Formal Methods and Real-World Cryptographic Protocols:

Advancing the Verification and Testing Frontiers for Specifications and Implementations

Lucca Hirschi · *Habilitation à Diriger des Recherches (HDR) defense*
2nd November 2026

RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

Inria

Loria

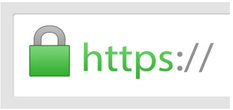
UNIVERSITÉ DE LORRAINE

Formal Methods and Real-World Cryptographic Protocols



Untitled – Calder, 1942

Cryptographic Protocols: Invisible and yet everywhere

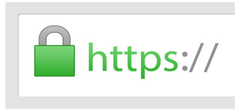


TLS



WPA3

Cryptographic Protocols: Invisible and yet everywhere



TLS



WPA3



5G AKA



PQXDH / X3DH



EMV



Car key



Electronic identification (e.g. PACE)

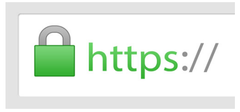


Belenios



OPC UA

Cryptographic Protocols: Invisible and yet everywhere



TLS



WPA3



5G AKA



PQXDH / X3DH



EMV



Car key



Electronic identification
(e.g. PACE)



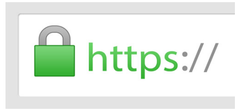
Belenios



OPC UA

concurrent programs relying on cryptography to
secure communications against an attacker/threat model 👺

Cryptographic Protocols: Invisible and yet everywhere



TLS



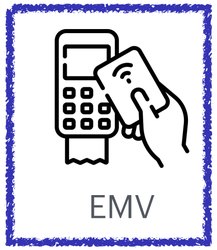
WPA3



5G AKA



PQXDH / X3DH



EMV



Car key



Electronic identification
(e.g. PACE)



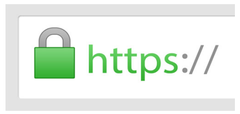
Belenios



OPC UA

concurrent programs relying on cryptography to
secure communications against an attacker/threat model 👺

Cryptographic Protocols: Invisible and yet everywhere



TLS



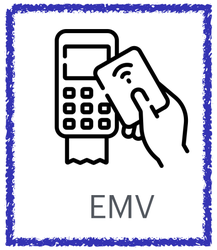
WPA3



5G AKA



PQXDH / X3DH



EMV



Car key



Electronic identification
(e.g. PACE)

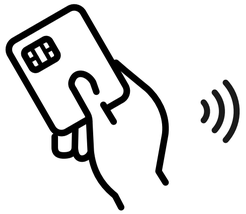


Belenios

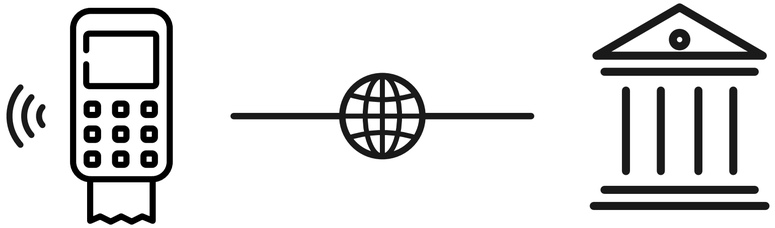


OPC UA

concurrent programs relying on cryptography to
secure communications against an attacker/threat model



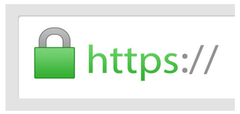
Credit Card



Shop Terminal

Bank

Cryptographic Protocols: Invisible and yet everywhere



TLS



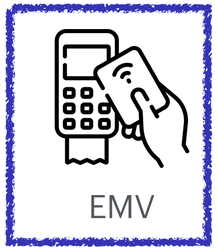
WPA3



5G AKA



PQXDH / X3DH



EMV



Car key



Electronic identification (e.g. PACE)

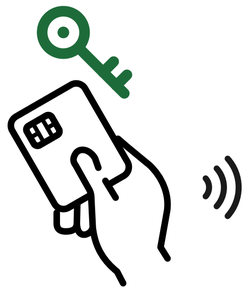


Belenios

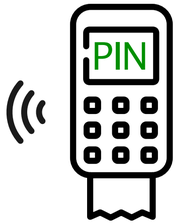


OPC UA

concurrent programs relying on cryptography to
secure communications against an attacker/threat model



Credit Card



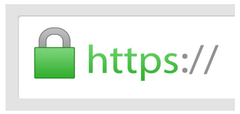
Shop Terminal



Bank

- Key : authenticates card + integrity
- PIN code authenticates cardholder (>50€)

Cryptographic Protocols: Invisible and yet everywhere



TLS



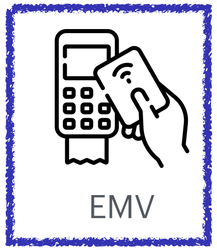
WPA3



5G AKA



PQXDH / X3DH



EMV



Car key



Electronic identification (e.g. PACE)



Belenios

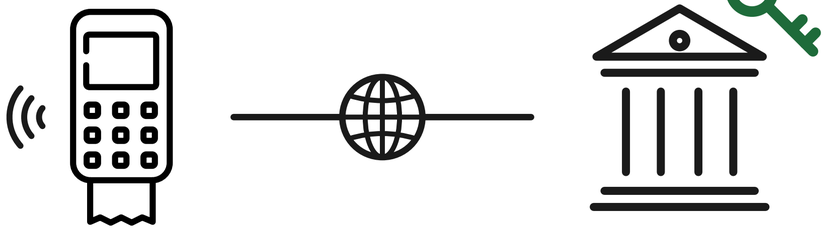


OPC UA

concurrent programs relying on cryptography to
secure communications against an attacker/threat model 😈



Smartphone

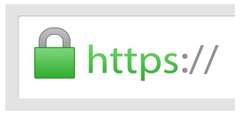


Shop Terminal

Bank

- Key 🔑: authenticates card + integrity
- PIN code authenticates cardholder (>50€)
- Phone directly authenticates cardholder

Cryptographic Protocols: Invisible and yet everywhere



TLS



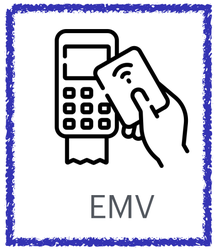
WPA3



5G AKA



PQXDH / X3DH



EMV



Car key



Electronic identification (e.g. PACE)

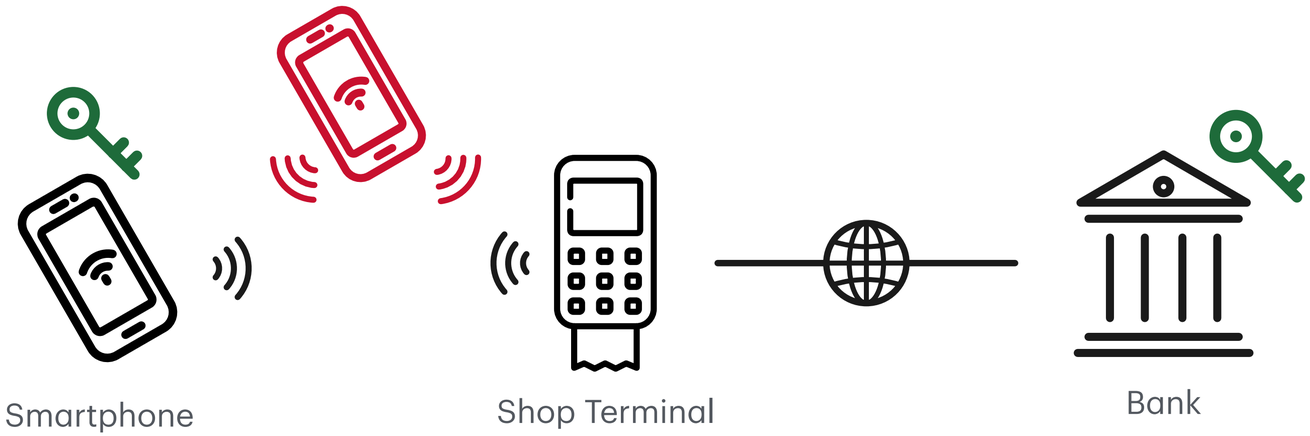


Belenios



OPC UA

concurrent programs relying on cryptography to
secure communications against an attacker/threat model



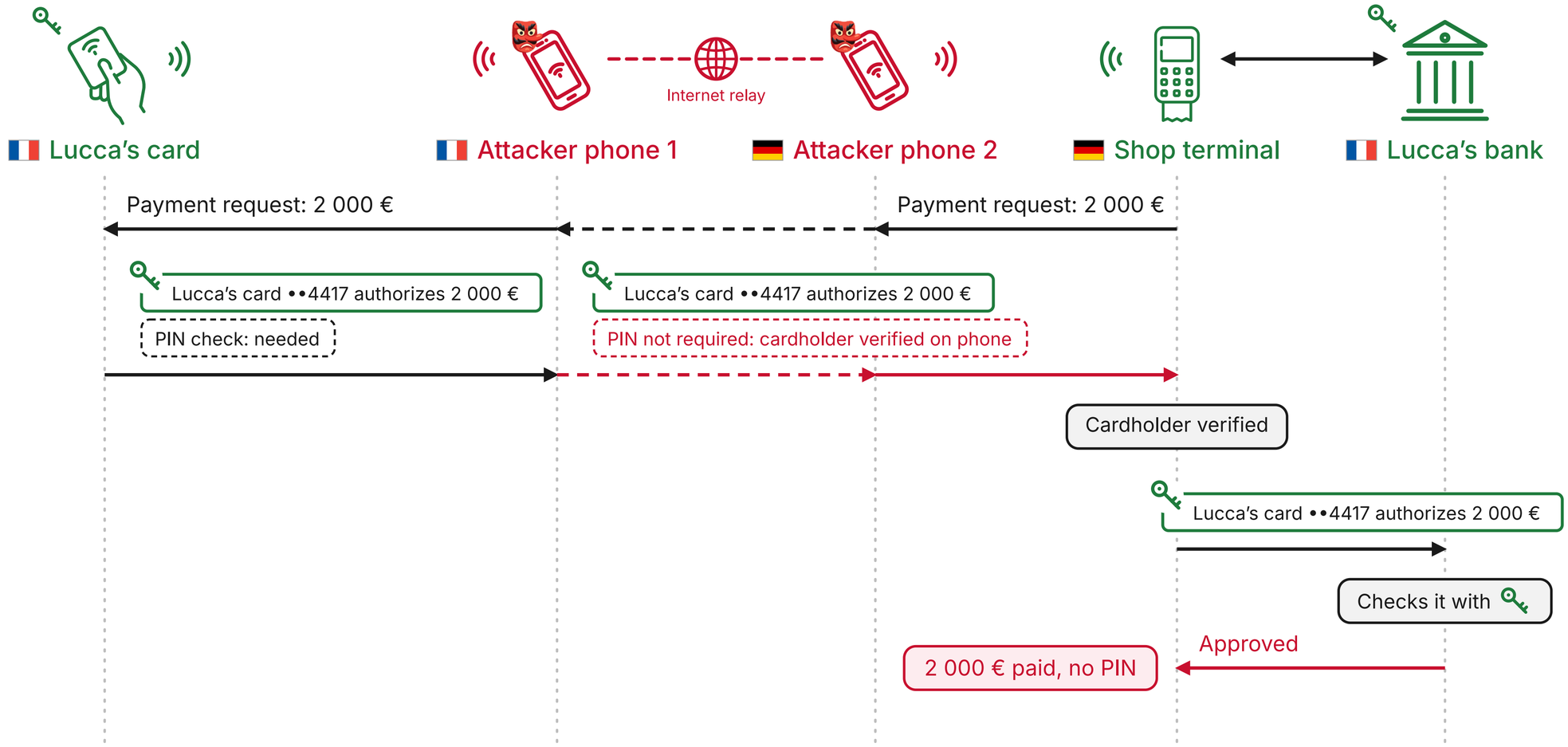
Smartphone

Shop Terminal

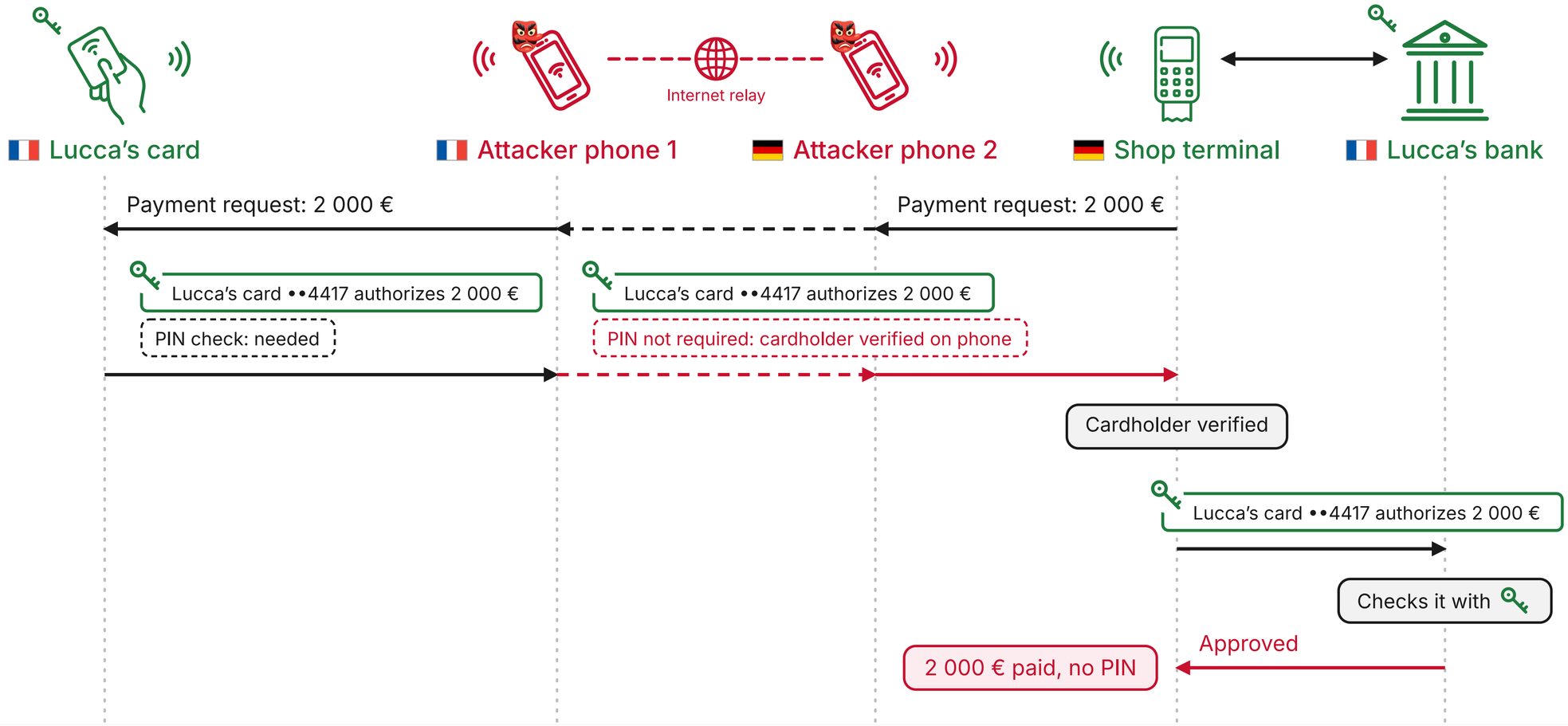
Bank

- Key : authenticates card + integrity
- PIN code authenticates cardholder (>50€)
- Phone directly authenticates cardholder

Example of logical attack against EMV [Basin et al. SP'21]



Example of logical attack against EMV [Basin et al. SP'21]

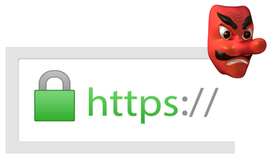


😈 pays 2 000€ with Lucca's money but knows neither 🔑 nor PIN

☞ Logical attack: 😈 never breaks cryptography but can intercept, replay, reorder, modify and inject messages



EMV: 40k LoC



OpenSSL: 200k LoC



WPA3: 50k LoC



5G AKA: 35k LoC



libsignal: 45k LoC



EMV: 40k LoC



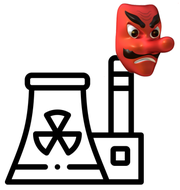
Car key: 30k LoC



PACE: 45k LoC

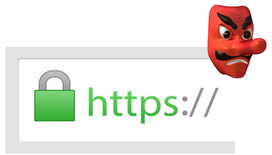


Belenios: 16k LoC



OPC UA: 120k LoC

Roger Needham in 1996:
« Security protocols are three-line programs that people still manage to get wrong.»



OpenSSL: 200k LoC



WPA3: 50k LoC



5G AKA: 35k LoC



libsignal: 45k LoC



EMV: 40k LoC



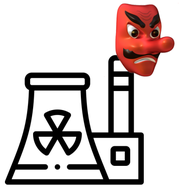
Car key: 30k LoC



PACE: 45k LoC



Belenios: 16k LoC



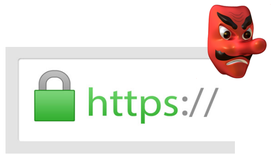
OPC UA: 120k LoC

Roger Needham in 1996:
« Security protocols are three-line programs that people still manage to get wrong.»

Designers'/Dev.' task is overly complex

- Design a whole distributed system
- Aim for security (not mere safety)
- Large, complex, yet efficient, legacy, …





OpenSSL: 200k LoC



WPA3: 50k LoC



5G AKA: 35k LoC



libsignal: 45k LoC



EMV: 40k LoC



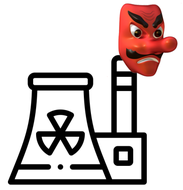
Car key: 30k LoC



PACE: 45k LoC



Belenios: 16k LoC



OPC UA: 120k LoC

Roger Needham in 1996:
« Security protocols are three-line programs that people still manage to get wrong.»



has some advantages:

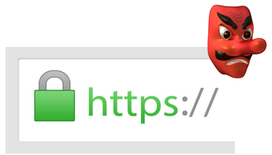
- One needs one « weakest link »
- Can participate, access network, compromise, …
- Many possible strategies

vs.

Designers'/Dev.' task is overly complex

- Design a whole distributed system
- Aim for security (not mere safety)
- Large, complex, yet efficient, legacy, …





OpenSSL: 200k LoC



WPA3: 50k LoC



5G AKA: 35k LoC



libsignal: 45k LoC



EMV: 40k LoC



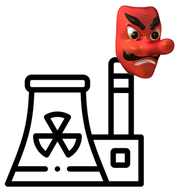
Car key: 30k LoC



PACE: 45k LoC



Belenios: 16k LoC



OPC UA: 120k LoC

Roger Needham in 1996:
« Security protocols are three-line programs that people still manage to get wrong.»



- has some advantages:
- One needs one « weakest link »
- Can participate, access network, compromise, …
- Many possible strategies

vs.

Designers'/Dev.' task is overly complex

- Design a whole distributed system
- Aim for security (not mere safety)
- Large, complex, yet efficient, legacy, …





How can we obtain higher assurance for the cryptographic protocols that underpin our digital world?

