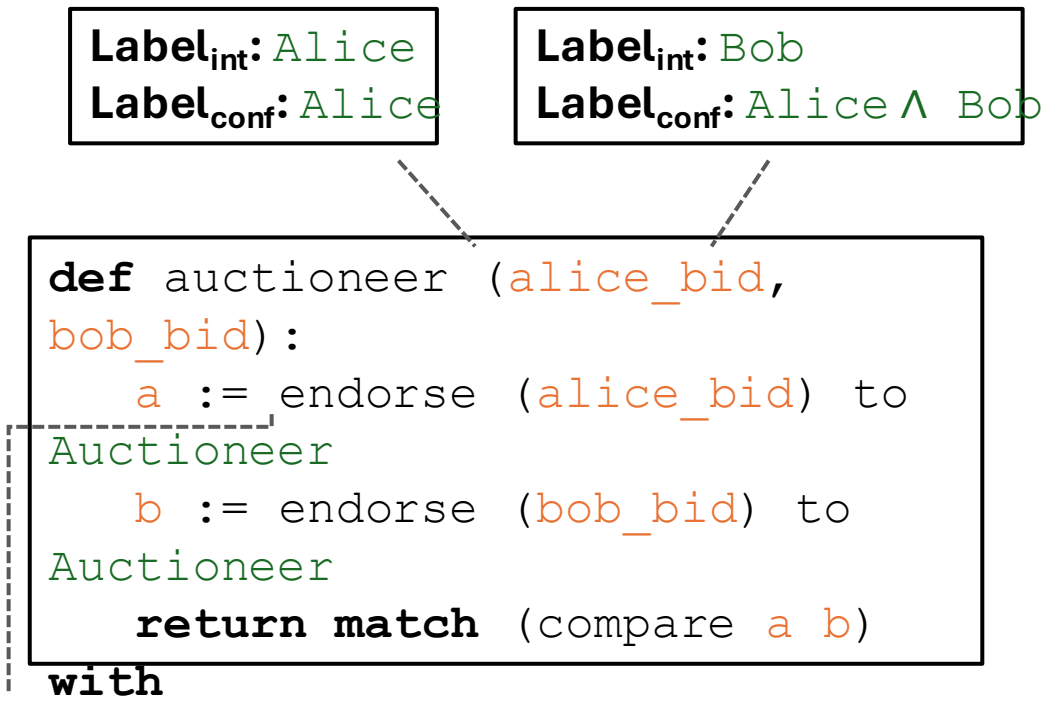
CCS'17 definition is intensional



Pins insecurity to this (internal) `endorse` statement.

These appear as **events** on the program trace!

CCS'17 definition is intensional

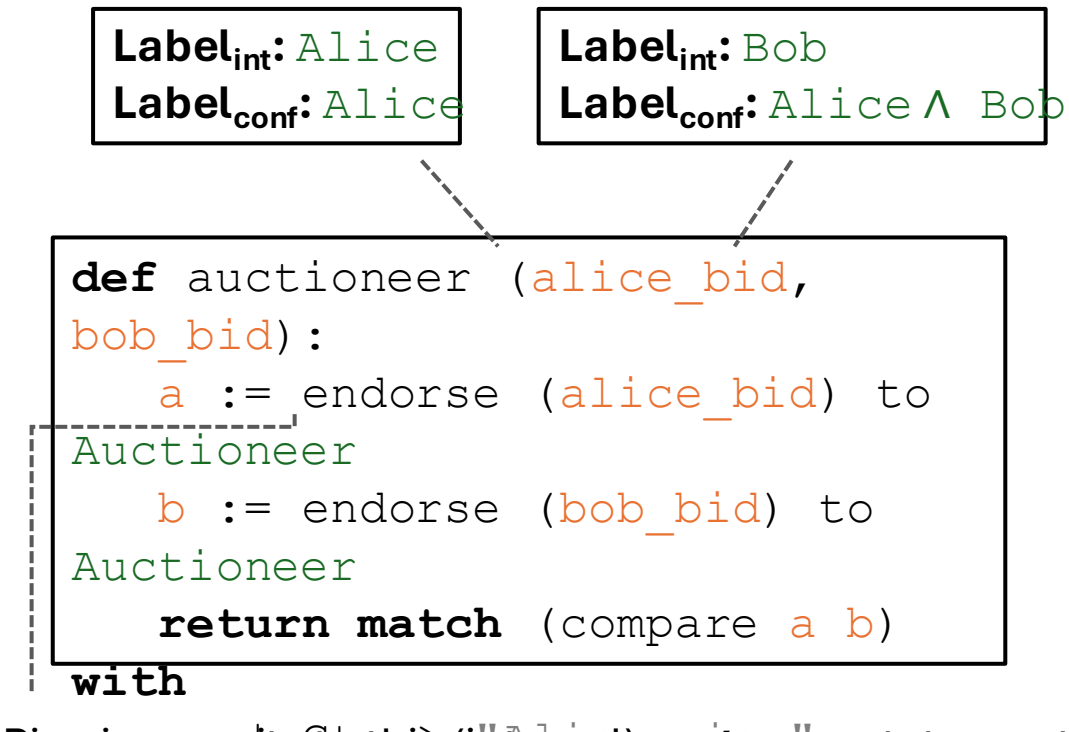


Pins insecurity to this (internal) `endorse` statement.

These appear as **events** on the program trace!

input	endorse alice_bid	endorse bob_bid	return

CCS'17 definition is intensional



Pins insecurity to this (internal) `endorse` statement.

These appear as **events** on the program trace!

input	endorse alice_bid	endorse bob_bid	return

CCS'17 definition is intensional

Label$_{int}$: Alice
Label$_{conf}$: Alice

Label$_{int}$: Bob
Label$_{conf}$: Alice ∧ Bob

```
def auctioneer (alice_bid,
bob_bid):
   a := endorse (alice_bid) to
Auctioneer
   b := endorse (bob_bid) to
Auctioneer
   return match (compare a b)
with
  | Gt -> "Alice wins"
  | _ -> "Bob wins"
end
```

Pins insecurity to this (internal) `endorse` statement.
These appear as **events** on the program trace!

input	endorse alice_bid	endorse bob_bid	return



CCS'17 definition is intensional

Label$_{int}$: Alice
Label$_{conf}$: Alice

Label$_{int}$: Bob
Label$_{conf}$: Alice ∧ Bob

```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b)
    with
    | Gt -> "Alice wins"
    | _ -> "Bob wins"
    end
```

```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return 0
```

Pins insecurity to this (internal) `endorse` statement.
These appear as **events** on the program trace!

input	endorse alice_bid	endorse bob_bid	return



CCS'17 definition is intensional

Label$_{int}$: Alice
Label$_{conf}$: Alice

Label$_{int}$: Bob
Label$_{conf}$: Alice ∧ Bob

```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b)
with
  | Gt -> "Alice wins"
  | _ -> "Bob wins"
end
```

```
def auctioneer (alice_bid, bob_bid):
return 0
```

```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return 0
```

Pins insecurity to this (internal) `endorse` statement.

These appear as **events** on the program trace!

input	endorse alice_bid	endorse bob_bid	return



CCS'17 definition is intensional

Label$_{int}$: Alice
Label$_{conf}$: Alice

Label$_{int}$: Bob
Label$_{conf}$: Alice ∧ Bob

```
def auctioneer (alice_bid,
bob_bid):
   a := endorse (alice_bid) to
Auctioneer
   b := endorse (bob_bid) to
Auctioneer
   return match (compare a b)
with
   | Gt -> "Alice wins"
   | _ -> "Bob wins"
   end
```

Pins insecurity to this (internal) `endorse` statement.

These appear as **events** on the program trace!

input	endorse alice_bid	endorse bob_bid	return



```
def auctioneer (alice_bid,
bob_bid):
return 0
```

```
def auctioneer (alice_bid,
bob_bid):
   a := endorse (alice_bid) to
Auctioneer
   b := endorse (bob_bid) to
```

input	endorse alice_bid	endorse bob_bid	return

CCS'17 definition is intensional

Label$_{int}$: Alice
Label$_{conf}$: Alice

Label$_{int}$: Bob
Label$_{conf}$: Alice ∧ Bob

```
def auctioneer (alice_bid,
bob_bid):
   a := endorse (alice_bid) to
Auctioneer
   b := endorse (bob_bid) to
Auctioneer
   return match (compare a b)
with
   | Gt -> "Alice wins"
   | _ -> "Bob wins"
   end
```

Pins insecurity to this (internal) `endorse` statement.

These appear as **events** on the program trace!

input	endorse alice_bid	endorse b_bid	return



```
def auctioneer (alice_bid,
bob_bid):
return 0
```

```
def auctioneer (alice_bid,
bob_bid):
   a := endorse (alice_bid) to
Auctioneer
   b := endorse (bob_bid) to
```

input	endorse alice_bid	endorse b_bid	return

CCS'17 definition is intensional

Label$_{int}$: Alice
Label$_{conf}$: Alice

Label$_{int}$: Bob
Label$_{conf}$: Alice ∧ Bob

```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to Auctioneer
    return match (compare a b)
with
  | Gt -> "Alice wins"
  | _ -> "Bob wins"
end
```

Pins insecurity to this (internal) `endorse` statement.
These appear as **events** on the program trace!

input	endorse alice_bid	endorse bob_bid	return



```
def auctioneer (alice_bid, bob_bid):
return 0
```

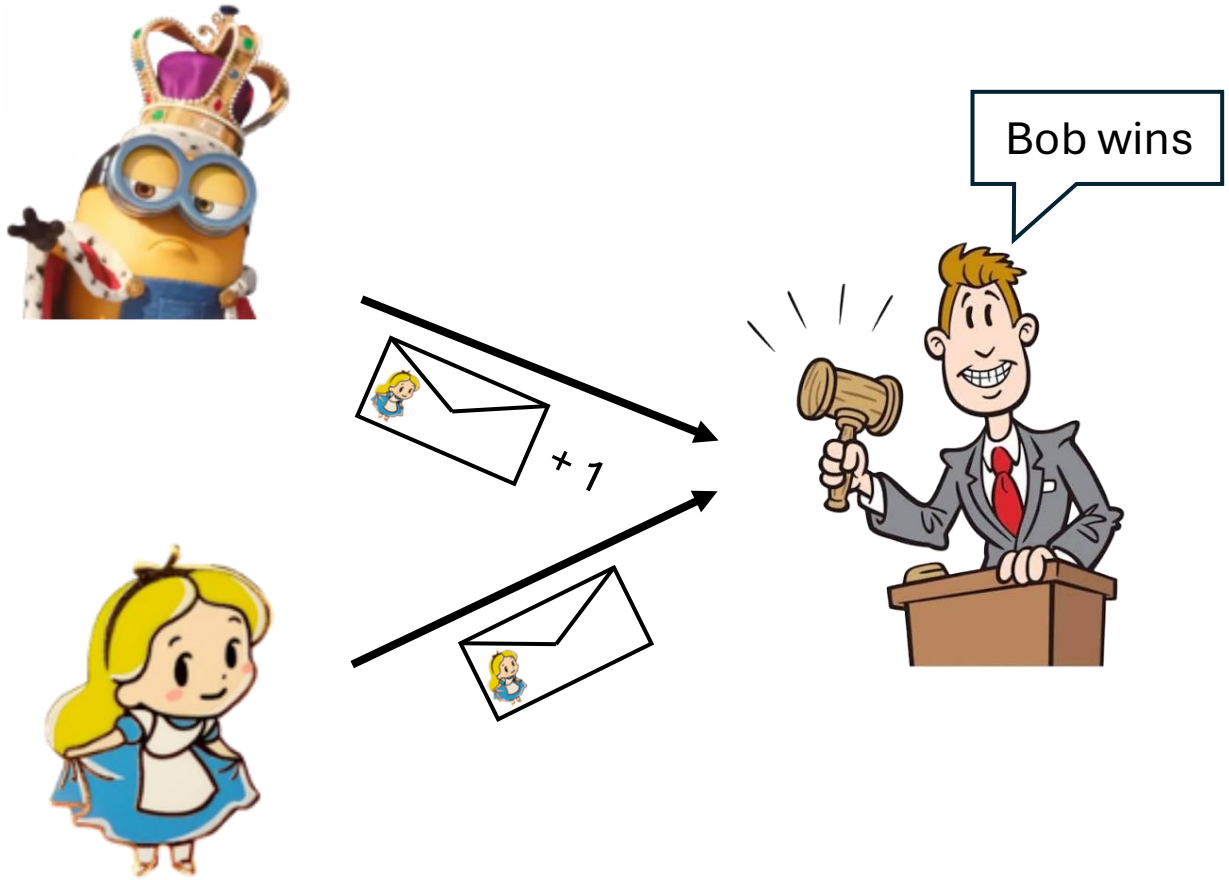
```
def auctioneer (alice_bid, bob_bid):
    a := endorse (alice_bid) to Auctioneer
    b := endorse (bob_bid) to
```

input	endorse alice_bid	endorse bob_bid	return

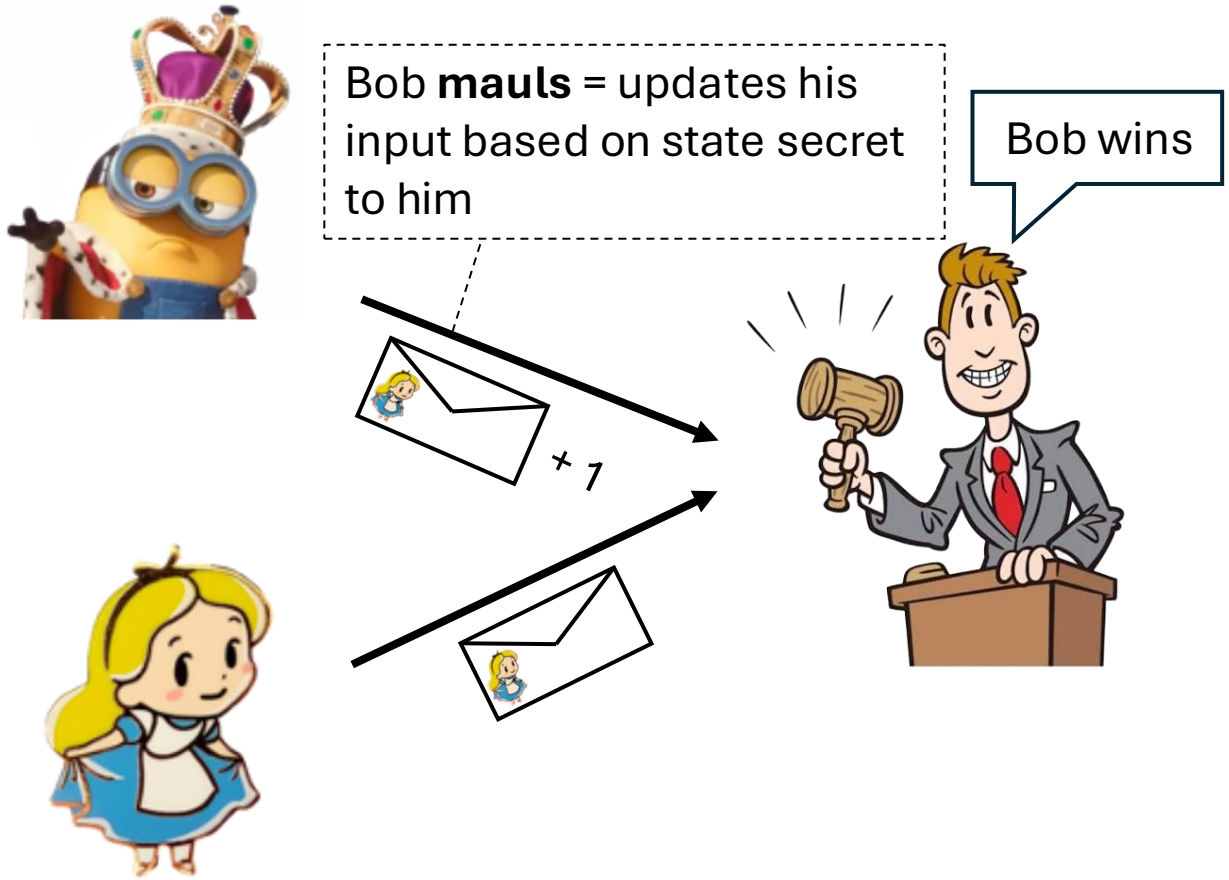


Can we do better?

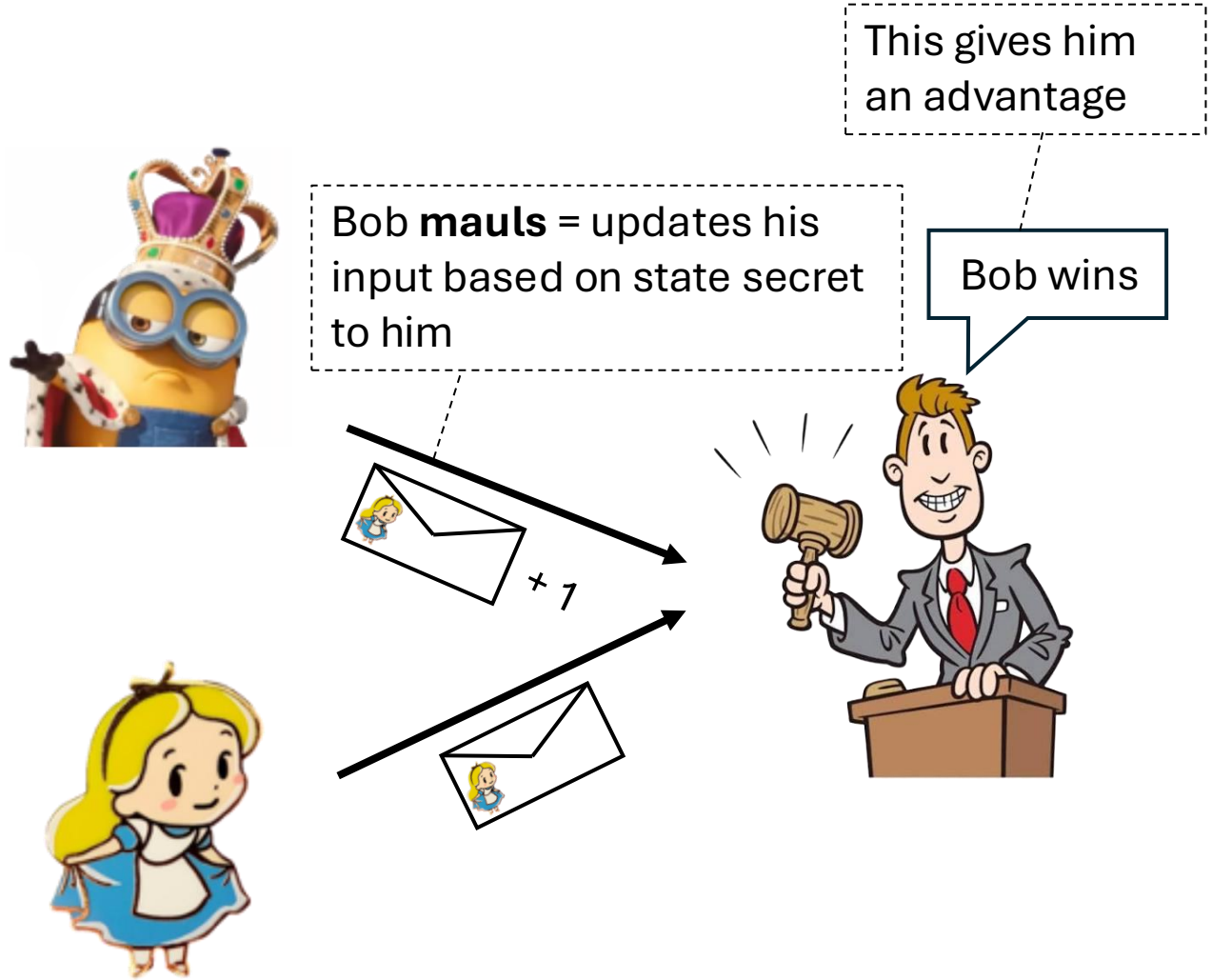
Can we do better?



Can we do better?

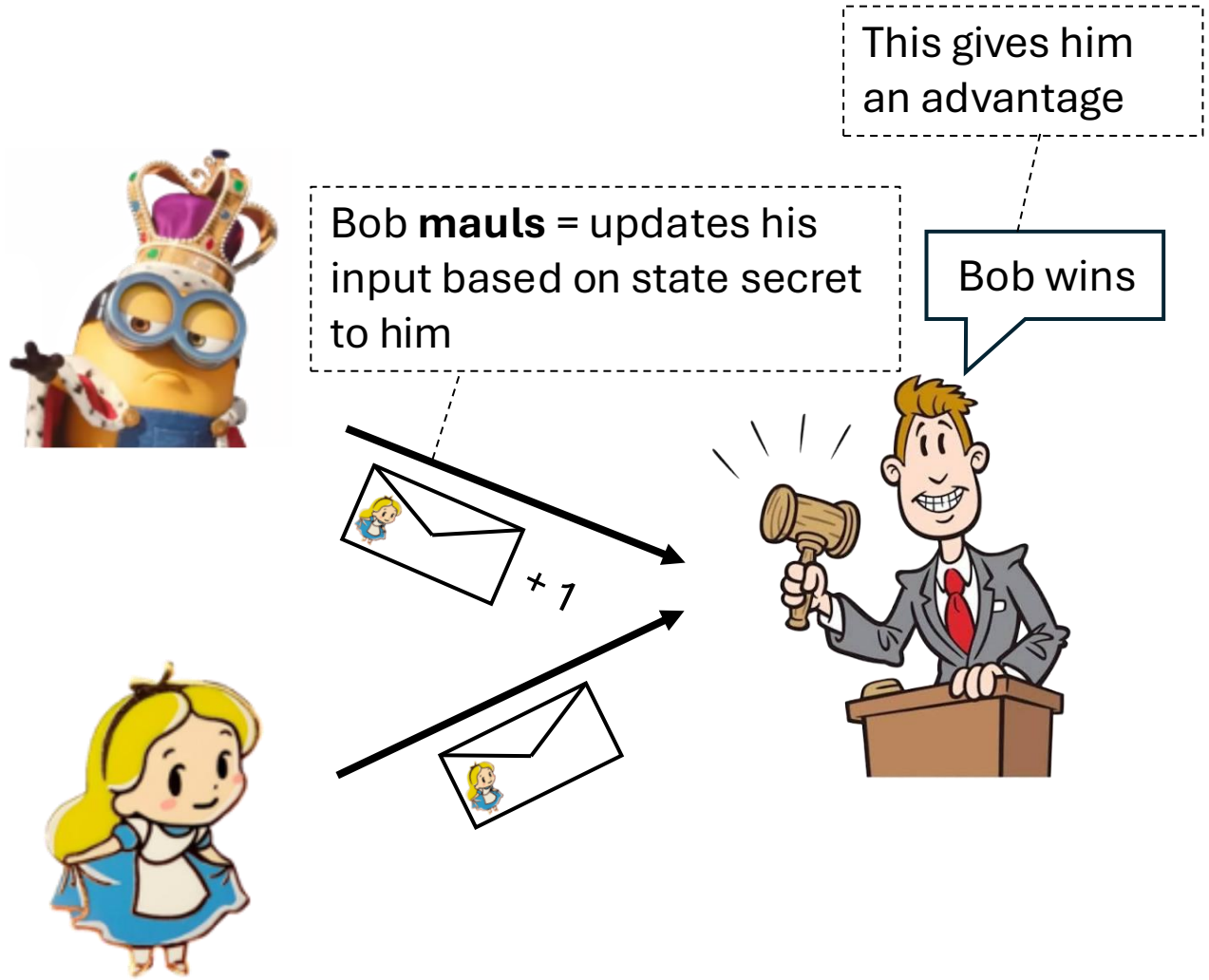


Can we do better?



Can we do better?

Can we eliminate Bob's ability to maul?

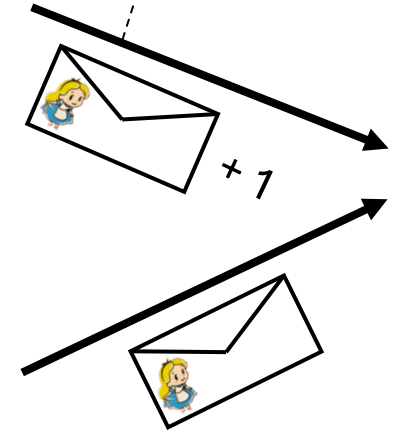


Can we do better?

Can we eliminate Bob's ability to maul? ✗









This gives him an advantage

Bob **mauls** = updates his input based on state secret to him

Bob wins

Can we do better?

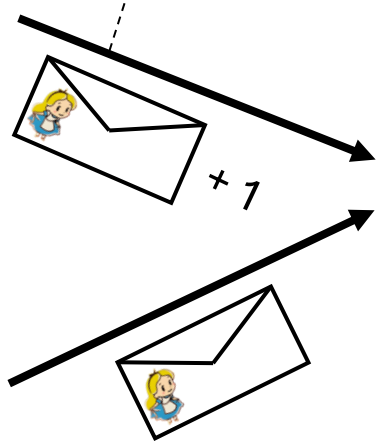
Can we eliminate Bob's ability to maul? ✗







Bob **mauls** = updates his input based on state secret to him



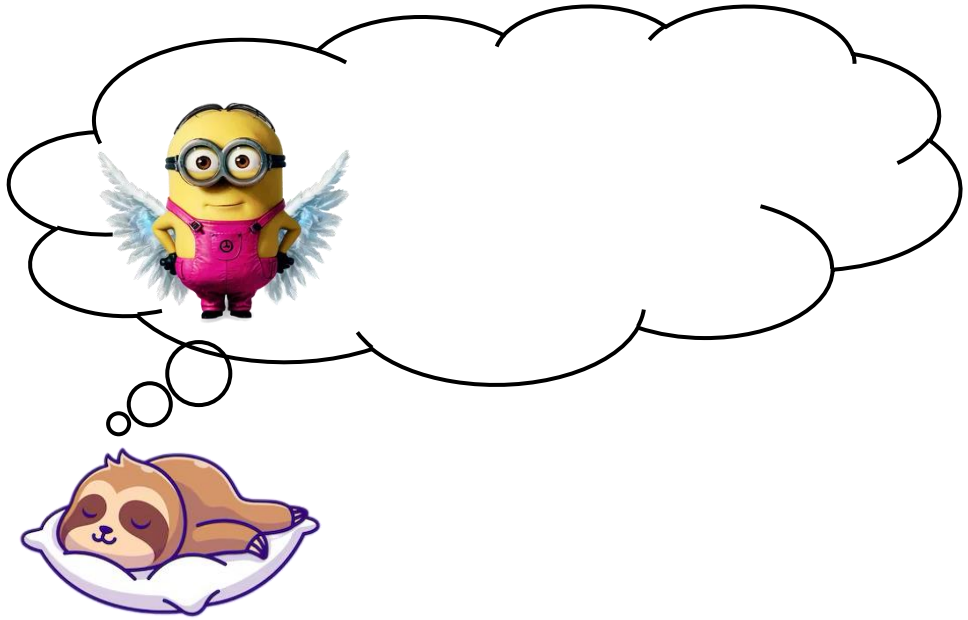
This gives him an advantage

Bob wins



Can we do better?

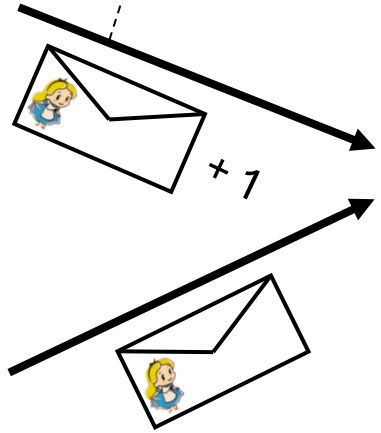
Can we eliminate Bob's ability to maul?







Bob **mauls** = updates his input based on state secret to him



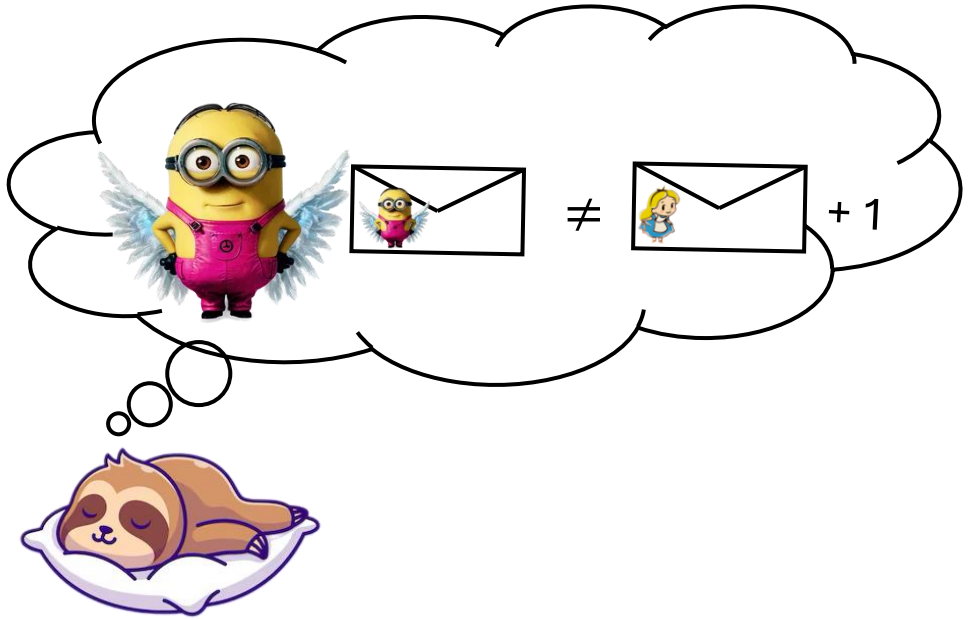
This gives him an advantage

Bob wins



Can we do better?

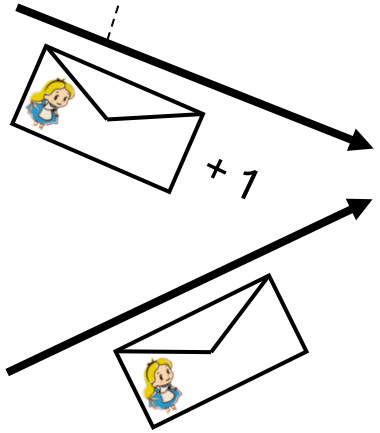
Can we eliminate Bob's ability to maul?