Scope

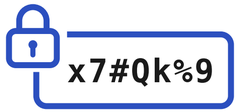
Crypto Primitives



MAC

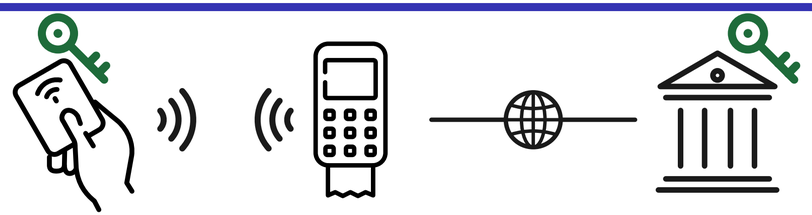


Digital signature



Encryption

Protocol Design



Standard, RFC, Specification

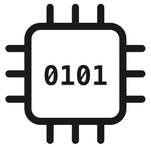
Implementation







Hardware







1. **Verification through Formal Methods**

Formal Analysis in Dolev-Yao Models



2. **Testing through DY Fuzzing**

Dolev-Yao Model-Guided Fuzzing



Scope

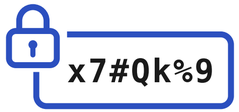
Crypto Primitives



MAC

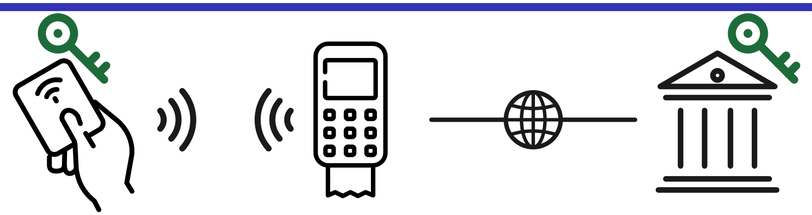


Digital signature



Encryption

Protocol Design



Standard, RFC, Specification

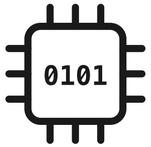
Implementation







Hardware







1. **Verification through Formal Methods**

Formal Analysis in Dolev-Yao Models



2. **Testing through DY Fuzzing**

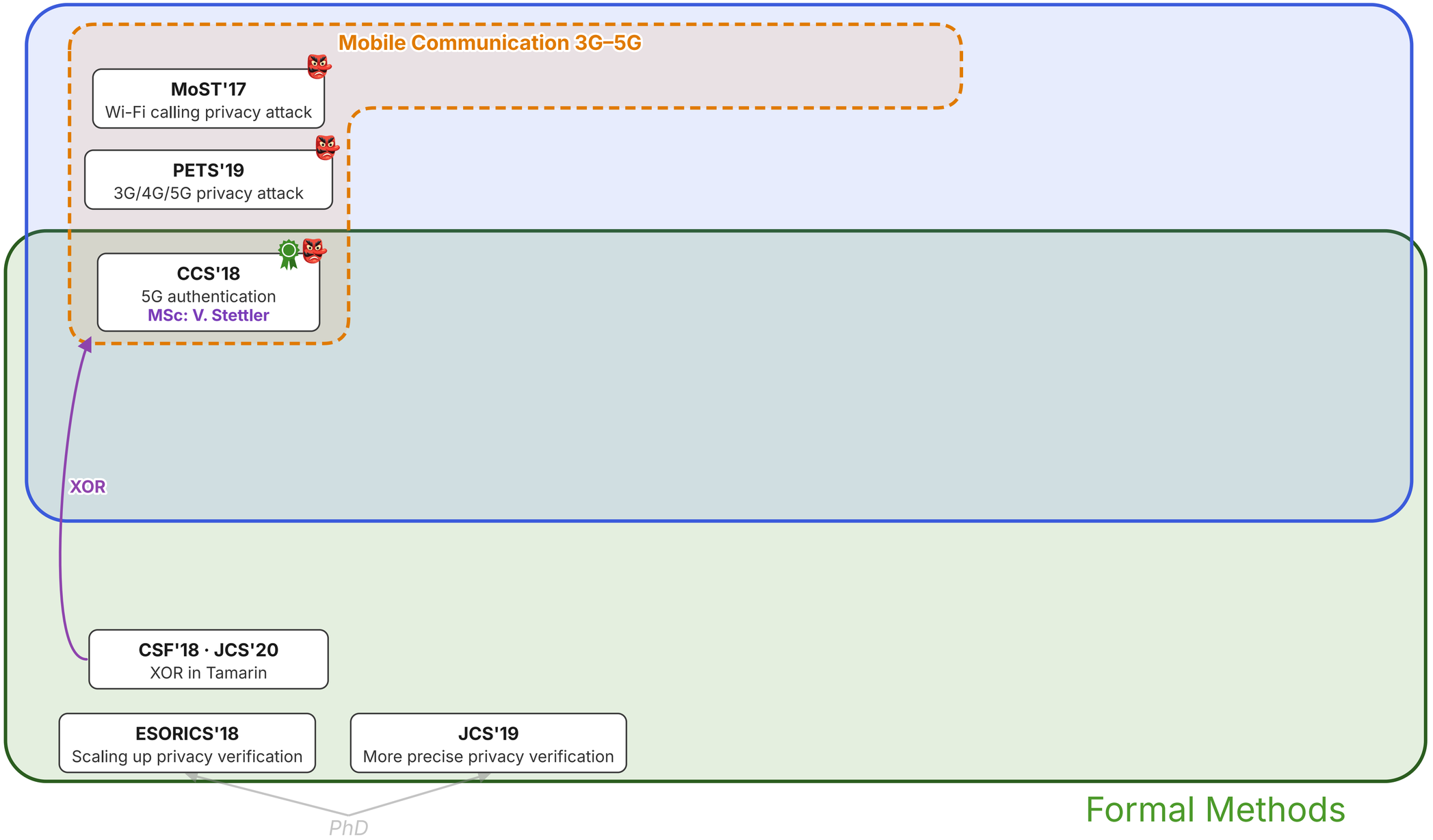
Dolev-Yao Model-Guided Fuzzing



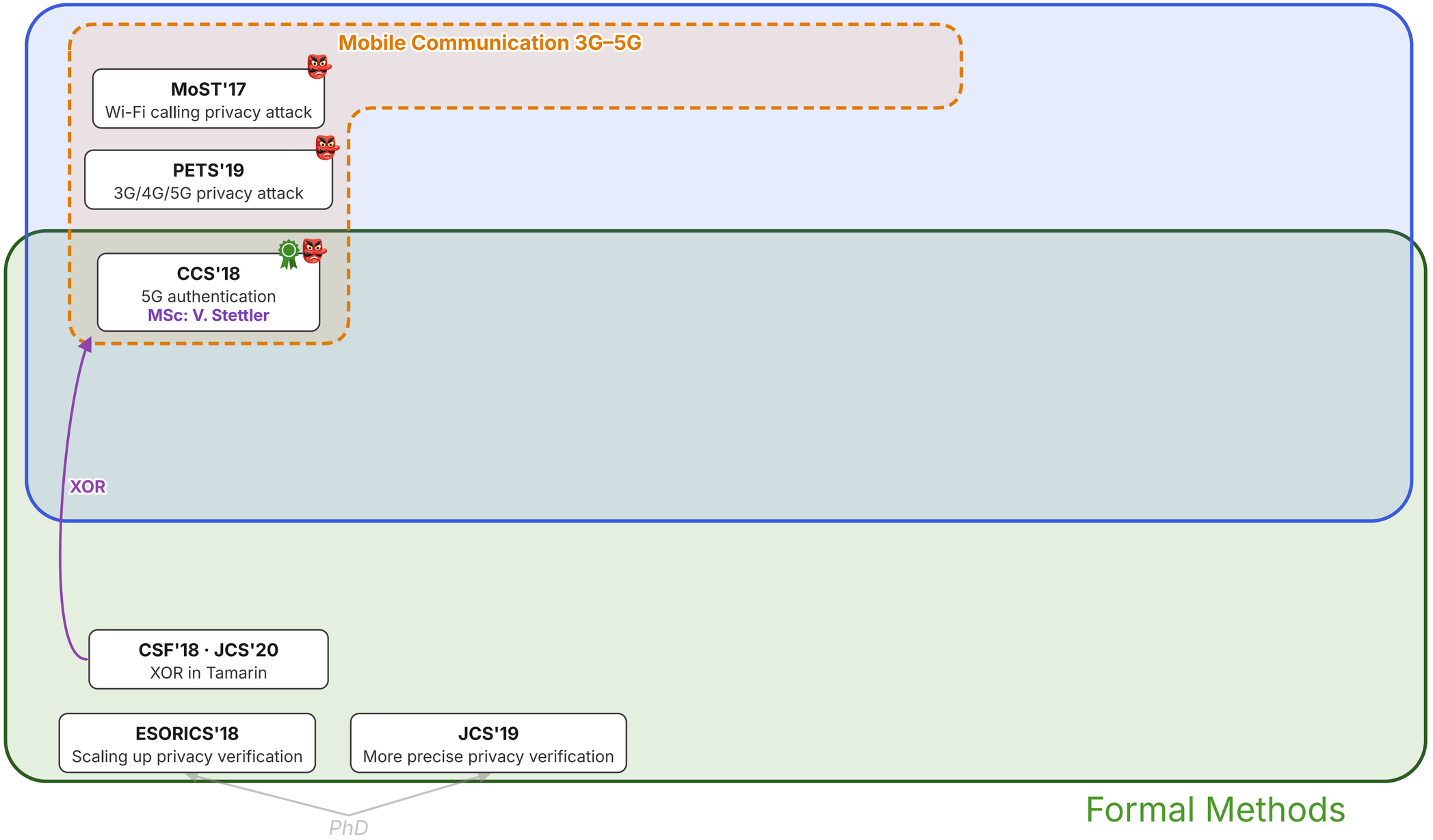
Before that: overall research journey — selection of works


CSF'18 · JCS'20
XOR in Tamarin
ESORICS'18
Scaling up privacy verification
JCS'19
More precise privacy verification
PhD
Formal Methods

Real-World Protocol Analysis



Real-World Protocol Analysis

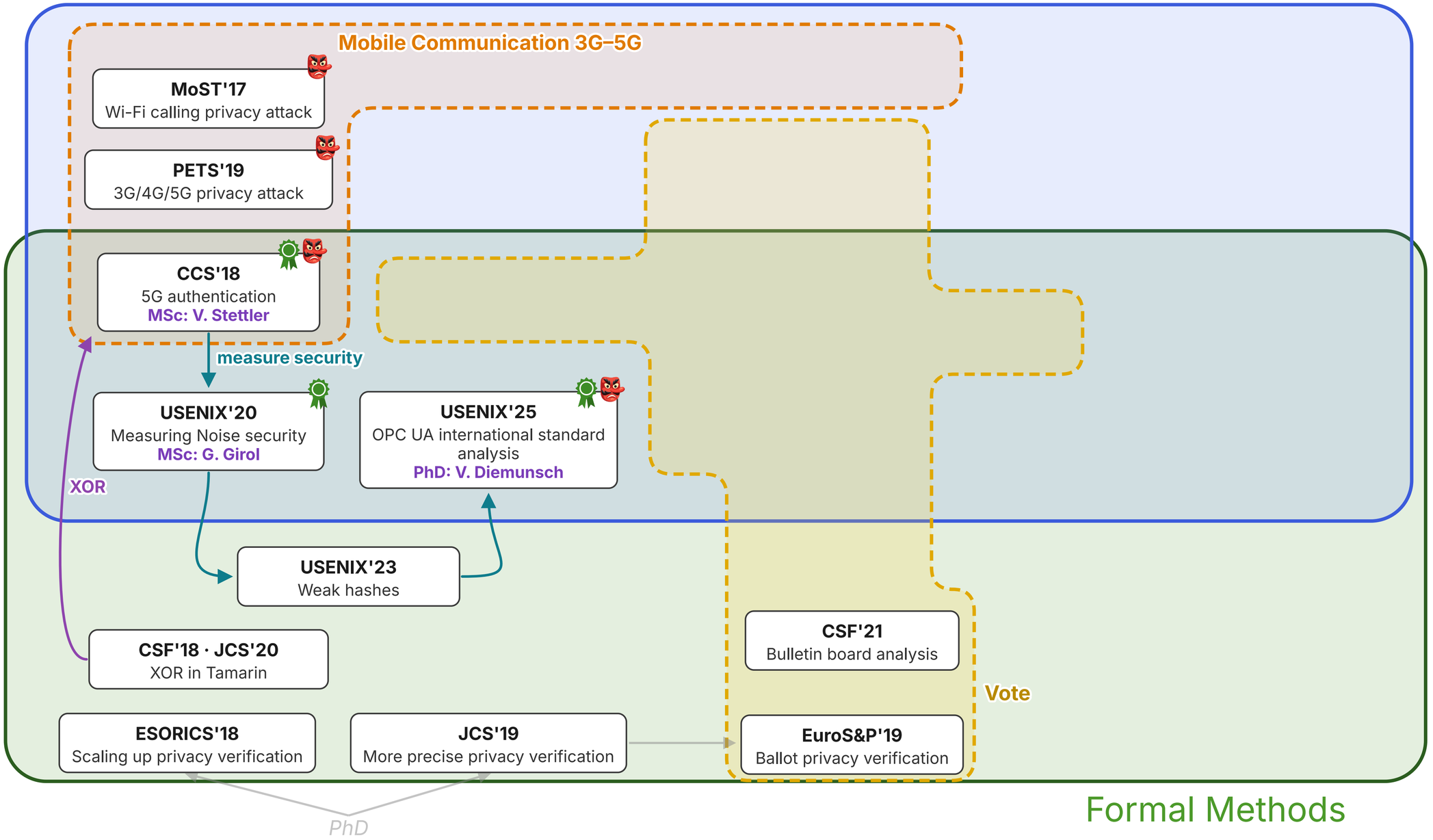


Attacks on real-world protocols, acknowledged by vendors

Large-scale formal analysis of a standard

PhD MSc With students we supervised

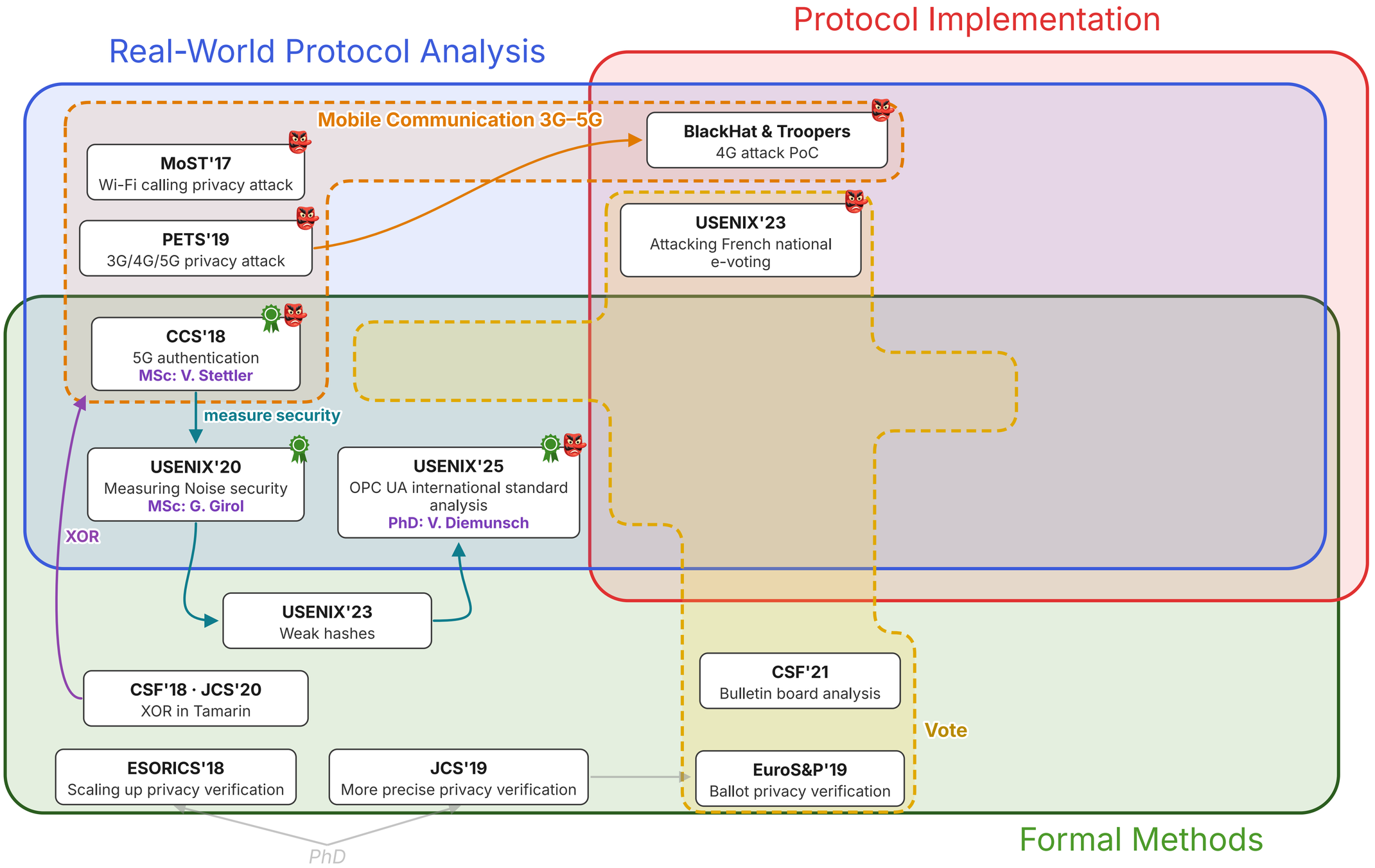
Real-World Protocol Analysis

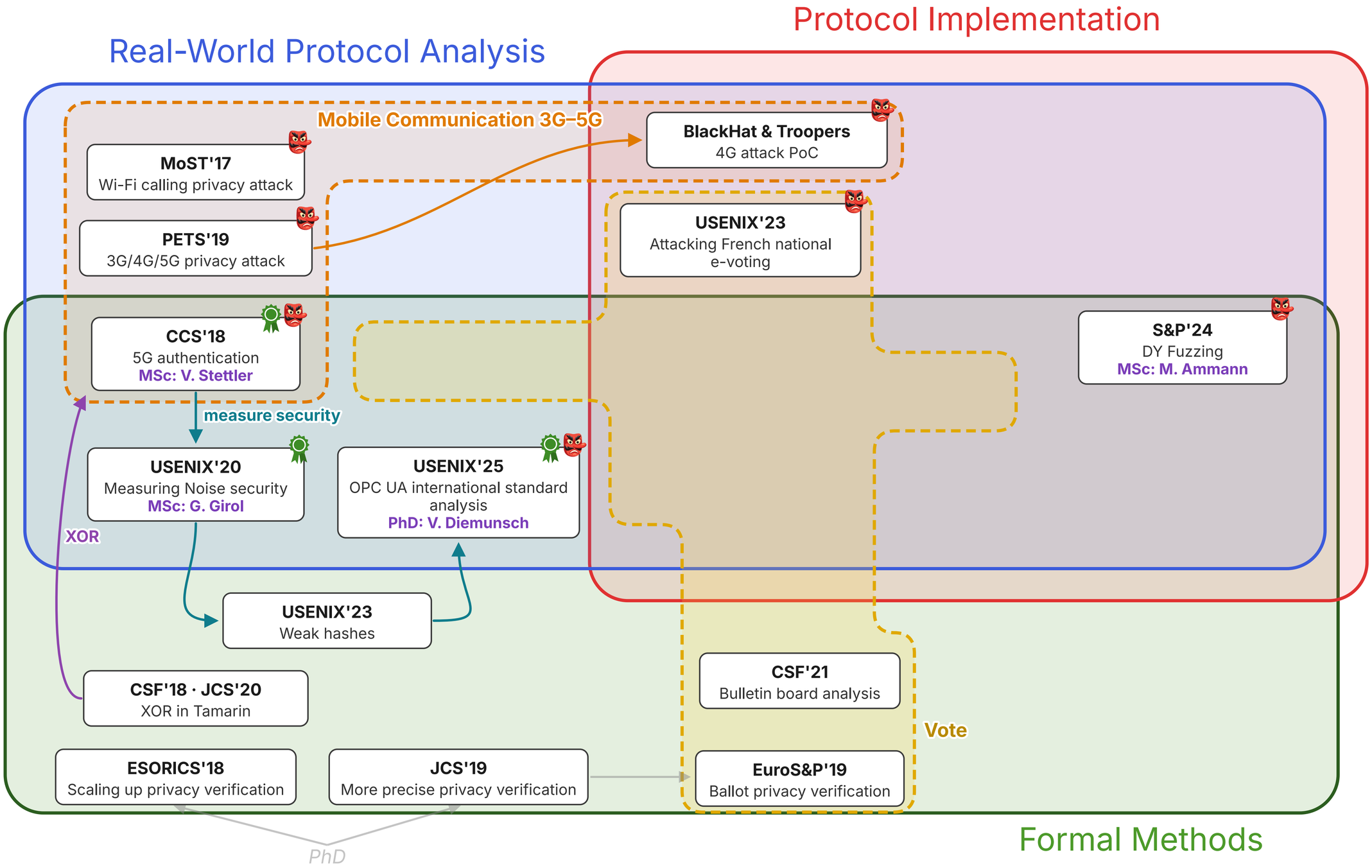


Attacks on real-world protocols, acknowledged by vendors

Large-scale formal analysis of a standard

PhD MSc With students we supervised

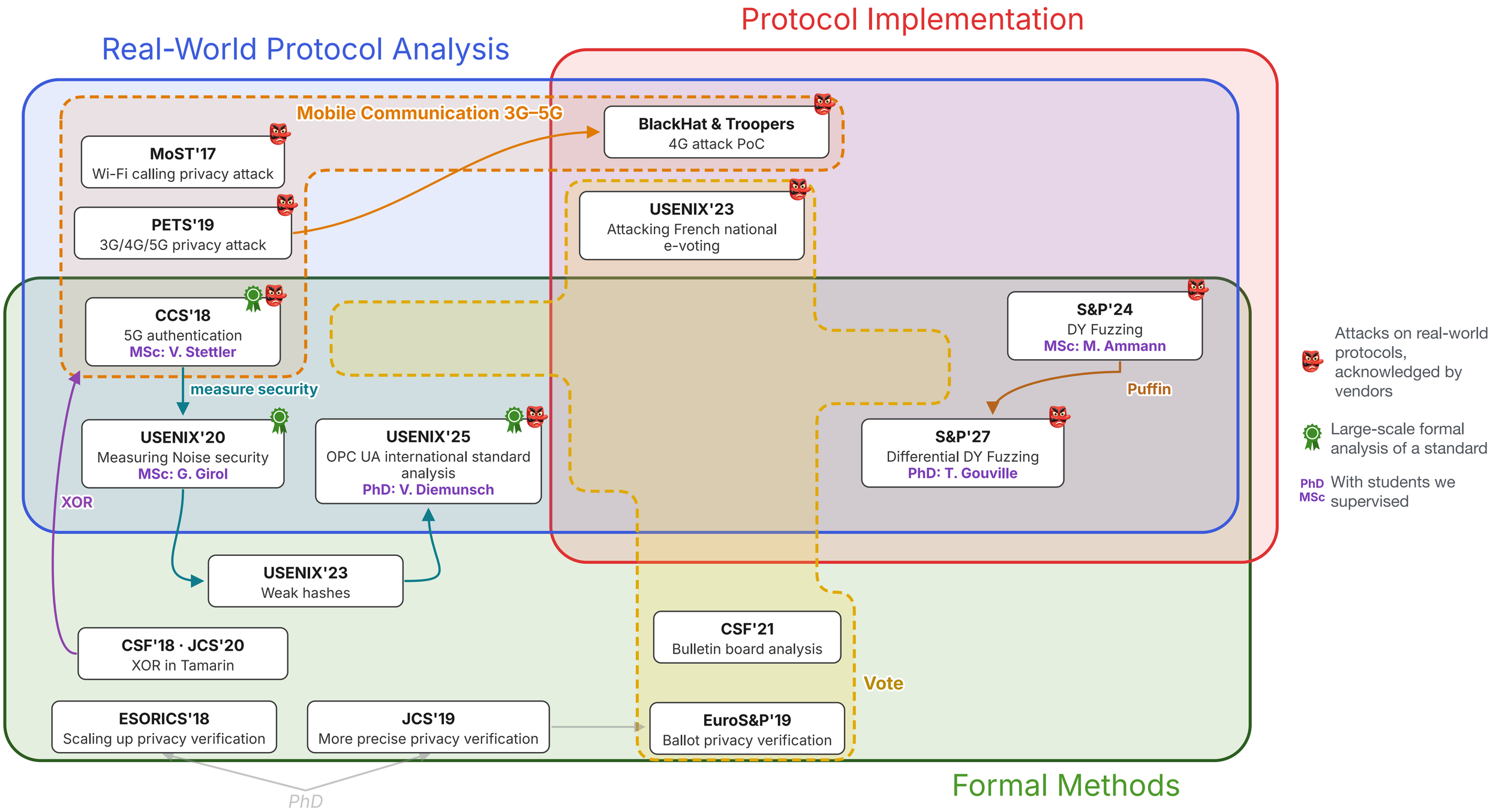

Real-World Protocol Analysis
Protocol Implementation
Formal Methods
Mobile Communication 3G–5G
MoST'17
Wi-Fi calling privacy attack
PETS'19
3G/4G/5G privacy attack
BlackHat & Troopers
4G attack PoC
USENIX'23
Attacking French national e-voting
CCS'18
5G authentication
MSc: V. Stettler
measure security
USENIX'20
Measuring Noise security
MSc: G. Girol
USENIX'25
OPC UA international standard analysis
PhD: V. Diemunsch
XOR
USENIX'23
Weak hashes
CSF'18 · JCS'20
XOR in Tamarin
CSF'21
Bulletin board analysis
Vote
ESORICS'18
Scaling up privacy verification
JCS'19
More precise privacy verification
EuroS&P'19
Ballot privacy verification
PhD
Attacks on real-world protocols, acknowledged by vendors
Large-scale formal analysis of a standard
PhD MSc With students we supervised

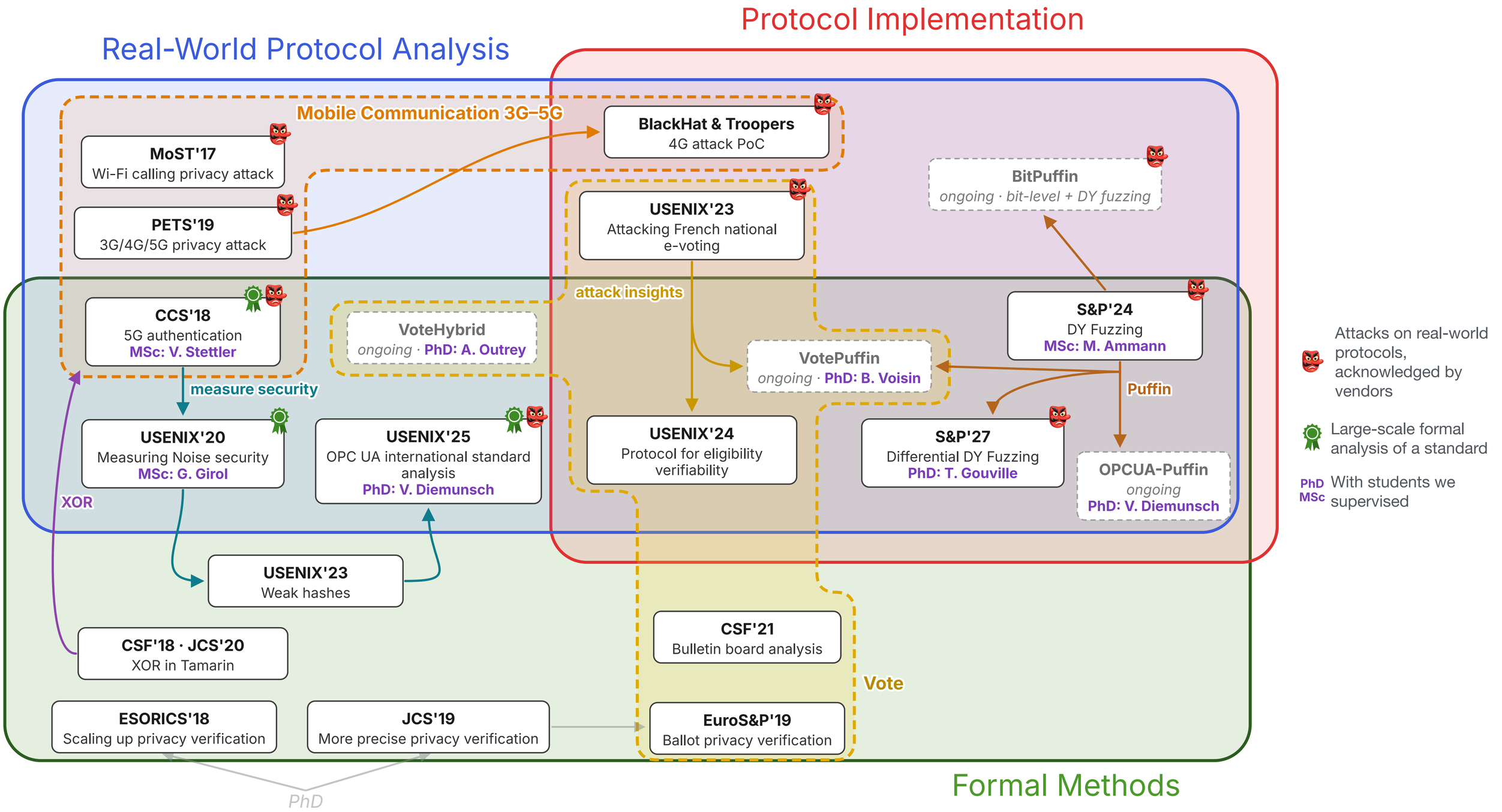


Attacks on real-world protocols, acknowledged by vendors

Large-scale formal analysis of a standard

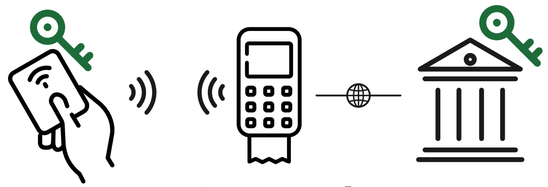
PhD MSc With students we supervised


Real-World Protocol Analysis
Protocol Implementation
Formal Methods
Mobile Communication 3G–5G
MoST'17
Wi-Fi calling privacy attack
PETS'19
3G/4G/5G privacy attack
BlackHat & Troopers
4G attack PoC
USENIX'23
Attacking French national e-voting
CCS'18
5G authentication
MSc: V. Stettler
measure security
S&P'24
DY Fuzzing
MSc: M. Ammann
Puffin
USENIX'20
Measuring Noise security
MSc: G. Girol
USENIX'25
OPC UA international standard analysis
PhD: V. Diemunsch
S&P'27
Differential DY Fuzzing
PhD: T. Gouville
XOR
USENIX'23
Weak hashes
CSF'18 · JCS'20
XOR in Tamarin
CSF'21
Bulletin board analysis
Vote
ESORICS'18
Scaling up privacy verification
JCS'19
More precise privacy verification
EuroS&P'19
Ballot privacy verification
PhD
Attacks on real-world protocols, acknowledged by vendors
Large-scale formal analysis of a standard
PhD MSc With students we supervised


Real-World Protocol Analysis
Protocol Implementation
Mobile Communication 3G–5G
MoST'17
Wi-Fi calling privacy attack
PETS'19
3G/4G/5G privacy attack
BlackHat & Troopers
4G attack PoC
USENIX'23
Attacking French national e-voting
BitPuffin
ongoing · bit-level + DY fuzzing
attack insights
CCS'18
5G authentication
MSc: V. Stettler
VoteHybrid
ongoing · PhD: A. Outrey
VotePuffin
ongoing · PhD: B. Voisin
S&P'24
DY Fuzzing
MSc: M. Ammann
Puffin
measure security
USENIX'20
Measuring Noise security
MSc: G. Girol
USENIX'25
OPC UA international standard analysis
PhD: V. Diemunsch
USENIX'24
Protocol for eligibility verifiability
S&P'27
Differential DY Fuzzing
PhD: T. Gouville
OPCUA-Puffin
ongoing
PhD: V. Diemunsch
XOR
USENIX'23
Weak hashes
CSF'18 · JCS'20
XOR in Tamarin
CSF'21
Bulletin board analysis
Vote
ESORICS'18
Scaling up privacy verification
JCS'19
More precise privacy verification
EuroS&P'19
Ballot privacy verification
PhD
Formal Methods
Attacks on real-world protocols, acknowledged by vendors
Large-scale formal analysis of a standard
PhD MSc With students we supervised

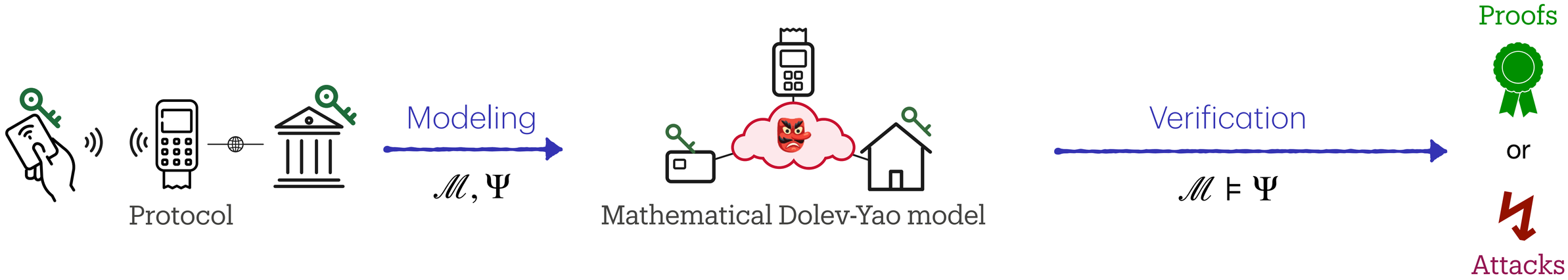
Part I:  Formal Methods for Cryptographic Protocol Security Verification

Formal verification in Dolev-Yao (DY) models

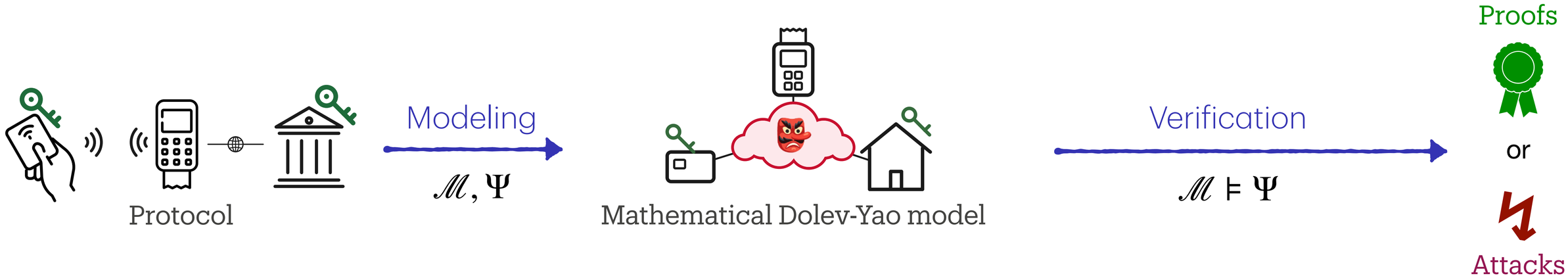


Protocol

Formal verification in Dolev-Yao (DY) models

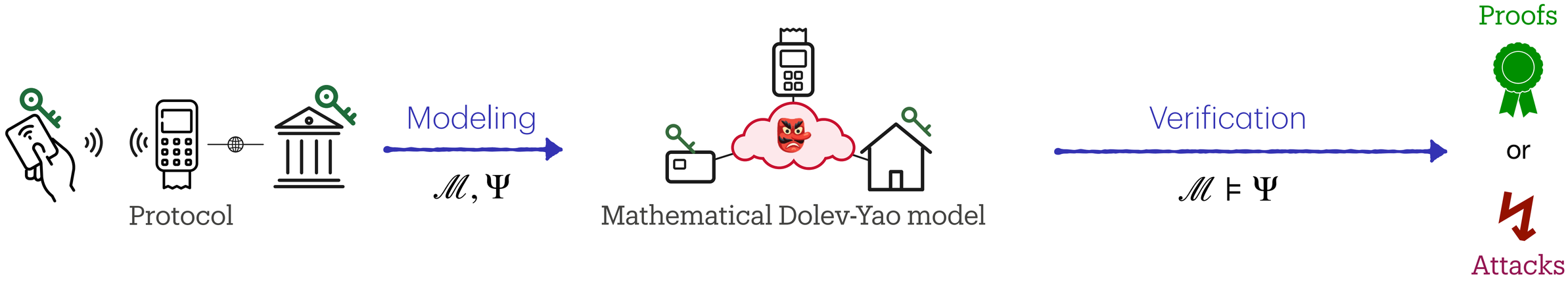


Formal verification in Dolev-Yao (DY) models



- Cryptography ↠ black-box functionalities modeled in a ***term algebra:***

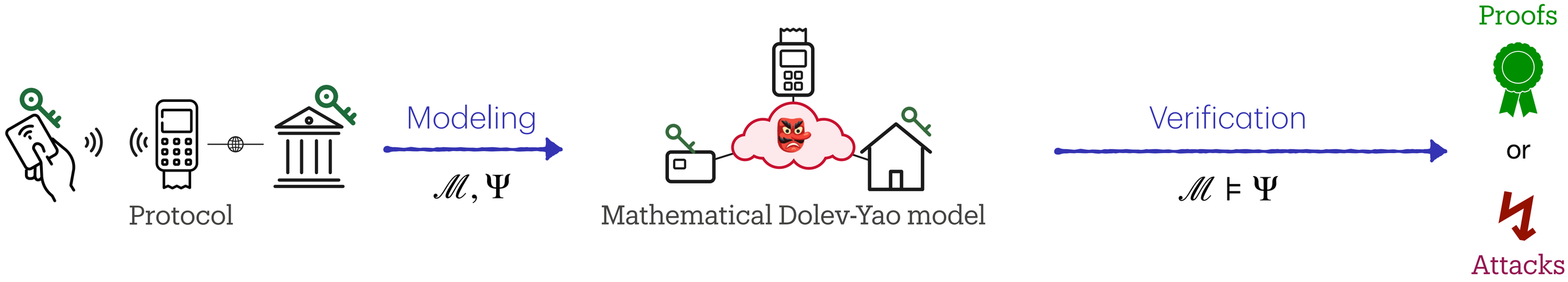
Formal verification in Dolev-Yao (DY) models



- Cryptography ↠ black-box functionalities modeled in a ***term algebra:***
  - function symbols $\mathscr{F}$ and names $\mathscr{N}$
  - formal terms $\mathscr{T}(\mathscr{F}, \mathscr{N})$

$\mathscr{M}$ { MAC(· , ·), true ∈ $\mathscr{F}$

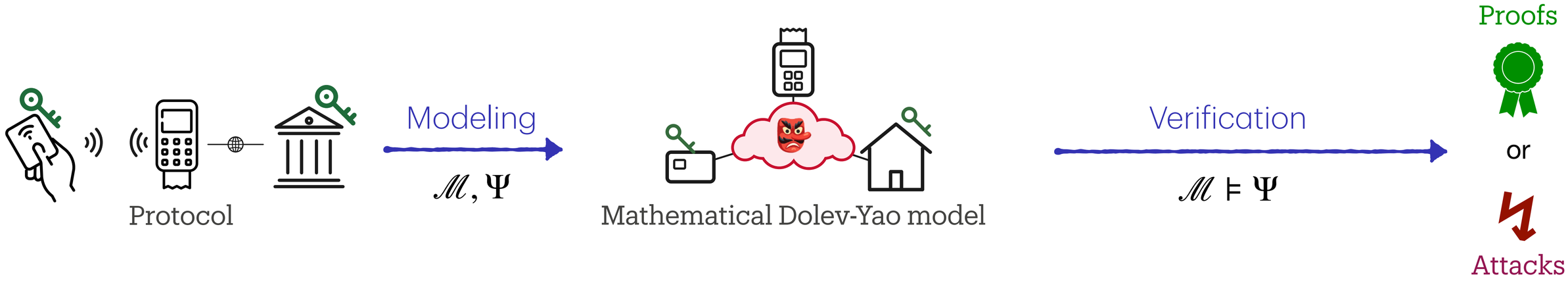
Formal verification in Dolev-Yao (DY) models



- Cryptography ↠ black-box functionalities modeled in a ***term algebra:***
  - function symbols $\mathscr{F}$ and names $\mathscr{N}$
  - formal terms $\mathscr{T}(\mathscr{F}, \mathscr{N})$
  - modulo equational theory $=_{\mathscr{E}}$

$$\mathscr{M} \begin{cases} \mathsf{MAC}(\cdot,\cdot), \mathsf{true} \in \mathscr{F} \\ \mathsf{check}(\mathsf{MAC}(x,y),x,y) =_{\mathscr{E}} \mathsf{true} \end{cases}$$

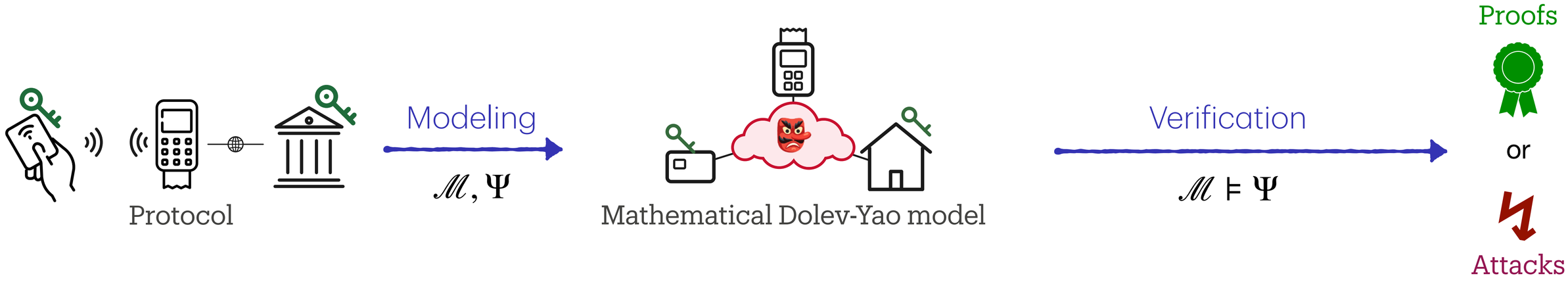
Formal verification in Dolev-Yao (DY) models



- Cryptography ↠ black-box functionalities modeled in a ***term algebra:***
  - function symbols $\mathscr{F}$ and names $\mathscr{N}$
  - formal terms $\mathscr{T}(\mathscr{F}, \mathscr{N})$
  - modulo equational theory $=_{\mathscr{E}}$
- Protocol roles ↠ programs exchanging terms
  Example: ***processes*** in applied π-calculus

$\mathscr{M}$ {
- MAC(· , ·), true ∈ $\mathscr{F}$
- check(MAC(x, y), x, y) $=_{\mathscr{E}}$ true
- `let P_card(` k_{MAC} `) =`
  `in(transaction);`
  `out(<MAC(…,…),'PinNeeded'>)`

Formal verification in Dolev-Yao (DY) models



- Cryptography ⇾ black-box functionalities modeled in a ***term algebra:***
  - function symbols $\mathscr{F}$ and names $\mathscr{N}$
  - formal terms $\mathscr{T}(\mathscr{F}, \mathscr{N})$
  - modulo equational theory $=_{\mathscr{E}}$

- Protocol roles ⇾ programs exchanging terms
  Example: ***processes*** in applied π-calculus

- Dolev Yao 👺 controls networks ⇾ learns outputs, computes over terms modulo $=_{\mathscr{E}}$, chooses inputs
  ☞ Captures logical attacks by design

- Security goals ⇾ 1st order formula over claims or ≈

$\mathscr{M}$

MAC(· , ·), true ∈ $\mathscr{F}$

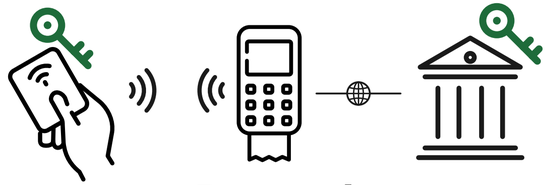
check(MAC(x, y), x, y) $=_{\mathscr{E}}$ true

```
let P_card( k_MAC ) =
 in(transaction);
 out(<MAC(…,…),'PinNeeded'>)
```

Ψ

```
TerminalValidate(transaction)
 ==>
CardSigned(transaction)
```

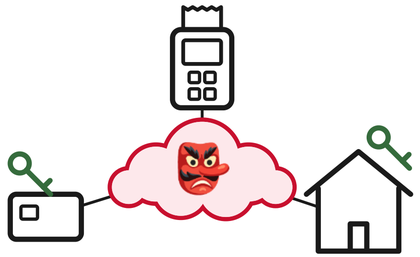
Formal verification in Dolev-Yao (DY) models



Protocol

Modeling

$\mathscr{M}, \Psi$



Mathematical Dolev-Yao model

Verification

$\mathscr{M} \models \Psi$

Proofs

or

Attacks

or

Non-termi.