Bob **mauls** = updates his input based on state secret to him



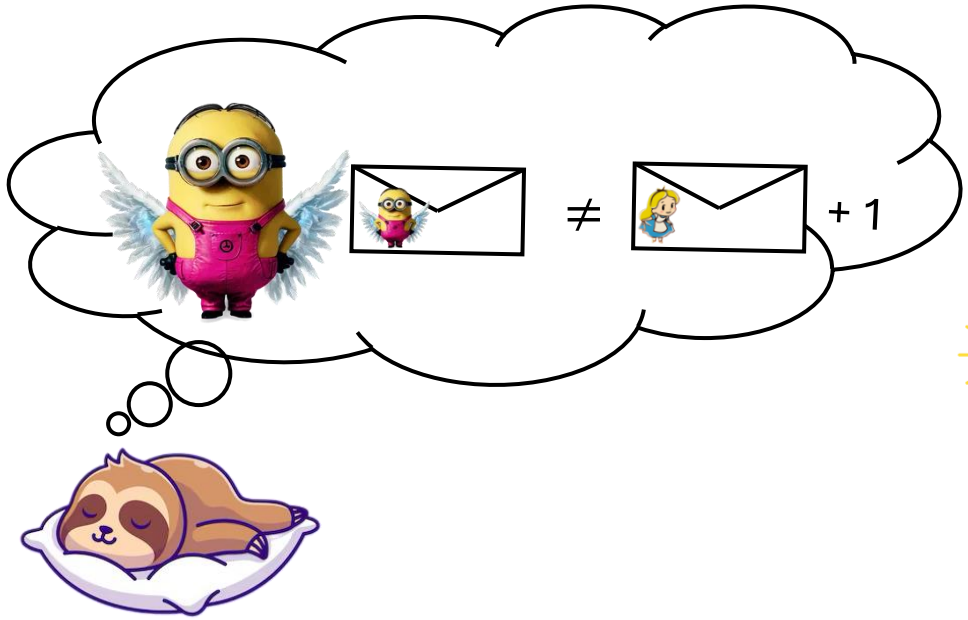
This gives him an advantage

Bob wins



Can we do better?

Can we eliminate Bob's ability to maul?

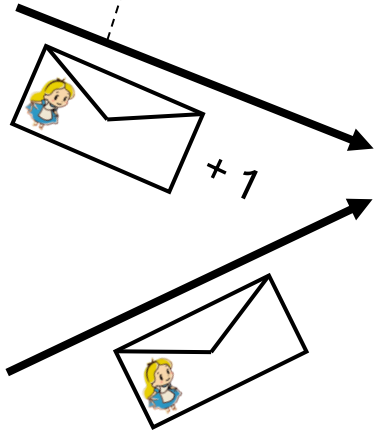








Bob **mauls** = updates his input based on state secret to him



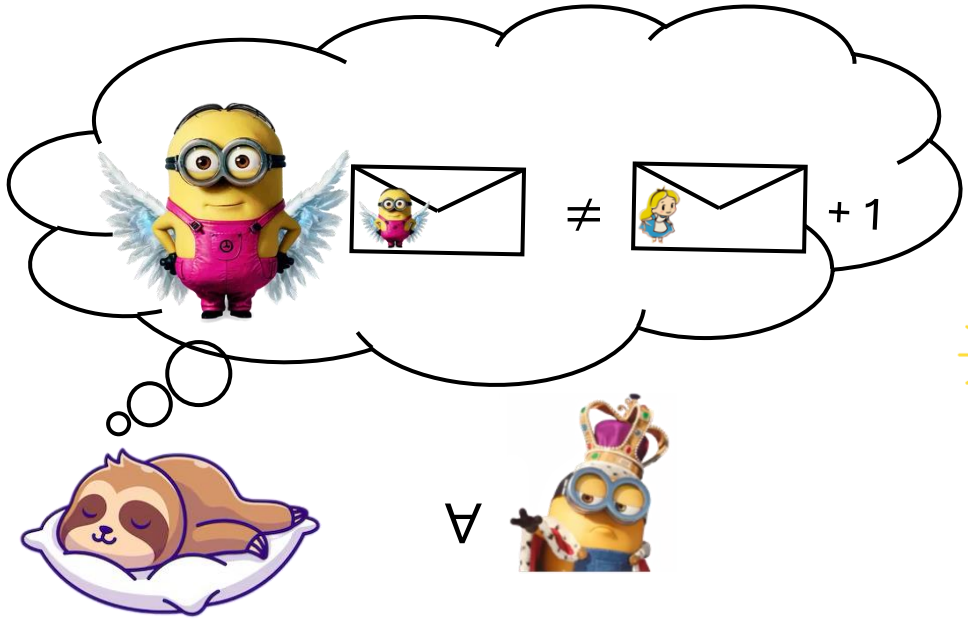
This gives him an advantage

Bob wins



Can we do better?

Can we eliminate Bob's ability to maul?

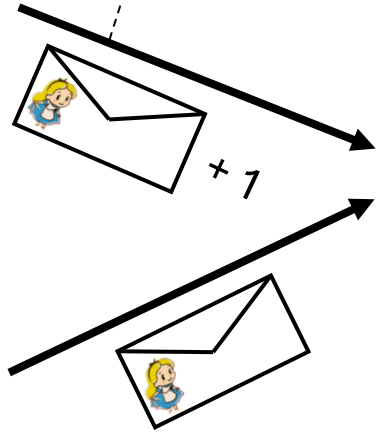








Bob **mauls** = updates his input based on state secret to him



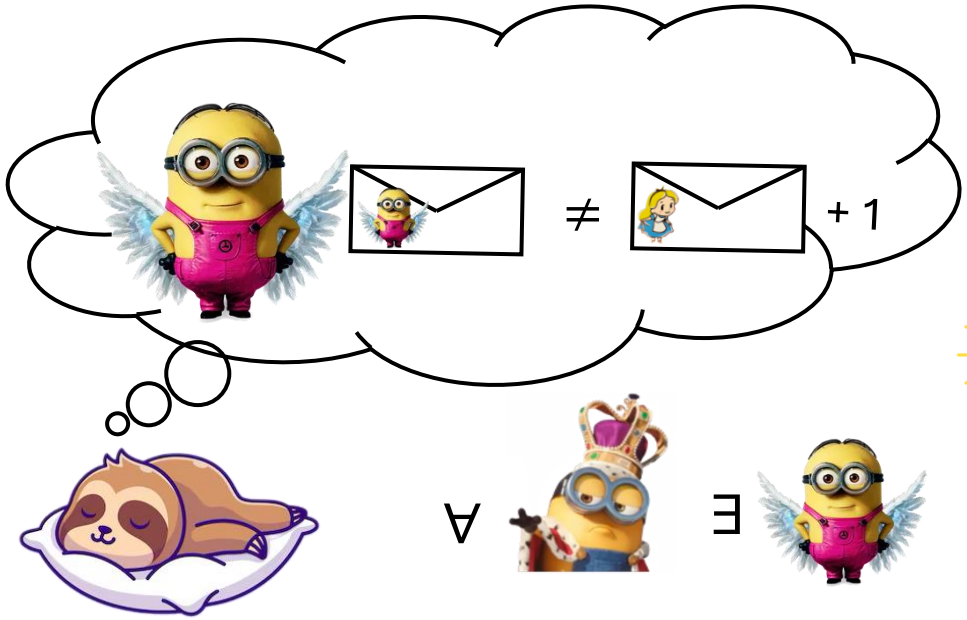
This gives him an advantage

Bob wins



Can we do better?

Can we eliminate Bob's ability to maul?

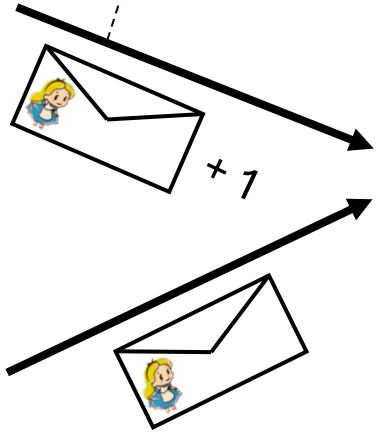








Bob **mauls** = updates his input based on state secret to him



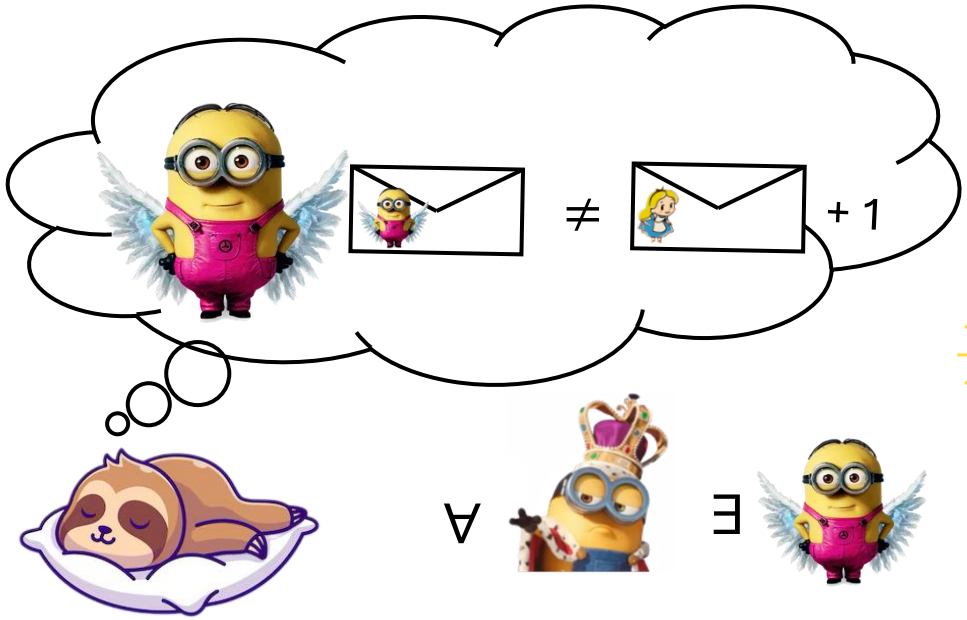
This gives him an advantage

Bob wins



Can we do better?

Can we eliminate Bob's ability to maul?





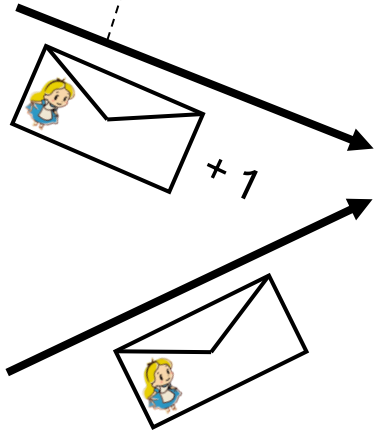




∀ ∃

with the same behavior

Bob **mauls** = updates his input based on state secret to him



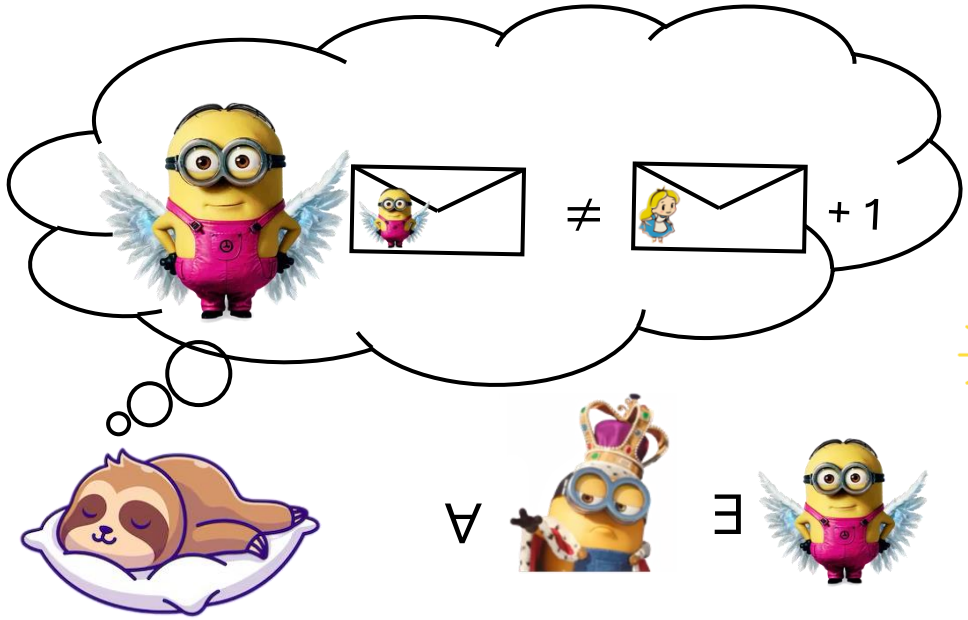
This gives him an advantage

Bob wins



Can we do better?

Can we eliminate Bob's ability to maul?

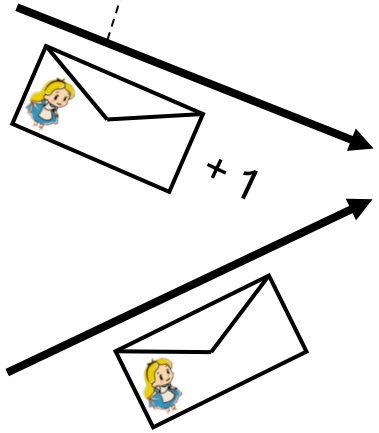


with the same behavior





Bob **mauls** = updates his input based on state secret to him



This gives him an advantage

Bob wins



This is a real–ideal statement!

ImpThreat: a language-based formalism

ImpThreat: a language-based formalism

ValExp	e_v	::=	$x \oplus y$
BoolExp	e_b	::=	$t(x)$
Command	c	::=	$x := e_v$ | `if` e_b `then` p `else` p
		|	`while` e_b `do` p | `output` x
Program	p	::=	ε | $c\,;p$

ImpThreat: a language-based formalism

ValExp	e_v	::=	$x \oplus y$
BoolExp	e_b	::=	$t(x)$
Command	c	::=	$x := e_v$ | `if` e_b `then` p `else` p
		|	`while` e_b `do` p | `output` x
Program	p	::=	ε | c ; p

ImpThreat: a language-based formalism

ValExp	e_v	::=	$x \oplus y$
BoolExp	e_b	::=	$t(x)$
Command	c	::=	$x := e_v$ | `if` e_b `then` p `else` p
		|	`while` e_b `do` p | `output` x
Program	p	::=	ε | c ; p

ImpThreat: a language-based formalism

ValExp	e_v	::=	$x \oplus y$
BoolExp	e_b	::=	$t(x)$
Command	c	::=	$x := e_v$ | `if` e_b `then` p `else` p
		|	`while` e_b `do` p | `output` x
Program	p	::=	ε | c ; p

ImpThreat: a language-based formalism

ValExp	e_v	::=	$x \oplus y$
BoolExp	e_b	::=	$t(x)$
Command	c	::=	$x := e_v$ | `if` e_b `then` p `else` p
		|	`while` e_b `do` p | `output` x
Program	p	::=	ε | c ; p

ImpThreat: a language-based formalism

ValExp	e_v	::=	$x \oplus y$
BoolExp	e_b	::=	$t(x)$
Command	c	::=	$x := e_v$ | `if` e_b `then` p `else` p
		|	`while` e_b `do` p | `output` x
Program	p	::=	ε | $c\,;p$

ImpThreat: a language-based formalism

$$
\begin{array}{llll}
\text{ValExp} & e_v & ::= & x \oplus y \\
\text{BoolExp} & e_b & ::= & t(x) \\
\text{Command} & c & ::= & x := e_v \mid \texttt{if } e_b \texttt{ then } p \texttt{ else } p \\
 & & \mid & \texttt{while } e_b \texttt{ do } p \mid \texttt{output } x \\
\text{Program} & p & ::= & \varepsilon \mid c\,;p
\end{array}
$$

S-ASSIGN

$$\frac{[\![e_v]\!]\sigma = v}{\langle x := e_v,\ \sigma\rangle \xrightarrow{\text{store}} \sigma[x \mapsto v]}$$

C-IF-T

$$\frac{[\![e_b]\!]\sigma = \texttt{true}}{\langle \texttt{if } e_b \texttt{ then } p_1 \texttt{ else } p_2,\ \sigma\rangle \xrightarrow{\text{cont}} p_1}$$

C-WHILE-T

$$\frac{[\![e_b]\!]\sigma = \texttt{true}}{\langle \texttt{while } e_b \texttt{ do } p,\ \sigma\rangle \xrightarrow{\text{cont}} p\,;\texttt{while } e_b \texttt{ do } p}$$

C-IF-F

$$\frac{[\![e_b]\!]\sigma = \texttt{false}}{\langle \texttt{if } e_b \texttt{ then } p_1 \texttt{ else } p_2,\ \sigma\rangle \xrightarrow{\text{cont}} p_2}$$

C-WHILE-F

$$\frac{[\![e_b]\!]\sigma = \texttt{false}}{\langle \texttt{while } e_b \texttt{ do } p,\ \sigma\rangle \xrightarrow{\text{cont}} \epsilon}$$

ImpThreat: a language-based formalism

$$
\begin{array}{llll}
\text{ValExp} & e_v & ::= & x \oplus y \\
\text{BoolExp} & e_b & ::= & t(x) \\
\text{Command} & c & ::= & x := e_v \mid \texttt{if } e_b \texttt{ then } p \texttt{ else } p \\
 & & \mid & \texttt{while } e_b \texttt{ do } p \mid \texttt{output } x \\
\text{Program} & p & ::= & \varepsilon \mid c\,;p
\end{array}
$$

S-ASSIGN

$$\frac{[\![e_v]\!]\sigma = v}{\langle x := e_v,\ \sigma\rangle \xrightarrow{\text{store}} \sigma[x \mapsto v]}$$

store evolution

C-IF-T

$$\frac{[\![e_b]\!]\sigma = \texttt{true}}{\langle \texttt{if } e_b \texttt{ then } p_1 \texttt{ else } p_2,\ \sigma\rangle \xrightarrow{\text{cont}} p_1}$$

C-WHILE-T

$$\frac{[\![e_b]\!]\sigma = \texttt{true}}{\langle \texttt{while } e_b \texttt{ do } p,\ \sigma\rangle \xrightarrow{\text{cont}} p\,;\texttt{while } e_b \texttt{ do } p}$$

C-IF-F

$$\frac{[\![e_b]\!]\sigma = \texttt{false}}{\langle \texttt{if } e_b \texttt{ then } p_1 \texttt{ else } p_2,\ \sigma\rangle \xrightarrow{\text{cont}} p_2}$$

C-WHILE-F

$$\frac{[\![e_b]\!]\sigma = \texttt{false}}{\langle \texttt{while } e_b \texttt{ do } p,\ \sigma\rangle \xrightarrow{\text{cont}} \epsilon}$$

ImpThreat: a language-based formalism

$$
\begin{array}{lllll}
\text{ValExp} & e_v & ::= & x \oplus y \\
\text{BoolExp} & e_b & ::= & t(x) \\
\text{Command} & c & ::= & x := e_v \mid \texttt{if } e_b \texttt{ then } p \texttt{ else } p \\
 & & \mid & \texttt{while } e_b \texttt{ do } p \mid \texttt{output } x \\
\text{Program} & p & ::= & \varepsilon \mid c\,;p
\end{array}
$$

S-ASSIGN

$$\frac{[\![e_v]\!]\sigma = v}{\langle x := e_v,\ \sigma\rangle \xrightarrow{\text{store}} \sigma[x \mapsto v]}$$

store evolution

C-IF-T

$$\frac{[\![e_b]\!]\sigma = \texttt{true}}{\langle \texttt{if } e_b \texttt{ then } p_1 \texttt{ else } p_2,\ \sigma\rangle \xrightarrow{\text{cont}} p_1}$$

C-IF-F

$$\frac{[\![e_b]\!]\sigma = \texttt{false}}{\langle \texttt{if } e_b \texttt{ then } p_1 \texttt{ else } p_2,\ \sigma\rangle \xrightarrow{\text{cont}} p_2}$$

C-WHILE-T

$$\frac{[\![e_b]\!]\sigma = \texttt{true}}{\langle \texttt{while } e_b \texttt{ do } p,\ \sigma\rangle \xrightarrow{\text{cont}} p\,;\texttt{while } e_b \texttt{ do } p}$$

C-WHILE-F

$$\frac{[\![e_b]\!]\sigma = \texttt{false}}{\langle \texttt{while } e_b \texttt{ do } p,\ \sigma\rangle \xrightarrow{\text{cont}} \epsilon}$$

program continuation

ImpThreat: a language-based formalism

ValExp	e_v	::=	$x \oplus y$
BoolExp	e_b	::=	$t(x)$
Command	c	::=	$x := e_v \mid \texttt{if } e_b \texttt{ then } p \texttt{ else } p$
		$\mid$	$\texttt{while } e_b \texttt{ do } p \mid \texttt{output } x$
Program	p	::=	$\varepsilon \mid c\,;p$

S-ASSIGN

$$\frac{[\![e_v]\!]\sigma = v}{\langle x := e_v,\ \sigma\rangle \xrightarrow{\text{store}} \sigma[x \mapsto v]}$$

store evolution

C-IF-T

$$\frac{[\![e_b]\!]\sigma = \texttt{true}}{\langle \texttt{if } e_b \texttt{ then } p_1 \texttt{ else } p_2,\ \sigma\rangle \xrightarrow{\text{cont}} p_1}$$

C-IF-F

$$\frac{[\![e_b]\!]\sigma = \texttt{false}}{\langle \texttt{if } e_b \texttt{ then } p_1 \texttt{ else } p_2,\ \sigma\rangle \xrightarrow{\text{cont}} p_2}$$

C-WHILE-T

$$\frac{[\![e_b]\!]\sigma = \texttt{true}}{\langle \texttt{while } e_b \texttt{ do } p,\ \sigma\rangle \xrightarrow{\text{cont}} p\,;\texttt{while } e_b \texttt{ do } p}$$

C-WHILE-F

$$\frac{[\![e_b]\!]\sigma = \texttt{false}}{\langle \texttt{while } e_b \texttt{ do } p,\ \sigma\rangle \xrightarrow{\text{cont}} \epsilon}$$

program continuation

ImpThreat: a language-based formalism

ValExp	e_v	::=	$x \oplus y$
BoolExp	e_b	::=	$t(x)$
Command	c	::=	$x := e_v \mid \texttt{if } e_b \texttt{ then } p \texttt{ else } p$
		$\mid$	$\texttt{while } e_b \texttt{ do } p \mid \texttt{output } x$
Program	p	::=	$\varepsilon \mid c\,;p$

S-ASSIGN

$$\frac{[\![e_v]\!]\sigma = v}{\langle x := e_v,\ \sigma\rangle \xrightarrow{\text{store}} \sigma[x \mapsto v]}$$

store evolution

C-IF-T

$$\frac{[\![e_b]\!]\sigma = \texttt{true}}{\langle \texttt{if } e_b \texttt{ then } p_1 \texttt{ else } p_2,\ \sigma\rangle \xrightarrow{\text{cont}} p_1}$$

C-IF-F

$$\frac{[\![e_b]\!]\sigma = \texttt{false}}{\langle \texttt{if } e_b \texttt{ then } p_1 \texttt{ else } p_2,\ \sigma\rangle \xrightarrow{\text{cont}} p_2}$$

C-WHILE-T

$$\frac{[\![e_b]\!]\sigma = \texttt{true}}{\langle \texttt{while } e_b \texttt{ do } p,\ \sigma\rangle \xrightarrow{\text{cont}} p\,;\texttt{while } e_b \texttt{ do } p}$$

C-WHILE-F

$$\frac{[\![e_b]\!]\sigma = \texttt{false}}{\langle \texttt{while } e_b \texttt{ do } p,\ \sigma\rangle \xrightarrow{\text{cont}} \epsilon}$$

program continuation

ImpThreat: a language-based formalism

ValExp	e_v	::=	$x \oplus y$
BoolExp	e_b	::=	$t(x)$
Command	c	::=	$x := e_v \mid \texttt{if } e_b \texttt{ then } p \texttt{ else } p$
		$\mid$	$\texttt{while } e_b \texttt{ do } p \mid \texttt{output } x$
Program	p	::=	$\varepsilon \mid c\,;p$

S-ASSIGN

$$\frac{[\![e_v]\!]\sigma = v}{\langle x := e_v,\ \sigma\rangle \xrightarrow{\text{store}} \sigma[x \mapsto v]}$$

store evolution

C-IF-T

$$\frac{[\![e_b]\!]\sigma = \texttt{true}}{\langle \texttt{if } e_b \texttt{ then } p_1 \texttt{ else } p_2,\ \sigma\rangle \xrightarrow{\text{cont}} p_1}$$

C-IF-F

$$\frac{[\![e_b]\!]\sigma = \texttt{false}}{\langle \texttt{if } e_b \texttt{ then } p_1 \texttt{ else } p_2,\ \sigma\rangle \xrightarrow{\text{cont}} p_2}$$

C-WHILE-T

$$\frac{[\![e_b]\!]\sigma = \texttt{true}}{\langle \texttt{while } e_b \texttt{ do } p,\ \sigma\rangle \xrightarrow{\text{cont}} p\,;\texttt{while } e_b \texttt{ do } p}$$

C-WHILE-F

$$\frac{[\![e_b]\!]\sigma = \texttt{false}}{\langle \texttt{while } e_b \texttt{ do } p,\ \sigma\rangle \xrightarrow{\text{cont}} \epsilon}$$

program continuation

ImpThreat: a language-based formalism

$$
\begin{array}{lllll}
\text{ValExp} & e_v & ::= & x \oplus y \\
\text{BoolExp} & e_b & ::= & t(x) \\
\text{Command} & c & ::= & x := e_v \mid \texttt{if } e_b \texttt{ then } p \texttt{ else } p \\
 & & \mid & \texttt{while } e_b \texttt{ do } p \mid \texttt{output } x \\
\text{Program} & p & ::= & \varepsilon \mid c\,;p
\end{array}
$$

S-ASSIGN

$$\frac{[\![e_v]\!]\sigma = v}{\langle x := e_v,\ \sigma\rangle \xrightarrow{\text{store}} \sigma[x \mapsto v]}$$

store evolution

C-IF-T

$$\frac{[\![e_b]\!]\sigma = \texttt{true}}{\langle \texttt{if } e_b \texttt{ then } p_1 \texttt{ else } p_2,\ \sigma\rangle \xrightarrow{\text{cont}} p_1}$$

C-IF-F

$$\frac{[\![e_b]\!]\sigma = \texttt{false}}{\langle \texttt{if } e_b \texttt{ then } p_1 \texttt{ else } p_2,\ \sigma\rangle \xrightarrow{\text{cont}} p_2}$$

C-WHILE-T

$$\frac{[\![e_b]\!]\sigma = \texttt{true}}{\langle \texttt{while } e_b \texttt{ do } p,\ \sigma\rangle \xrightarrow{\text{cont}} p\,;\texttt{while } e_b \texttt{ do } p}$$

C-WHILE-F

$$\frac{[\![e_b]\!]\sigma = \texttt{false}}{\langle \texttt{while } e_b \texttt{ do } p,\ \sigma\rangle \xrightarrow{\text{cont}} \epsilon}$$

program continuation

Threat model and sandwich semantics

Threat model and sandwich semantics



Threat model and sandwich semantics

Variable
v_1
v_2
v_3
…



Threat model and sandwich semantics

Variable	Public?
v_1	✗
v_2	✗
v_3	✓
…	…



Threat model and sandwich semantics

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…



Threat model and sandwich semantics

Untrusted variables can be **mauled**

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…

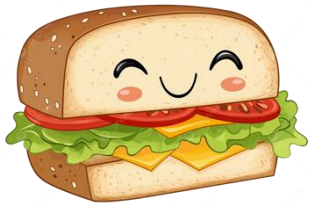


Threat model and sandwich semantics

Untrusted variables can be **mauled**

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…





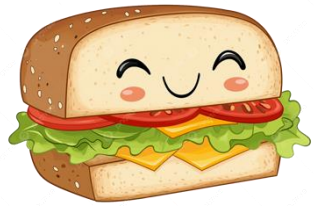
Threat model and sandwich semantics

Untrusted variables can be **mauled**

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
...	...	...

$$v_p := v_s * v_u$$





Threat model and sandwich semantics

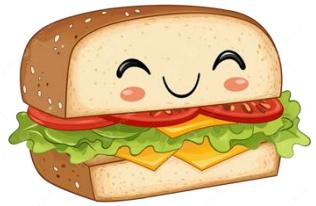
Untrusted variables can be **mauled**

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…

$v_p\ 0\ \ v_s\ 42\ \ v_u\ 0$

$v_p := v_s * v_u$



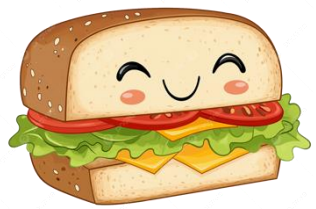


Threat model and sandwich semantics

Untrusted variables can be **mauled**

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…





v_p 0 v_s 42 v_u 0



$v_p := v_s * v_u$

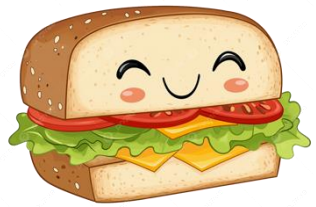


Threat model and sandwich semantics

Untrusted variables can be **mauled**

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…







Writes to untrusted

$v_p \ 0 \ v_s \ 42 \ v_u \ 0$

$v_p := v_s * v_u$

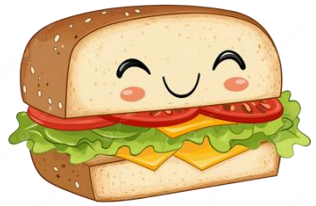


Threat model and sandwich semantics

Untrusted variables can be **mauled**

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…





$v_p\ 0\ \ v_s\ 42\ \ v_u\ 0$



Writes to untrusted

$v_p := v_s * v_u$



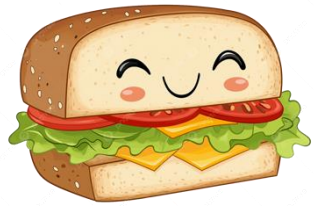
Reads from public

Threat model and sandwich semantics

Untrusted variables can be **mauled**

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
...	...	...







Writes to untrusted

v_p 0 v_s 42 v_u 0

$v_p := v_s * v_u$



Reads from public

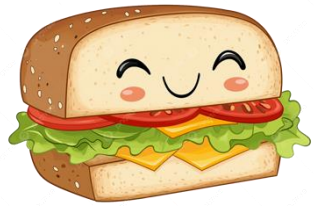
v_p 0 v_s 42 v_u 0 ☠ corrupts v_u to 1 v_p 0 v_s 42 v_u 1

Threat model and sandwich semantics

Untrusted variables can be **mauled**

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
...	...	...