- Cryptography ⇝ black-box functionalities modeled in a ***term algebra:***
  - function symbols $\mathscr{F}$ and names $\mathscr{N}$
  - formal terms $\mathscr{T}(\mathscr{F}, \mathscr{N})$
  - modulo equational theory $=_{\mathscr{E}}$
- Protocol roles ⇝ programs exchanging terms
  Example: ***processes*** in applied π-calculus
- Dolev Yao 👺 controls networks ⇝ learns outputs, computes over terms modulo $=_{\mathscr{E}}$, chooses inputs
  ☞ Captures logical attacks by design
- Security goals ⇝ 1st order formula over claims or ≈

$\mathscr{M}$

MAC(· , ·), true ∈ $\mathscr{F}$

check(MAC(x, y), x, y) $=_{\mathscr{E}}$ true

```
let P_card( k_MAC ) =
 in(transaction);
 out(<MAC(…,…),'PinNeeded'>)
```

Ψ

```
TerminalValidate(transaction)
 ==>
CardSigned(transaction)
```

Undecidable problem but sound algorithms and tools:

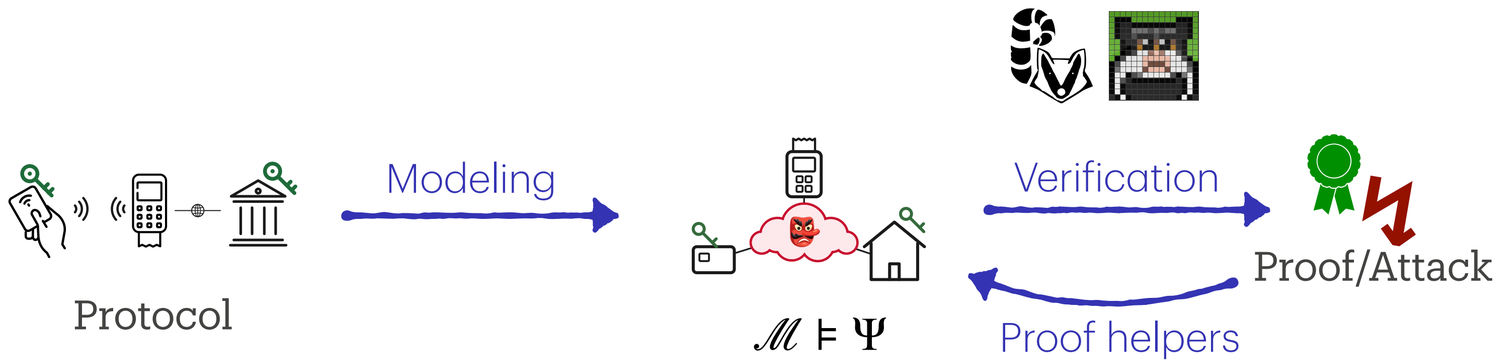




Proverif & Tamarin

Sound proof helpers

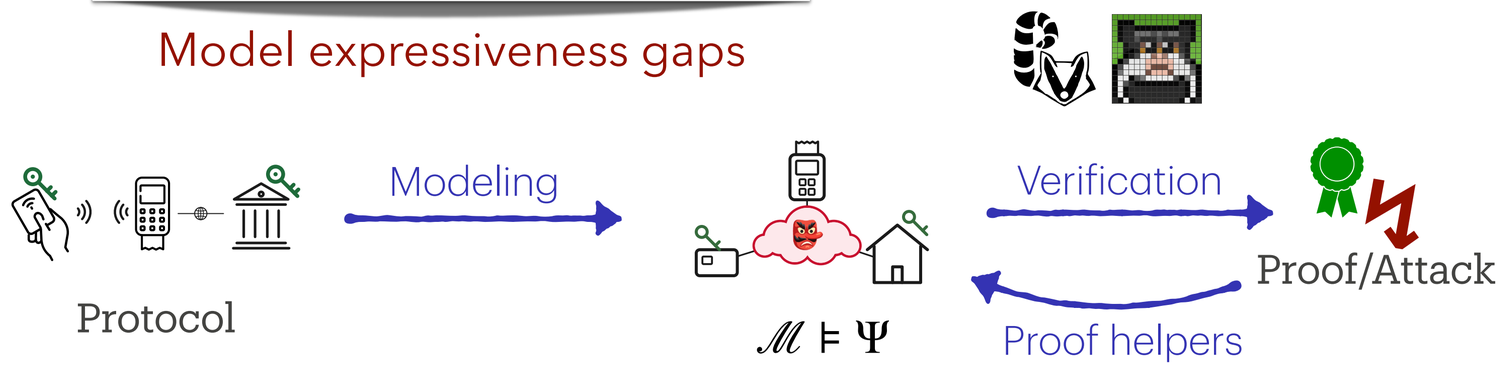
Contributions summary



Contributions summary

1. New modeling techniques
   Weak hash [USENIX'23] · BB [CSF'21]

Model expressiveness gaps



Hash modeling limitations

[USENIX'23]



Usual crypto assumption for H: Random Oracle Model (ROM)

DY model: $\mathsf{hash}(\cdot) \in \mathcal{F}$



Usual practitioners' belief: second-preimage resistant H is enough

If $x \in H$
then $n := H[x]$
else pick fresh $n \in \mathcal{A}$
$H[x] := n$

ROM

Given x, Hard to find y ≠ x such that H(y) = H(x)

Second-preimage resistance

Hash modeling limitations

[USENIX'23]

Usual crypto assumption for H: Random Oracle Model (ROM)
DY model: $\mathsf{hash}(\cdot) \in \mathcal{F}$

Usual practitioners' belief: second-preimage resistant H is enough

If $x \in H$
then $n := H[x]$
else pick fresh $n \in \mathcal{A}$
$H[x] := n$

ROM

Given x, Hard to find y ≠ x such that H(y) = H(x)

Second-preimage resistance

Reality:

- Deviations from the ROM: e.g. length extension (Merkle–Damgård)
- Practical structured collisions (CPC, IPC), yet still second-preimage resistant

(i) H(x) = H(y) => H(x||z) = H(y||z)
(ii) From H(x) and y, one can compute H(x||y)

Length-extension

Given p_1 and p_2, find x ≠ y such that $H(p_1||x) = H(p_2||y)$

Chosen-Prefix Collision (CPC)
IPC: same when $p_1 = p_2$

[USENIX'23]

Hash modeling limitations

Usual crypto assumption for H: Random Oracle Model (ROM)
DY model: $\mathsf{hash}(\cdot) \in \mathcal{F}$

Usual practitioners' belief: second-preimage resistant H is enough

If $x \in H$
then $n := H[x]$
else pick fresh $n \in \mathcal{A}$
$H[x] := n$

ROM

Given x, Hard to find y ≠ x such that H(y) = H(x)

Second-preimage resistance

Reality:

- Deviations from the ROM: e.g. length extension (Merkle–Damgård)
- Practical structured collisions (CPC, IPC), yet still second-preimage resistant
- H becomes weaker over time

(i) H(x) = H(y) =>
H(x||z) = H(y||z)
(ii) From H(x) and y, one can compute H(x||y)

Length-extension

Given p_1 and p_2, find x ≠ y such that $H(p_1||x) = H(p_2||y)$

Chosen-Prefix Collision (CPC)
IPC: same when $p_1 = p_2$

Hash modeling limitations

[USENIX'23]

Usual crypto assumption for H: Random Oracle Model (ROM)

DY model: hash(·) ∈ $\mathcal{F}$

If $x \in H$
then $n := H[x]$
else pick fresh $n \in \mathcal{A}$
$H[x] := n$

ROM

Usual practitioners' belief: second-preimage resistant H is enough

Given x, Hard to find y ≠ x such that H(y) = H(x)

Second-preimage resistance

GAP

- **SLOTH** attack against TLS 1.2 using **CPC** (CVE-2015-7575) [NDSS'16]
- **CPC-based** authentication attack on **RADIUS** protocol [USENIX'24]
- **Nostradamus** attack against Merkle-Damgård, biasing outputs
- Flickr API attack due to **length-extension**

(i) H(x) = H(y) =>
H(x||z) = H(y||z)
(ii) From H(x) and y, one can compute H(x||y)

Length-extension

Reality:

- Deviations from the ROM: e.g. length extension (Merkle–Damgård)
- Practical structured collisions (CPC, IPC), yet still second-preimage resistant
- H becomes weaker over time

Given p_1 and p_2, find x ≠ y such that H(p_1||x) = H(p_2||y)

Chosen-Prefix Collision (CPC)

IPC: same when p_1=p_2

Hash modeling limitations

[USENIX'23]

Usual crypto assumption for H: Random Oracle Model (ROM)

DY model: $\mathsf{hash}(\cdot) \in \mathcal{F}$

If $x \in H$
then $n := H[x]$
else pick fresh $n \in \mathcal{A}$
$H[x] := n$

ROM

Usual practitioners' belief: second-preimage resistant H is enough

Given x, Hard to find y ≠ x such that H(y) = H(x)

Second-preimage resistance

GAP

Revisit this gap: extend protocol formal analyses with these weaknesses:

- Under which minimal H assumptions is a protocol secure?
- Can existing/future H weaknesses be weaponized against a protocol?

(i) H(x) = H(y) => H(x||z) = H(y||z)
(ii) From H(x) and y, one can compute H(x||y)

Length-extension

Reality:

- Deviations from the ROM: e.g. length extension (Merkle–Damgård)
- Practical structured collisions (CPC, IPC), yet still second-preimage resistant
- H becomes weaker over time

Given p_1 and p_2, find x ≠ y such that $H(p_1||x) = H(p_2||y)$

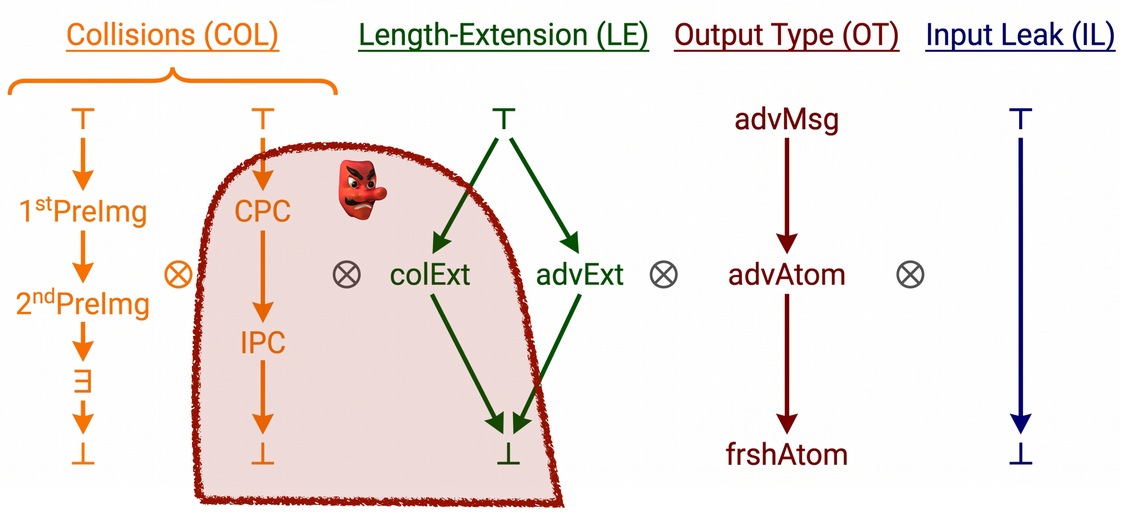
Chosen-Prefix Collision (CPC)
IPC: same when $p_1 = p_2$

[USENIX'23]

New hash modeling techniques

Modeling $\mathscr{M} \vDash \Psi$

- 4 dimensions lattice of hash weaknesses
  Partial order ≈ weaker attacker

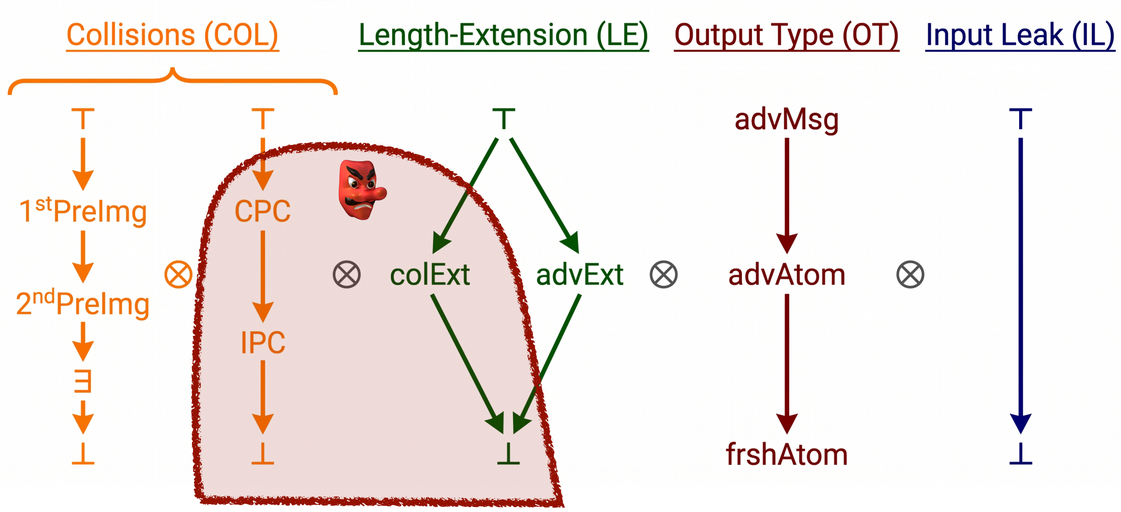


[USENIX'23]

New hash modeling techniques

Modeling $\mathscr{M} \vDash \Psi$

- 4 dimensions lattice of hash weaknesses
  Partial order ≈ weaker attacker
  Ex: {CPC,colExt}: 👺 finds CPC and can extend

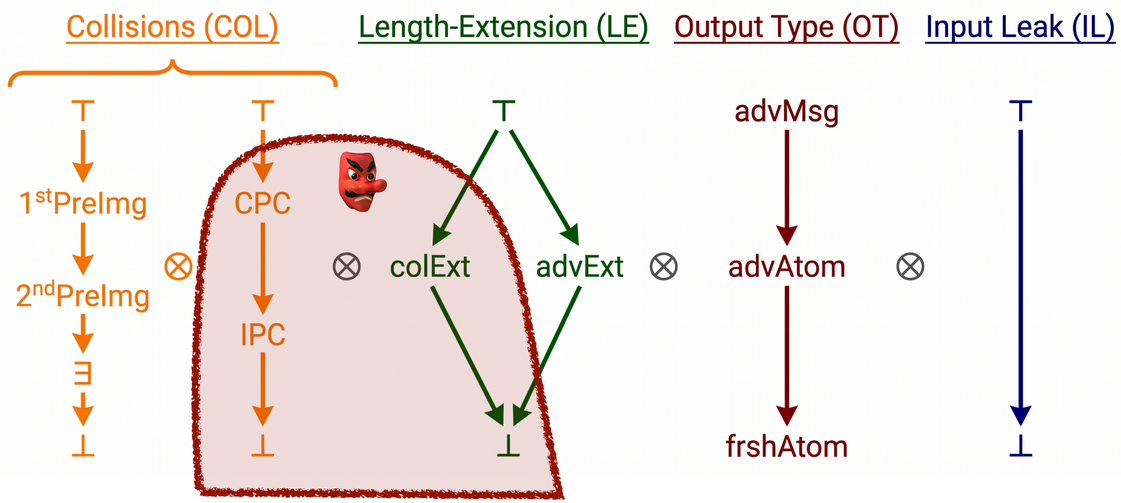


[USENIX'23]

New hash modeling techniques

Modeling $\mathscr{M} \vDash \Psi$

- 4 dimensions lattice of hash weaknesses
  Partial order ≈ weaker attacker
  Ex: {CPC,colExt}: 👺 finds CPC and can extend



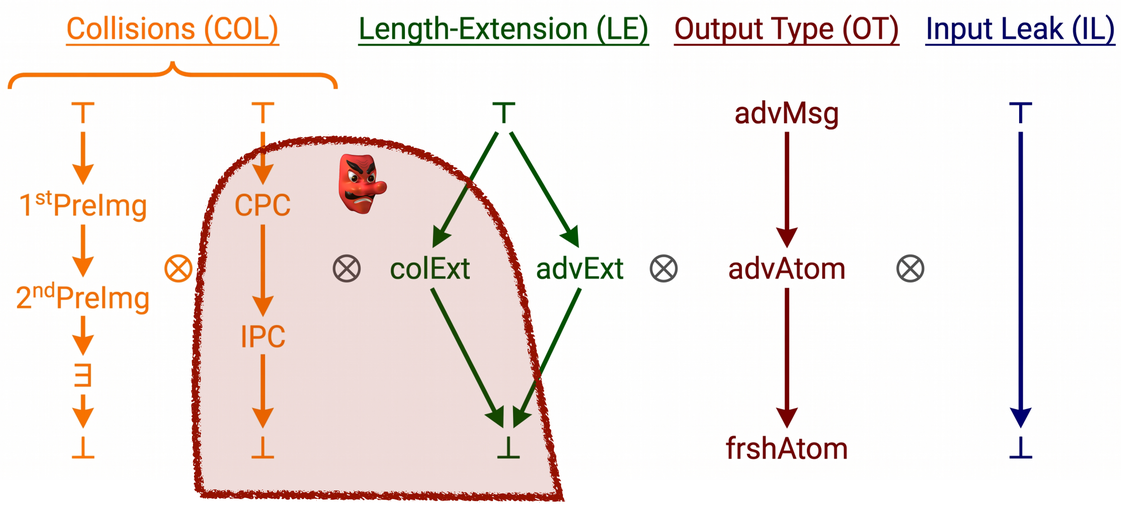
- Some are easy to model: e.g. 2ndPreImg: $H(pi2(z)) = H(z)$

- Other « Allowed collisions » cannot be expressed by equational theories; instead:

[USENIX'23]

New hash modeling techniques

Modeling $\mathcal{M} \vDash \Psi$



- 4 dimensions lattice of hash weaknesses
  Partial order ≈ weaker attacker
  Ex: {CPC,colExt}: 😈 finds CPC and can extend

- Some are easy to model: e.g. 2ndPreImg: $\mathrm{H}(\mathrm{pi2}(z)) = \mathrm{H}(z)$

- Other « Allowed collisions » cannot be expressed by equational theories; instead:
  - Function symbols modeling collisions. E.g. for $\mathrm{cp}_1(\cdot,\cdot)$: computes a CP-collision
  - **Over-approximation first**: 😈 chooses Hash outputs, logged with $\mathrm{H}^L(\mathrm{in},\mathrm{out})$
  - **Restrict then**: exclude traces where attacker is outside given threat model. Ex for: {CPC,colExt}

$$\forall i_1, i_2, o\,.\ \mathrm{H}^L(i_1,o)\ \&\ \mathrm{H}^L(i_2,o)\ \&\ i_1 \neq i_2 \implies \exists p_1, p_2, s\,.\,(i_1 = p_1 \parallel \mathrm{cp}_1(p_1,p_2) \parallel s\ \ \&\ \ i_2 = p_2 \parallel \mathrm{cp}_2(p_1,p_2) \parallel s)$$

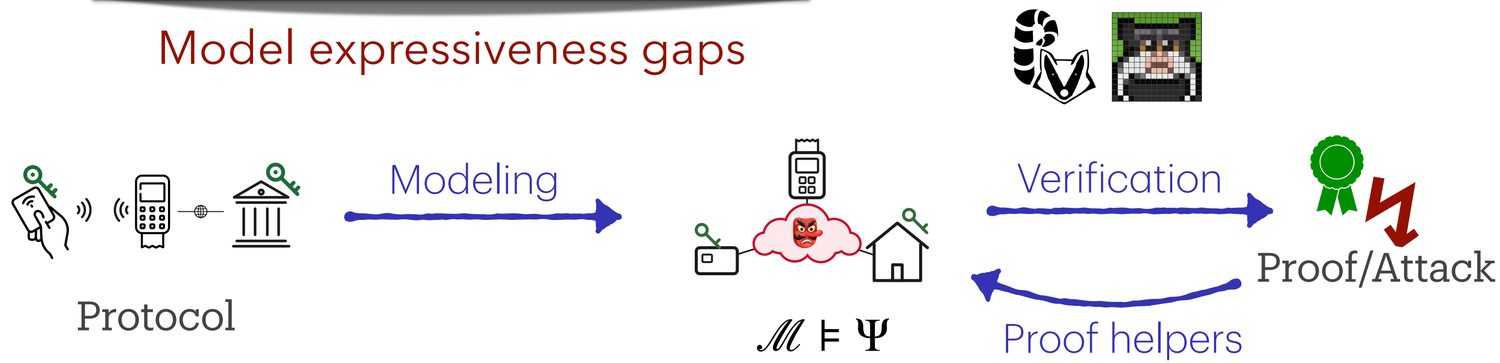
- Concatenation $\cdot \parallel \cdot$ of hashed messages must be **associative**. Ex: $\mathrm{H}((x \parallel y) \parallel z) = \mathrm{H}(x \parallel (y \parallel z))$

Contributions

1. New modeling techniques

Weak hash [USENIX'23] · BB [CSF'21]

Model expressiveness gaps



Contributions

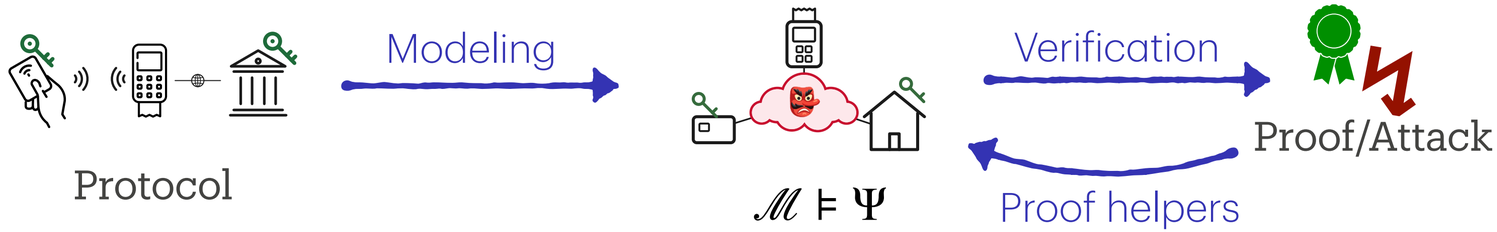
1. New modeling techniques
   - Weak hash [USENIX'23]
   - BB [CSF'21]

Model expressiveness gaps

2. Extend verifier capabilities
   - XOR [CSF'18,JCS'20] ≈ [EuroS&P'19]
   - Assoc [USENIX'23] ≈ POR [Esorics'18]

Expressiveness and scaling gaps





XOR and associative symbols

[CSF'18,JCS'20,USENIX'23]



$\mathscr{M} \models \Psi$ Verification

Tamarin: supported neither $\cdot \| \cdot$ (required for hash modeling) nor XOR (often used in protocols)

[CSF'18,JCS'20,USENIX'23]



XOR and associative symbols

$\mathscr{M} \vDash \Psi$ Verification →

Tamarin: supported neither $\cdot \,||\, \cdot$ (required for hash modeling) nor XOR (often used in protocols)

[CSF'18,JCS'20]

- XOR: Associative, Commutative (AC) (supported) + cancellation properties ($x \oplus y \oplus x = y$) ↛ non-termination

☞ Soundly restrict explored (symbolic) traces with new normal form condition + relax equivalence verification

☞ New analyses of RFID protocols and 5G; otherwise impossible

[CSF'18,JCS'20,USENIX'23]



XOR and associative symbols

$\mathscr{M} \vDash \Psi$ Verification →

Tamarin: supported neither $\cdot \,\|\, \cdot$ (required for hash modeling) nor XOR (often used in protocols)

[CSF'18,JCS'20]

- XOR: Associative, Commutative (AC) (supported) + cancellation properties ($x \oplus y \oplus x = y$) ⇾ non-termination

☞ Soundly restrict explored (symbolic) traces with new normal form condition + relax equivalence verification

☞ New analyses of RFID protocols and 5G; otherwise impossible

[USENIX'23]

- Associative concatenation $\cdot \,\|\, \cdot$ is algebraically simpler (purely Associative)
  - ⇾ Dropping Commutativity can break unification finiteness Ex: $0 \,\|\, x \stackrel{?}{=} x \,\|\, 0$ (yields $\{x \mapsto 0^n \mid n \in \mathbb{N}\}$)
  - ⇾ Out of scope

[CSF'18,JCS'20,USENIX'23]



XOR and associative symbols

$\mathscr{M} \vDash \Psi$ Verification →

Tamarin: supported neither $\cdot \parallel \cdot$ (required for hash modeling) nor XOR (often used in protocols)

[CSF'18,JCS'20]

- XOR: Associative, Commutative (AC) (supported) + cancellation properties ($x \oplus y \oplus x = y$) ↠ non-termination

☞ Soundly restrict explored (symbolic) traces with new normal form condition + relax equivalence verification

☞ New analyses of RFID protocols and 5G; otherwise impossible

[USENIX'23]

- Associative concatenation $\cdot \parallel \cdot$ is algebraically simpler (purely Associative)
  - ↠ Dropping Commutativity can break unification finiteness Ex: $0 \parallel x =^? x \parallel 0$ (yields $\{x \mapsto 0^n \mid n \in \mathbb{N}\}$)
  - ↠ Out of scope

☞ Extended Tamarin soundness/completeness proof for general case; aborts verif. only if infinite solutions occur + new (sound) normal form conditions

☞ Enabled practical analysis of our new hash models (We also extended ProVerif for $\cdot \parallel \cdot$)

Contributions

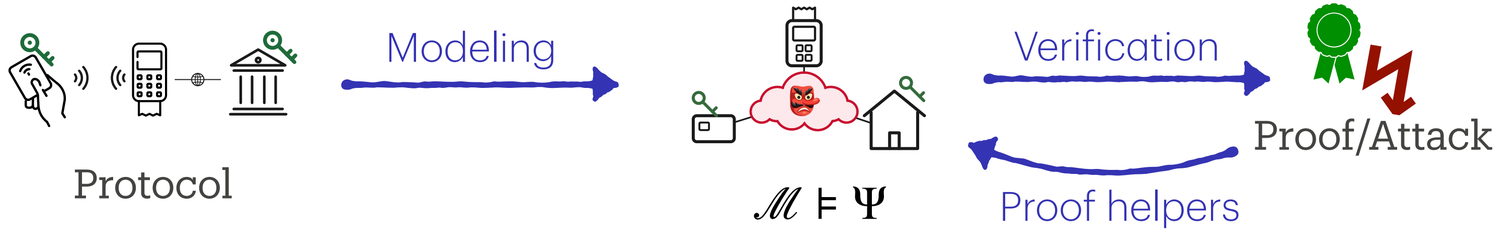
1. New modeling techniques
   - Weak hash [USENIX'23]
   - BB [CSF'21]

Model expressiveness gaps

2. Extend verifier capabilities
   - XOR [CSF'18,JCS'20] ≈ [EuroS&P'19]
   - Assoc [USENIX'23] ≈ POR [Esorics'18]

Expressiveness and scaling gaps





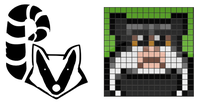
Contributions

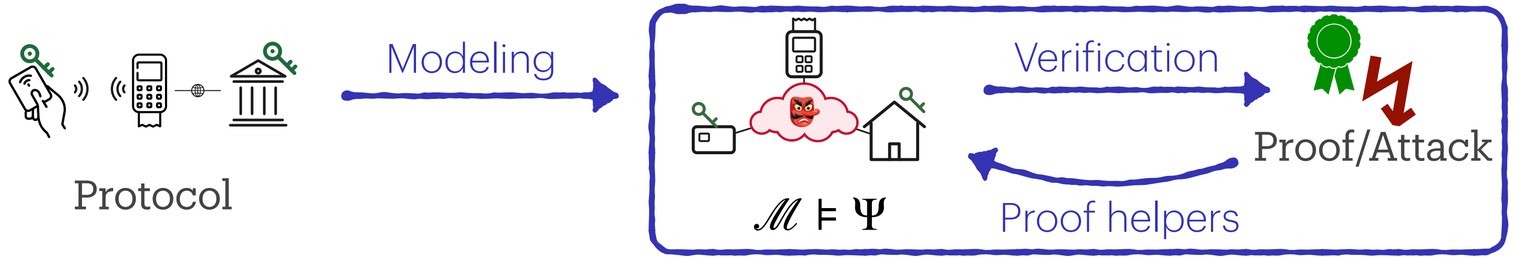
1. New modeling techniques
   - Weak hash [USENIX'23]
   - BB [CSF'21]

Model expressiveness gaps

2. Extend verifier capabilities
   - XOR [CSF'18,JCS'20]
   - ≈ [EuroS&P'19]
   - Assoc [USENIX'23]
   - POR [Esorics'18]

Expressiveness and scaling gaps





Measuring security limits



From binary to « measurement »

3. Proof methodologies
   - Lattice definition and exploration: [CCS'18,USENIX'20 USENIX'23 USENIX'25]
   - Iterative proof helpers: [USENIX'25]

[USENIX'23] 

Hash model results

Verification 

Lattice of 👺: 264 proof obligations for each security property!

[USENIX'23] 

Hash model results

Verification 

Lattice of 👺: 264 proof obligations for each security property!

1. Extend the partial order:
   Ex: 2nd Preimage ∧ InputLeak ⇒ 1st Preimage
2. Use partial order to prune proof obligations after each result
3. Iterative lattice exploration strategy optimizing pruning pay-off
   ☞ 70% fewer Tamarin calls

[USENIX'23]

Hash model results

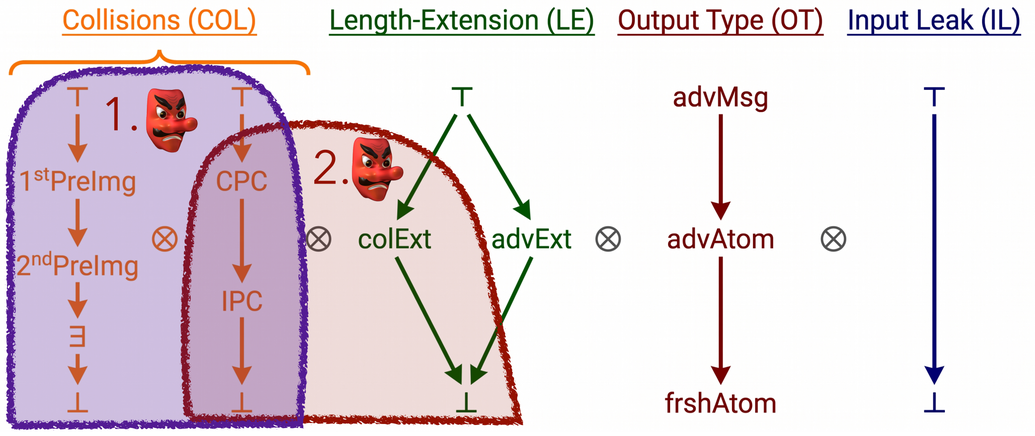
Verification 

Lattice of 😈: 264 proof obligations for each security property!

1. Extend the partial order:
   Ex: 2ndPreimage ∧ InputLeak ⇒ 1stPreimage
2. Use partial order to prune proof obligations after each result
3. Iterative lattice exploration strategy optimizing pruning pay-off
   ☞ 70% fewer Tamarin calls
4. Present compact results leveraging lattice structure

Example: Sigma' protocol authentication

- Minimal attack threat models:
  1. T, T
  2. CPC, colExt
- Maximal secure threat models:
  A. 1stPreimage, CPC, advExt, advMsg, inLeak
  B. 1stPreimage, IPC, T, advMsg, inLeak



[USENIX'23]

Hash model results

Verification 

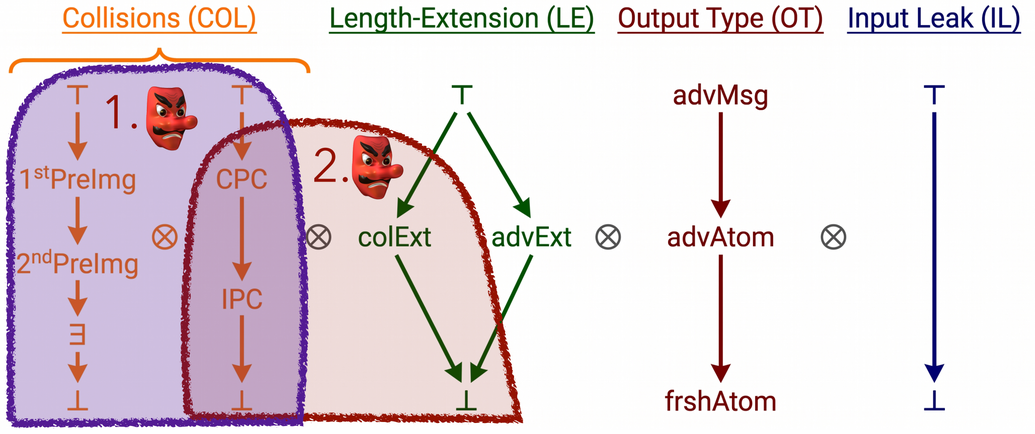
Lattice of 😈: 264 proof obligations for each security property!

1. Extend the partial order:
   Ex: 2ndPreimage ∧ InputLeak ⇒ 1stPreimage
2. Use partial order to prune proof obligations after each result
3. Iterative lattice exploration strategy optimizing pruning pay-off
   ☞ 70% fewer Tamarin calls
4. Present compact results leveraging lattice structure

- Analyzed SSH, IKEv2, Telegram, Sigma', Flickr
  ☞ automatically rediscovered all known hash-related attacks
  ☞ and new attack variants + formal proofs for the others 😈

Example: Sigma' protocol authentication

- Minimal attack threat models:
  1. T,T 😈
  2. CPC, colExt
- Maximal secure threat models:
  A. 1stPreimage, CPC, advExt, advMsg, inLeak
  B. 1stPreimage, IPC, T, advMsg, inLeak



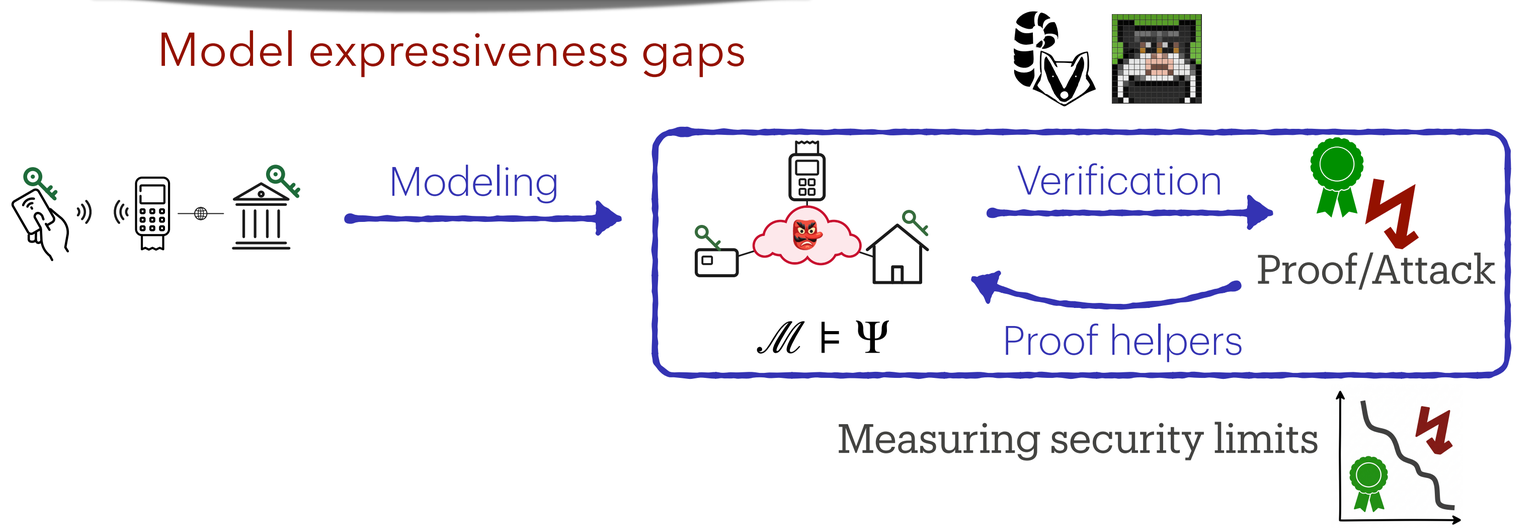
Contributions

1. New modeling techniques
Weak hash [USENIX'23] · BB [CSF'21]

Model expressiveness gaps

2. Extend verifier capabilities
XOR [CSF'18,JCS'20] · ≈ [EuroS&P'19]
Assoc [USENIX'23] · POR [Esorics'18]

Expressiveness and scaling gaps



From binary to « measurement »

3. Proof methodologies
· Lattice definition and exploration: [CCS'18,USENIX'20 USENIX'23 USENIX'25]
· Iterative proof helpers: [USENIX'25]

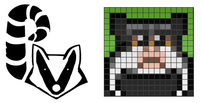
Contributions

1. New modeling techniques
Weak hash [USENIX'23] · BB [CSF'21]

Model expressiveness gaps

2. Extend verifier capabilities
XOR [CSF'18,JCS'20] · ≈ [EuroS&P'19]
Assoc [USENIX'23] · POR [Esorics'18]

Expressiveness and scaling gaps





Standard,
RFCs, etc.

OPC-UA: 528 pages
5G AKA: 722 pages

Specify



Precise system and
threat specifications

Modeling



Measuring security limits



From binary to « measurement »

3. Proof methodologies
· Lattice definition and exploration: [CCS'18,USENIX'20 USENIX'23 USENIX'25]
· Iterative proof helpers: [USENIX'25]

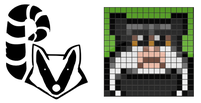
Contributions

1. New modeling techniques
Weak hash [USENIX'23] · BB [CSF'21]

Model expressiveness gaps

2. Extend verifier capabilities
XOR [CSF'18,JCS'20] · ≈ [EuroS&P'19]
Assoc [USENIX'23] · POR [Esorics'18]

Expressiveness and scaling gaps





Standard,
RFCs, etc.

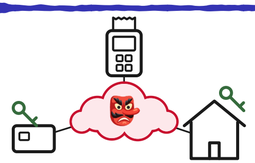
OPC-UA: 528 pages
5G AKA: 722 pages

Specify



Precise system and
threat specifications

Modeling



$\mathscr{M} \models \Psi$

Verification

Proof/Attack

Proof helpers

Security
Analysis



Security analysis,
Attacks, Fixes,
Recommendations,
Lessons

Measuring security limits



From binary to « measurement »

3. Proof methodologies
· Lattice definition and exploration: [CCS'18,USENIX'20 USENIX'23 USENIX'25]
· Iterative proof helpers: [USENIX'25]

4. Large-scale formal analyses of standards
Industrial standards: 5G [CCS'18] · Noise [USENIX'20] · OPC-UA [USENIX'25]

Large-scale analyses: 3GPP standards



3G / 4G standard
(3GPP)



5G standard,
722 pages (3GPP)

Large-scale analyses: 3GPP standards



3G / 4G standard (3GPP)

Manual analysis

[PETS'19]

[MoST'17]



5G standard, 722 pages (3GPP)

2 practical privacy logical attacks 😈:

1. On 3G/4G/5G authentication, ⊕
2. On Wi-Fi calling

😈 PoC on commercial 4G networks

Formal analysis of fixes



Large-scale analyses: 3GPP standards



3G / 4G standard (3GPP)

Manual analysis

[PETS'19]

[MoST'17]

2 practical privacy logical attacks 😈:

1. On 3G/4G/5G authentication, ⊕
2. On Wi-Fi calling

😈 PoC on commercial 4G networks

Formal analysis of fixes



5G standard, 722 pages (3GPP)

Formal analysis with Tamarin

[CCS'18]

XOR ↛ Tamarin extension [CSF'18]

2 authentication weaknesses 😈

Proven secure with fixes

With V. Stettler (MSc)

Large-scale analyses: 3GPP standards



3G / 4G standard (3GPP)

Manual analysis

[PETS'19]

[MoST'17]

2 practical privacy logical attacks 😈:

1. On 3G/4G/5G authentication, ⊕
2. On Wi-Fi calling

😈 PoC on commercial 4G networks

Formal analysis of fixes



5G standard, 722 pages (3GPP)

Formal analysis with Tamarin

[CCS'18]

XOR ↛ Tamarin extension [CSF'18]

2 authentication weaknesses 😈

Proven secure with fixes

With V. Stettler (MSc)

Responsible disclosure

Privacy attacks were eventually fixed. For 1.:

Meetings w/ 3GPP & GSMA (operators) ('17)

↛ flagged as 5G key issues ('19)

↛ 7+ vendors proposed various fixes ('18-'22)

↛ New 5G specification version with fix ('22)

Push back authentication weakness:

« 5G standards are for interoperability »

Large-scale analyses: 3GPP standards



3G / 4G standard (3GPP)

Manual analysis

[PETS'19]

[MoST'17]

2 practical privacy logical attacks 😈:
1. On 3G/4G/5G authentication, ⊕
2. On Wi-Fi calling

😈 PoC on commercial 4G networks

Formal analysis of fixes



5G standard, 722 pages (3GPP)

Formal analysis with Tamarin

[CCS'18]

XOR ↛ Tamarin extension [CSF'18]

2 authentication weaknesses 😈

Proven secure with fixes

With V. Stettler (MSc)

Responsible disclosure

Privacy attacks were eventually fixed. For 1.:

Meetings w/ 3GPP & GSMA (operators) ('17)

↛ flagged as 5G key issues ('19)

↛ 7+ vendors proposed various fixes ('18-'22)

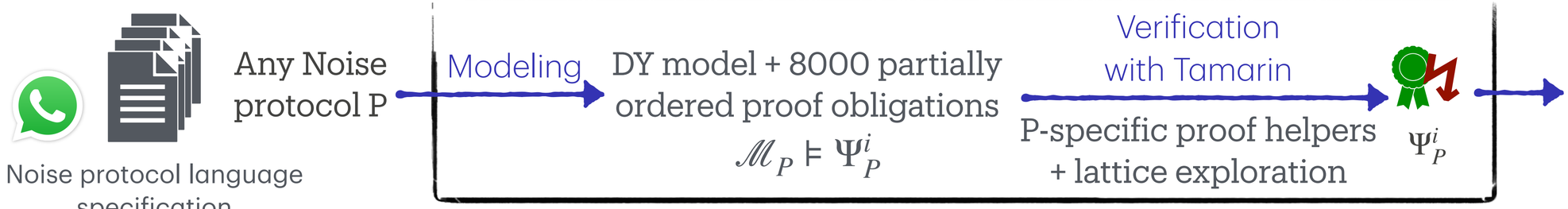
↛ New 5G specification version with fix ('22)

Push back authentication weakness:

« 5G standards are for interoperability »

3GPP won't move until practical attack?

Large-scale analyses: Noise & OPC UA



Vacarme compiler With G. Girol (MSc)

[USENIX'20]



Protocol comparison

→ which protocol to use

Large-scale analyses: Noise & OPC UA



Noise protocol language specification

Any Noise protocol P

Modeling

DY model + 8000 partially ordered proof obligations $\mathscr{M}_P \vDash \Psi_P^i$

Verification with Tamarin

P-specific proof helpers + lattice exploration

Ψ_P^i

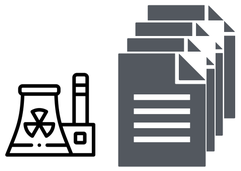
Vacarme compiler With G. Girol (MSc)

[USENIX'20]



Protocol comparison

→ which protocol to use



OPC UA industrial standard, 528p (OPC Foundation)

Formal analysis with ProVerif

[USENIX'25]

Challenges: multi-layer, state machine

→ non-termination → new proof methods

6 severe logical attacks +2 weaknesses

provably secure fixes

With V. Diemunsch (PhD: 2022-2026)

Responsible disclosure

3-year-process! Meetings with OPC, CISA, BSI, ANSSI and 100 emails later:

8 GCVE + standard update with fixes

[USENIX'23] 

French national e-voting protocol

- **National, legislative election**: 1.6M voters (citizens abroad) used a new e-voting protocol in 2022

French national e-voting protocol

- **National, legislative election**: 1.6M voters (citizens abroad) used a new e-voting protocol in 2022



- No (precise) specification to verify against



Code

- We instead reverse-engineered the protocol from the obfuscated voting client (sent to voters)

- We found practical logical attacks caused by two vulnerabilities:

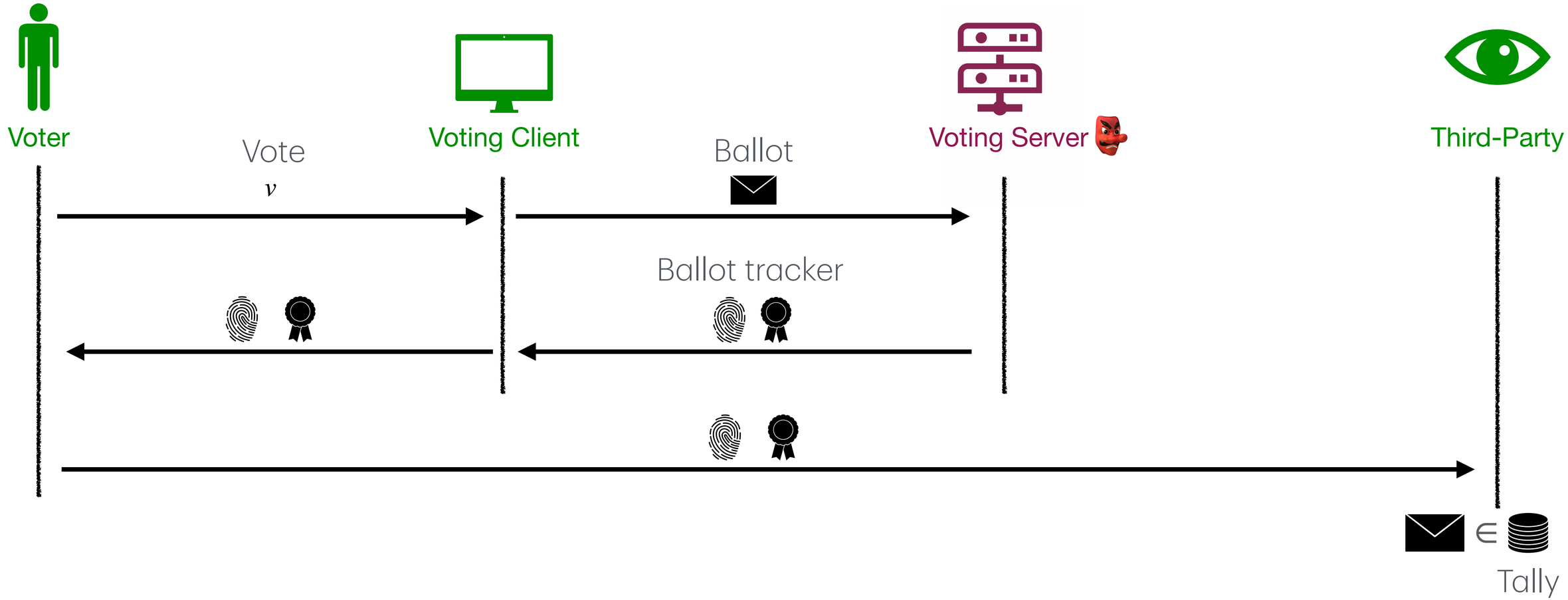


Protocol design flaw: sub-election ID (consulate) not cryptographically bound to ballots
→ weaponized against ballot privacy