Writes to untrusted

$v_p\ 0\ v_s\ 42\ v_u\ 0$

$v_p := v_s * v_u$



Reads from public

$v_p\ 0\ v_s\ 42\ v_u\ 0$ → ☠ corrupts v_u to 1 → $v_p\ 0\ v_s\ 42\ v_u\ 1$

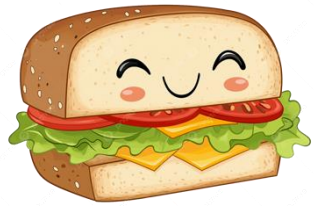
$v_p\ 0\ v_s\ 42\ v_u\ 1$ → $v_p := v_s * v_u$ runs → $v_p\ 42\ v_s\ 42\ v_u\ 1$

Threat model and sandwich semantics

Untrusted variables can be **mauled**

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…







Writes to untrusted

v_p 0 v_s 42 v_u 0

$v_p := v_s * v_u$



Reads from public

v_p 0 v_s 42 v_u 0 → ☠ corrupts v_u to 1 → v_p 0 v_s 42 v_u 1

v_p 0 v_s 42 v_u 1 → $v_p := v_s * v_u$ runs → v_p 42 v_s 42 v_u 1

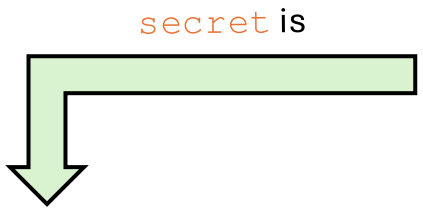
v_p 42 v_s 42 v_u 1 → ☠ learns public variables → “v_p is 42”

Not so easy: control flow

Not so easy: control flow

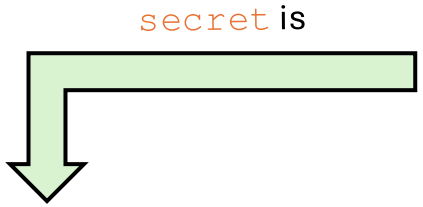
```
 if secret then
  then skip;
skip; skip
  else skip
public := 1
```

Not so easy: control flow




```
 if secret then
  then skip;
skip; skip
  else skip
public := 1
```

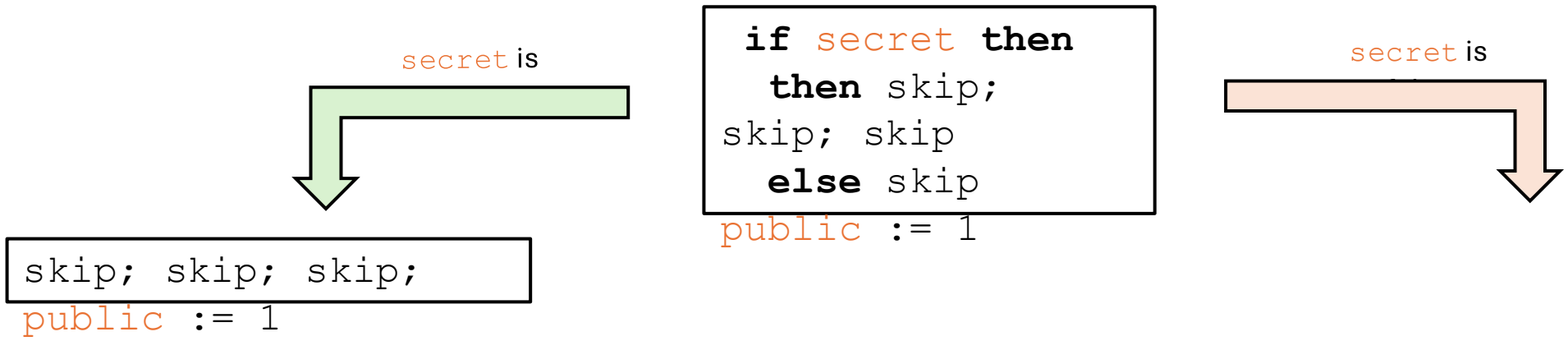
Not so easy: control flow



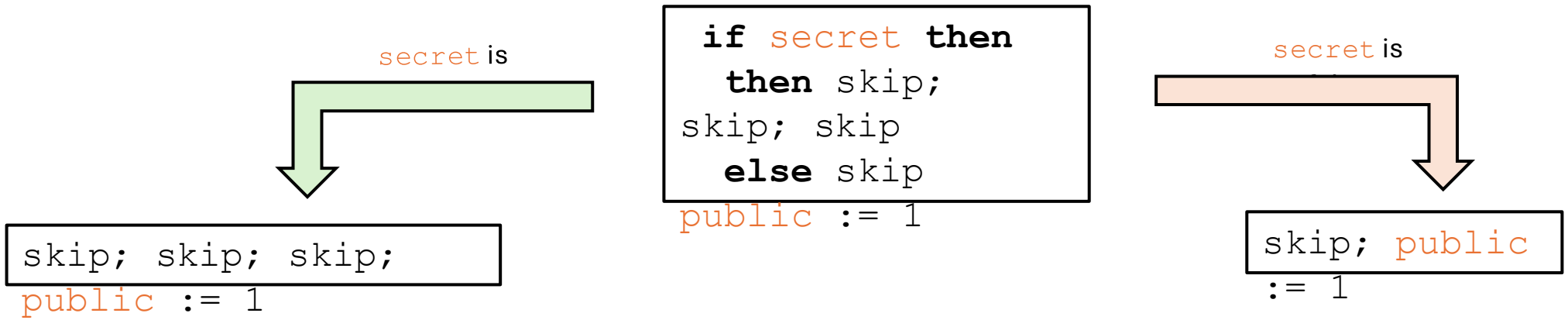

```
skip; skip; skip;
public := 1
```

```
 if secret then
  then skip;
skip; skip
  else skip
public := 1
```

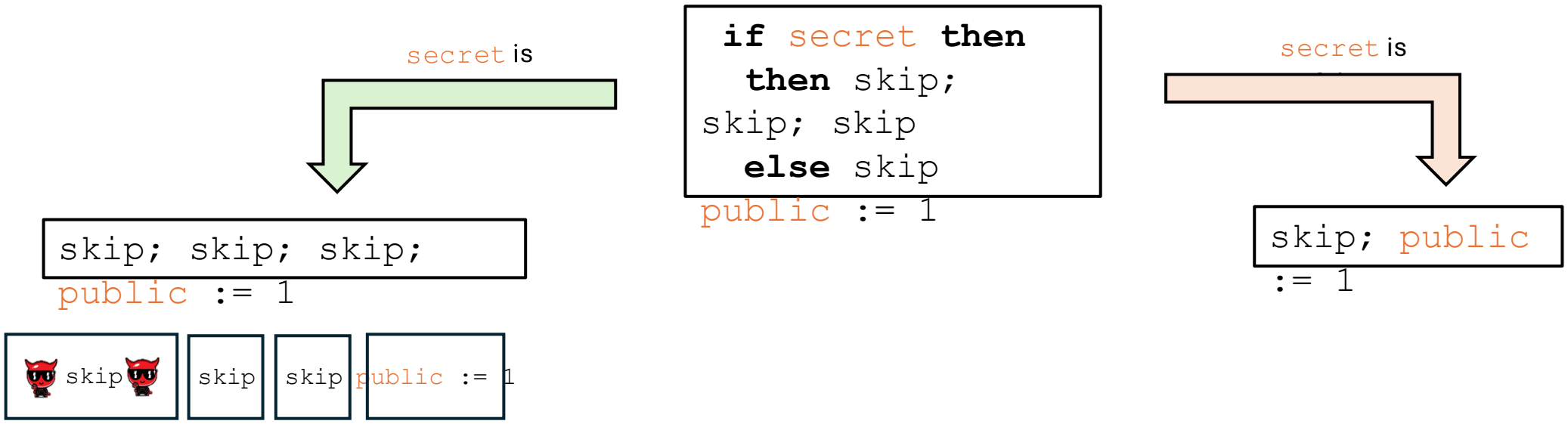
Not so easy: control flow



Not so easy: control flow



Not so easy: control flow



Not so easy: control flow

```
if secret then
  then skip;
skip; skip
  else skip
public := 1
```

secret is

```
skip; skip; skip;
public := 1
```



secret is

```
skip; public
:= 1
```

Not so easy: control flow

```
if secret then
  then skip;
skip; skip
  else skip
public := 1
```

secret is

```
skip; skip; skip;
public := 1
```



secret is

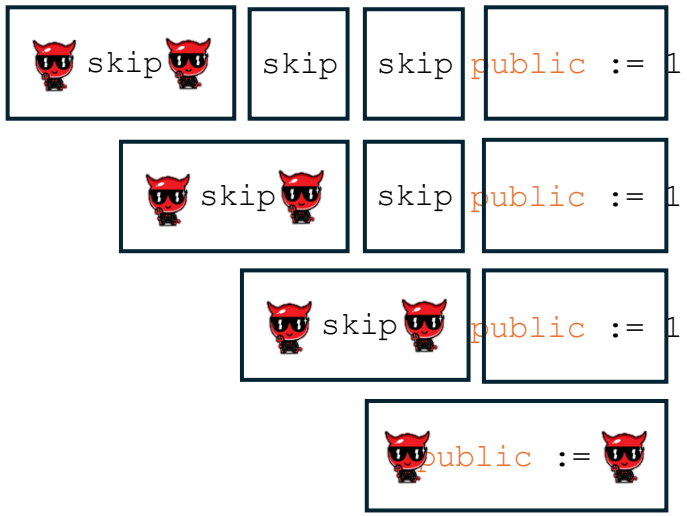
```
skip; public
:= 1
```

Not so easy: control flow

```
if secret then
  then skip;
skip; skip
  else skip
public := 1
```

secret is

```
skip; skip; skip;
public := 1
```



secret is

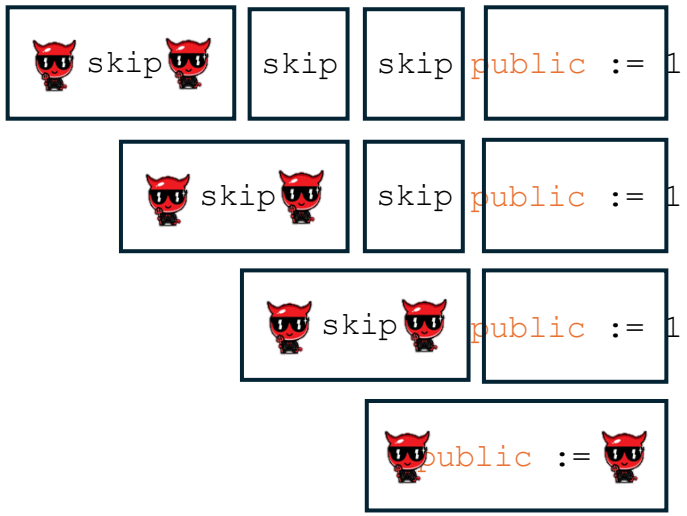
```
skip; public
:= 1
```

Not so easy: control flow

```
if secret then
  then skip;
skip; skip
  else skip
public := 1
```

secret is

```
skip; skip; skip;
public := 1
```



secret is

```
skip; public
:= 1
```

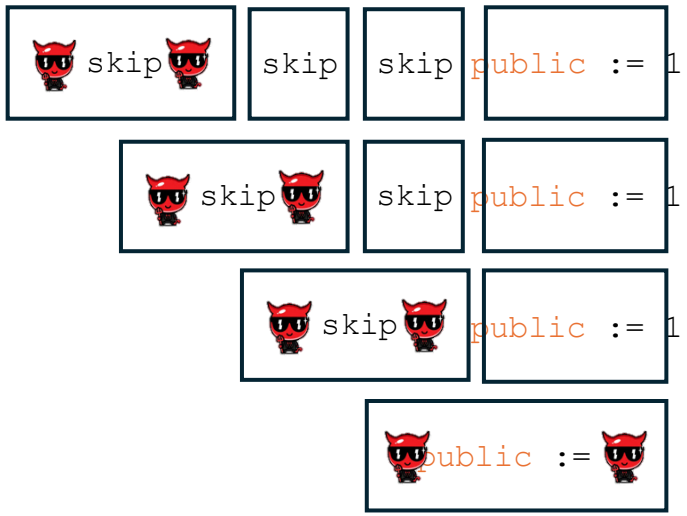


Not so easy: control flow

```
if secret then
  then skip;
skip; skip
  else skip
public := 1
```

secret is

```
skip; skip; skip;
public := 1
```



secret is

```
skip; public
:= 1
```

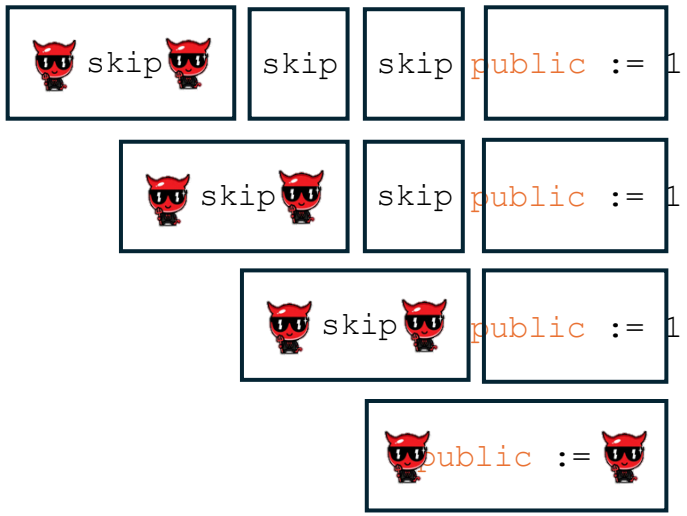


Not so easy: control flow

```
if secret then
  then skip;
skip; skip
  else skip
public := 1
```

secret is

```
skip; skip; skip;
public := 1
```



secret is

```
skip; public
:= 1
```



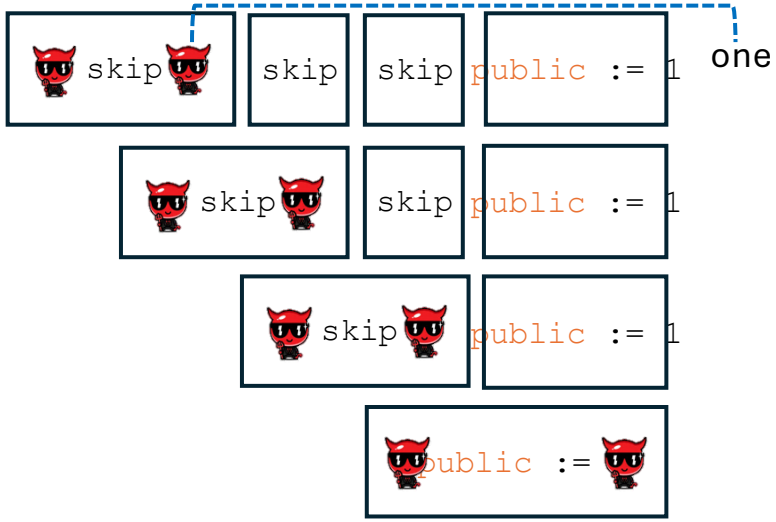


Not so easy: control flow

```
if secret then
  then skip;
skip; skip
  else skip
public := 1
```

secret is

```
skip; skip; skip;
public := 1
```



secret is

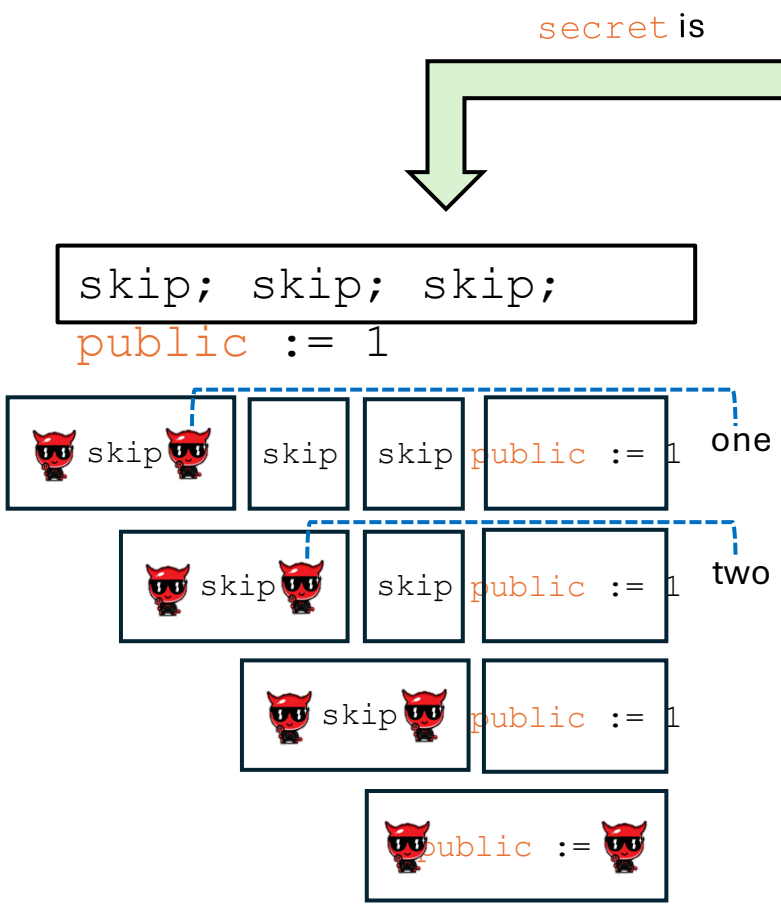
```
skip; public
:= 1
```

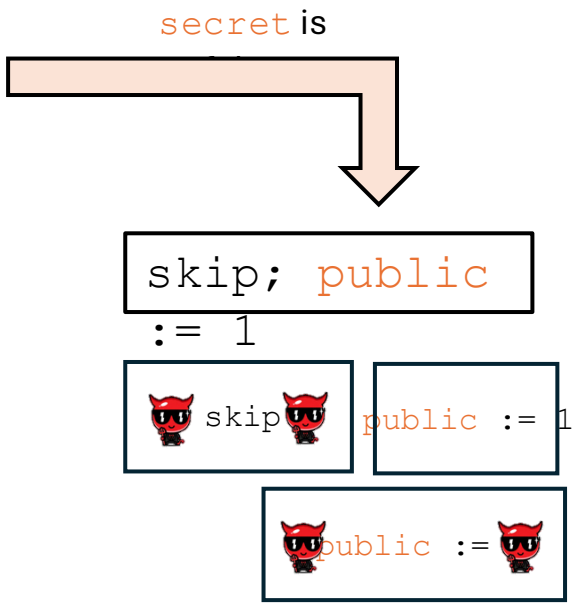




Not so easy: control flow

```
if secret then
  then skip;
skip; skip
  else skip
public := 1
```

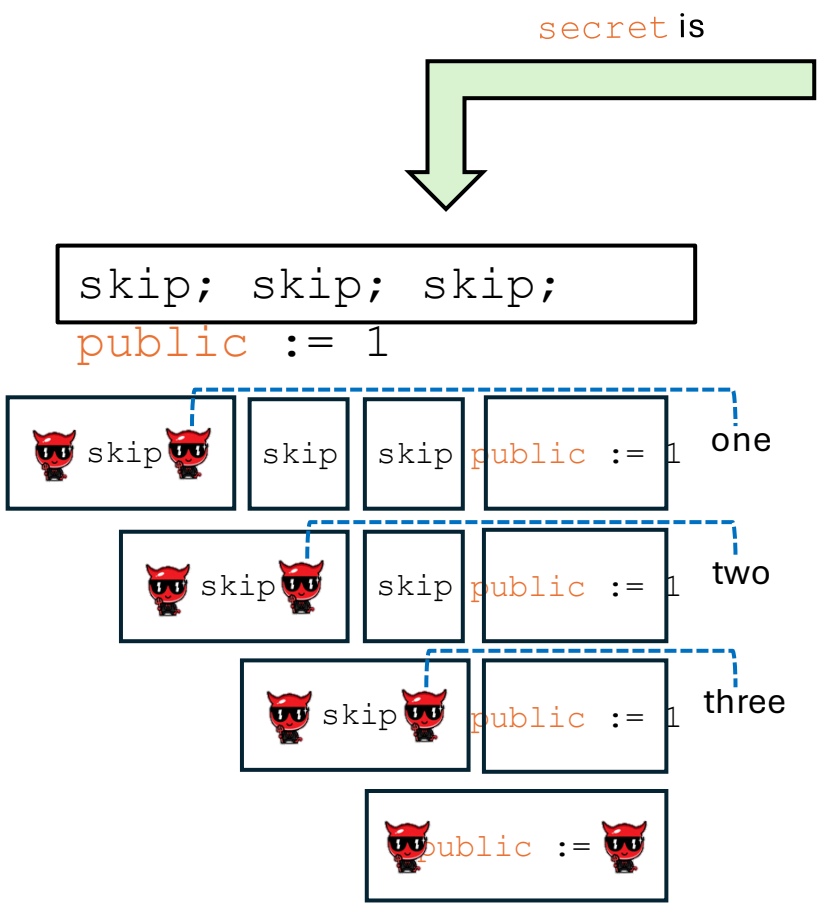


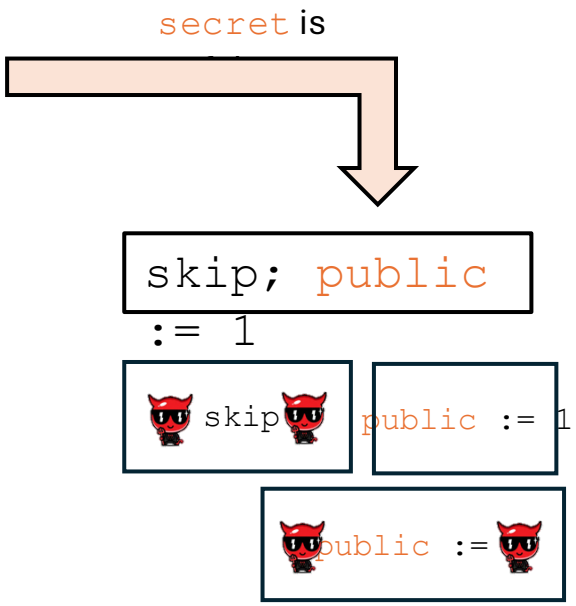




Not so easy: control flow

```
if secret then
  then skip;
skip; skip
  else skip
public := 1
```

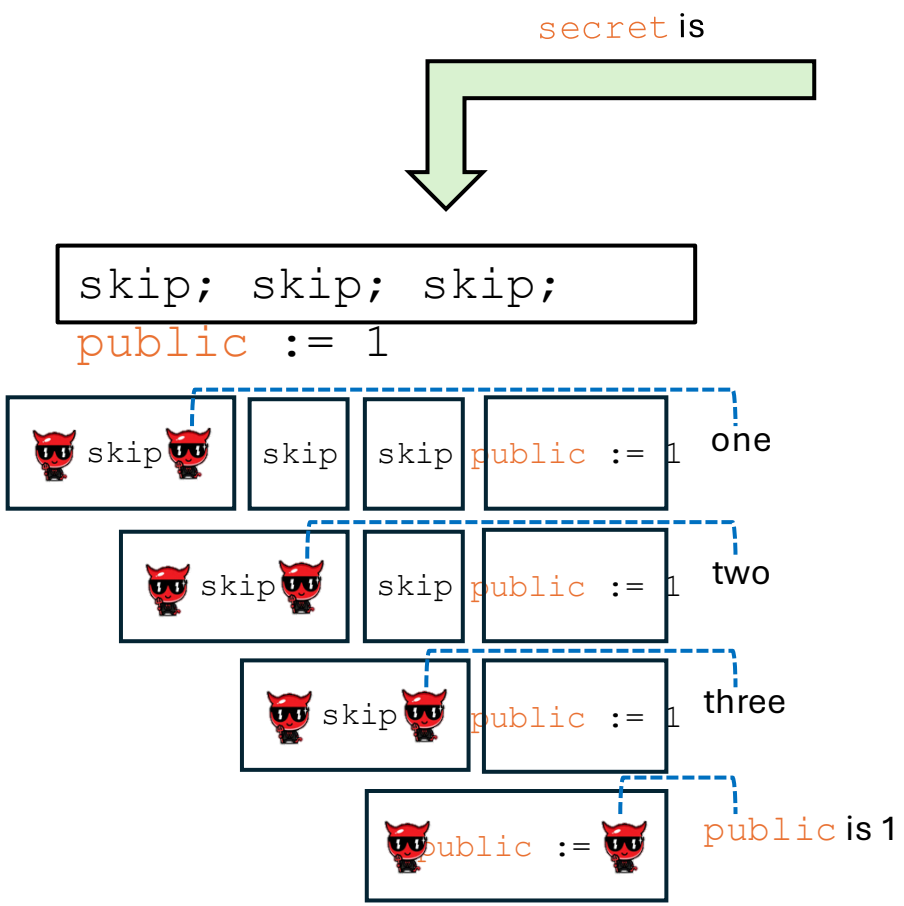


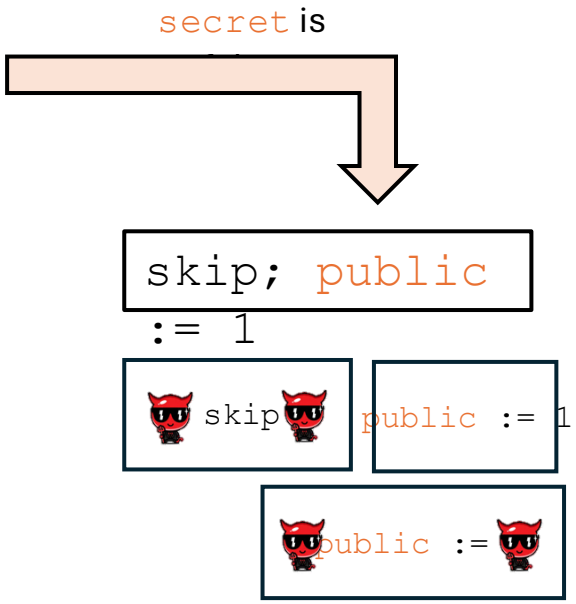




Not so easy: control flow

```
if secret then
  then skip;
skip; skip
  else skip
public := 1
```

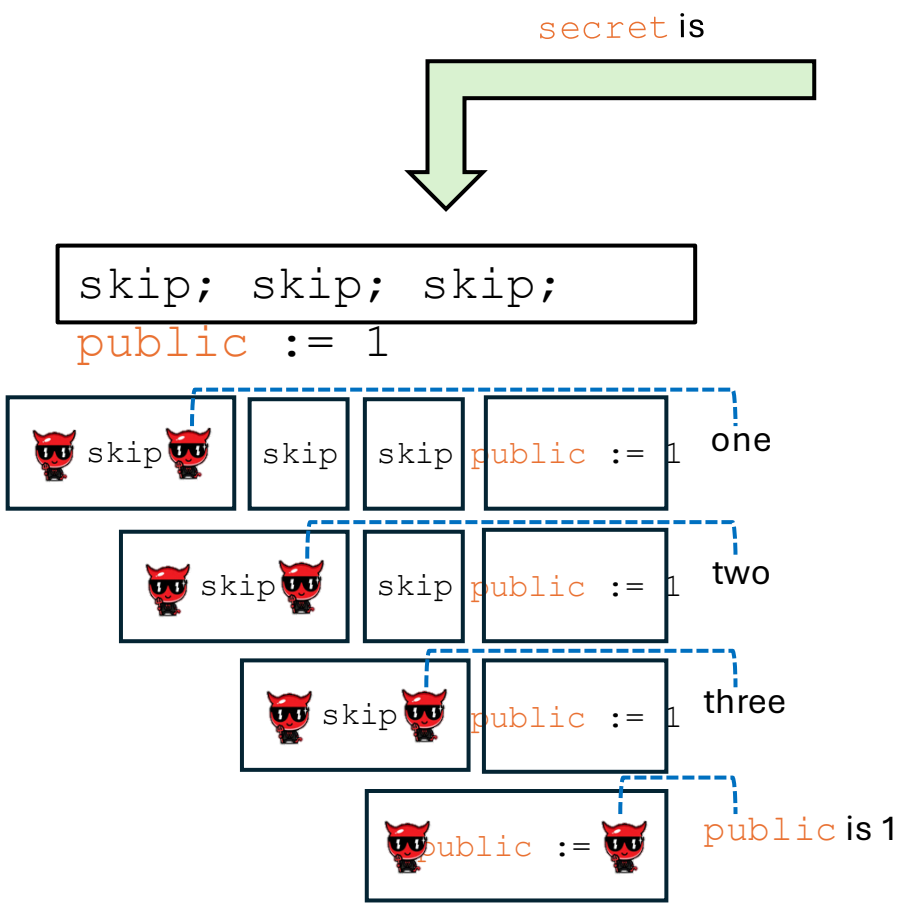


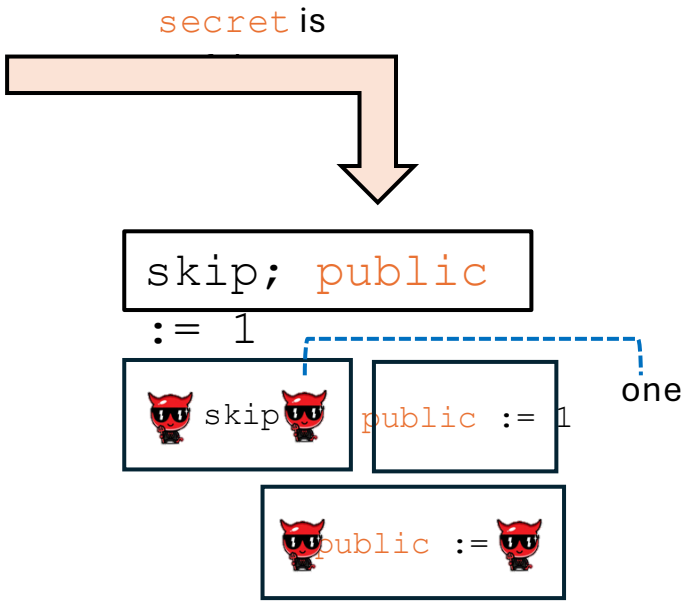




Not so easy: control flow

```
if secret then
  then skip;
skip; skip
  else skip
public := 1
```

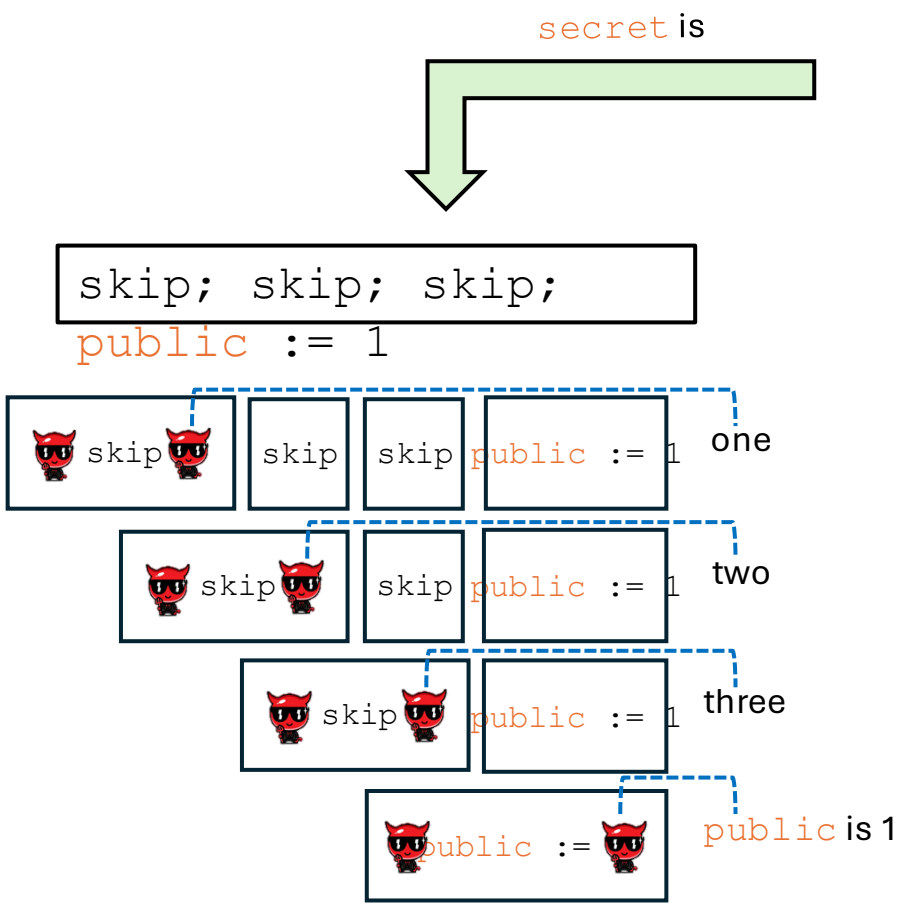




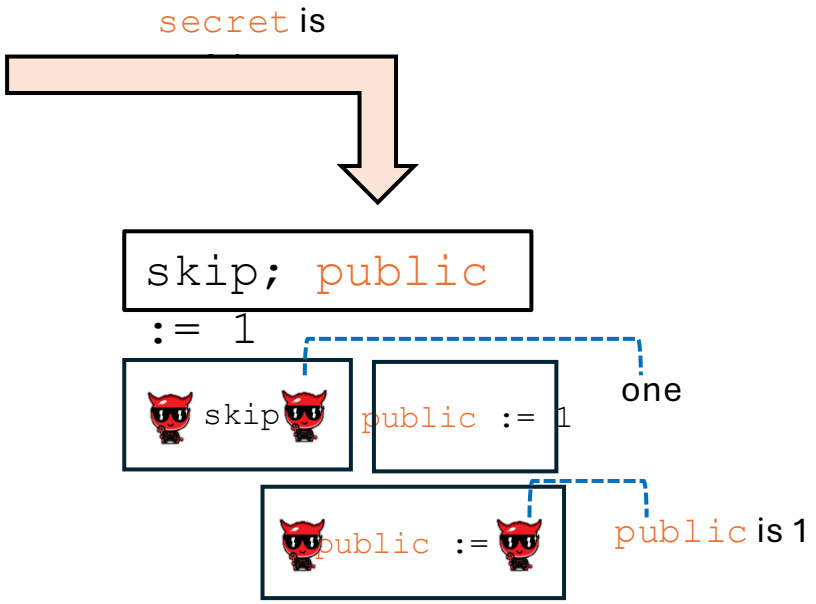


Not so easy: control flow

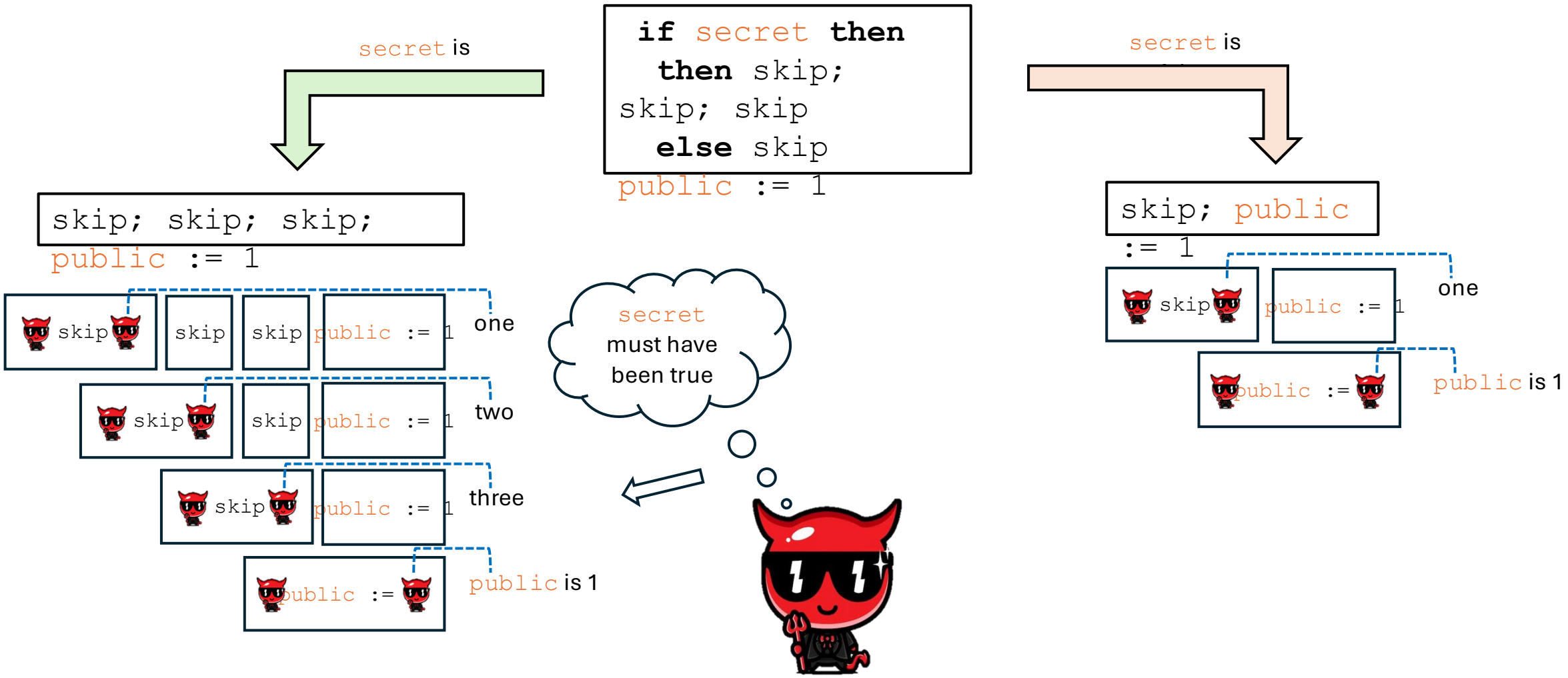
```
if secret then
  then skip;
skip; skip
  else skip
public := 1
```



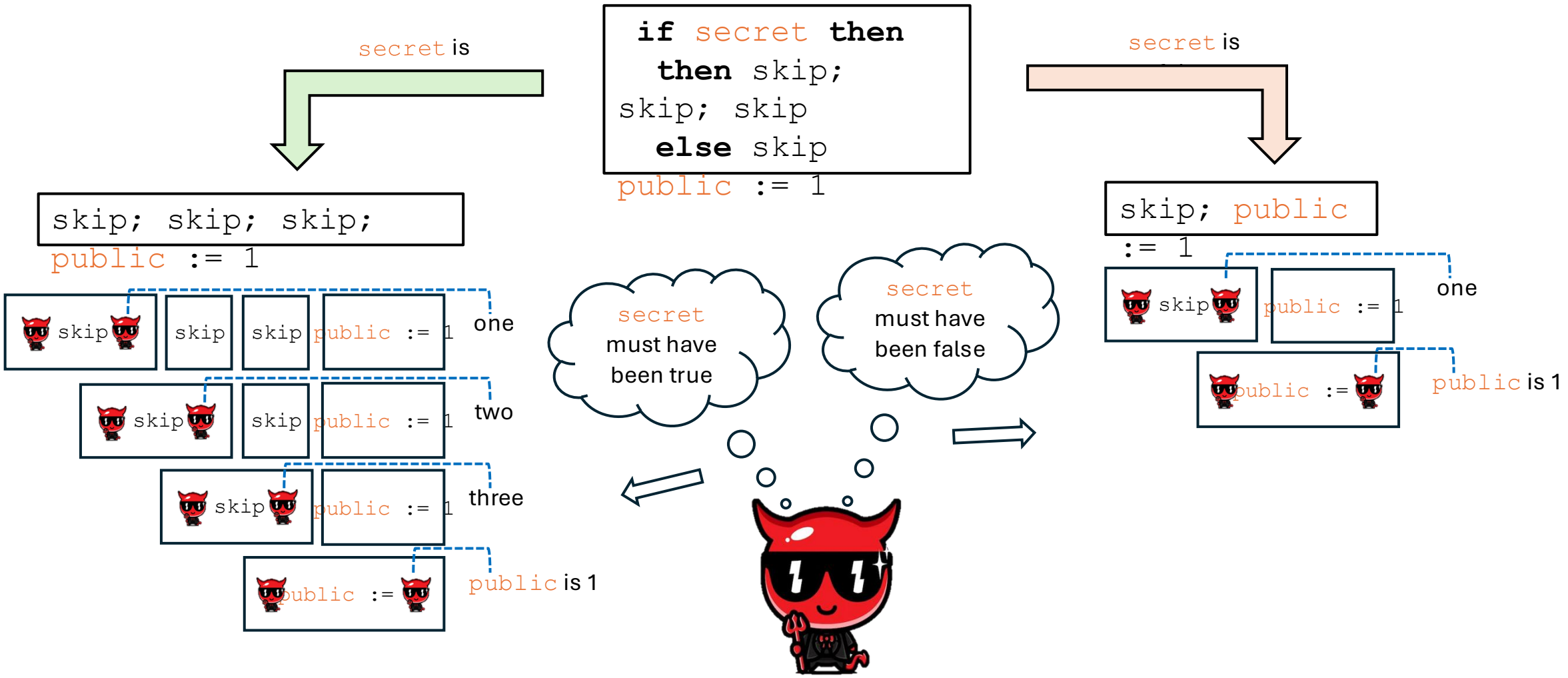




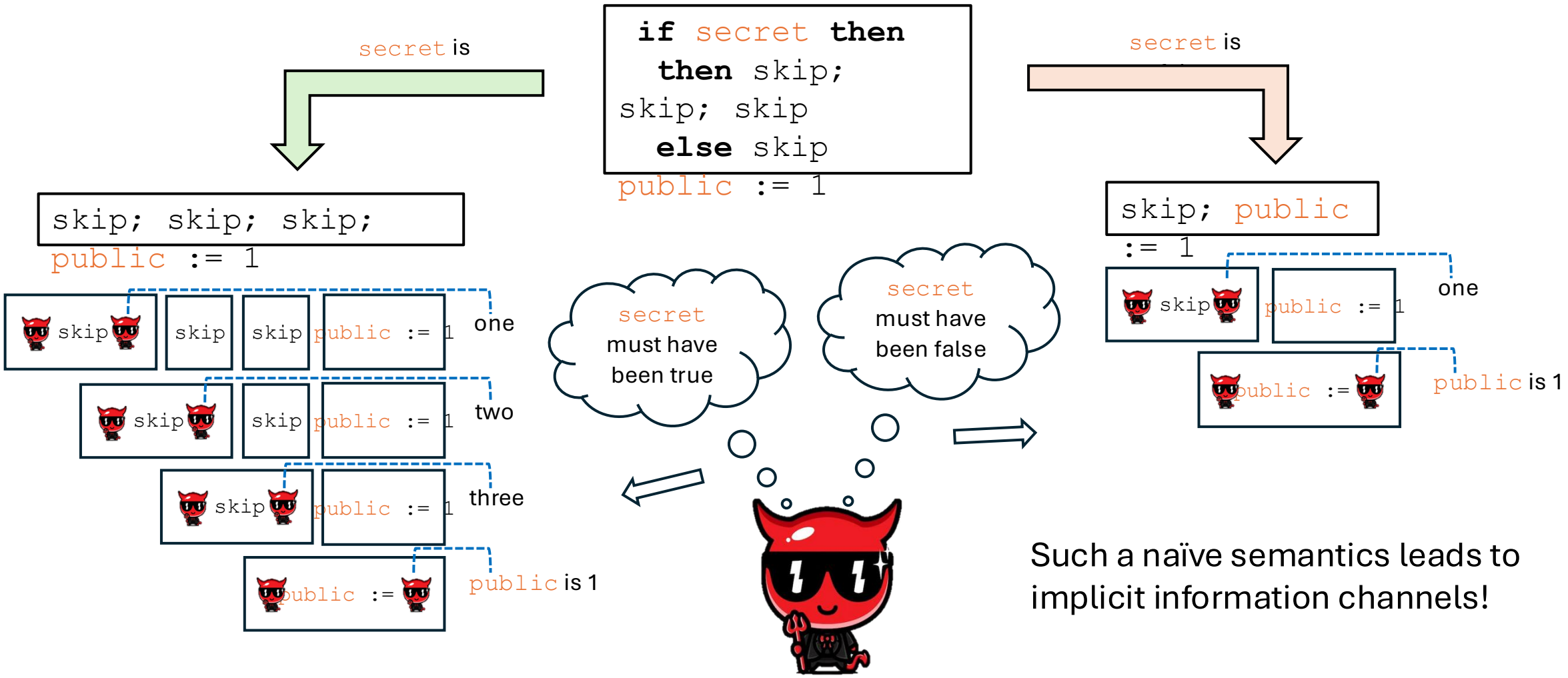
Not so easy: control flow



Not so easy: control flow

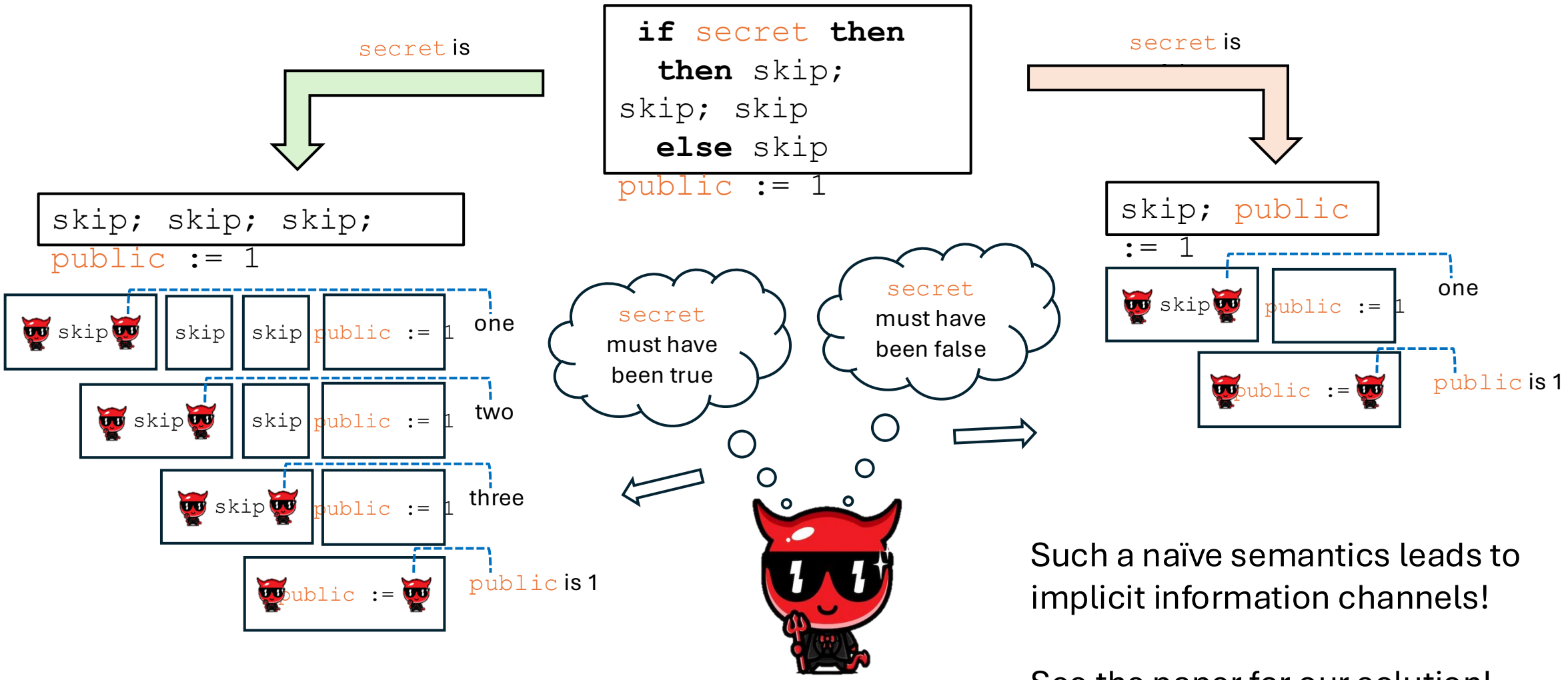


Not so easy: control flow



Such a naïve semantics leads to implicit information channels!

Not so easy: control flow



Such a naïve semantics leads to implicit information channels!

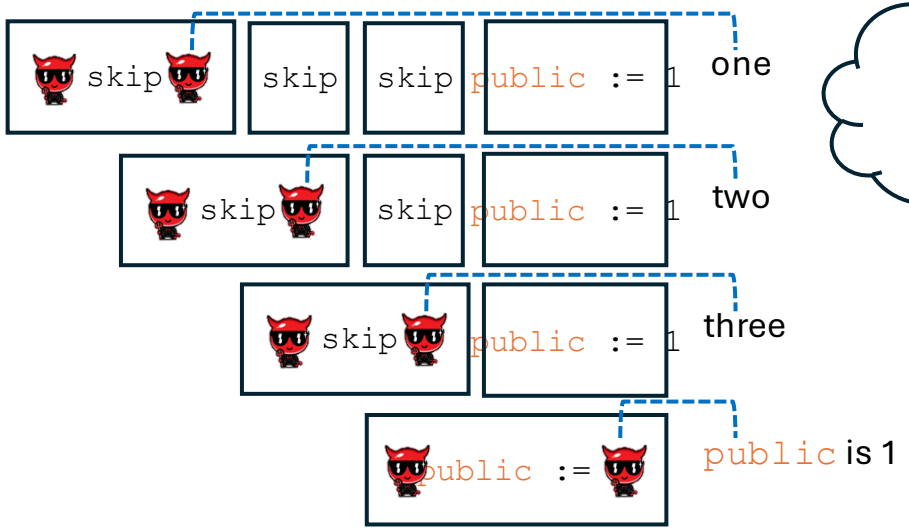
See the paper for our solution!

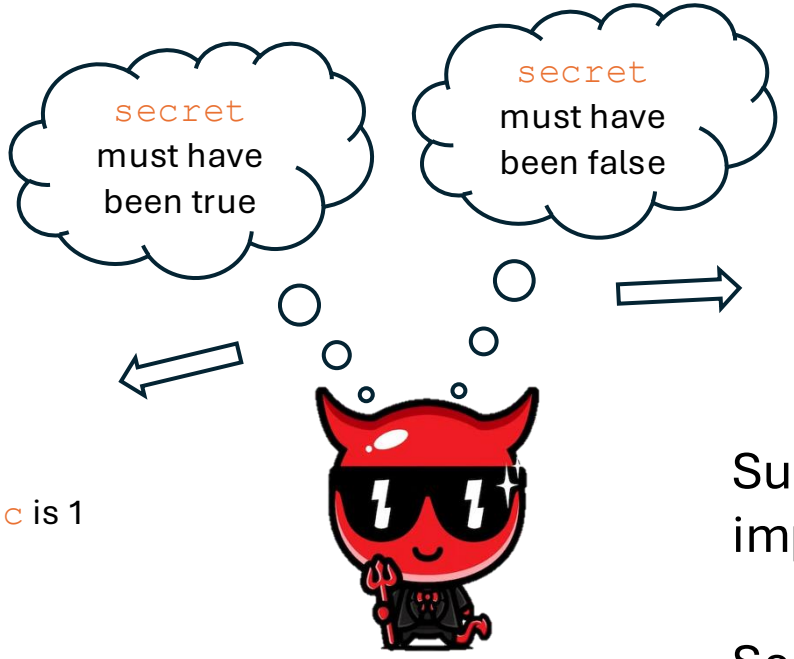
Not so easy: control flow

```
if secret then
  then skip;
skip; skip
  else skip
public := 1
```

secret is

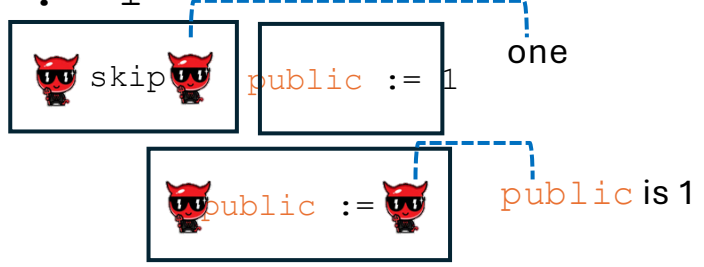
```
skip; skip; skip;
public := 1
```





secret is

```
skip; public
:= 1
```



Such a naïve semantics leads to implicit information channels!

See the paper for our solution!



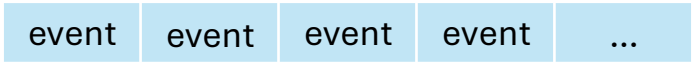
Defining behavior ⟦ ⟧

Defining behavior ⟦ ⟧

What are the traces?

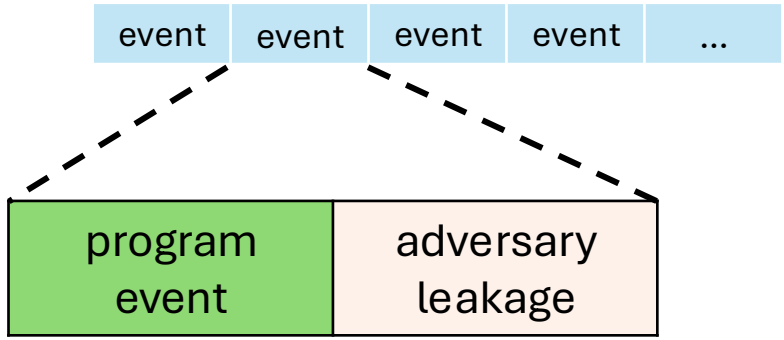
Defining behavior ⟦ ⟧

What are the traces?



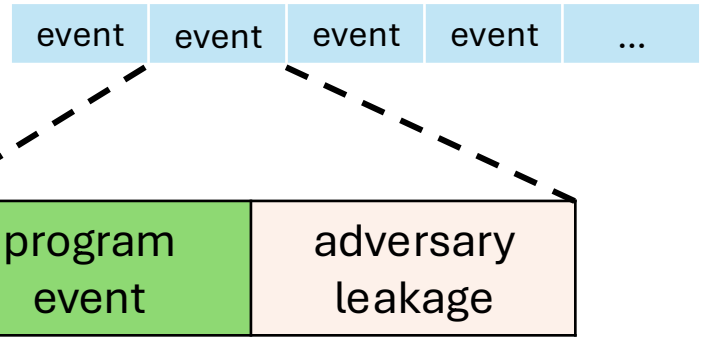
Defining behavior ⟦ ⟧

What are the traces?



Defining behavior ⟦ ⟧

What are the traces?

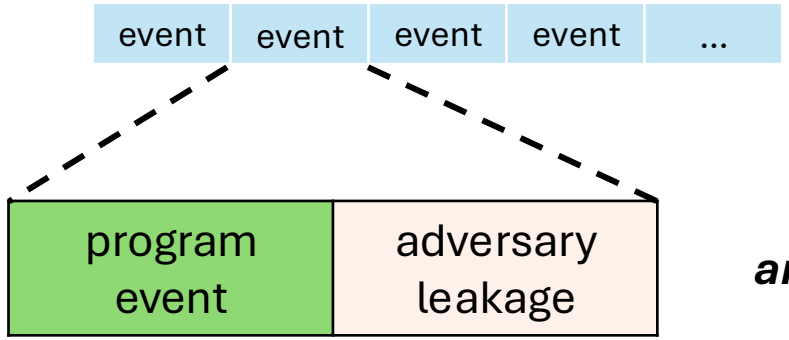


Command	c	::=	$x := e_v$ | if e_b then p else p
		|	while e_b do p | output x

Defining behavior ⟦ ⟧

What are the traces?

Command	c	::=	$x := e_v$ | `if` e_b `then` p `else` p
		|	`while` e_b `do` p | `output` x



anything from 

Defining behavior ⟦ ⟧

What are the traces?

event	event	event	event	...

program event	adversary leakage

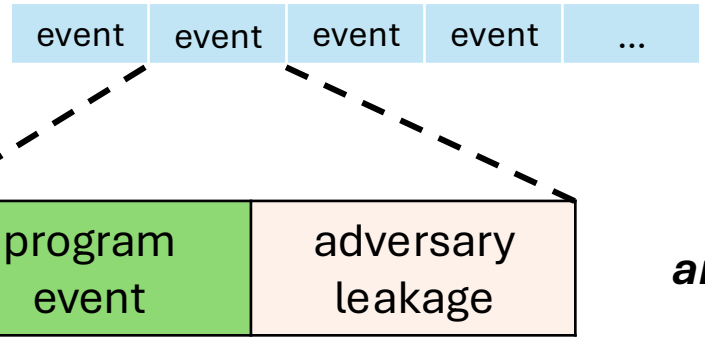
anything from 

Command c	::=	$x := e_v$ | if e_b then p else p
	|	while e_b do p | output x

```
pub :=
42;
output
pub
```

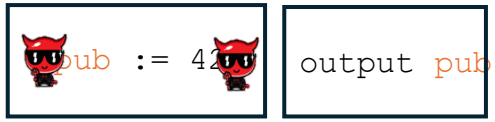
Defining behavior ⟦ ⟧

What are the traces?



Command	c	::=	$x := e_v$ | if e_b then p else p | while e_b do p | output x

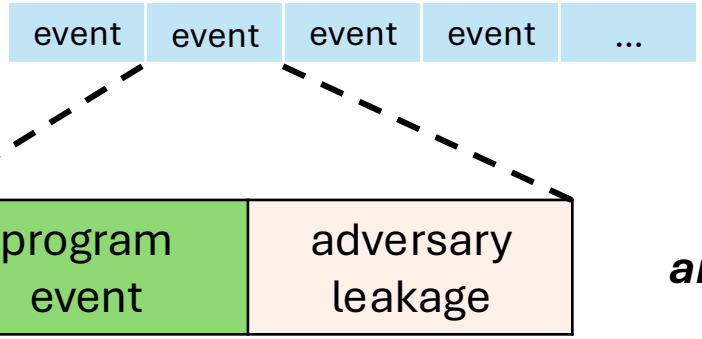
anything from 




```
pub :=
42;
output
pub
```

Defining behavior ⟦ ⟧

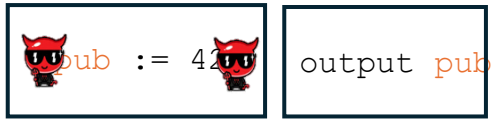
What are the traces?

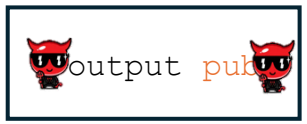


Command	c	::=	$x := e_v$ | if e_b then p else p | while e_b do p | output x

anything from 

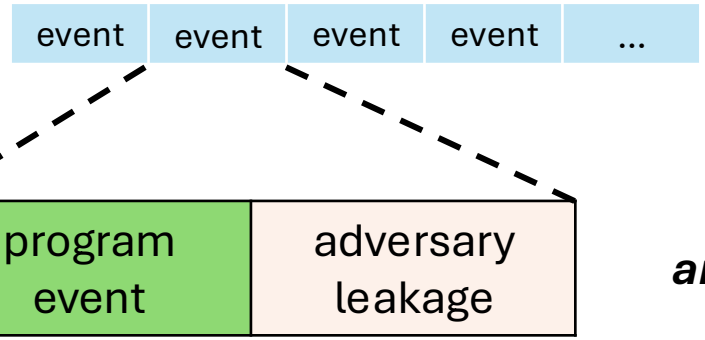
```
pub := 42;
output pub
```





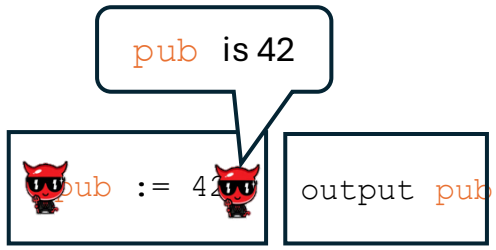
Defining behavior ⟦ ⟧

What are the traces?