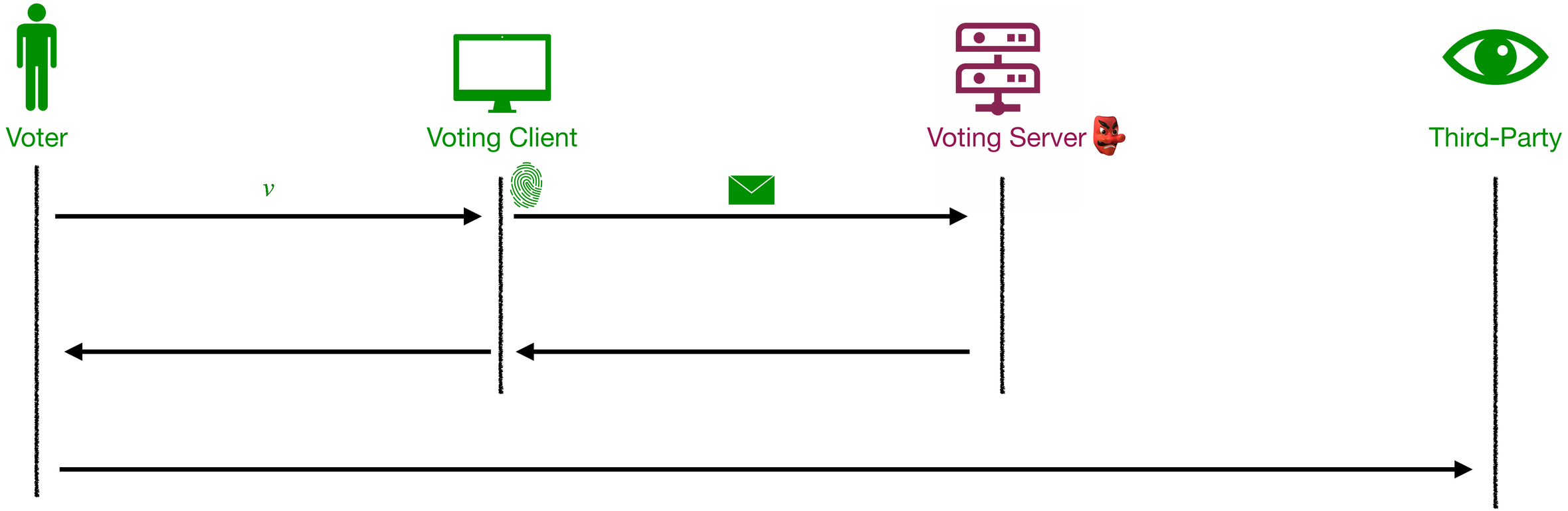


Implementation bug…

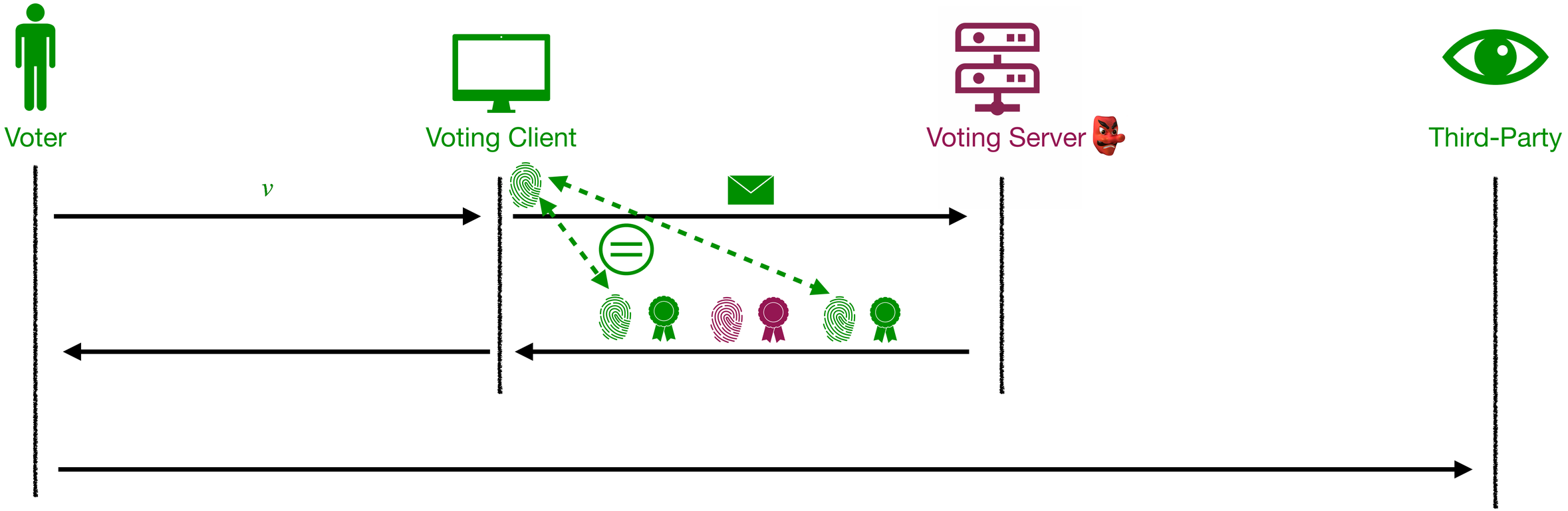
Attacking verifiability and election integrity



Attacking verifiability and election integrity

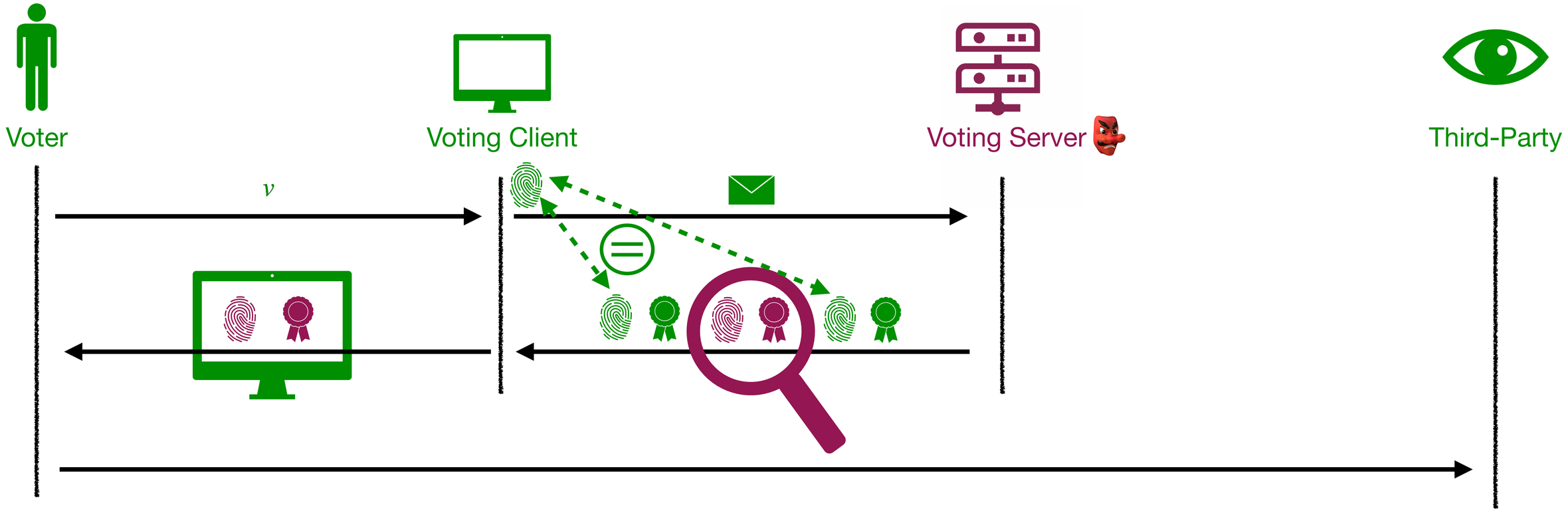


Attacking verifiability and election integrity



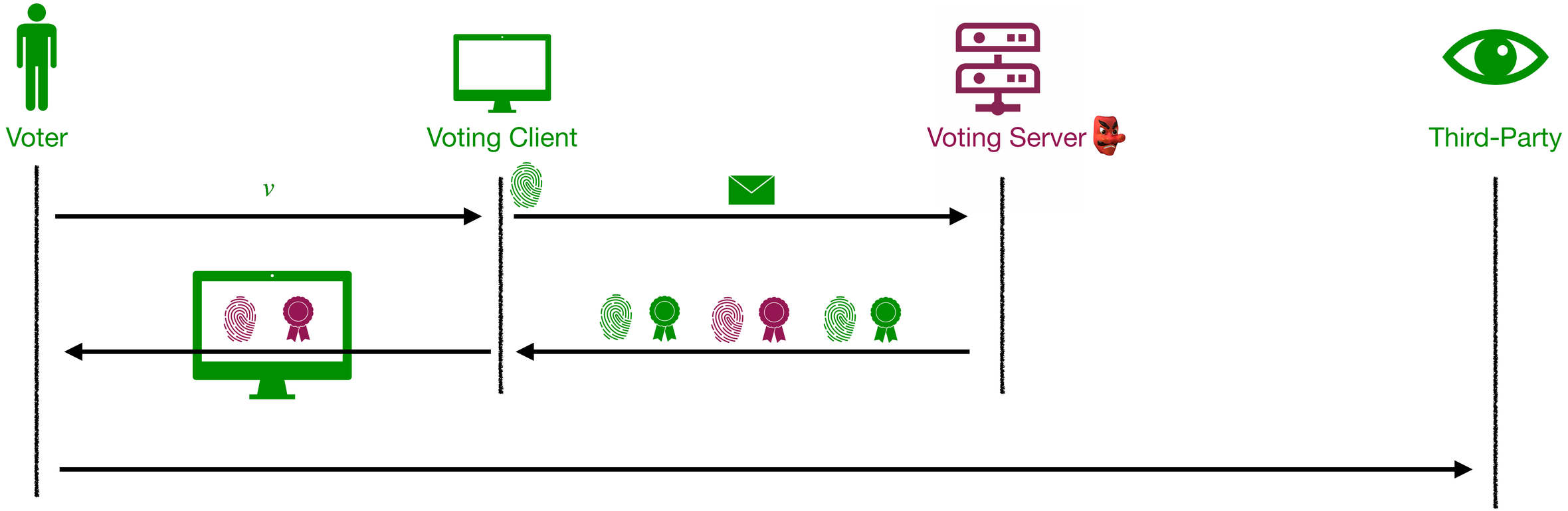
- There are 4 versions of  with various consistency checks in the voting client

Attacking verifiability and election integrity



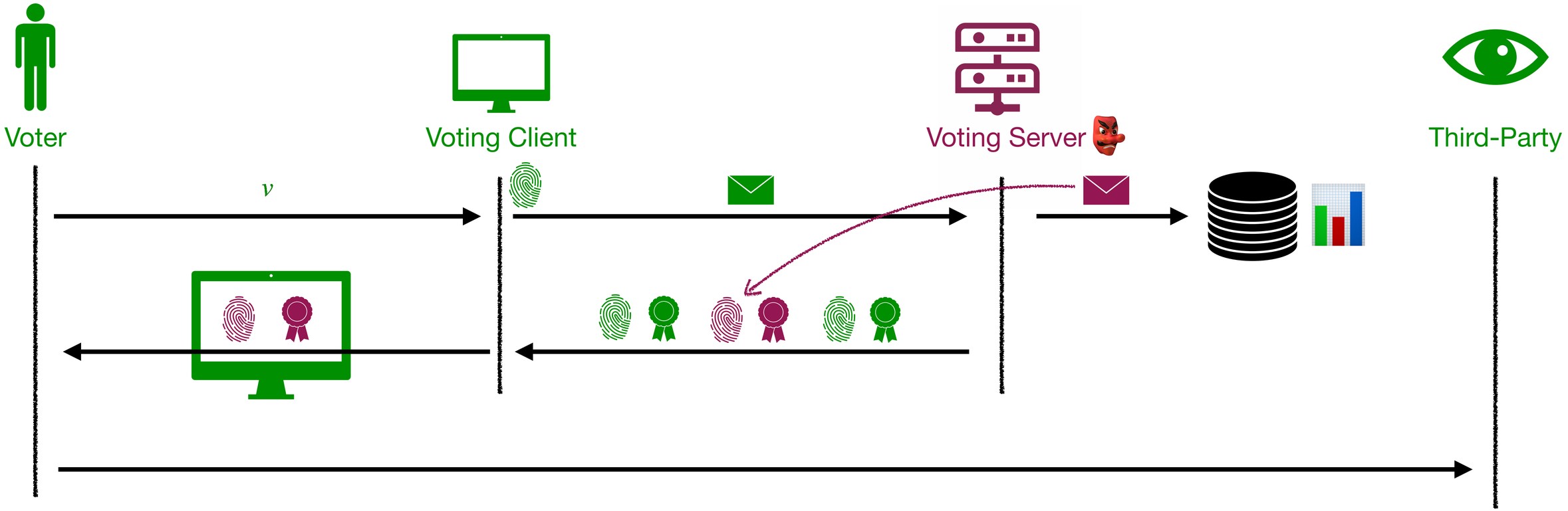
- There are 4 versions of with various consistency checks in the voting client
- Implemented checks are buggy and actually displayed to the voter can be attacker-controlled

Attacking verifiability and election integrity



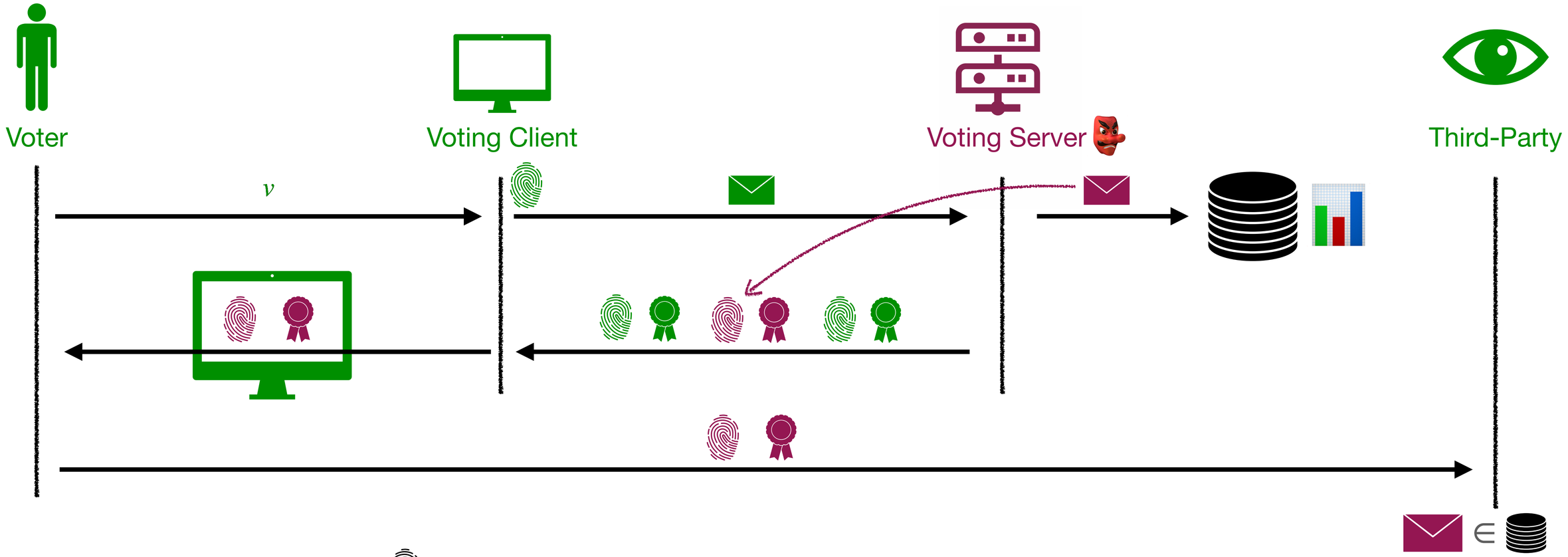
- There are 4 versions of with various consistency checks in the voting client
- Implemented checks are buggy and actually displayed to the voter can be attacker-controlled

Attacking verifiability and election integrity



- There are 4 versions of with various consistency checks in the voting client
- Implemented checks are buggy and actually displayed to the voter can be attacker-controlled

Attacking verifiability and election integrity



- There are 4 versions of fingerprint with various consistency checks in the voting client
- Implemented checks are buggy and fingerprint + ribbon actually displayed to the voter can be attacker-controlled
- Impact: channel or server attacker can stealthily modify the election outcome by replacing ballots

French national e-voting protocol

- **National, legislative election**: 1.6M eligible voters (abroad) used a new e-voting protocol in 2022



- No (precise) specification ⇾ no formal analysis possible



Code

- We instead reverse-engineered the protocol from the obfuscated voting client (sent to voters)

- We found 2 practical logical attacks caused by:



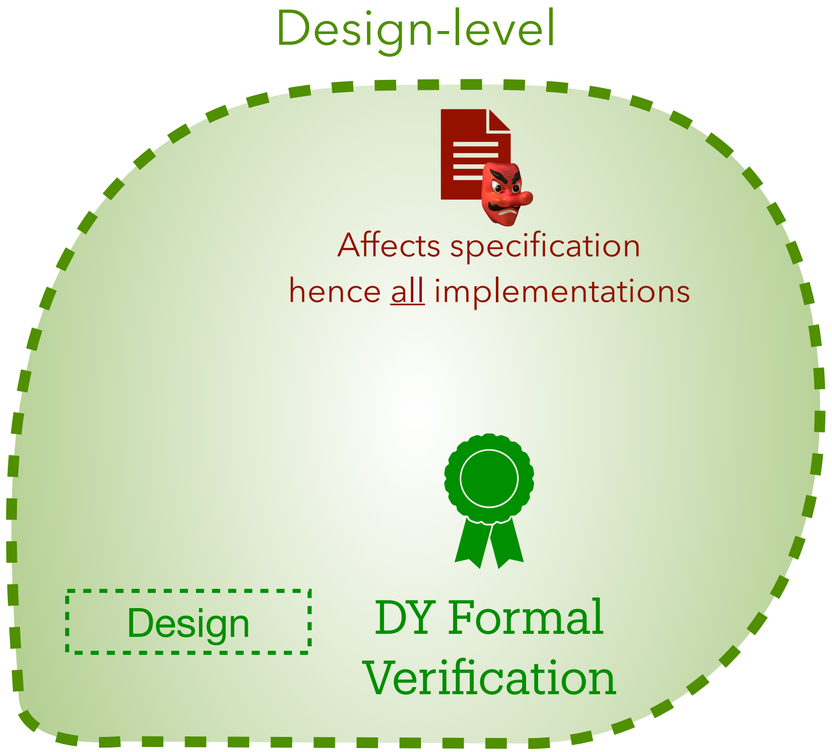
Protocol design flaw: sub-election ID (consulate) not cryptographically bound to ballots
⇾ weaponized against ballot privacy



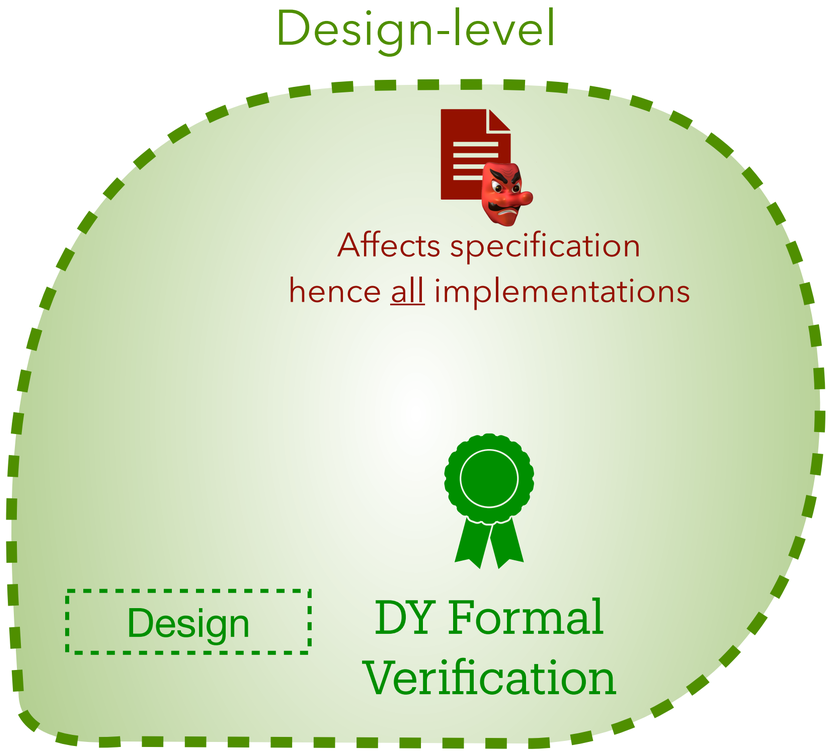
Implementation bug:
⇾ weaponized against verifiability

- Reported with 5 fixes to ANSSI, EFA Ministry, vendor ('22) ⇾ (partially) fixed in next elections ('23)