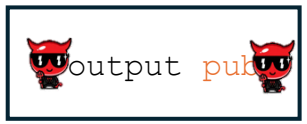


anything from 

Command c ::= $x := e_v$ | if e_b then p else p | while e_b do p | output x

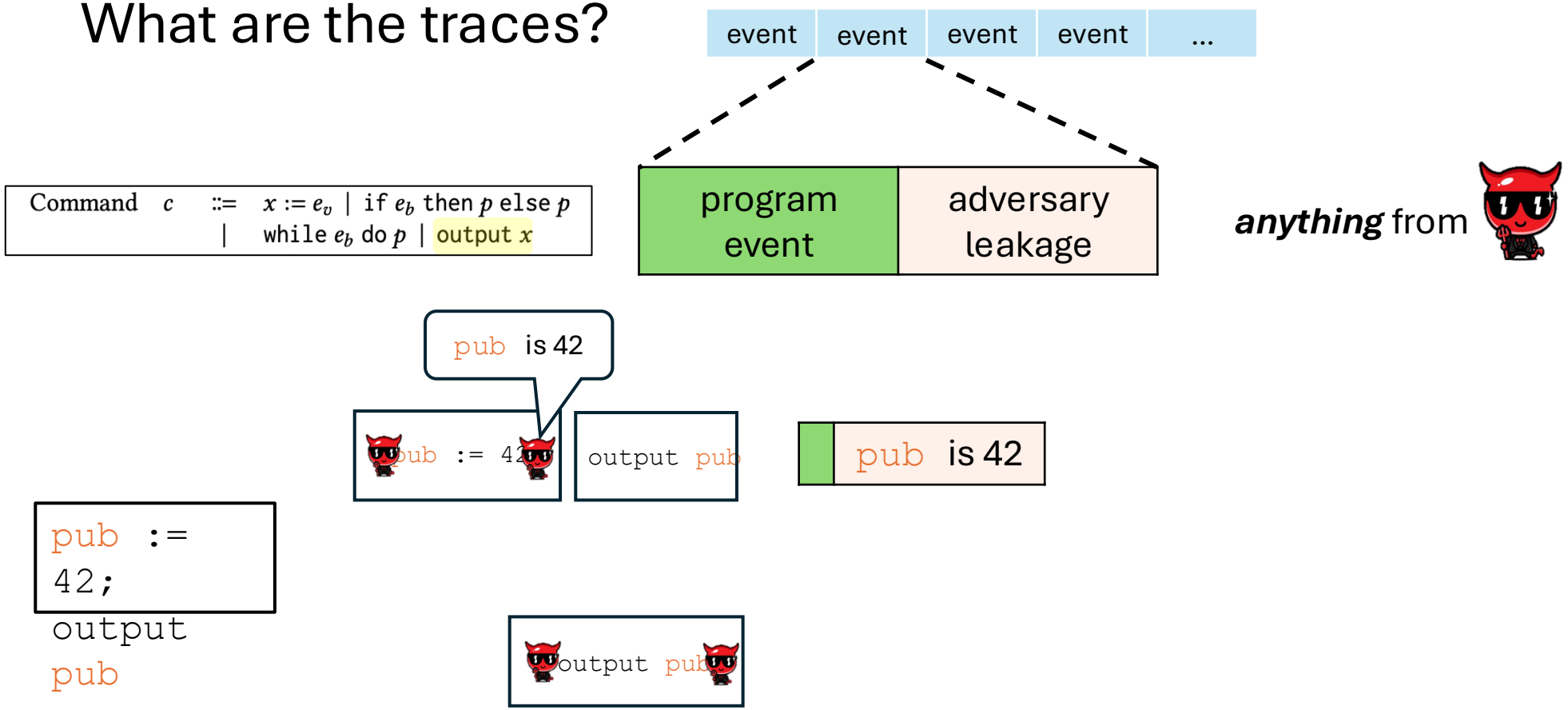



```
pub :=
42;
output
pub
```



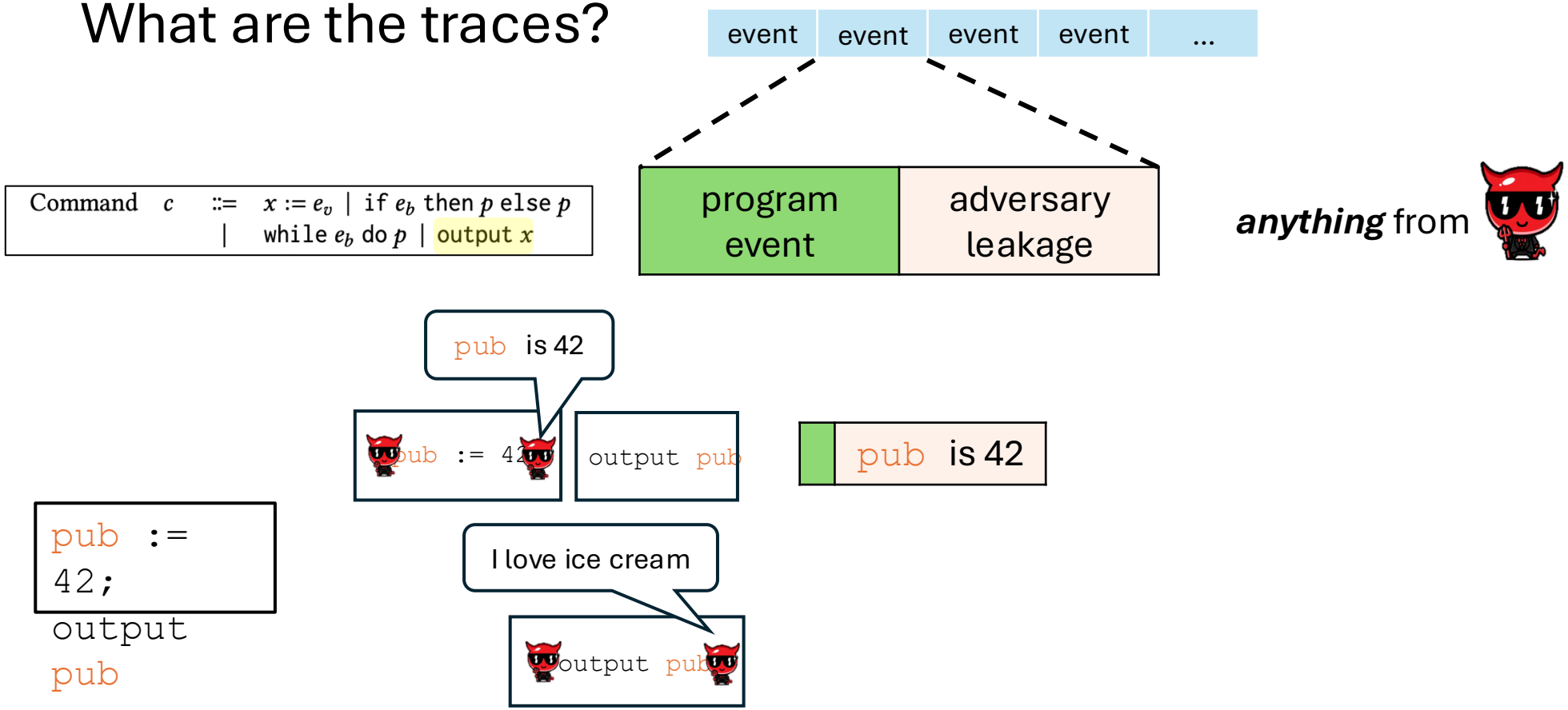
Defining behavior ⟦ ⟧

What are the traces?



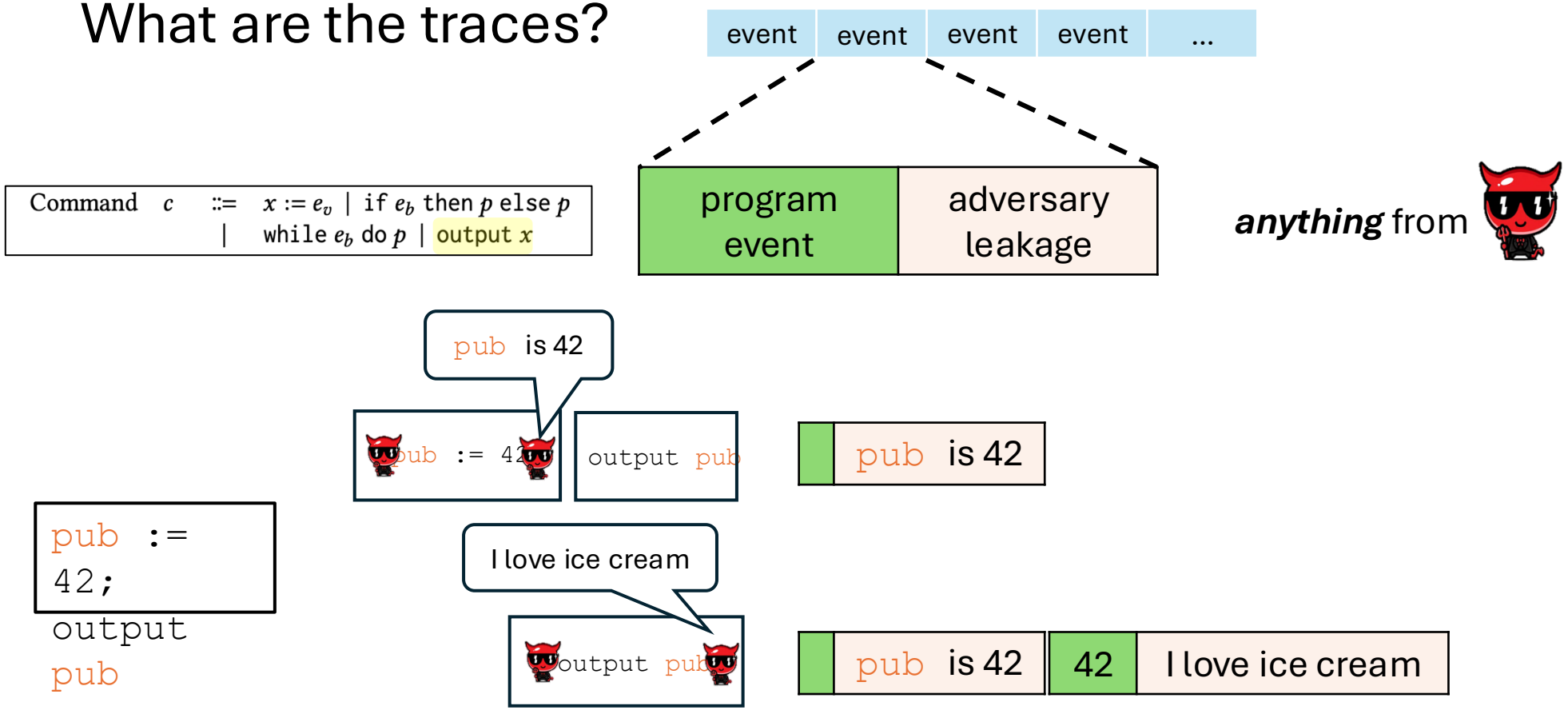
Defining behavior ⟦ ⟧

What are the traces?



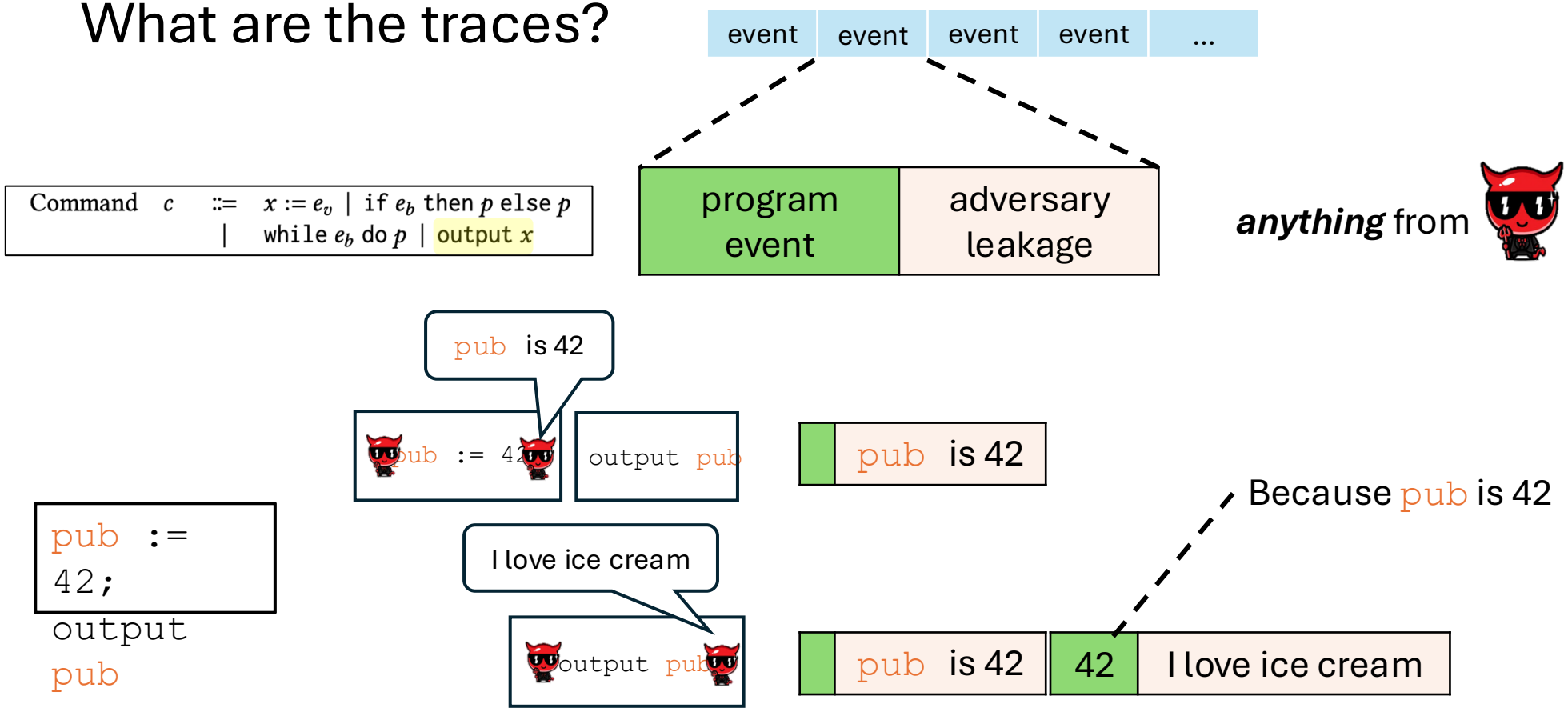
Defining behavior ⟦ ⟧

What are the traces?



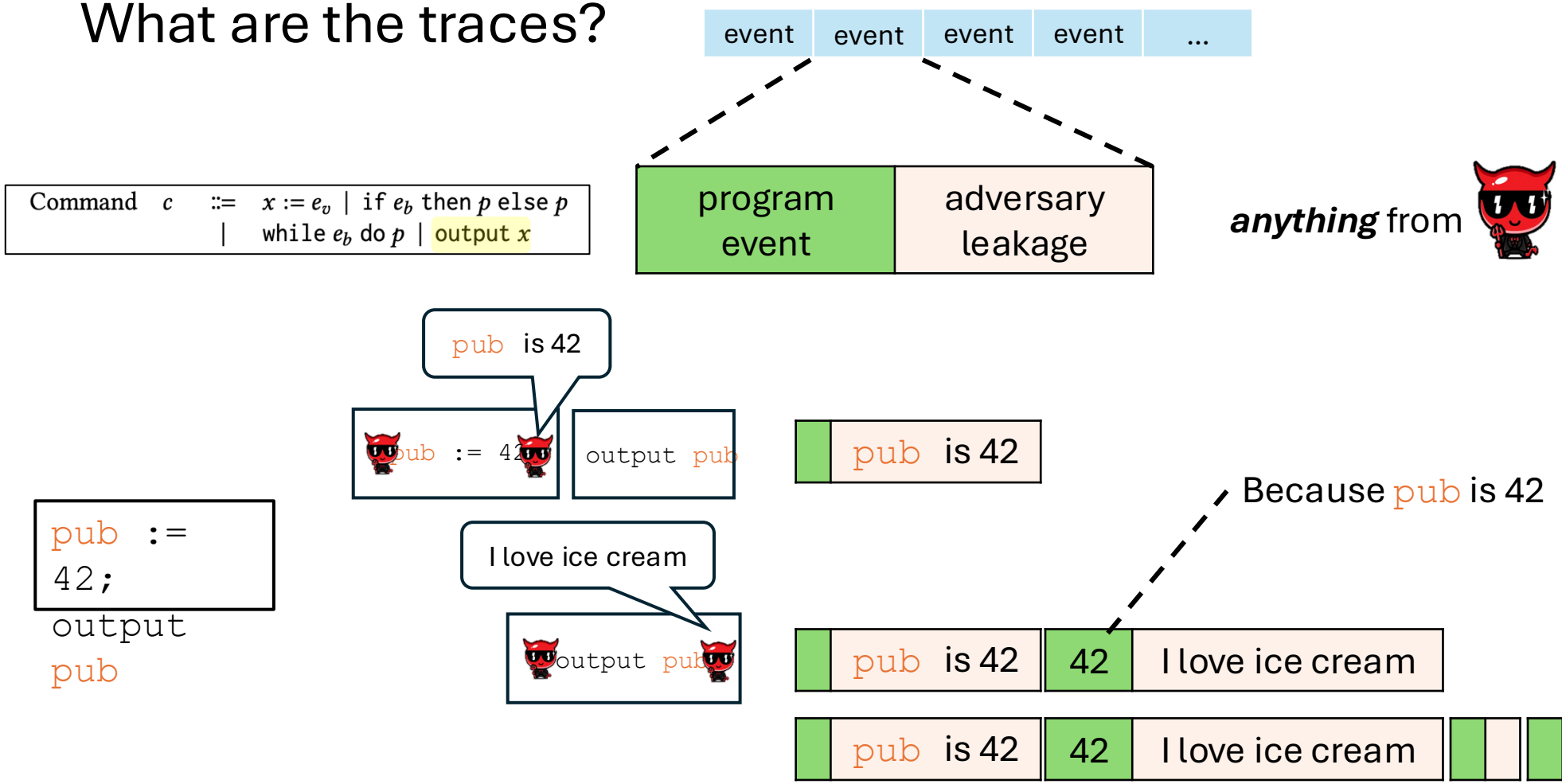
Defining behavior ⟦ ⟧

What are the traces?



Defining behavior ⟦ ⟧

What are the traces?



(Unary) real–ideal emulation in ImpThreat

(Unary) real–ideal emulation in ImpThreat

“Every real attack is equivalent to some ideal attack”

(Unary) real–ideal emulation in ImpThreat

“Every real attack is equivalent to some ideal attack”

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

(Unary) real–ideal emulation in ImpThreat

“Every real attack is equivalent to some ideal attack”

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

(Unary) real–ideal emulation in ImpThreat

“Every real attack is equivalent to some ideal attack”

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

(Unary) real–ideal emulation in ImpThreat

“Every real attack is equivalent to some ideal attack”

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

(Unary) real–ideal emulation in ImpThreat

“Every real attack is equivalent to some ideal attack”

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

$$\forall \sigma_{inp}.[\![p \mid A_r]\!](\sigma_{inp}) \approx^{t} [\![p \mid A_i]\!](\sigma_{inp})$$

(Unary) real–ideal emulation in ImpThreat

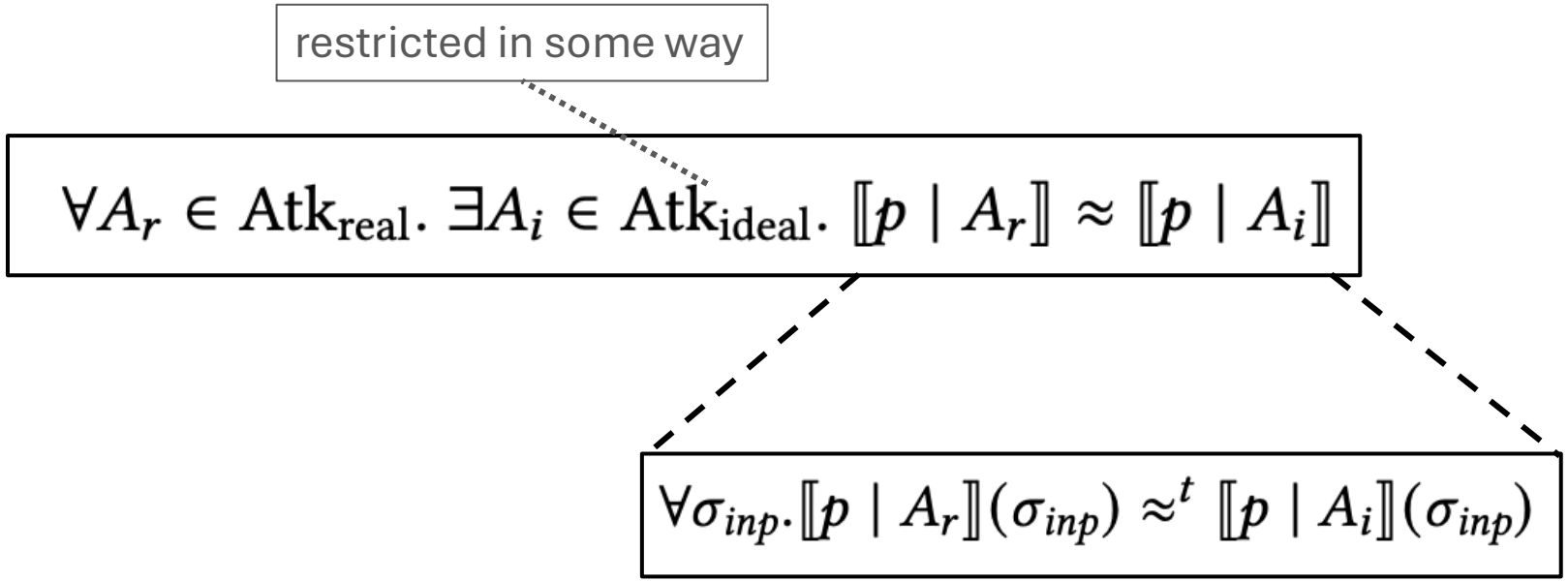
“Every real attack is equivalent to some ideal attack”

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

$$\forall \sigma_{inp}.[\![p \mid A_r]\!](\sigma_{inp}) \approx^{t} [\![p \mid A_i]\!](\sigma_{inp})$$

(Unary) real–ideal emulation in ImpThreat

“Every real attack is equivalent to some ideal attack”



Revisiting nonmalleable information flow

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Revisiting nonmalleable information flow

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{nmif}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Revisiting nonmalleable information flow

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

these attackers cannot **maul.**

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{nmif}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Revisiting nonmalleable information flow

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

these attackers cannot **maul.**

Nonmalleable = cannot maul!

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{nmif}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Revisiting nonmalleable information flow

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

these attackers cannot **maul.**

Nonmalleable = cannot maul!

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{nmif}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)

Revisiting nonmalleable information flow

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

these attackers cannot **maul.**

Nonmalleable = cannot maul!

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{nmif}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)

Intensional 4-hyperproperty formulation

Revisiting nonmalleable information flow

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

these attackers cannot **maul.**

Nonmalleable = cannot maul!

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{nmif}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)

Intensional 4-hyperproperty formulation

Has a complex exclusion of 'inept' attackers

Revisiting nonmalleable information flow

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

these attackers cannot **maul.**

Nonmalleable = cannot maul!

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{nmif}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)

Intensional 4-hyperproperty formulation

Has a complex exclusion of 'inept' attackers



Revisiting nonmalleable information flow

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

these attackers cannot **maul.**

Nonmalleable = cannot maul!

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{nmif}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$



Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)

Intensional 4-hyperproperty formulation

Has a complex exclusion of 'inept' attackers

Revisiting nonmalleable information flow

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

these attackers cannot **maul.**

Nonmalleable = cannot maul!

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{nmif}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Let's pretend that I just didn't see that



Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)

Intensional 4-hyperproperty formulation

Has a complex exclusion of 'inept' attackers

This work

Revisiting nonmalleable information flow

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

these attackers cannot **maul.**

Nonmalleable = cannot maul!

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{nmif}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Let's pretend that I just didn't see that



Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)

Intensional 4-hyperproperty formulation

Has a complex exclusion of 'inept' attackers

This work

Simple and extensional

Revisiting nonmalleable information flow

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

these attackers cannot **maul.**

Nonmalleable = cannot maul!

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{nmif}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$



Nonmalleable information flow control
Ethan Cecchetti, Andrew C. Myers, Owen Arden (CCS '17)

Intensional 4-hyperproperty formulation

Has a complex exclusion of 'inept' attackers

This work

Simple and extensional

No exclusion of inept attackers

One paradigm, many properties

Nonmalleable information flow

One paradigm, many properties

Nonmalleable information flow

Transparent endorsement

Robust declassification

Noninterference

One paradigm, many properties

Nonmalleable information flow

Transparent endorsement

Robust declassification

Very strict ---------- Noninterference

One paradigm, many properties

Integrity and confidentiality relaxed

Nonmalleable information flow

Transparent endorsement

Robust declassification

Very strict

Noninterference

One paradigm, many properties

Integrity and confidentiality relaxed ----- Nonmalleable information flow

Transparent endorsement

Confidentiality relaxed ----- Robust declassification

Very strict ---------- Noninterference

One paradigm, many properties

Integrity and confidentiality relaxed ----- Nonmalleable information flow

Integrity relaxed ---------- Transparent endorsement

Confidentiality relaxed ----- Robust declassification

Very strict ---------- Noninterference

One paradigm, many properties

Integrity and confidentiality relaxed ----- Nonmalleable information flow

Integrity relaxed ---------- Transparent endorsement

Confidentiality relaxed ----- Robust declassification

Very strict ---------- Noninterference

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

All properties are the same statement!

One paradigm, many properties

Integrity and confidentiality relaxed	-----	Nonmalleable information flow	$\mathrm{Atk}_{\mathrm{nmif}}$
Integrity relaxed	----------	Transparent endorsement	$\mathrm{Atk}_{\mathrm{te}}$
Confidentiality relaxed	-----	Robust declassification	$\mathrm{Atk}_{\mathrm{rd}}$
Very strict	----------	Noninterference	$\mathrm{Atk}_{\mathrm{ni}}$

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

All properties are the same statement!

One paradigm, many properties

Integrity and confidentiality relaxed	Nonmalleable information flow	$\mathrm{Atk}_{\mathrm{nmif}}$	nonmauling
Integrity relaxed	Transparent endorsement	$\mathrm{Atk}_{\mathrm{te}}$	
Confidentiality relaxed	Robust declassification	$\mathrm{Atk}_{\mathrm{rd}}$	
Very strict	Noninterference	$\mathrm{Atk}_{\mathrm{ni}}$	

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

All properties are the same statement!

One paradigm, many properties

Integrity and confidentiality relaxed	Nonmalleable information flow	$\mathrm{Atk}_{\mathrm{nmif}}$	nonmauling
Integrity relaxed	Transparent endorsement	$\mathrm{Atk}_{\mathrm{te}}$	nonmauling & incurious
Confidentiality relaxed	Robust declassification	$\mathrm{Atk}_{\mathrm{rd}}$	
Very strict	Noninterference	$\mathrm{Atk}_{\mathrm{ni}}$	

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

All properties are the same statement!

One paradigm, many properties

Integrity and confidentiality relaxed	Nonmalleable information flow	$\mathrm{Atk}_{\mathrm{nmif}}$	nonmauling
Integrity relaxed	Transparent endorsement	$\mathrm{Atk}_{\mathrm{te}}$	nonmauling & incurious
Confidentiality relaxed	Robust declassification	$\mathrm{Atk}_{\mathrm{rd}}$	passive
Very strict	Noninterference	$\mathrm{Atk}_{\mathrm{ni}}$	

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

All properties are the same statement!

One paradigm, many properties

Integrity and confidentiality relaxed	Nonmalleable information flow	$\mathrm{Atk}_{\mathrm{nmif}}$	nonmauling
Integrity relaxed	Transparent endorsement	$\mathrm{Atk}_{\mathrm{te}}$	nonmauling & incurious
Confidentiality relaxed	Robust declassification	$\mathrm{Atk}_{\mathrm{rd}}$	passive
Very strict	Noninterference	$\mathrm{Atk}_{\mathrm{ni}}$	passive & incurious

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

All properties are the same statement!

One paradigm, many properties

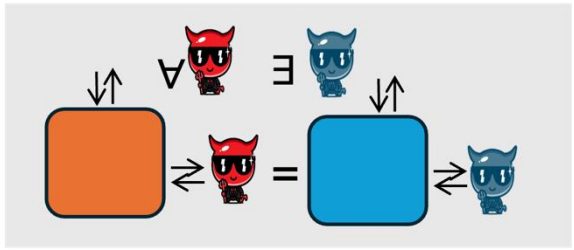
Integrity and confidentiality relaxed -----	Nonmalleable information flow	$\mathrm{Atk}_{\mathrm{nmif}}$	nonmauling
Integrity relaxed ----------	Transparent endorsement	$\mathrm{Atk}_{\mathrm{te}}$	nonmauling & incurious
Confidentiality relaxed -----	Robust declassification	$\mathrm{Atk}_{\mathrm{rd}}$	passive
Very strict ----------	Noninterference	$\mathrm{Atk}_{\mathrm{ni}}$	passive & incurious

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

passive = never corrupts
incurious = never reads

All properties are the same statement!

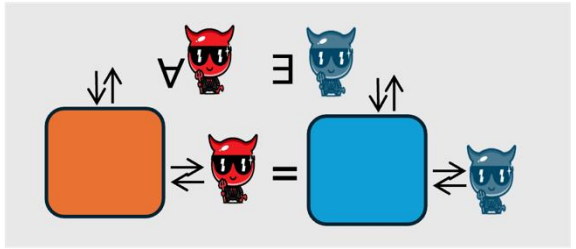
Our Work











$$\llbracket \rrbracket$$

$$\Gamma \vdash t : \tau$$

Information flow needs downgrading

Information flow needs downgrading

```
Alice: <A, A>          Bob: <B,B>

let alice_networth = endorse(input from Alice, A∧B)
let bob_networth = endorse(input from Bob, A∧B)

let greater = declassify(alice_networth > bob_networth, A∨B )

output greater to Alice, Bob
```

Information flow needs downgrading



```
Alice: <A, A>          Bob: <B,B>

let alice_networth = endorse(input from Alice, A∧B)
let bob_networth = endorse(input from Bob, A∧B)

let greater = declassify(alice_networth > bob_networth, A∨B )

output greater to Alice, Bob
```

Information flow needs downgrading



```
Alice: <A, A>          Bob: <B,B>

let alice_networth = endorse(input from Alice, A∧B)
let bob_networth = endorse(input from Bob, A∧B)

let greater = declassify(alice_networth > bob_networth, A∨B )

output greater to Alice, Bob
```

Information flow needs downgrading

```
Alice: <A, A>          Bob: <B,B>

let alice_networth = endorse(input from Alice, A∧B)
let bob_networth = endorse(input from Bob, A∧B)

let greater = declassify(alice_networth > bob_networth, A∨B )

output greater to Alice, Bob
```





Information flow needs downgrading

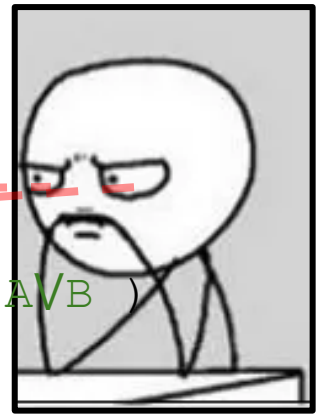
```
Alice: <A, A>          Bob: <B,B>

let alice_networth = endorse(input from Alice, A∧B)
let bob_networth = endorse(input from Bob, A∧B)

let greater = declassify(alice_networth > bob_networth, A∨B )

output greater to Alice, Bob
```





Information flow needs downgrading

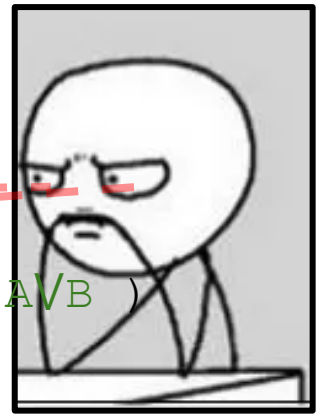


```
Alice: <A, A>          Bob: <B,B>

let alice_networth = endorse(input from Alice, A∧B)
let bob_networth = endorse(input from Bob, A∧B)

let greater = declassify(alice_networth > bob_networth, A∨B )

output greater to Alice, Bob
```



Can we make precise this requirement of needing to just ‘stare hard at downgrading’?

Real–ideal strikes again

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Real–ideal strikes again

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{dgrst}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Real–ideal strikes again

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{dgrst}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…



$\in \mathrm{Atk}_{\mathrm{real}}$

Real–ideal strikes again

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{dgrst}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…



$\in \mathrm{Atk}_{\mathrm{real}}$

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…



$\in \mathrm{Atk}_{\mathrm{dgrst}}$

Real–ideal strikes again

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{dgrst}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…

```
Alice: <A, A>                                   Bob: <B,B>

let alice_networth = endorse(input from Alice,
A∧B)
let bob_networth = endorse(input from Bob, A∧B)

let greater = declassify(alice_networth >
bob_networth, A∨B )

output greater to Alice, Bob
```



$\in \mathrm{Atk}_{\mathrm{real}}$



$\in \mathrm{Atk}_{\mathrm{dgrst}}$

Real–ideal strikes again

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{dgrst}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…

can corrupt

```
Alice: <A, A>                                   Bob: <B,B>

let alice_networth = endorse(input from Alice,
A∧B)
let bob_networth = endorse(input from Bob, A∧B)

let greater = declassify(alice_networth >
bob_networth, A∨B )

output greater to Alice, Bob
```



$\in \mathrm{Atk}_{\mathrm{real}}$



$\in \mathrm{Atk}_{\mathrm{dgrst}}$

Real–ideal strikes again

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{dgrst}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…

can corrupt

```
Alice: <A, A>                                   Bob: <B,B>

let alice_networth = endorse(input from Alice,
A∧B)
let bob_networth = endorse(input from Bob, A∧B)

let greater = declassify(alice_networth >
bob_networth, A∨B )

output greater to Alice, Bob
```

can read



$\in \mathrm{Atk}_{\mathrm{real}}$



$\in \mathrm{Atk}_{\mathrm{dgrst}}$

Real–ideal strikes again

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{dgrst}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…

```
Alice: <A, A>                                   Bob: <B,B>

let alice_networth = endorse(input from Alice,
A∧B)
let bob_networth = endorse(input from Bob, A∧B)

let greater = declassify(alice_networth >
bob_networth, A∨B )

output greater to Alice, Bob
```

can corrupt

can read



$\in \mathrm{Atk}_{\mathrm{real}}$



$\in \mathrm{Atk}_{\mathrm{dgrst}}$

Real adversaries have no more power than if they only interacted at explicit downgrading points!

Real–ideal strikes again

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{ideal}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{dgrst}}.\ [\![p \mid A_r]\!] \approx [\![p \mid A_i]\!]$$

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…

Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✓
v_3	✓	✓
…	…	…

can corrupt

```
Alice: <A, A>                                   Bob: <B,B>

let alice_networth = endorse(input from Alice,
A∧B)
let bob_networth = endorse(input from Bob, A∧B)

let greater = declassify(alice_networth >
bob_networth, A∨B )

output greater to Alice, Bob
```

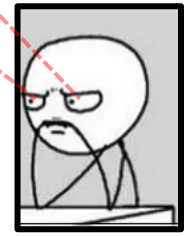
can read



$\in \mathrm{Atk}_{\mathrm{real}}$



$\in \mathrm{Atk}_{\mathrm{dgrst}}$



Real adversaries have no more power than if they only interacted at explicit downgrading points!

A real–ideal type soundness theorem

A real–ideal type soundness theorem

$\boxed{pc \vdash_{\skull} c}$

ST-Assign
$$\frac{\skull|\mathrm{src}(a)| \sqsubseteq \skull|\mathrm{dst}(a)| \qquad pc \sqsubseteq \skull|\mathrm{dst}(a)|}{pc \vdash_{\skull} a}$$

$\boxed{pc \vdash_{\skull} p}$

ST-Empty
$$\frac{}{pc \vdash_{\skull} \epsilon}$$

ST-Endorse
$$\frac{\skull|y| = \mathsf{PU} \qquad \skull|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{\skull} x := \mathtt{endorse}(y)}$$

ST-Declassify
$$\frac{\skull|y| = \mathsf{ST} \qquad \skull|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{\skull} x := \mathtt{declassify}(y)}$$

ST-Seq
$$\frac{pc \vdash_{\skull} c \qquad pc \vdash_{\skull} p}{pc \vdash_{\skull} c\,;p}$$

ST-Output
$$\frac{pc \sqsubseteq \skull|x|}{pc \vdash_{\skull} \mathtt{output}\ x}$$

ST-If
$$\frac{\skull|x| \sqcup pc \vdash_{\skull} p_1 \qquad \skull|x| \sqcup pc \vdash_{\skull} p_2}{pc \vdash_{\skull} \mathtt{if}\ t(x)\ \mathtt{then}\ p_1\ \mathtt{else}\ p_2}$$

$\skull|-|,\ pc \in \{\mathsf{PT}, \mathsf{PU}, \mathsf{ST}, \mathsf{SU}\}$

ST-While
$$\frac{\skull|x| \sqcup pc \vdash_{\skull} p}{pc \vdash_{\skull} \mathtt{while}\ t(x)\ \mathtt{do}\ p}$$

A real–ideal type soundness theorem

$pc \vdash_{☠} c$

ST-Assign

$$\frac{☠|\mathrm{src}(a)| \sqsubseteq ☠|\mathrm{dst}(a)| \qquad pc \sqsubseteq ☠|\mathrm{dst}(a)|}{pc \vdash_{☠} a}$$

$pc \vdash_{☠} p$

ST-Empty

$$\frac{}{pc \vdash_{☠} \epsilon}$$

ST-Endorse

$$\frac{☠|y| = \mathsf{PU} \qquad ☠|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{☠} x := \mathtt{endorse}(y)}$$

ST-Declassify

$$\frac{☠|y| = \mathsf{ST} \qquad ☠|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{☠} x := \mathtt{declassify}(y)}$$

ST-Seq

$$\frac{pc \vdash_{☠} c \qquad pc \vdash_{☠} p}{pc \vdash_{☠} c\,;p}$$

ST-Output

$$\frac{pc \sqsubseteq ☠|x|}{pc \vdash_{☠} \mathtt{output}\ x}$$

ST-If

$$\frac{☠|x| \sqcup pc \vdash_{☠} p_1 \qquad ☠|x| \sqcup pc \vdash_{☠} p_2}{pc \vdash_{☠} \mathtt{if}\ t(x)\ \mathtt{then}\ p_1\ \mathtt{else}\ p_2}$$

$☠|-|,\ pc \in \{\mathsf{PT}, \mathsf{PU}, \mathsf{ST}, \mathsf{SU}\}$

ST-While

$$\frac{☠|x| \sqcup pc \vdash_{☠} p}{pc \vdash_{☠} \mathtt{while}\ t(x)\ \mathtt{do}\ p}$$

A real–ideal type soundness theorem

$pc \vdash_{☠} c$

ST-Assign

$$\frac{☠|\mathrm{src}(a)| \sqsubseteq ☠|\mathrm{dst}(a)| \qquad pc \sqsubseteq ☠|\mathrm{dst}(a)|}{pc \vdash_{☠} a}$$

$pc \vdash_{☠} p$

ST-Empty

$$\frac{}{pc \vdash_{☠} \epsilon}$$

ST-Endorse

$$\frac{☠|y| = \mathsf{PU} \qquad ☠|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{☠} x := \mathtt{endorse}(y)}$$

ST-Declassify

$$\frac{☠|y| = \mathsf{ST} \qquad ☠|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{☠} x := \mathtt{declassify}(y)}$$

ST-Seq

$$\frac{pc \vdash_{☠} c \qquad pc \vdash_{☠} p}{pc \vdash_{☠} c\,;p}$$

ST-Output

$$\frac{pc \sqsubseteq ☠|x|}{pc \vdash_{☠} \mathtt{output}\ x}$$

ST-If

$$\frac{☠|x| \sqcup pc \vdash_{☠} p_1 \qquad ☠|x| \sqcup pc \vdash_{☠} p_2}{pc \vdash_{☠} \mathtt{if}\ t(x)\ \mathtt{then}\ p_1\ \mathtt{else}\ p_2}$$

$☠|-|,\ pc \in \{\mathsf{PT}, \mathsf{PU}, \mathsf{ST}, \mathsf{SU}\}$

ST-While

$$\frac{☠|x| \sqcup pc \vdash_{☠} p}{pc \vdash_{☠} \mathtt{while}\ t(x)\ \mathtt{do}\ p}$$

A real–ideal type soundness theorem

$\boxed{pc \vdash_{\skull} c}$

ST-Assign
$$\frac{\skull|\mathrm{src}(a)| \sqsubseteq \skull|\mathrm{dst}(a)| \qquad pc \sqsubseteq \skull|\mathrm{dst}(a)|}{pc \vdash_{\skull} a}$$

$\boxed{pc \vdash_{\skull} p}$

ST-Empty
$$\frac{}{pc \vdash_{\skull} \epsilon}$$

ST-Endorse
$$\frac{\skull|y| = \mathsf{PU} \qquad \skull|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{\skull} x := \mathtt{endorse}(y)}$$

ST-Declassify
$$\frac{\skull|y| = \mathsf{ST} \qquad \skull|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{\skull} x := \mathtt{declassify}(y)}$$

ST-Seq
$$\frac{pc \vdash_{\skull} c \qquad pc \vdash_{\skull} p}{pc \vdash_{\skull} c\,;p}$$

ST-Output
$$\frac{pc \sqsubseteq \skull|x|}{pc \vdash_{\skull} \mathtt{output}\ x}$$

ST-If
$$\frac{\skull|x| \sqcup pc \vdash_{\skull} p_1 \qquad \skull|x| \sqcup pc \vdash_{\skull} p_2}{pc \vdash_{\skull} \mathtt{if}\ t(x)\ \mathtt{then}\ p_1\ \mathtt{else}\ p_2}$$

$\skull|-|,\ pc \in \{\mathsf{PT}, \mathsf{PU}, \mathsf{ST}, \mathsf{SU}\}$

ST-While
$$\frac{\skull|x| \sqcup pc \vdash_{\skull} p}{pc \vdash_{\skull} \mathtt{while}\ t(x)\ \mathtt{do}\ p}$$

A real–ideal type soundness theorem

$\boxed{pc \vdash_{\text{💀}} c}$

ST-Assign
$$\frac{\text{💀}|\mathrm{src}(a)| \sqsubseteq \text{💀}|\mathrm{dst}(a)| \qquad pc \sqsubseteq \text{💀}|\mathrm{dst}(a)|}{pc \vdash_{\text{💀}} a}$$

$\boxed{pc \vdash_{\text{💀}} p}$

ST-Empty
$$\frac{}{pc \vdash_{\text{💀}} \epsilon}$$

ST-Endorse
$$\frac{\text{💀}|y| = \mathsf{PU} \qquad \text{💀}|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{\text{💀}} x := \mathtt{endorse}(y)}$$

ST-Declassify
$$\frac{\text{💀}|y| = \mathsf{ST} \qquad \text{💀}|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{\text{💀}} x := \mathtt{declassify}(y)}$$

ST-Seq
$$\frac{pc \vdash_{\text{💀}} c \qquad pc \vdash_{\text{💀}} p}{pc \vdash_{\text{💀}} c\,;p}$$

ST-Output
$$\frac{pc \sqsubseteq \text{💀}|x|}{pc \vdash_{\text{💀}} \mathtt{output}\ x}$$

ST-If
$$\frac{\text{💀}|x| \sqcup pc \vdash_{\text{💀}} p_1 \qquad \text{💀}|x| \sqcup pc \vdash_{\text{💀}} p_2}{pc \vdash_{\text{💀}} \mathtt{if}\ t(x)\ \mathtt{then}\ p_1\ \mathtt{else}\ p_2}$$

$\text{💀}|-|,\ pc \in \{\mathsf{PT}, \mathsf{PU}, \mathsf{ST}, \mathsf{SU}\}$

ST-While
$$\frac{\text{💀}|x| \sqcup pc \vdash_{\text{💀}} p}{pc \vdash_{\text{💀}} \mathtt{while}\ t(x)\ \mathtt{do}\ p}$$

A real–ideal type soundness theorem

$\boxed{pc \vdash_{☠} c}$

ST-Assign
$$\frac{☠|\mathrm{src}(a)| \sqsubseteq ☠|\mathrm{dst}(a)| \qquad pc \sqsubseteq ☠|\mathrm{dst}(a)|}{pc \vdash_{☠} a}$$

ST-Endorse
$$\frac{☠|y| = \mathsf{PU} \qquad ☠|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{☠} x := \mathtt{endorse}(y)}$$

ST-Declassify
$$\frac{☠|y| = \mathsf{ST} \qquad ☠|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{☠} x := \mathtt{declassify}(y)}$$

ST-Output
$$\frac{pc \sqsubseteq ☠|x|}{pc \vdash_{☠} \mathtt{output}\ x}$$

ST-If
$$\frac{☠|x| \sqcup pc \vdash_{☠} p_1 \qquad ☠|x| \sqcup pc \vdash_{☠} p_2}{pc \vdash_{☠} \mathtt{if}\ t(x)\ \mathtt{then}\ p_1\ \mathtt{else}\ p_2}$$

ST-While
$$\frac{☠|x| \sqcup pc \vdash_{☠} p}{pc \vdash_{☠} \mathtt{while}\ t(x)\ \mathtt{do}\ p}$$

$\boxed{pc \vdash_{☠} p}$

ST-Empty
$$\frac{}{pc \vdash_{☠} \epsilon}$$

ST-Seq
$$\frac{pc \vdash_{☠} c \qquad pc \vdash_{☠} p}{pc \vdash_{☠} c\,;p}$$

$☠|-|,\ pc \in \{\mathsf{PT}, \mathsf{PU}, \mathsf{ST}, \mathsf{SU}\}$

A real–ideal type soundness theorem

$\boxed{pc \vdash_{\skull} c}$

ST-Assign
$$\frac{\skull|\mathrm{src}(a)| \sqsubseteq \skull|\mathrm{dst}(a)| \qquad pc \sqsubseteq \skull|\mathrm{dst}(a)|}{pc \vdash_{\skull} a}$$

$\boxed{pc \vdash_{\skull} p}$

ST-Empty
$$\frac{}{pc \vdash_{\skull} \epsilon}$$

ST-Endorse
$$\frac{\skull|y| = \mathsf{PU} \qquad \skull|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{\skull} x := \mathtt{endorse}(y)}$$

ST-Declassify
$$\frac{\skull|y| = \mathsf{ST} \qquad \skull|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{\skull} x := \mathtt{declassify}(y)}$$

ST-Seq
$$\frac{pc \vdash_{\skull} c \qquad pc \vdash_{\skull} p}{pc \vdash_{\skull} c\,;p}$$

ST-Output
$$\frac{pc \sqsubseteq \skull|x|}{pc \vdash_{\skull} \mathtt{output}\ x}$$

ST-If
$$\frac{\skull|x| \sqcup pc \vdash_{\skull} p_1 \qquad \skull|x| \sqcup pc \vdash_{\skull} p_2}{pc \vdash_{\skull} \mathtt{if}\ t(x)\ \mathtt{then}\ p_1\ \mathtt{else}\ p_2}$$

$\skull|-|,\ pc \in \{\mathsf{PT}, \mathsf{PU}, \mathsf{ST}, \mathsf{SU}\}$

ST-While
$$\frac{\skull|x| \sqcup pc \vdash_{\skull} p}{pc \vdash_{\skull} \mathtt{while}\ t(x)\ \mathtt{do}\ p}$$

$$\vdash_{\skull} p \implies$$
$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{dgrst}}.\ [\![p \mid A_r]\!] \approx_w [\![p \mid A_i]\!]$$

A real–ideal type soundness theorem

$pc \vdash_{☠} c$

$$\textsc{ST-Assign}\ \frac{☠|\mathrm{src}(a)| \sqsubseteq ☠|\mathrm{dst}(a)| \qquad pc \sqsubseteq ☠|\mathrm{dst}(a)|}{pc \vdash_{☠} a}$$

$pc \vdash_{☠} p$

$$\textsc{ST-Empty}\ \frac{}{pc \vdash_{☠} \epsilon}$$

$$\textsc{ST-Endorse}\ \frac{☠|y| = \mathsf{PU} \qquad ☠|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{☠} x := \mathtt{endorse}(y)}$$

$$\textsc{ST-Declassify}\ \frac{☠|y| = \mathsf{ST} \qquad ☠|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{☠} x := \mathtt{declassify}(y)}$$

$$\textsc{ST-Seq}\ \frac{pc \vdash_{☠} c \qquad pc \vdash_{☠} p}{pc \vdash_{☠} c\,;p}$$

$$\textsc{ST-Output}\ \frac{pc \sqsubseteq ☠|x|}{pc \vdash_{☠} \mathtt{output}\ x}$$

$$\textsc{ST-If}\ \frac{☠|x| \sqcup pc \vdash_{☠} p_1 \qquad ☠|x| \sqcup pc \vdash_{☠} p_2}{pc \vdash_{☠} \mathtt{if}\ t(x)\ \mathtt{then}\ p_1\ \mathtt{else}\ p_2}$$

$☠|-|,\ pc \in \{\mathsf{PT}, \mathsf{PU}, \mathsf{ST}, \mathsf{SU}\}$

$$\textsc{ST-While}\ \frac{☠|x| \sqcup pc \vdash_{☠} p}{pc \vdash_{☠} \mathtt{while}\ t(x)\ \mathtt{do}\ p}$$

$$\vdash_{☠} p \implies$$

$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{dgrst}}.\ [\![p \mid A_r]\!] \approx_w [\![p \mid A_i]\!]$$

A real–ideal type soundness theorem

$\boxed{pc \vdash_{☠} c}$

ST-Assign
$$\frac{☠|\mathrm{src}(a)| \sqsubseteq ☠|\mathrm{dst}(a)| \qquad pc \sqsubseteq ☠|\mathrm{dst}(a)|}{pc \vdash_{☠} a}$$

$\boxed{pc \vdash_{☠} p}$

ST-Empty
$$\frac{}{pc \vdash_{☠} \epsilon}$$

ST-Endorse
$$\frac{☠|y| = \mathsf{PU} \qquad ☠|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{☠} x := \mathtt{endorse}(y)}$$

ST-Declassify
$$\frac{☠|y| = \mathsf{ST} \qquad ☠|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{☠} x := \mathtt{declassify}(y)}$$

ST-Seq
$$\frac{pc \vdash_{☠} c \qquad pc \vdash_{☠} p}{pc \vdash_{☠} c\,;p}$$

ST-Output
$$\frac{pc \sqsubseteq ☠|x|}{pc \vdash_{☠} \mathtt{output}\ x}$$

ST-If
$$\frac{☠|x| \sqcup pc \vdash_{☠} p_1 \qquad ☠|x| \sqcup pc \vdash_{☠} p_2}{pc \vdash_{☠} \mathtt{if}\ t(x)\ \mathtt{then}\ p_1\ \mathtt{else}\ p_2}$$

$☠|-|,\ pc \in \{\mathsf{PT}, \mathsf{PU}, \mathsf{ST}, \mathsf{SU}\}$

ST-While
$$\frac{☠|x| \sqcup pc \vdash_{☠} p}{pc \vdash_{☠} \mathtt{while}\ t(x)\ \mathtt{do}\ p}$$

$$\vdash_{☠} p \implies \forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{dgrst}}.\ [\![p \mid A_r]\!] \approx_w [\![p \mid A_i]\!]$$

A real–ideal type soundness theorem

$\boxed{pc \vdash_{\skull} c}$

ST-Assign
$$\frac{\skull|\mathrm{src}(a)| \sqsubseteq \skull|\mathrm{dst}(a)| \qquad pc \sqsubseteq \skull|\mathrm{dst}(a)|}{pc \vdash_{\skull} a}$$

$\boxed{pc \vdash_{\skull} p}$

ST-Empty
$$\frac{}{pc \vdash_{\skull} \epsilon}$$

ST-Endorse
$$\frac{\skull|y| = \mathsf{PU} \qquad \skull|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{\skull} x := \mathtt{endorse}(y)}$$

ST-Declassify
$$\frac{\skull|y| = \mathsf{ST} \qquad \skull|x| = \mathsf{PT}}{\mathsf{PT} \vdash_{\skull} x := \mathtt{declassify}(y)}$$

ST-Seq
$$\frac{pc \vdash_{\skull} c \qquad pc \vdash_{\skull} p}{pc \vdash_{\skull} c\,;p}$$

ST-Output
$$\frac{pc \sqsubseteq \skull|x|}{pc \vdash_{\skull} \mathtt{output}\ x}$$

ST-If
$$\frac{\skull|x| \sqcup pc \vdash_{\skull} p_1 \qquad \skull|x| \sqcup pc \vdash_{\skull} p_2}{pc \vdash_{\skull} \mathtt{if}\ t(x)\ \mathtt{then}\ p_1\ \mathtt{else}\ p_2}$$

$\skull|-|,\ pc \in \{\mathsf{PT}, \mathsf{PU}, \mathsf{ST}, \mathsf{SU}\}$

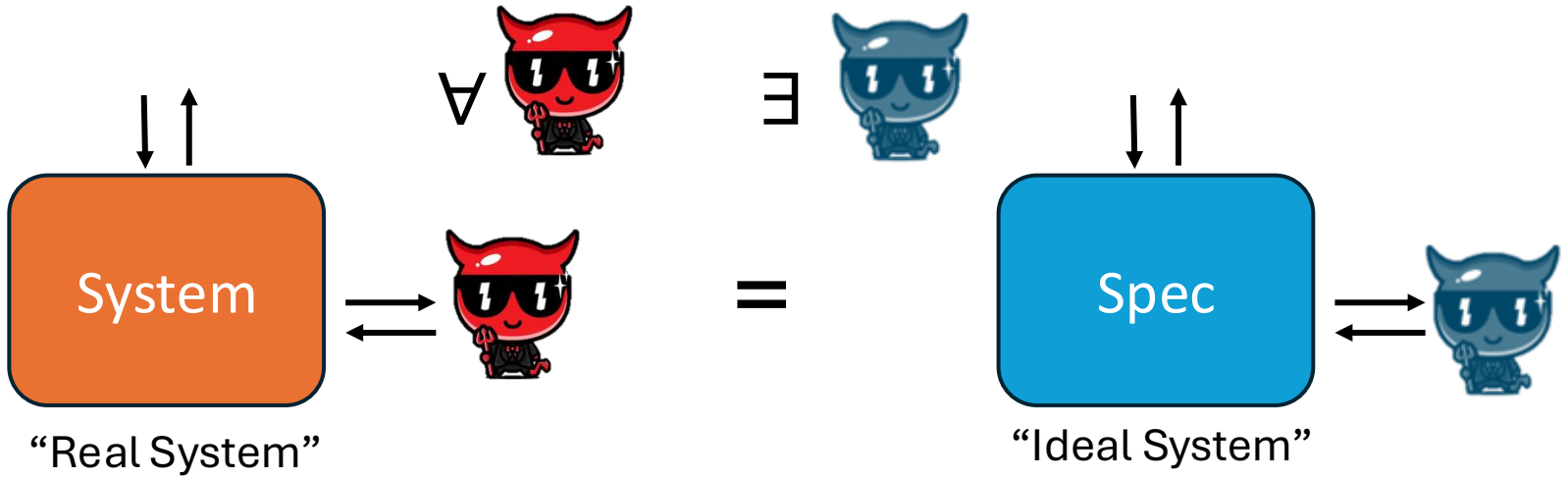
ST-While
$$\frac{\skull|x| \sqcup pc \vdash_{\skull} p}{pc \vdash_{\skull} \mathtt{while}\ t(x)\ \mathtt{do}\ p}$$

Checked in PT

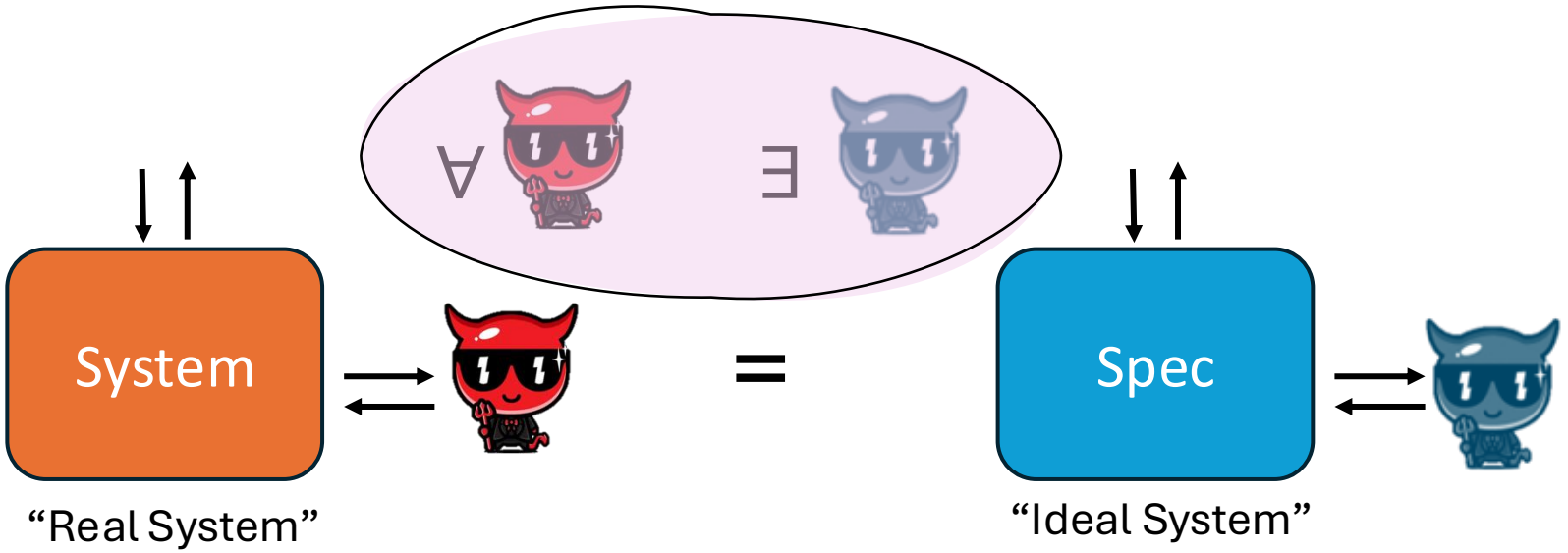
$$\vdash_{\skull} p \implies$$
$$\forall A_r \in \mathrm{Atk}_{\mathrm{real}}.\ \exists A_i \in \mathrm{Atk}_{\mathrm{dgrst}}.\ [\![p \mid A_r]\!] \approx_w [\![p \mid A_i]\!]$$