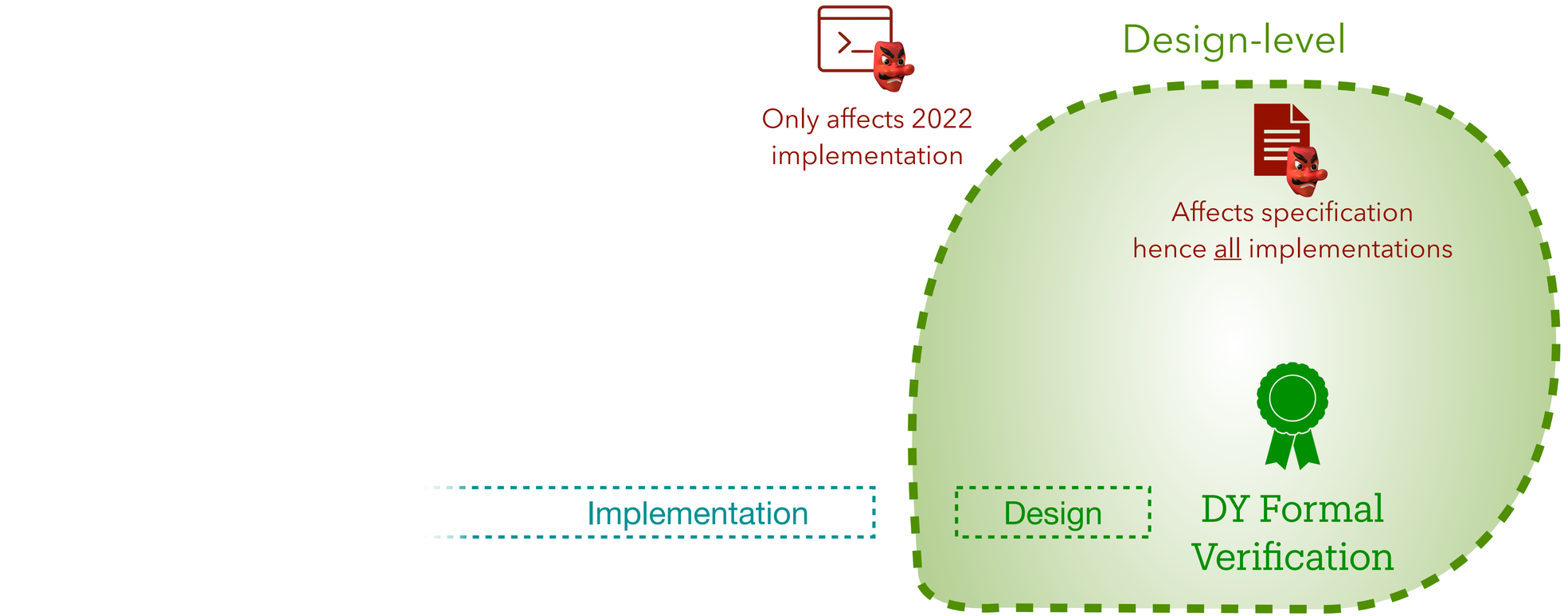
Beyond protocol specification analysis



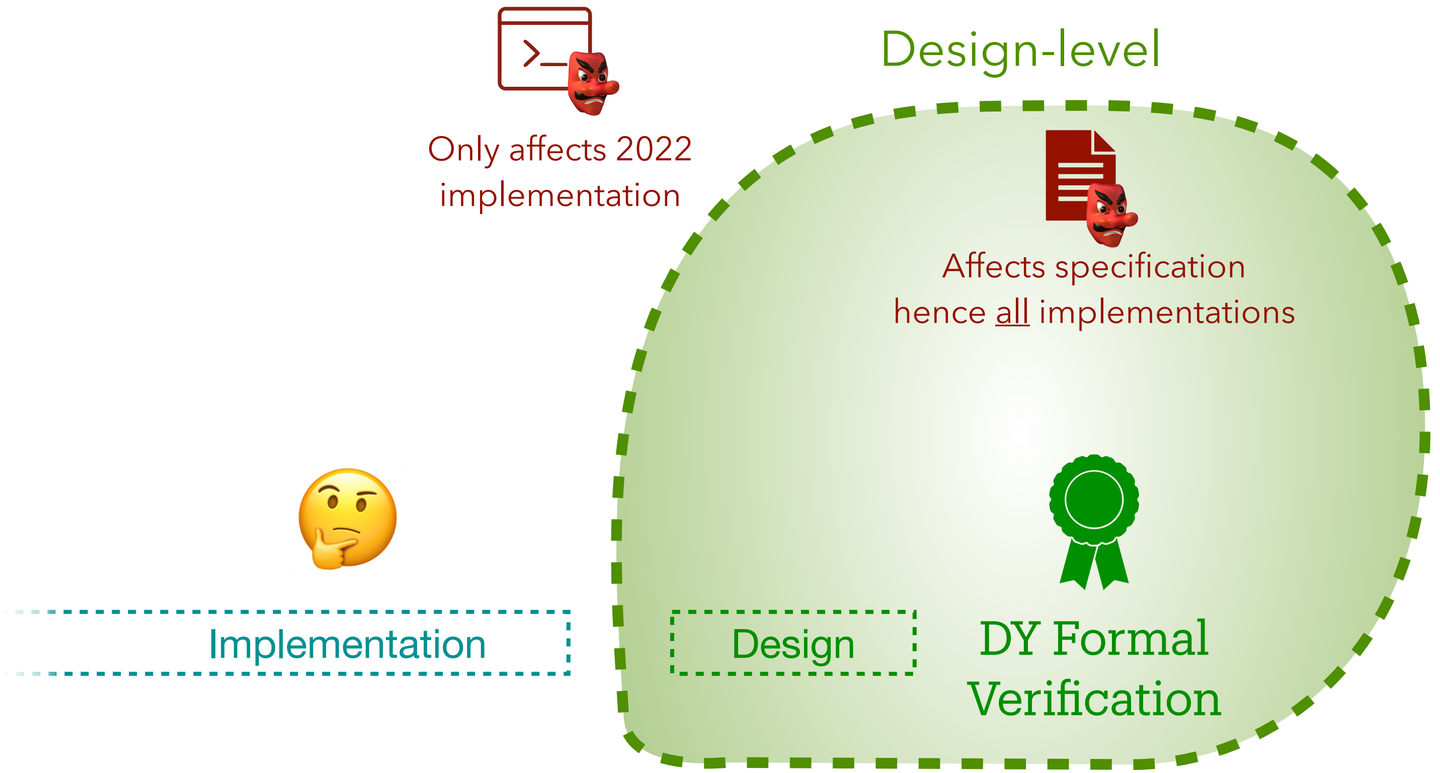
Beyond protocol specification analysis



Beyond protocol specification analysis



Beyond protocol specification analysis



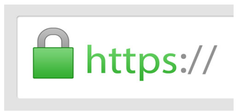
Beyond protocol specification analysis

Prove the code?





Belenios: 16k LoC



OpenSSL: 200k LoC

- Scale to whole protocol codebases?
- Rewrite and prove but existing deployed codebases?



Only affects 2022 implementation



Implementation

Design-level



Affects specification hence all implementations

Design



DY Formal Verification

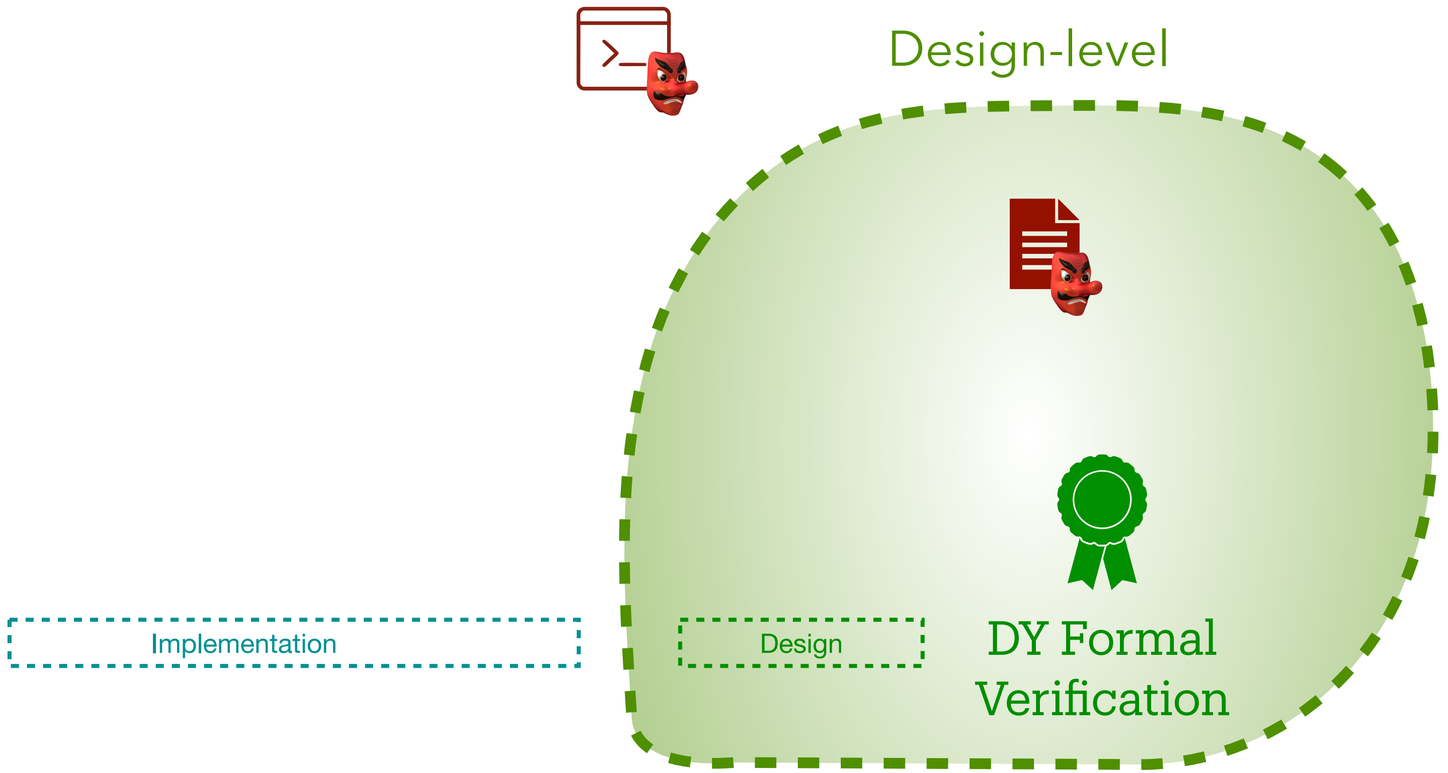
Part II: 

Dolev-Yao Model-Guided Fuzzing for Cryptographic Protocol Implementations

Beyond protocol specification analysis

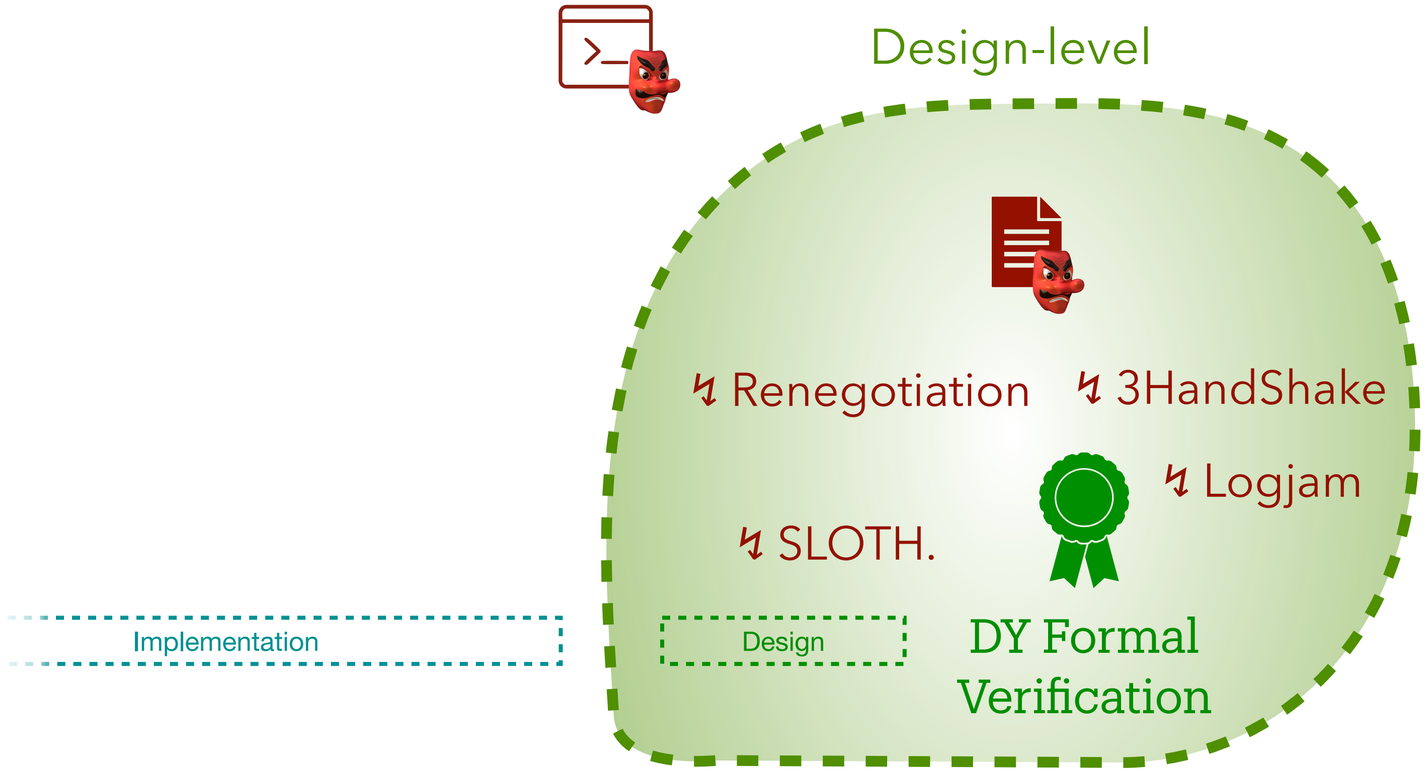
Further illustrated on some attacks ↯ on TLS





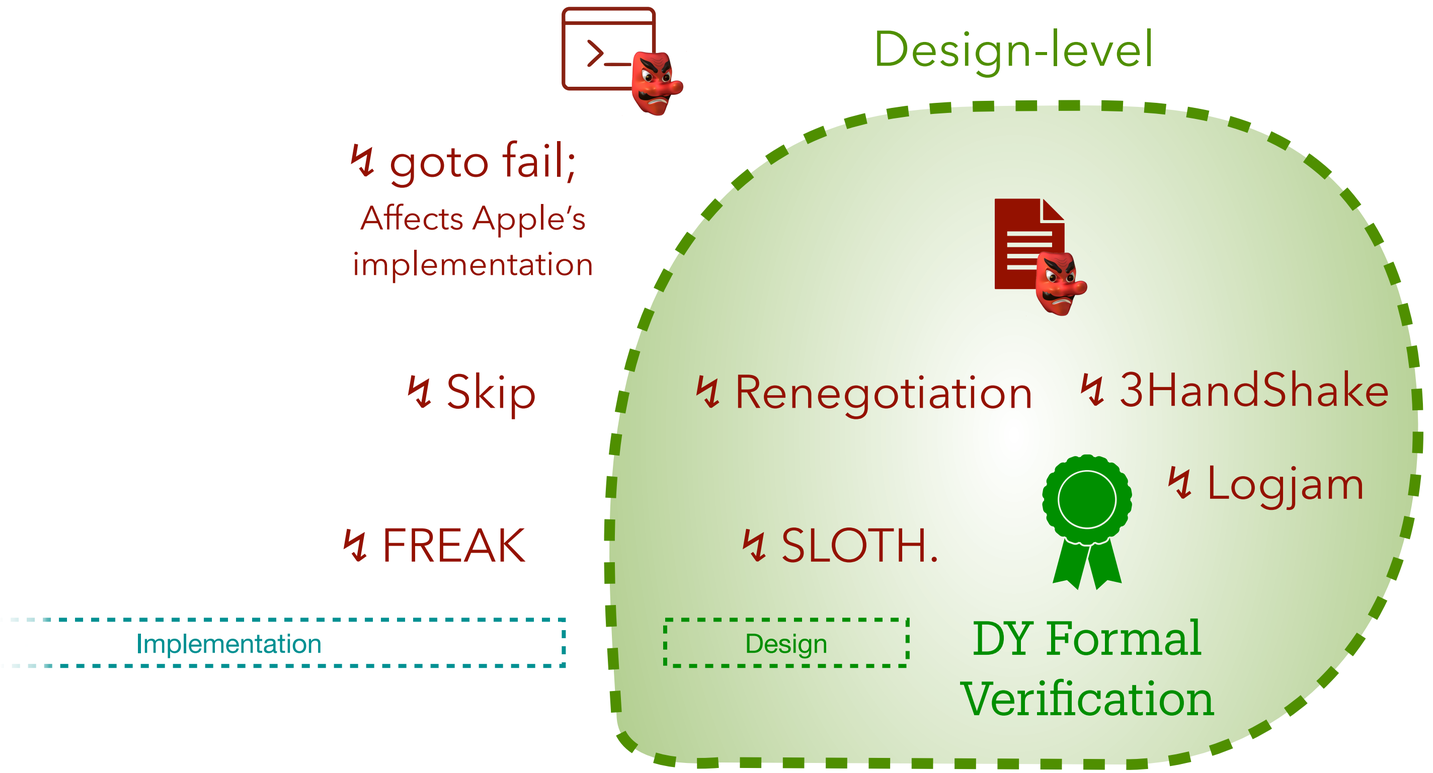
Beyond protocol specification analysis

Further illustrated on some attacks ↯ on TLS https://



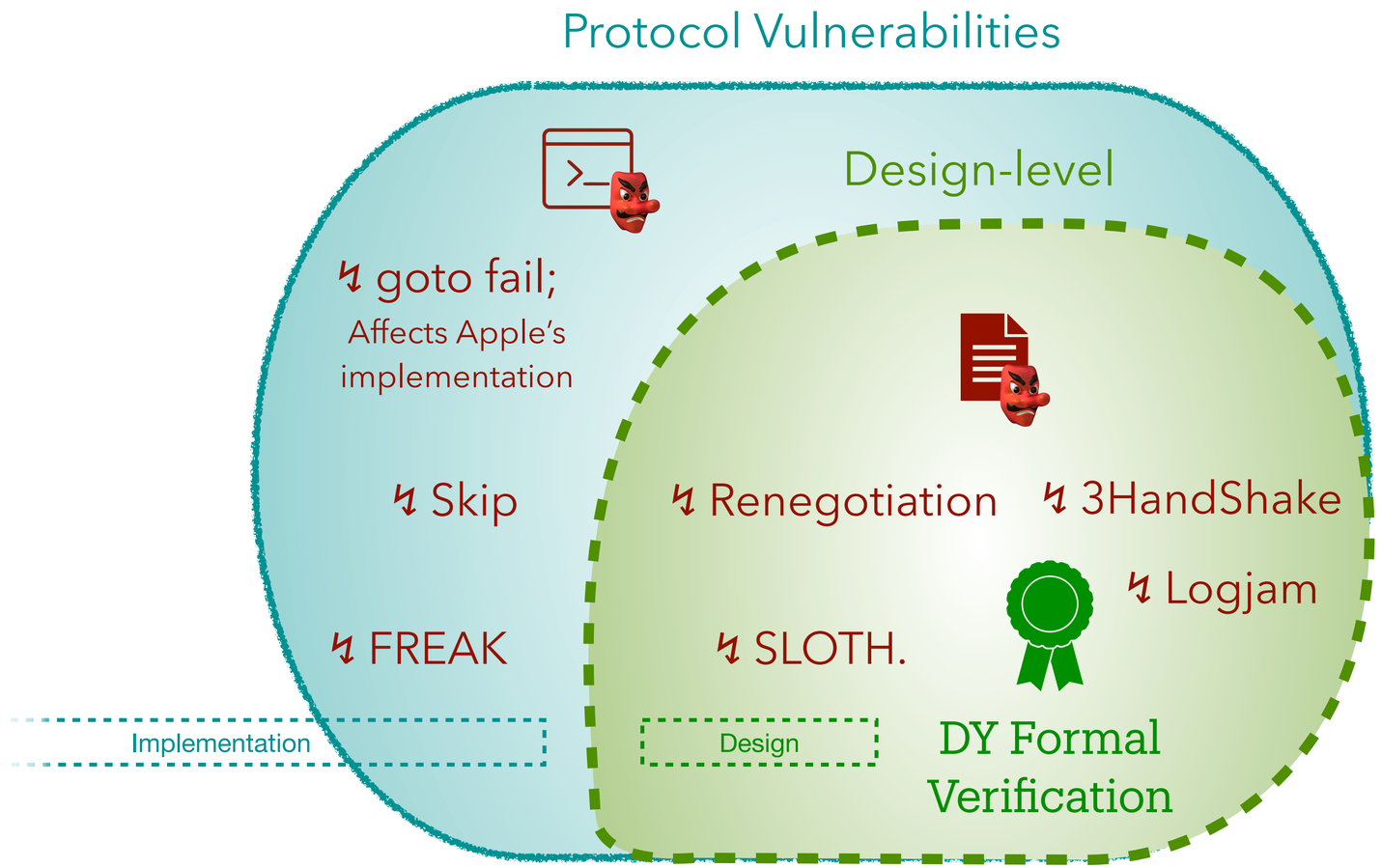
Beyond protocol specification analysis

Further illustrated on some attacks ↯ on TLS https://



Beyond protocol specification analysis

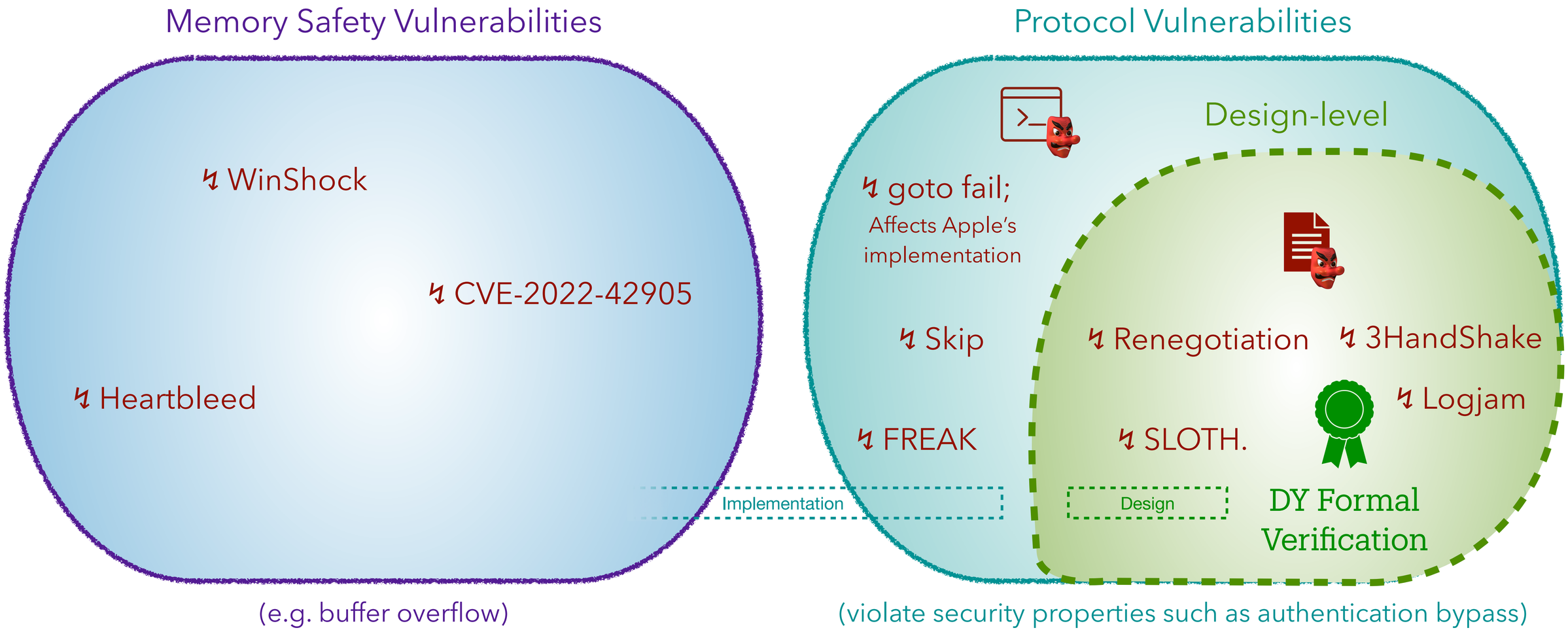
Further illustrated on some attacks ↯ on TLS https://