Proving real-ideal emulation is hard

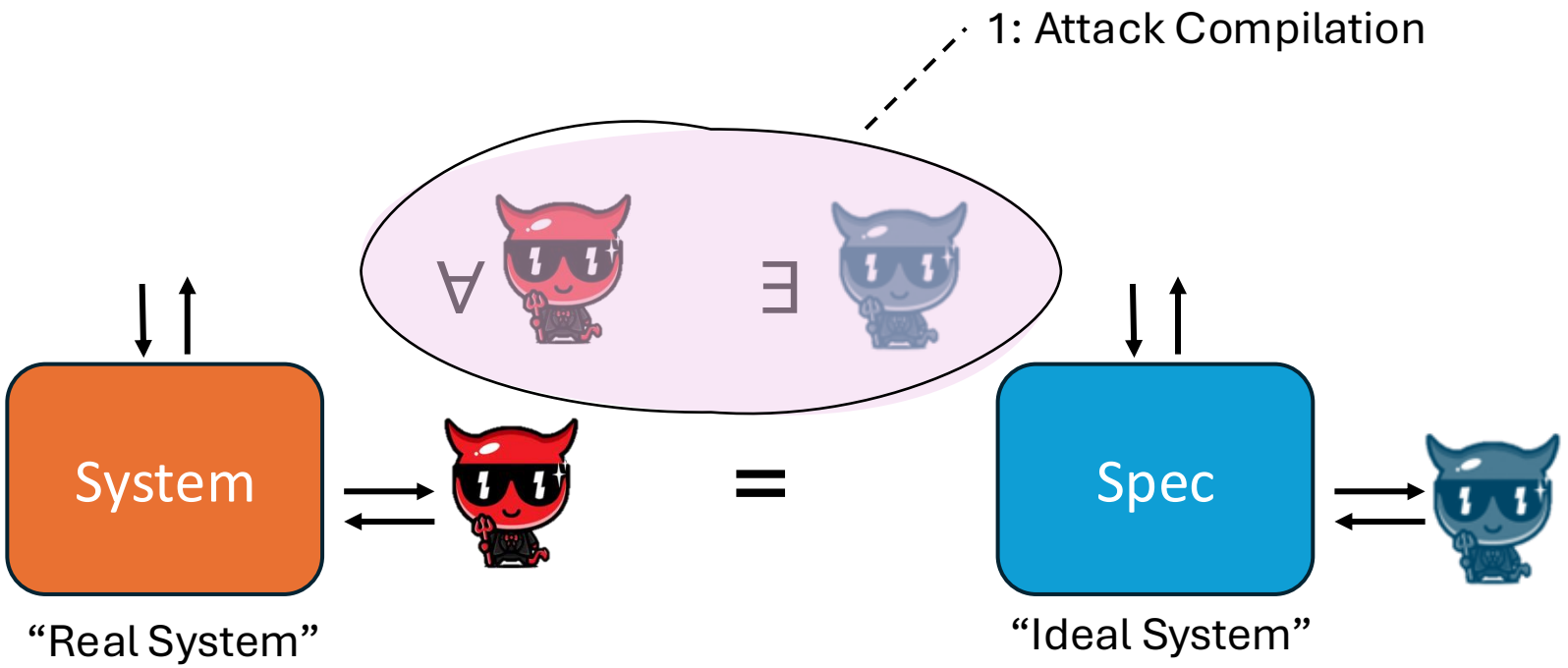
Proving real-ideal emulation is hard



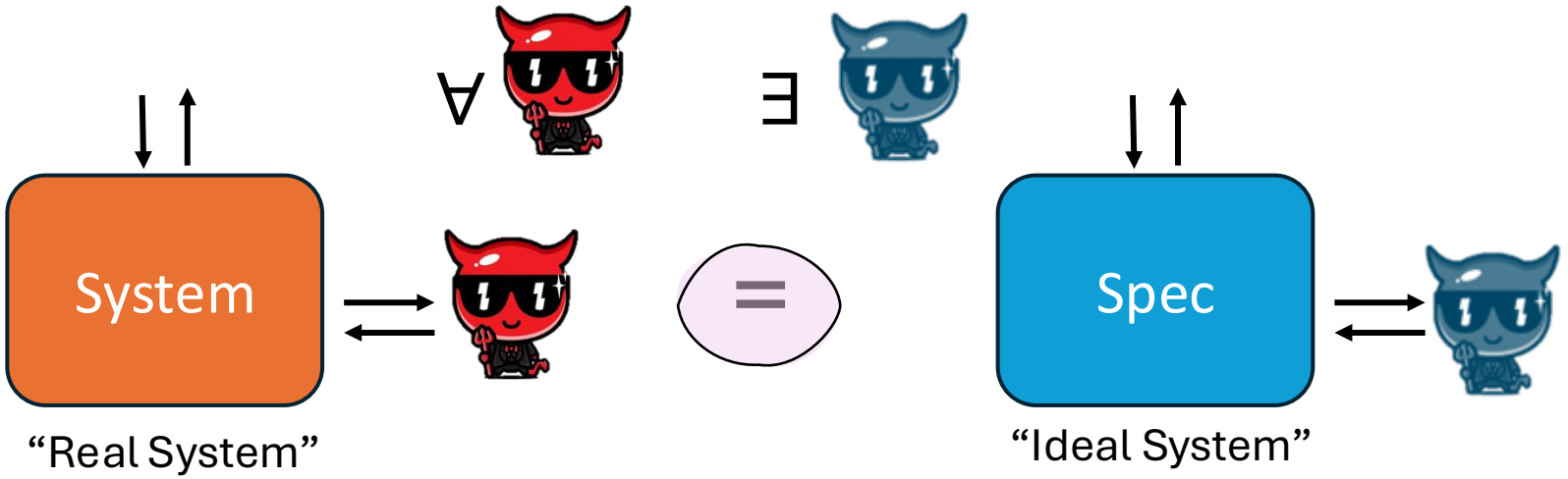
Proving real-ideal emulation is hard



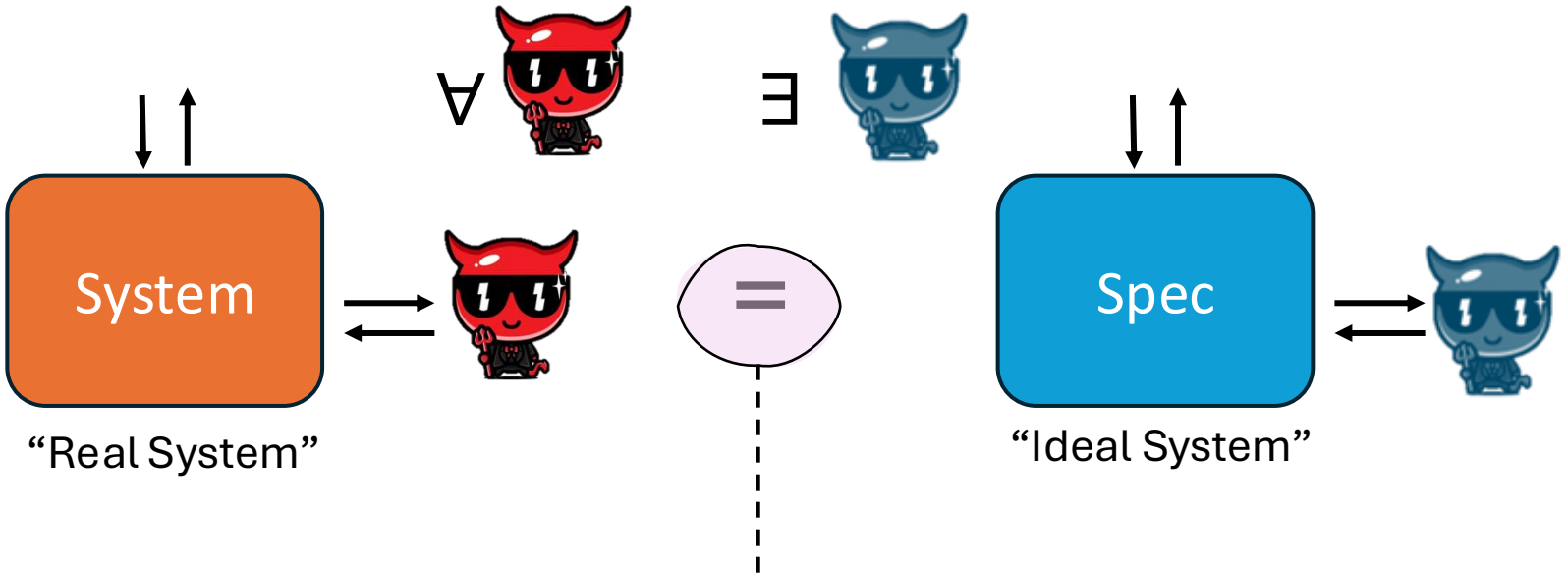
Proving real-ideal emulation is hard



Proving real-ideal emulation is hard



Proving real-ideal emulation is hard



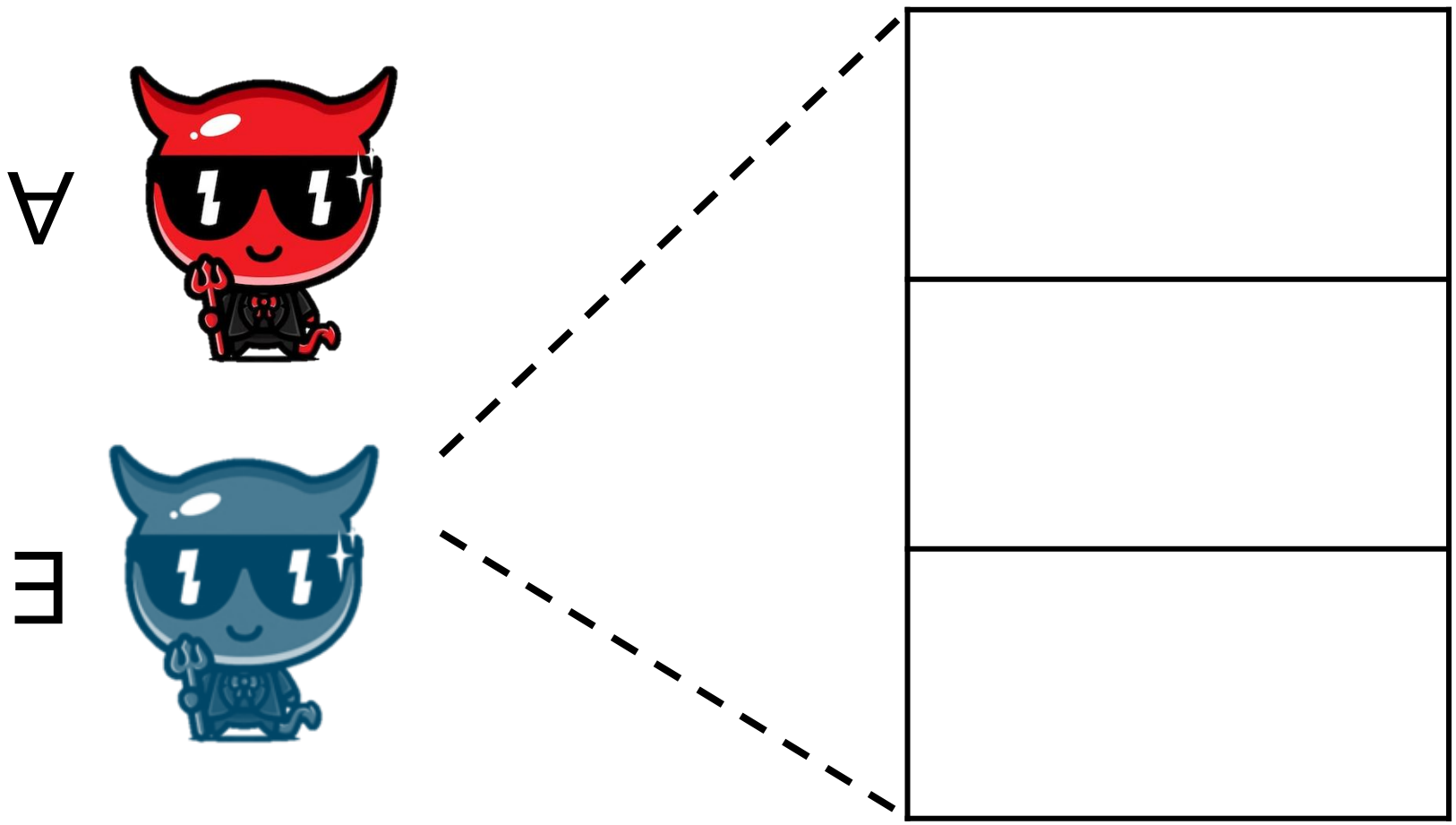
2: Behavioral equivalence – Usually bisimulation

A taste: the ideal attack construction

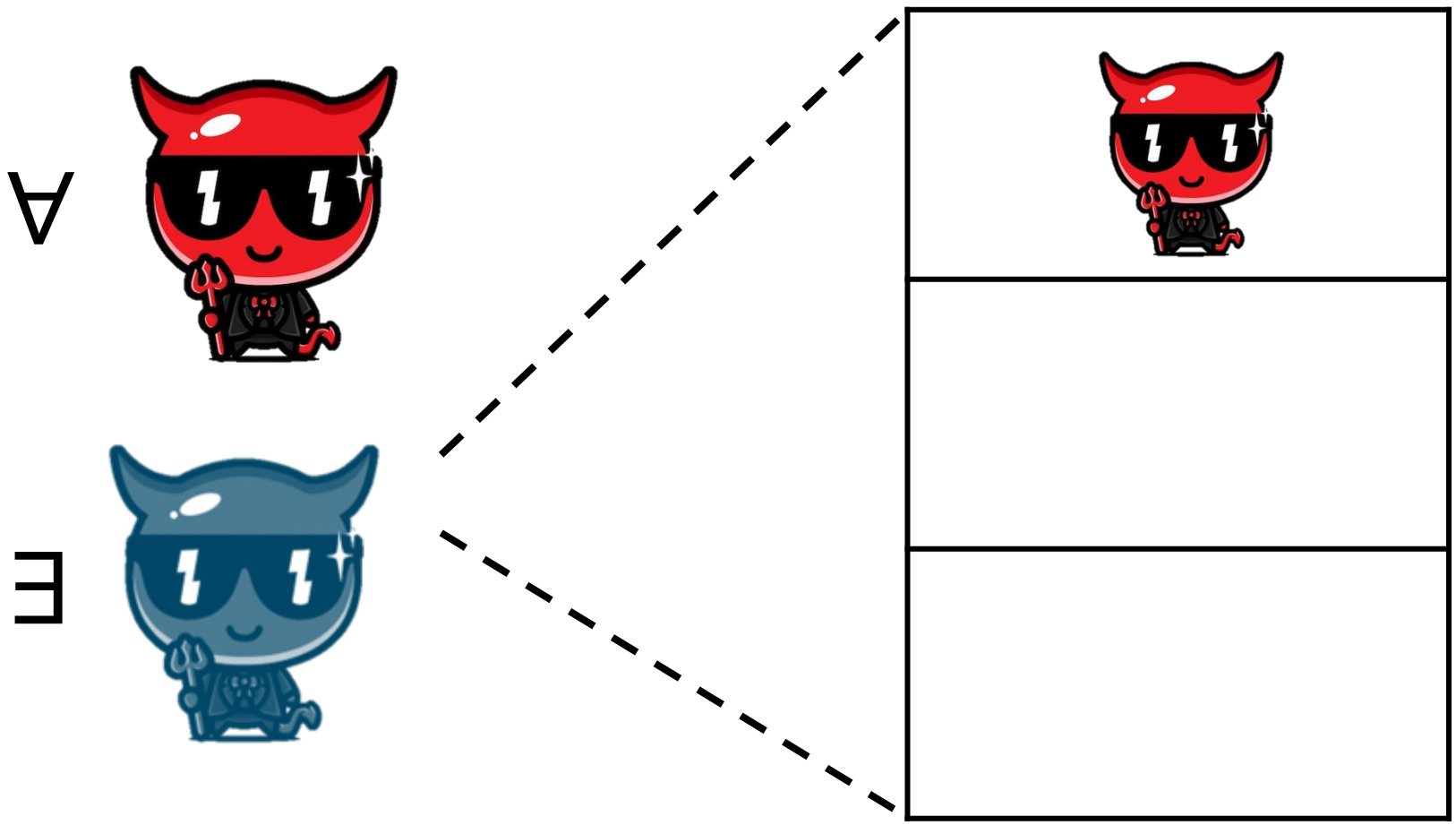
A taste: the ideal attack construction

$\forall$ 

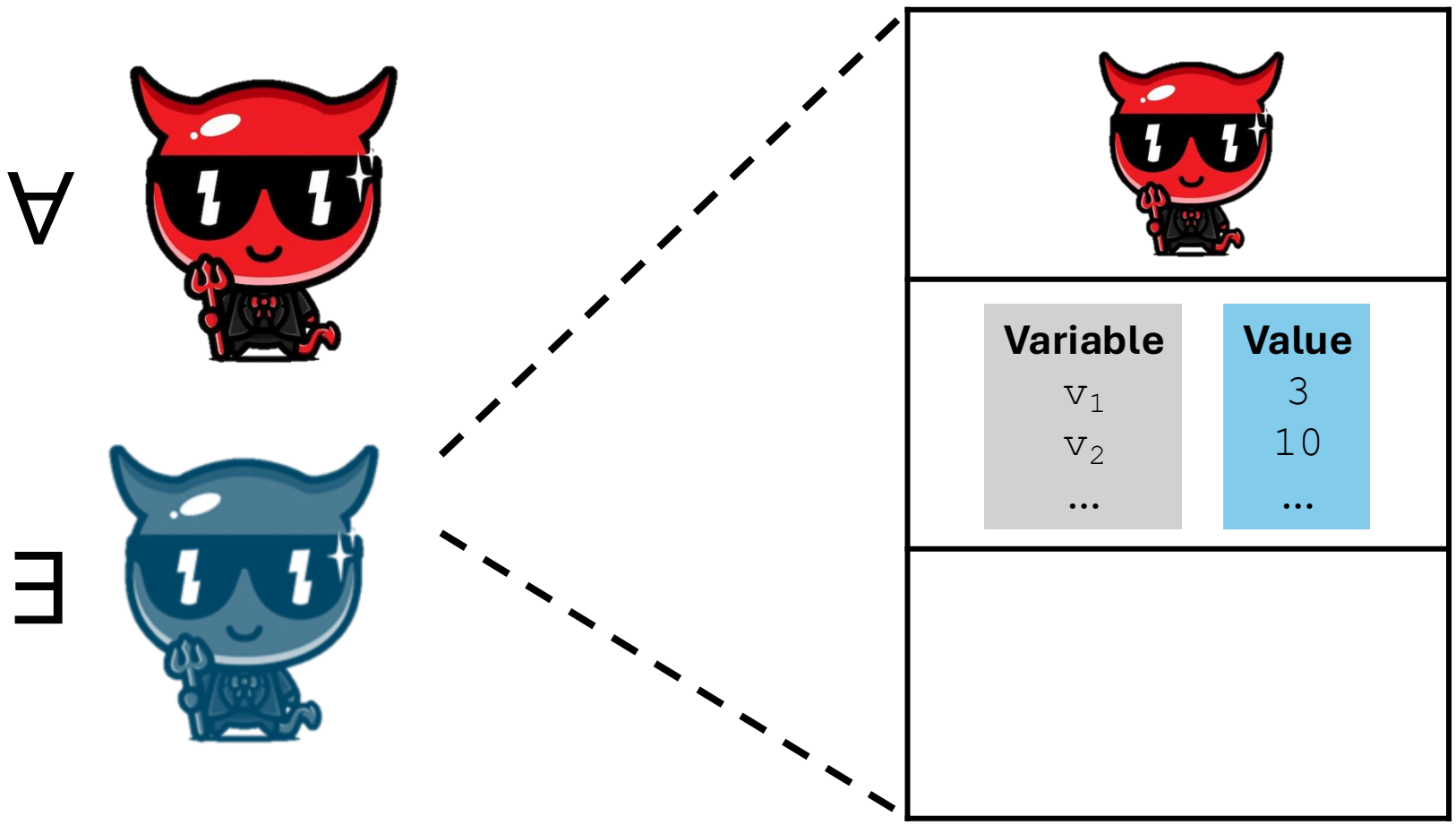
A taste: the ideal attack construction



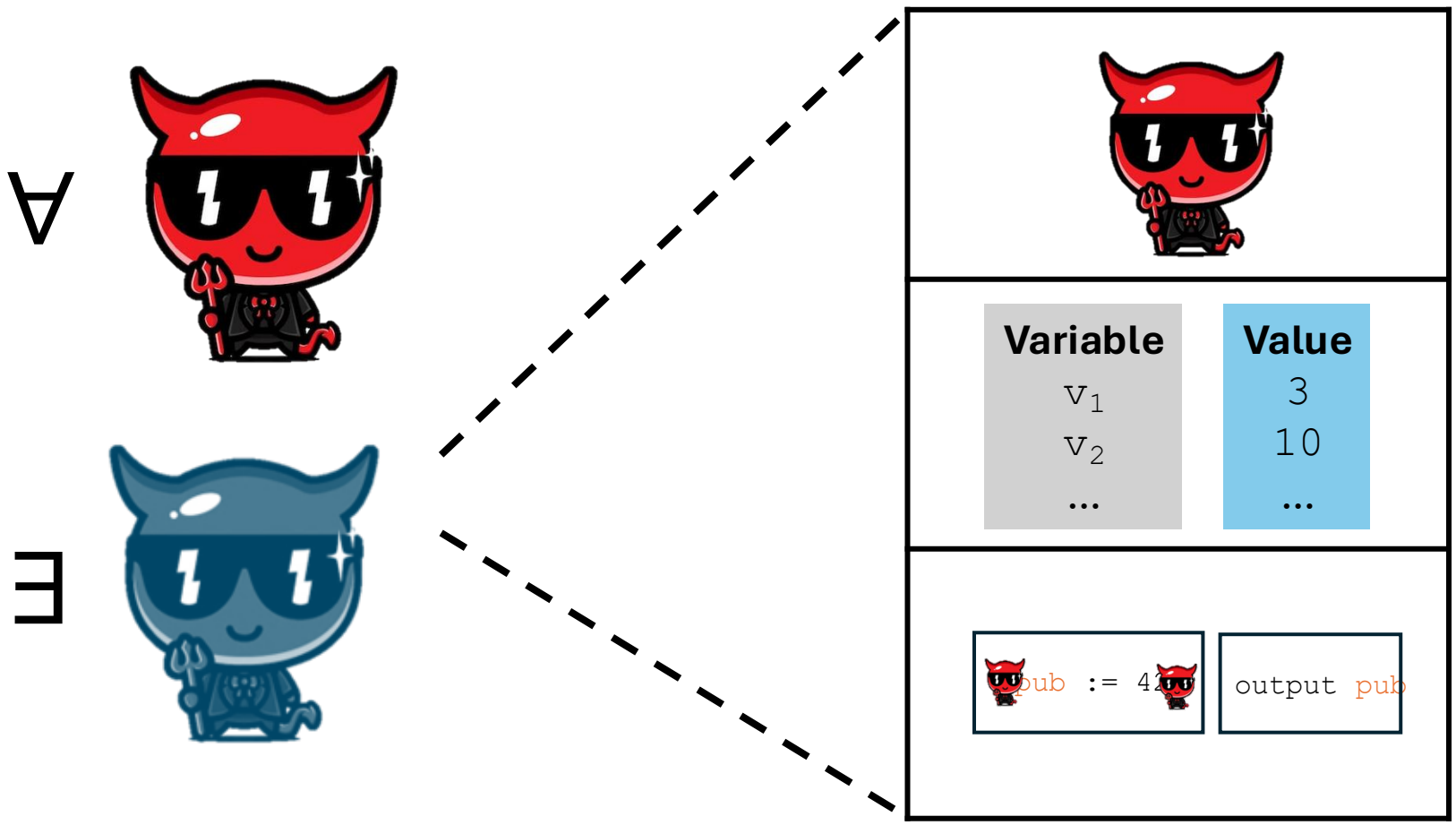
A taste: the ideal attack construction



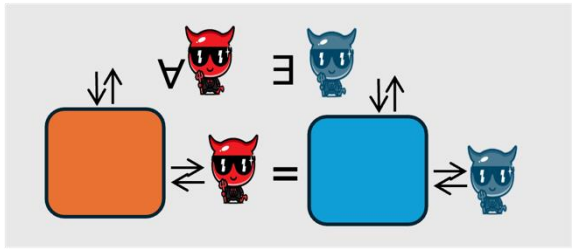
A taste: the ideal attack construction



A taste: the ideal attack construction



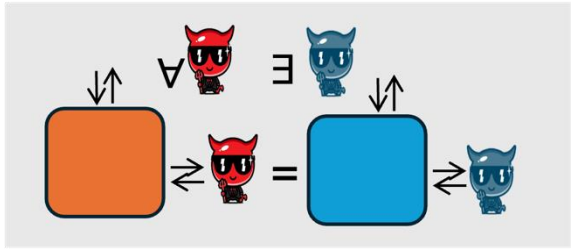
Our Work











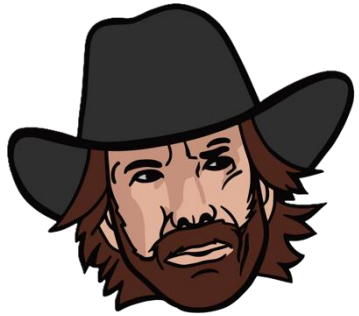
$[\![\ ]\!]$

$\Gamma \vdash t : \tau$



Heterogenous security

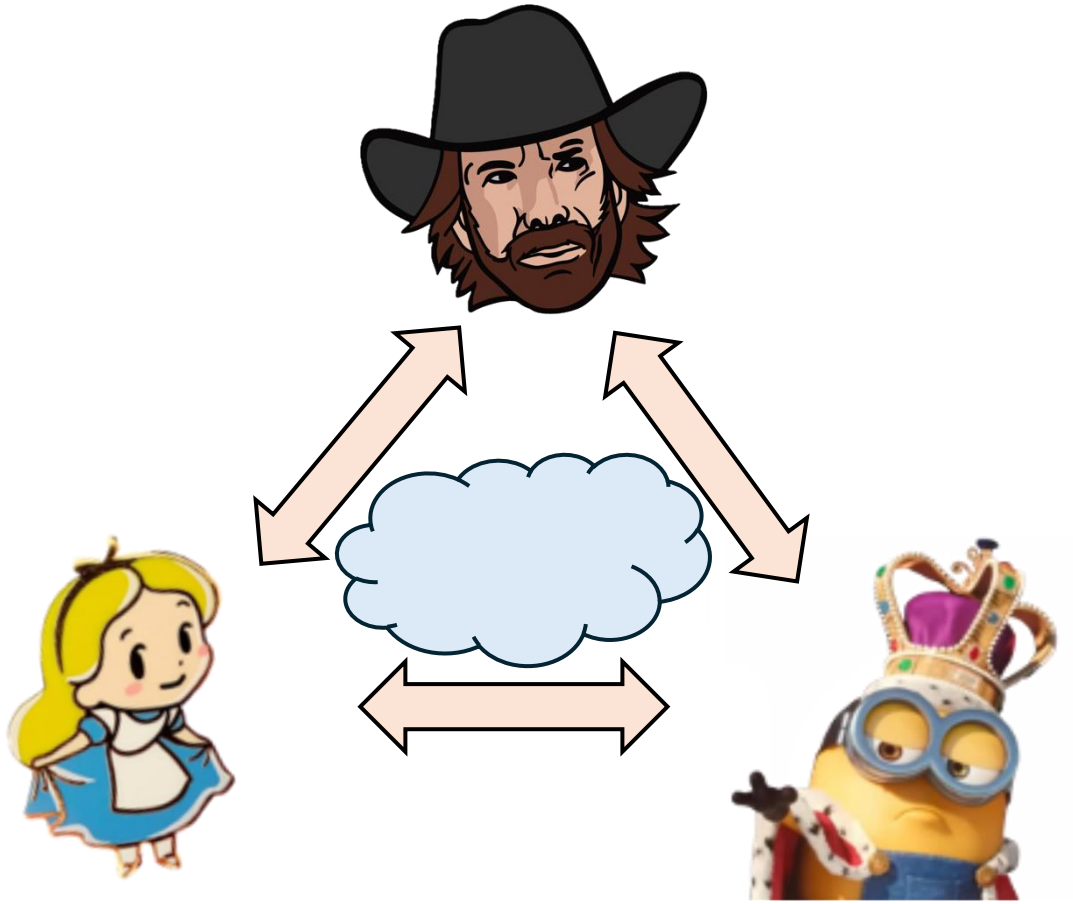
Heterogenous security



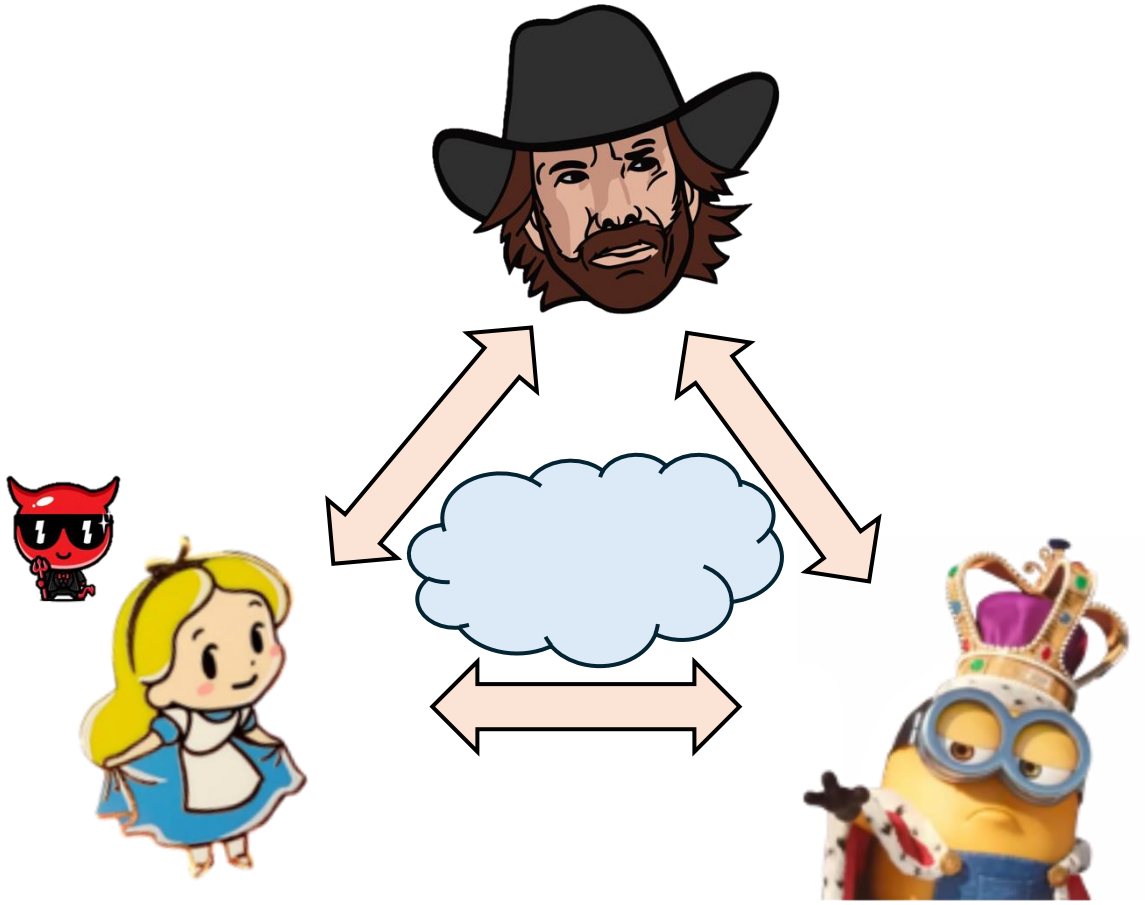




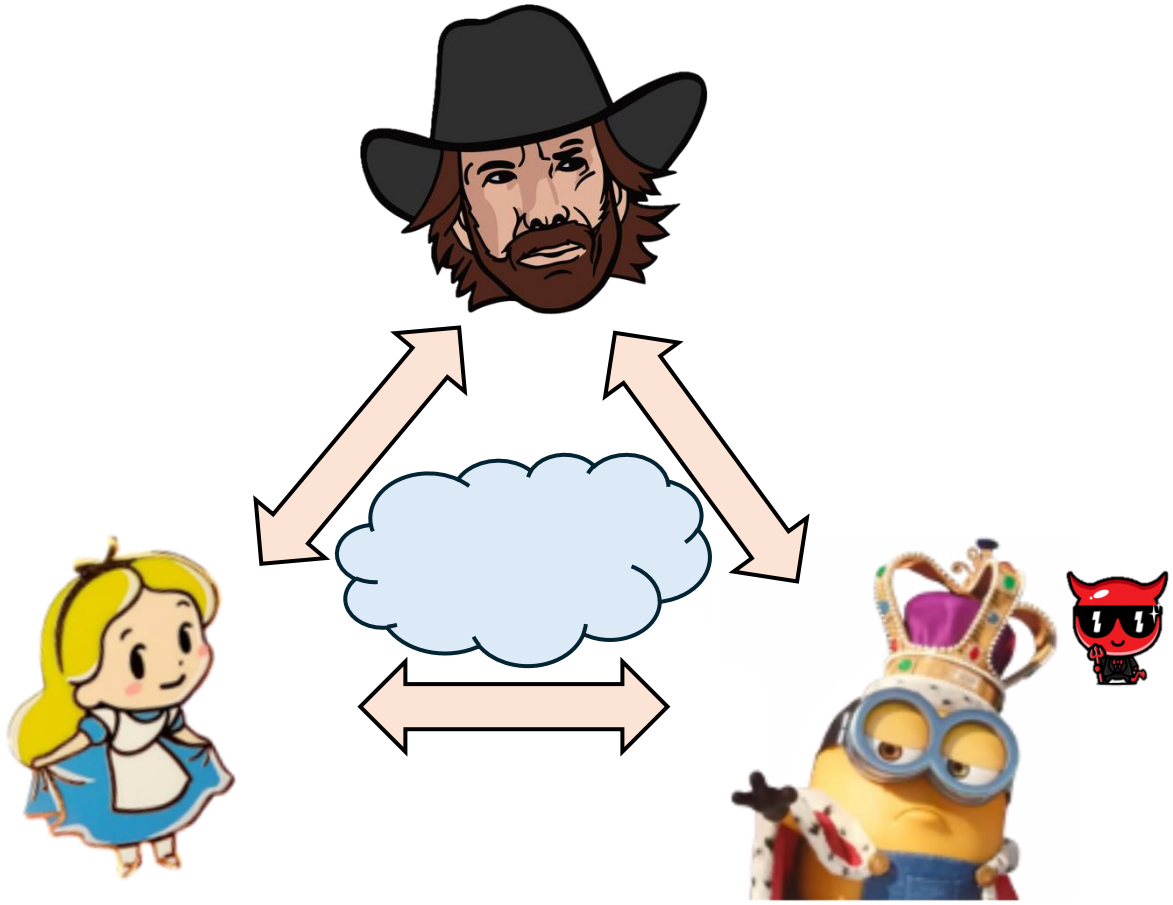
Heterogenous security



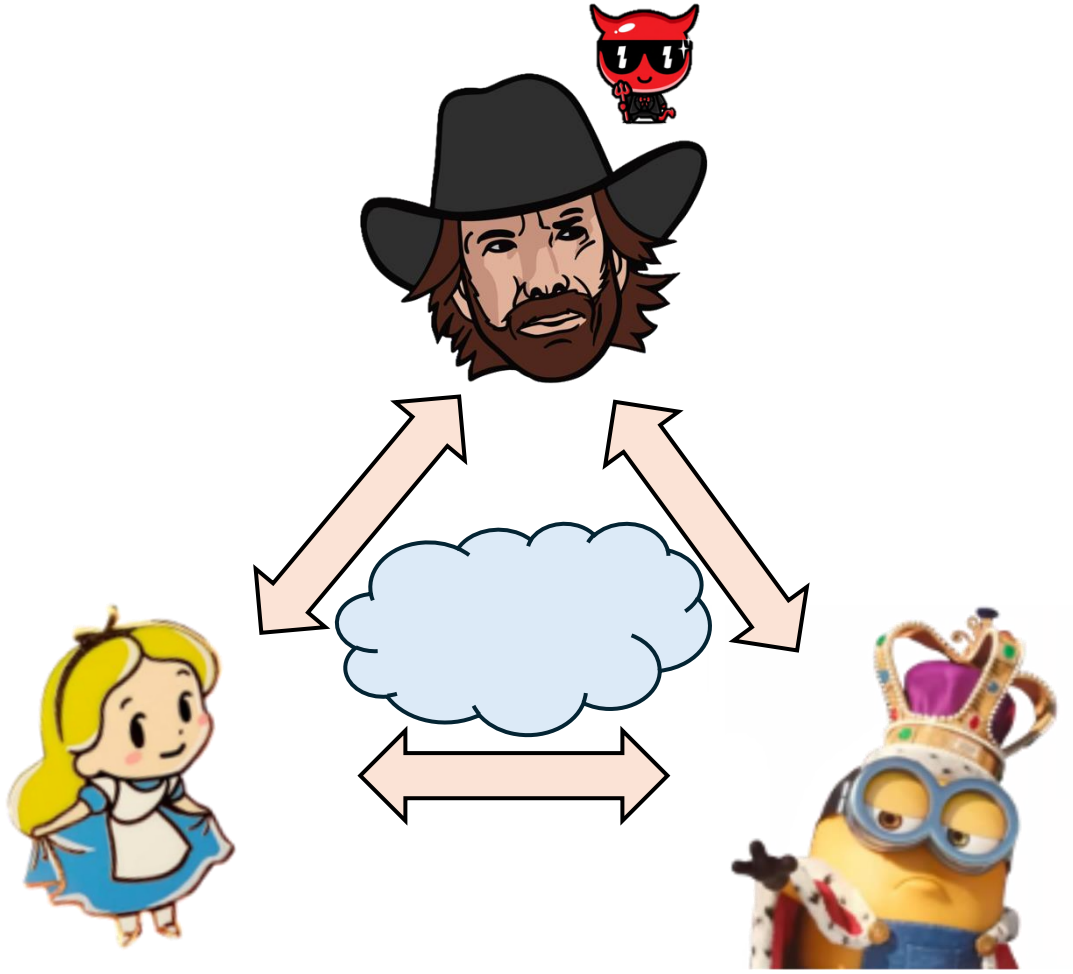
Heterogenous security



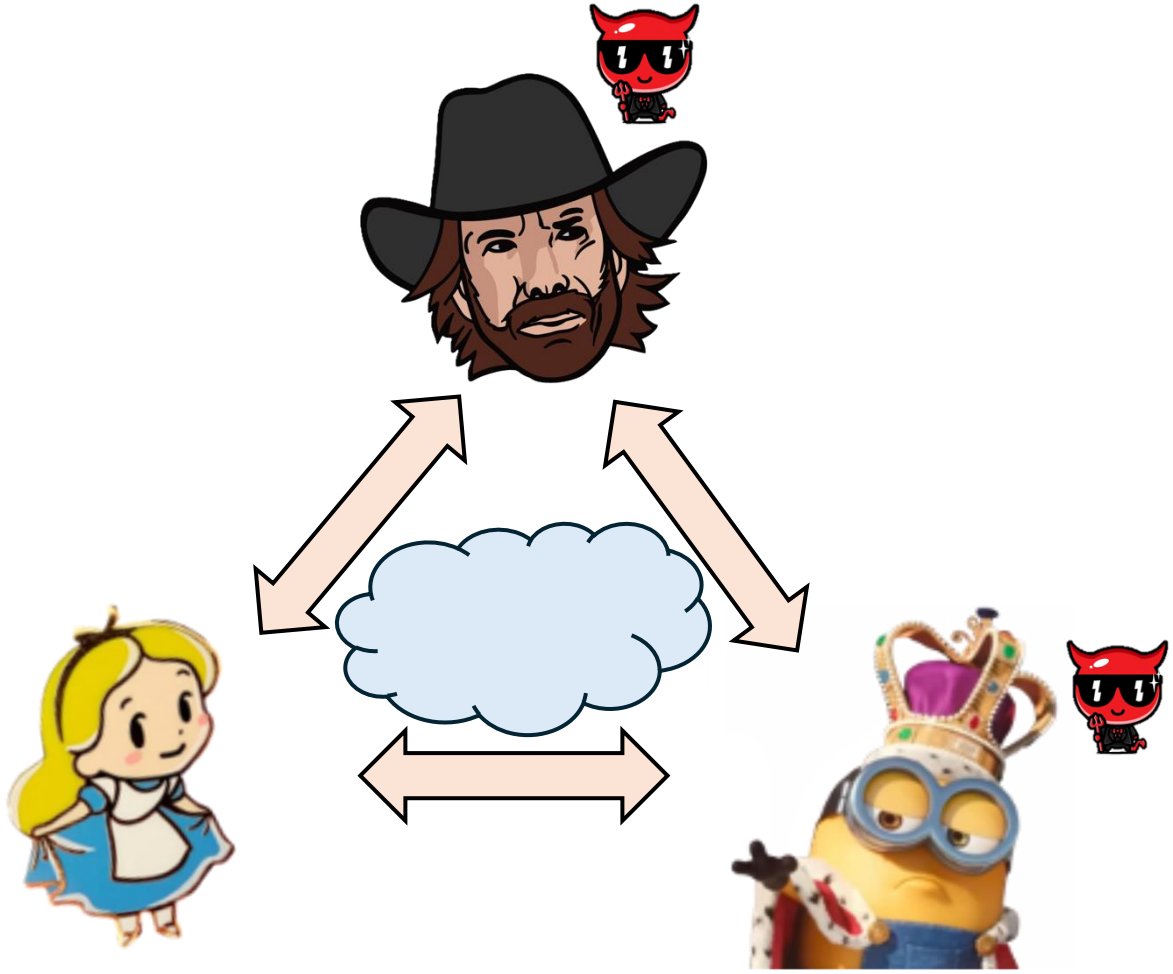
Heterogenous security



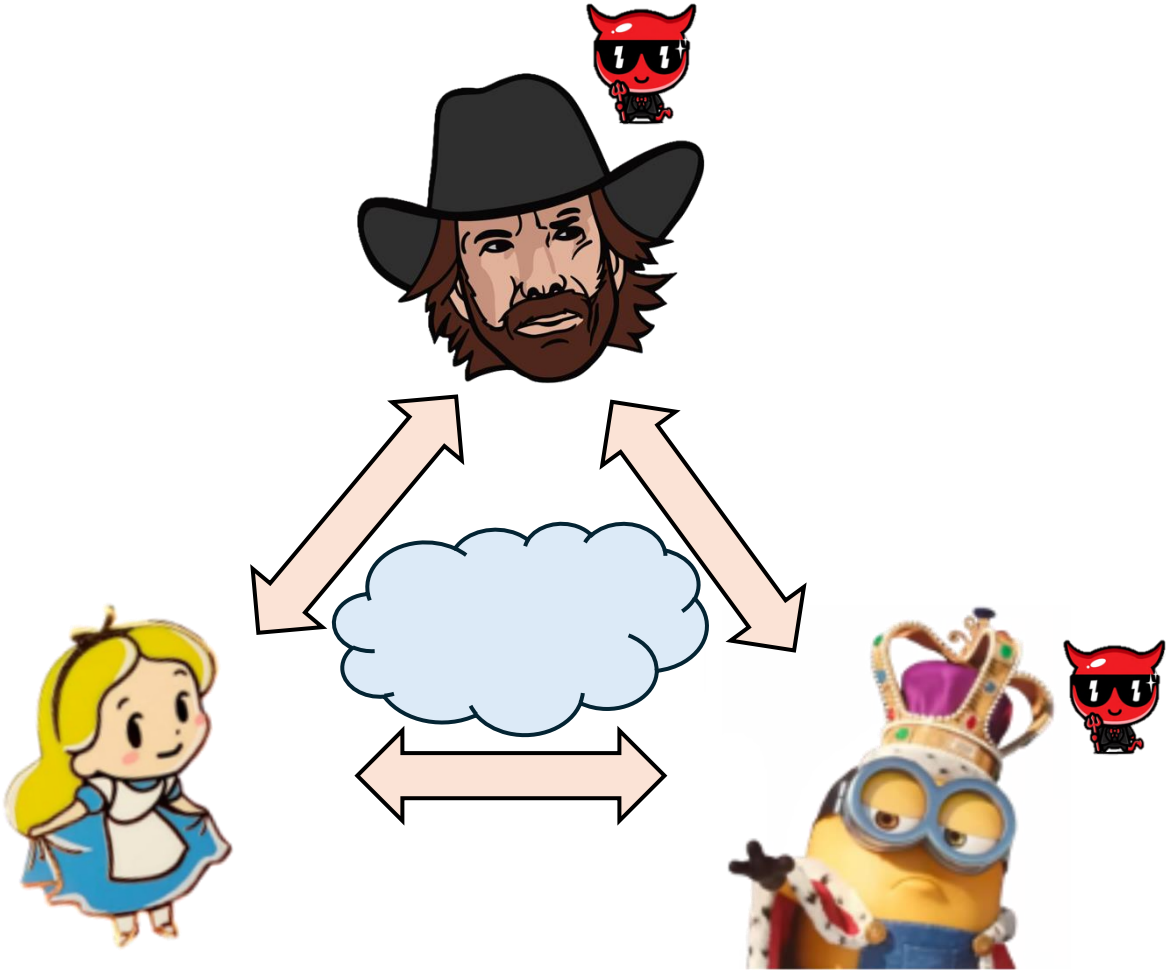
Heterogenous security



Heterogenous security

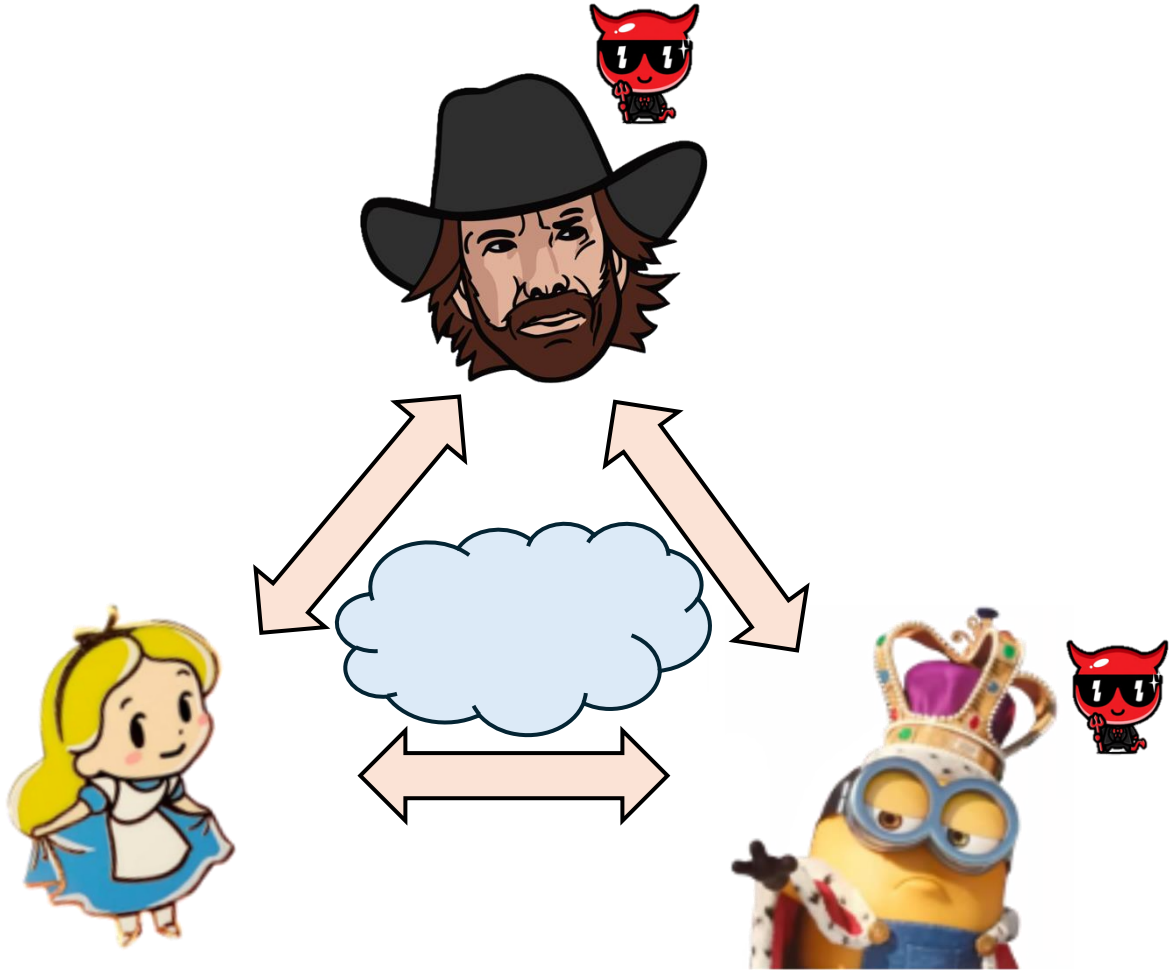


Heterogenous security





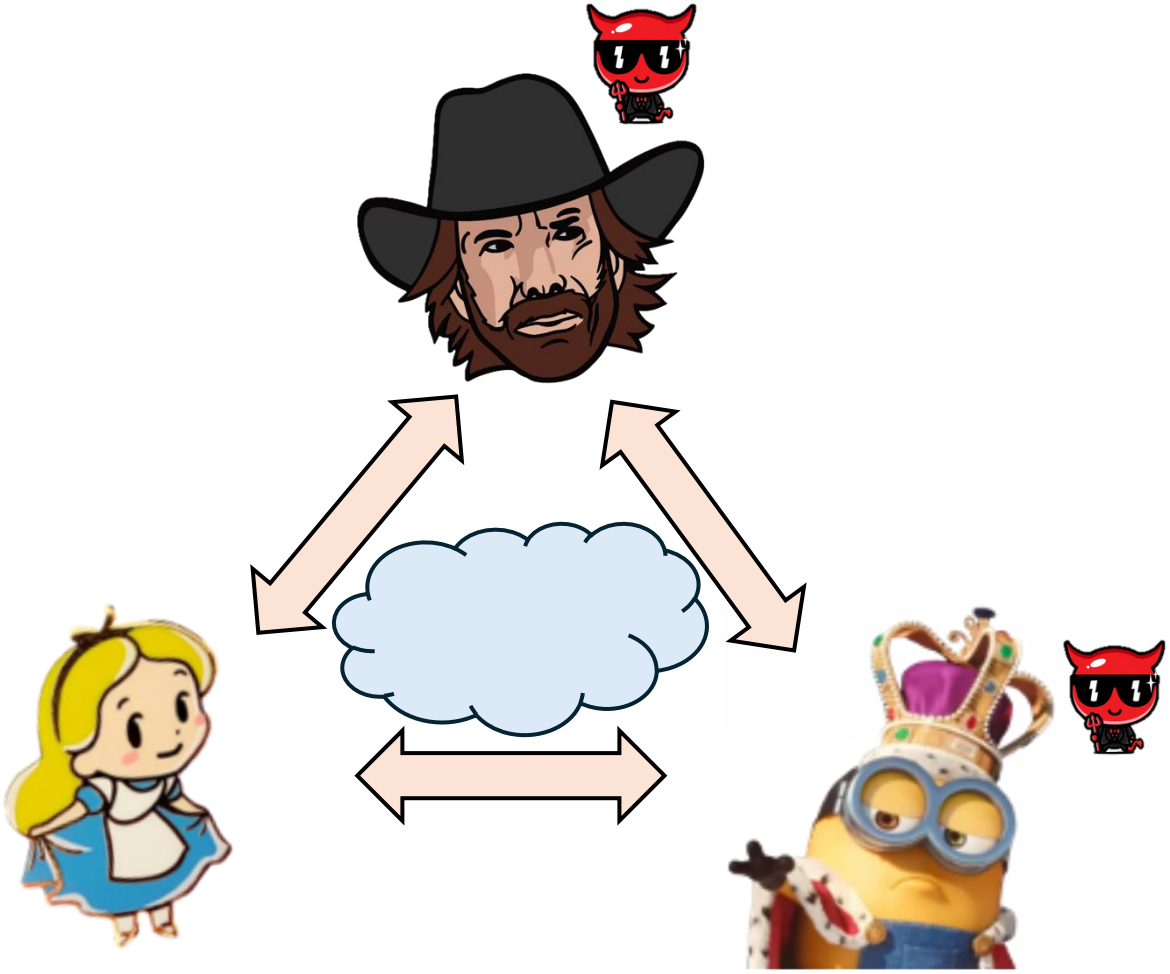
Heterogenous security





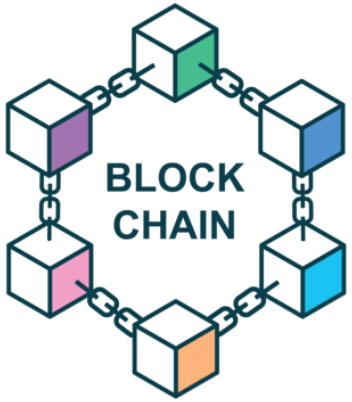


Heterogenous security

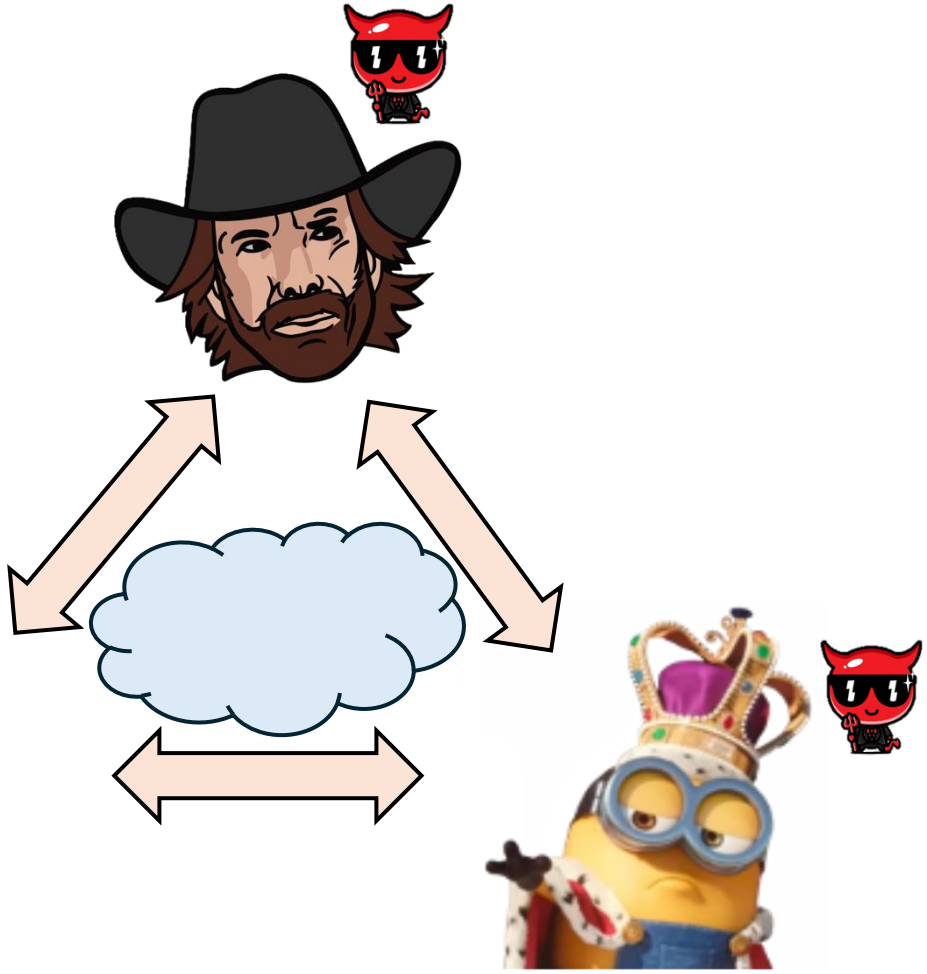






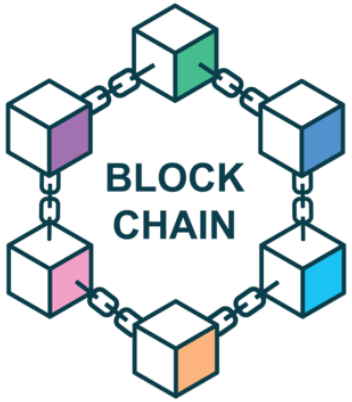


Heterogenous security









How to enforce security here?

Enforcing heterogenous security

Enforcing heterogenous security

Use **information flow labels**

Enforcing heterogenous security

Use **information flow labels**

Variable
v_1
v_2
v_3
…

Enforcing heterogenous security

Use **information flow labels**

Variable	Read Label
v_1	Alice, Bob
v_2	Bob
v_3	Chuck
…	…

Enforcing heterogenous security

Use **information flow labels**

Variable	Read Label	Write Label
v_1	Alice, Bob	Alice
v_2	Bob	Bob
v_3	Chuck	Chuck
…	…	…

Enforcing heterogenous security

Use **information flow labels**

Variable	Read Label	Write Label
v_1	Alice, Bob	Alice
v_2	Bob	Bob
v_3	Chuck	Chuck
…	…	…

Variable	Public?	Untrusted?
v_1	✓	✓
v_2	✗	✗
v_3	✗	✗
…	…	…



Variable	Public?	Untrusted?
v_1	✓	✗
v_2	✓	✓
v_3	✗	✗
…	…	…



Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✗
v_3	✓	✓
…	…	…



Enforcing heterogenous security

Use **information flow labels**

Variable	Read Label	Write Label
v_1	Alice, Bob	Alice
v_2	Bob	Bob
v_3	Chuck	Chuck
…	…	…

We require real–ideal emulation for all these scenarios!

Variable	Public?	Untrusted?
v_1	✓	✓
v_2	✗	✗
v_3	✗	✗
…	…	…



Variable	Public?	Untrusted?
v_1	✓	✗
v_2	✓	✓
v_3	✗	✗
…	…	…



Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✗
v_3	✓	✓
…	…	…



Enforcing heterogenous security

Use **information flow labels**

Variable	Read Label	Write Label
v_1	Alice, Bob	Alice
v_2	Bob	Bob
v_3	Chuck	Chuck
…	…	…

We require real–ideal emulation for all these scenarios!

$pc \vdash c$

T-Assign
$$\frac{\ell_x \sqsubseteq \ell_z \qquad \ell_y \sqsubseteq \ell_z \qquad pc \sqsubseteq \ell_z}{pc \vdash z := x \oplus y}$$

T-Endorse
$$\frac{\ell_y^{\rightarrow} = \ell_x^{\rightarrow} \qquad \ell_x^{\leftarrow} \succcurlyeq \ell_y^{\leftarrow} \qquad pc \sqsubseteq \ell_x \qquad \blacktriangledown \ell_y \sqcup pc}{pc \vdash x := \mathsf{endorse}(y)}$$

T-Declassify
$$\frac{\ell_y^{\leftarrow} = \ell_x^{\leftarrow} \qquad \ell_y^{\rightarrow} \succcurlyeq \ell_x^{\rightarrow} \qquad pc \sqsubseteq \ell_x \qquad \blacktriangledown \ell_y \sqcup pc}{pc \vdash x := \mathsf{declassify}(y)}$$

T-Output
$$\frac{pc \sqsubseteq \ell_x}{pc \vdash \mathsf{output}\ x}$$

T-If
$$\frac{\ell_x \sqcup pc \vdash p_1 \qquad \ell_x \sqcup pc \vdash p_2}{pc \vdash \mathsf{if}\ t(x)\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2}$$

T-While
$$\frac{\ell_x \sqcup pc \vdash p}{pc \vdash \mathsf{while}\ t(x)\ \mathsf{do}\ p}$$

$pc \vdash p$

T-Empty
$$\frac{}{pc \vdash \epsilon}$$

T-Seq
$$\frac{pc \vdash c \qquad pc \vdash p}{pc \vdash c\,;p}$$

Variable	Public?	Untrusted?
v_1	✓	✓
v_2	✗	✗
v_3	✗	✗
…	…	…



Variable	Public?	Untrusted?
v_1	✓	✗
v_2	✓	✓
v_3	✗	✗
…	…	…



Variable	Public?	Untrusted?
v_1	✗	✗
v_2	✗	✗
v_3	✓	✓
…	…	…



Evaluation

Evaluation



Evaluation



~ 5K LoC

Evaluation

~ 5K LoC

```
Theorem extract_step_commute_event c:
  CombinedInvariant c ->
  two_event (extract_2cfg c) = (comb_event c, comb_event c).
Proof.
  case: c.
  rewrite/CombinedInvariant/extract_2cfg.
  move=> ? [] // ? ? []. Forall_inv_toph.
  rewrite cmd_map_cons /==> *.
  do 2 cnstrct_eq_gl; cycle 1.
    { rewrite/pre_cmd_event/construct_simulator/sim_evnt cmd_map_sim_cons -/(pre_cmd_event _)/sim_active.
    case e: sim_active'=> //.
    move: e=> /sim_active'_false [] -> //. }
  pattern_eq_2 [⟨_|_⟩] [⟨_|_⟩]=> [|<-].
    { revert_hyp_pat (StateInv _).
    rewrite/StateInv=> - [] //. }
  move: (X in ⟨X|_⟩)=> ?.
  revert_hyp_pat [(CmbCmdInv _), (StateInv _)].
  rewrite/CmbCmdInv/TypeInv/StateInv.
  case=> + _. case. case: (pc_lbl _)=> - [] ? /= rwt=> [ |<- //].
  move=> iwt [] *.
  pattern_eq [LHS] with None=> [|->] ;[|pattern_eq [RHS] with None=> [|->]]=>//.
  1: case: (ideal_cmd _) iwt => // c //= wt.
  2: case: (real_cmd _) rwt => // c //= wt.
  1,2: case: c wt=> // ? /wf_output_inv /flows_from_untr /=.
  1,2: evar_spec_toph=>//.
  1,2: rewrite /⟨_|_⟩/is_trusted/is_untrusted=> -> //.
Qed.
```

Summing it up

Summing it up

- With AI writing most of our code and even proofs, getting the right system specifications is crucial.

Summing it up

- With AI writing most of our code and even proofs, getting the right system specifications is crucial.
- Our work shows how the cryptographic real–ideal paradigm offers a simple and unified approach to language-based security.

Summing it up

- With AI writing most of our code and even proofs, getting the right system specifications is crucial.
- Our work shows how the cryptographic real–ideal paradigm offers a simple and unified approach to language-based security.
- Less understood semantic security properties such as robust declassification and nonmalleable information flow are given intuitive definitions using the real–ideal paradigm.

Summing it up

- With AI writing most of our code and even proofs, getting the right system specifications is crucial.
- Our work shows how the cryptographic real–ideal paradigm offers a simple and unified approach to language-based security.
- Less understood semantic security properties such as robust declassification and nonmalleable information flow are given intuitive definitions using the real–ideal paradigm.
- Our work opens up a lot of directions for future research. Hyperproperties? Composition theorems? etc.

Demystifying information flow downgrading using the real–ideal paradigm

Suraaj Kanniwadi, Andrew Myers

in submission to POPL ‘27

Being the first PLDG speaker



As you know, you go to war with the army you have, not the army you might want or wish to have at a later time.

(Donald Rumsfeld)