(violate security properties such as authentication bypass)

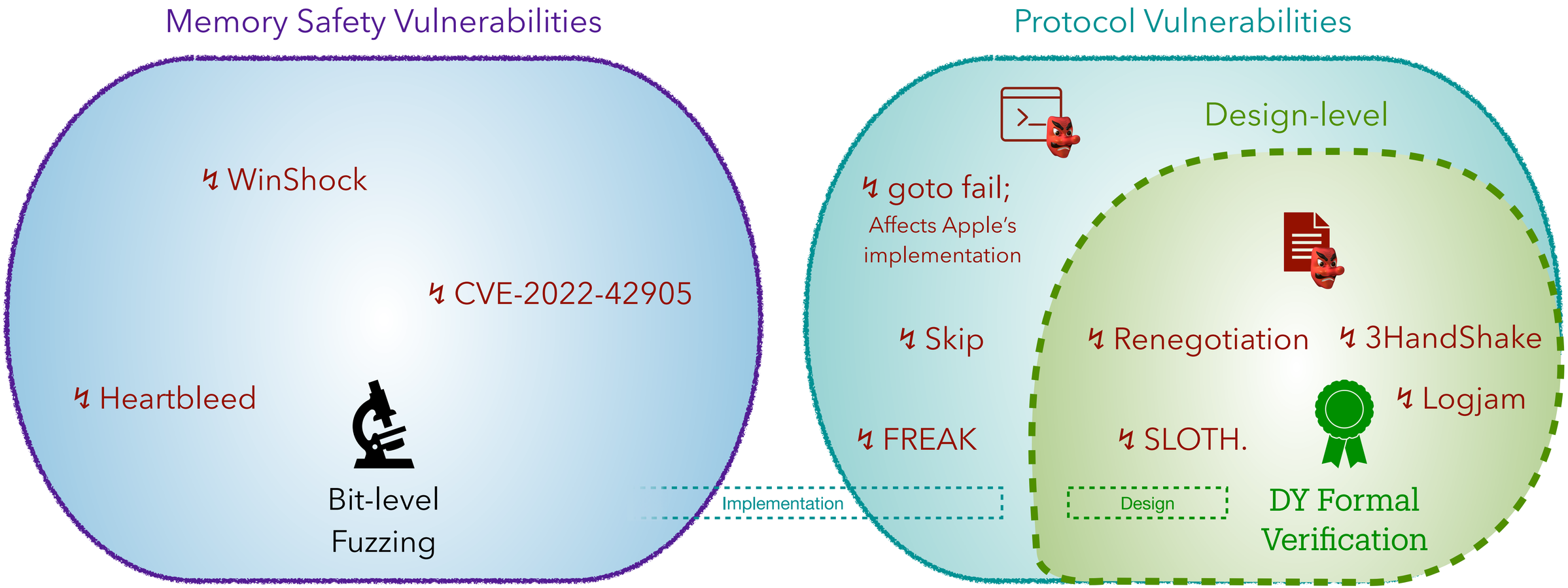
Beyond protocol specification analysis

Further illustrated on some attacks ↯ on TLS https://

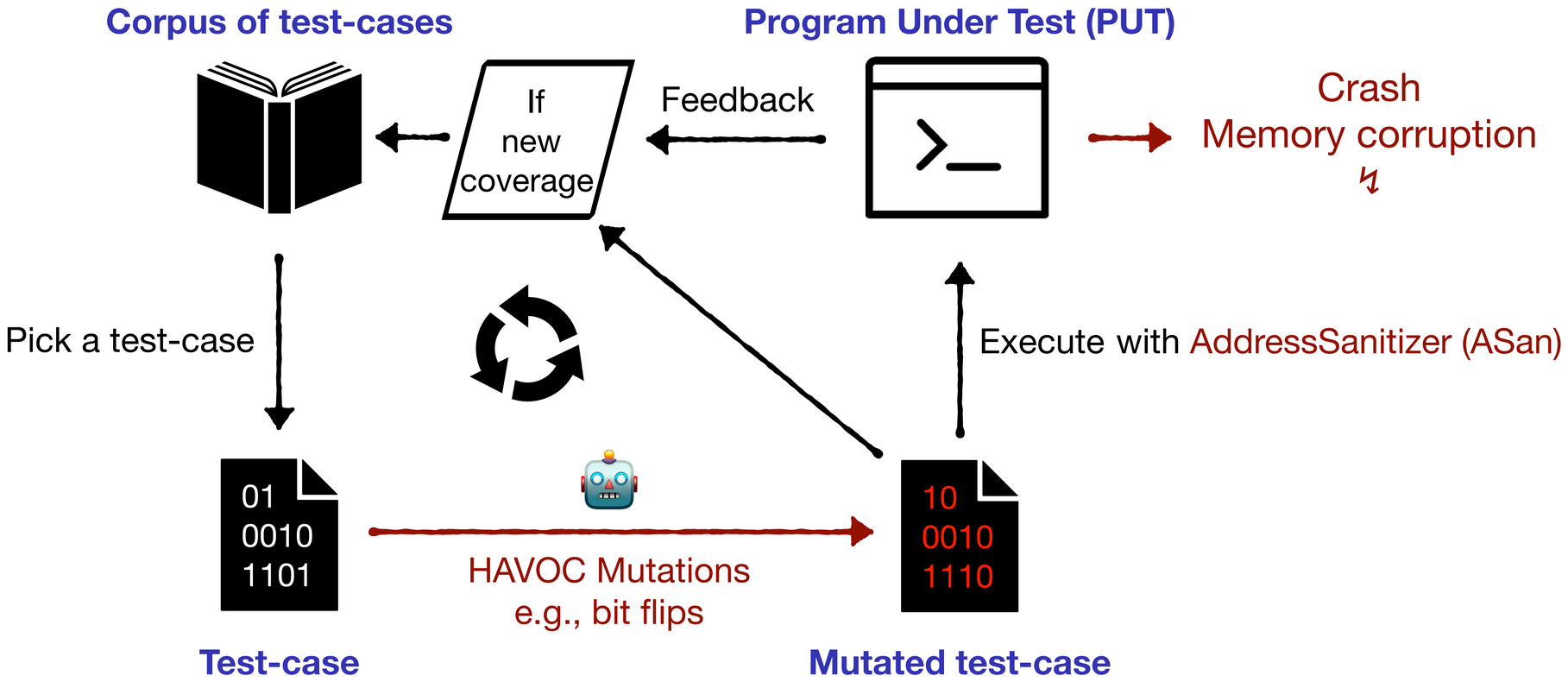


Beyond protocol specification analysis

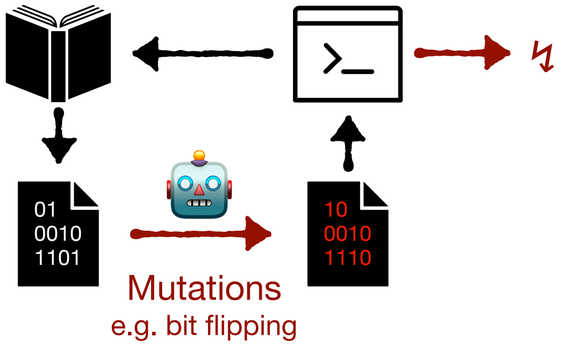
Further illustrated on some attacks ↯ on TLS https://



Bit-level Fuzzing (AFL-like)



Bit-level Fuzzing Limitations

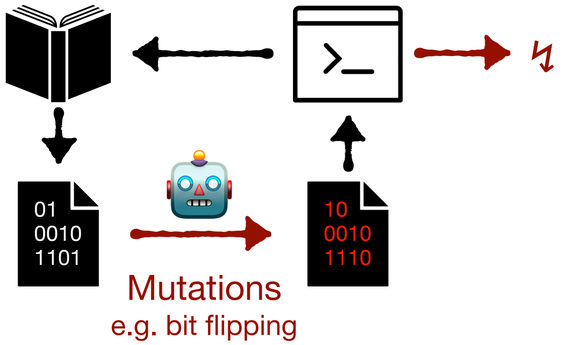



```
int FuzzerTestOneInput(const uint8_t *buf)
{
    server = SSL_new();
    SSL_set_bio(server);
    IO_write(in, buf);

    if (SSL_do_handshake(server) == 1) {
        /* Keep reading data */
        uint8_t tmp[1024];
        for (;;) {
            if (SSL_read(server, tmp) <= 0) {
                break;
            }
        }
    }
    return 0;
}
```

OpenSSL libFuzzer/AFL harness

Bit-level Fuzzing Limitations



Crypto protocols use crypto!

P[bit mutations 🤖: modify message
∧ adapt ECDSA signatures**]**
≤ **P[**break ECDSA**]**

Crypto protocols are interactive programs

Bit-level: 1 test-case = 1 bitstring

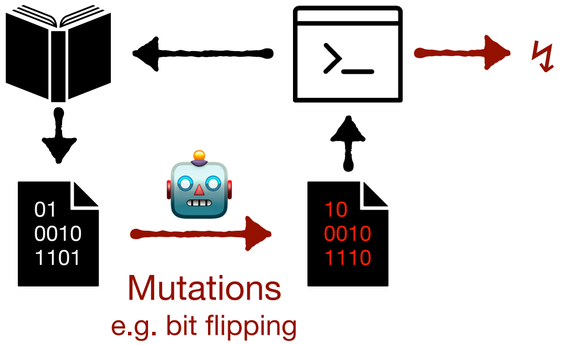
✗ Fuzzing Limitation 1: **Reachability**
No structural message and flow modification
☞ Fails to capture DY 👺 capabilities and logical attacks

```
int FuzzerTestOneInput(const uint8_t *buf)
{
    server = SSL_new();
    SSL_set_bio(server);
    IO_write(in, buf);

    if (SSL_do_handshake(server) == 1) {
        /* Keep reading data */
        uint8_t tmp[1024];
        for (;;) {
            if (SSL_read(server, tmp) <= 0) {
                break;
            }
        }
    }
    return 0;
}
```

OpenSSL libFuzzer/AFL harness

Bit-level Fuzzing Limitations



Crypto protocols use crypto!

P[bit mutations 🤖: modify message
∧ adapt ECDSA signatures**]**
≤ **P[**break ECDSA**]**

Crypto protocols are interactive programs

Bit-level: 1 test-case = 1 bitstring

✗ Fuzzing Limitation 1: **Reachability**
No structural message and flow modification
☞ Fails to capture DY 👺 capabilities and logical attacks

```c
int FuzzerTestOneInput(const uint8_t *buf)
{
    server = SSL_new();
    SSL_set_bio(server);
    IO_write(in, buf);

    if (SSL_do_handshake(server) == 1) {
        /* Keep reading data */
        uint8_t tmp[1024];
        for (;;) {
            if (SSL_read(server, tmp) <= 0) {
                break;
            }
        }
    }
    return 0;
}
```

OpenSSL libFuzzer/AFL harness

Crypto protocols aim for security >> mere safety

✗ Fuzzing Limitation 2: **Detection**
☞ Fails to detect protocol vulnerabilities

Memory Safety Vulnerabilities

Triggered by logical attacks

↯ Heartbleed



Bit-level Fuzzing

↯ CVE-2022-42905

(e.g. buffer overflow)

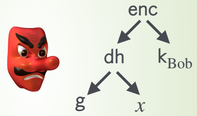
Protocol Vulnerabilities

↯ Skip

Design-level



DY Formal Verification



↯ goto fail;

↯ 3HandShake

Design

(e.g. authentication bypass)

Memory Safety Vulnerabilities

✗ Not reachable

Triggered by logical attacks

↯ Heartbleed



Bit-level Fuzzing

↯ CVE-2022-42905

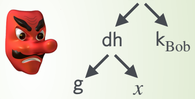
(e.g. buffer overflow)

Protocol Vulnerabilities

↯ Skip

Design-level

DY Formal Verification



↯ goto fail;

↯ 3HandShake

Design

(e.g. authentication bypass)

Memory Safety Vulnerabilities

✗ Not reachable

Triggered by logical attacks

↯ Heartbleed



Bit-level Fuzzing

↯ CVE-2022-42905

(e.g. buffer overflow)

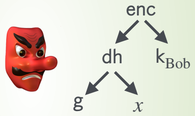
Protocol Vulnerabilities

✗ Not reachable

↯ Skip

Design-level

DY Formal Verification



↯ goto fail;

↯ 3HandShake

Design

(e.g. authentication bypass)

Memory Safety Vulnerabilities

✗ Not reachable

Triggered by logical attacks

↯ Heartbleed



Bit-level Fuzzing

↯ CVE-2022-42905

(e.g. buffer overflow)

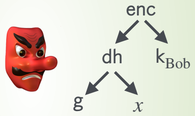
Protocol Vulnerabilities

✗ Not reachable
✗ Not detectable

↯ Skip

Design-level

DY Formal Verification



↯ goto fail;

↯ 3HandShake

Design

(e.g. authentication bypass)

Memory Safety Vulnerabilities

✗ Not reachable

Triggered by logical attacks

↯ Heartbleed



Bit-level Fuzzing

↯ CVE-2022-42905

(e.g. buffer overflow)



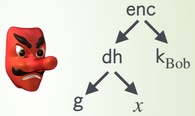
Protocol Vulnerabilities

✗ Not reachable
✗ Not detectable

↯ Skip

Design-level

DY Formal Verification



↯ goto fail;

↯ 3HandShake

Design

(e.g. authentication bypass)

Memory Safety Vulnerabilities

Protocol Vulnerabilities

✗ Not reachable

Triggered by logical attacks

✗ Not reachable
✗ Not detectable

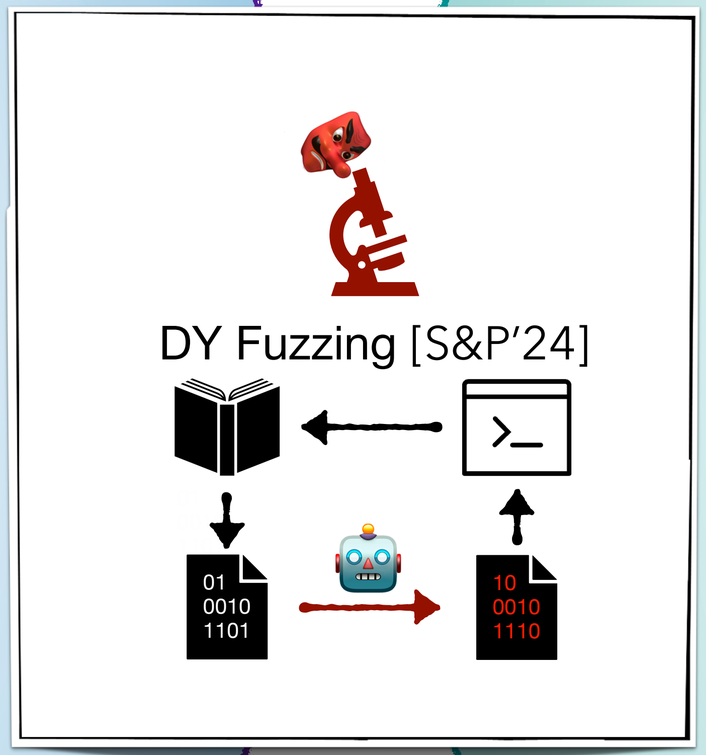
↯ Heartbleed

↯ Skip

Design-level

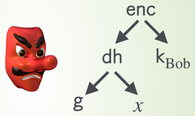
Bit-level Fuzzing







DY Formal Verification



↯ CVE-2022-42905

↯ 3HandShake

Design

(e.g. buffer overflow)

(e.g. authentication bypass)

Memory Safety Vulnerabilities

Protocol Vulnerabilities

✗ Not reachable

Triggered by logical attacks

✗ Not reachable
✗ Not detectable

↯ Skip

↯ Heartbleed

Design-level

Bit-level Fuzzing

DY Fuzzing [S&P'24]

01
0010
1101

10
0010
1110

DY Formal Verification

enc
dh
k_{Bob}
g
x

DY attacker model

↯ 3HandShake

Design

↯ CVE-2022-42905

(e.g. buffer overflow)

(e.g. authentication bypass)

Memory Safety Vulnerabilities

Protocol Vulnerabilities

✗ Not reachable

Triggered by logical attacks

✗ Not reachable
✗ Not detectable

↯ Heartbleed

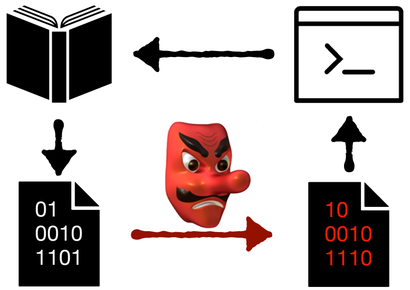
↯ Skip

Design-level

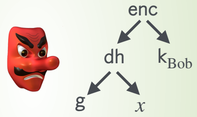
Bit-level Fuzzing



DY Fuzzing [S&P'24]



DY Formal Verification



↯ CVE-2022-42905

↯ 3HandShake

Design

(e.g. buffer overflow)

(e.g. authentication bypass)

Memory Safety Vulnerabilities

Protocol Vulnerabilities

✗ Not reachable

Triggered by logical attacks

✗ Not reachable
✗ Not detectable

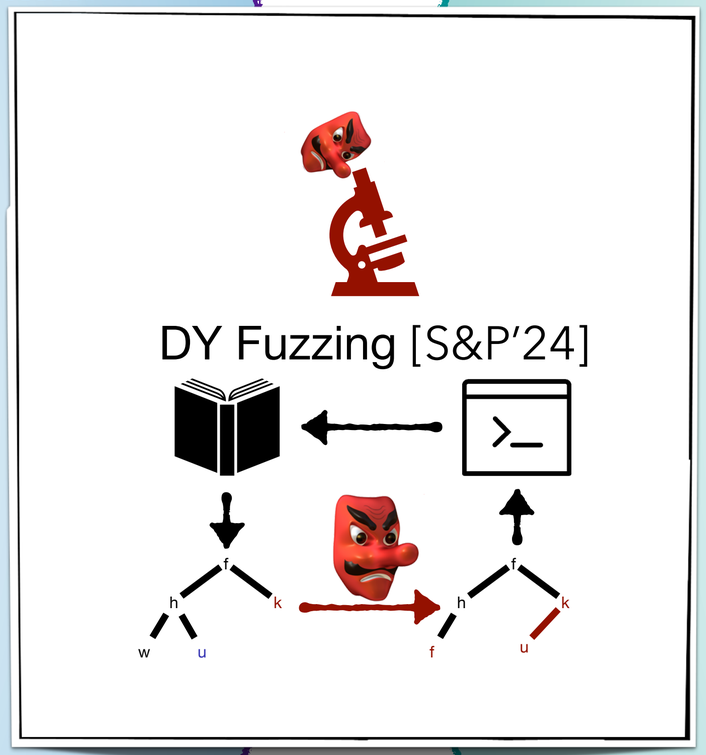
↯ Heartbleed

↯ Skip

Design-level

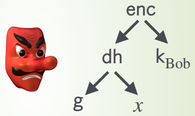


Bit-level Fuzzing





DY Formal Verification



↯ CVE-2022-42905

↯ 3HandShake

Design

(e.g. buffer overflow)

(e.g. authentication bypass)

Memory Safety Vulnerabilities

Protocol Vulnerabilities

✓ Reachable

Triggered by logical attacks

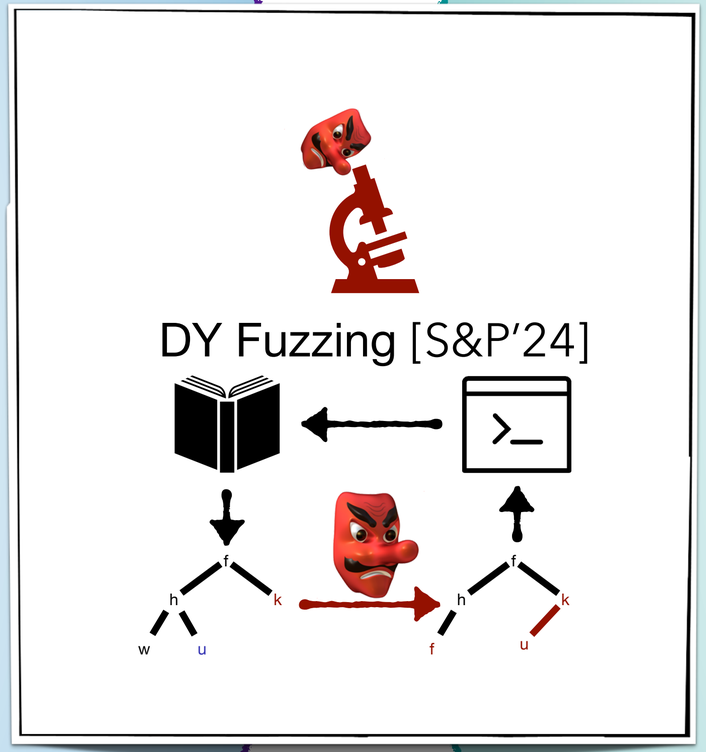
✓ Reachable
✓ Detectable

↯ Heartbleed

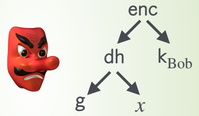
↯ Skip

Design-level

Bit-level Fuzzing



DY Formal Verification



↯ CVE-2022-42905

↯ 3HandShake

Design

DY Fuzzer = DY attacker in a fuzzing loop

(e.g. buf

on bypass)

Memory Safety Vulnerabilities

Protocol Vulnerabilities

✓ Reachable

Triggered by logical attacks

✓ Reachable
✓ Detectable

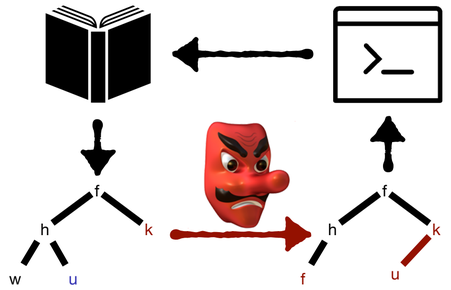
↯ Heartbleed

↯ Skip

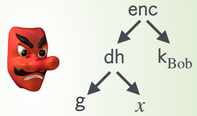
Design-level

Bit-level Fuzzing

DY Fuzzing [S&P'24]



DY Formal Verification



↯ CVE-2022-42905

↯ 3HandShake

Design

(e.g. buf

on bypass)

DY Fuzzer = DY attacker in a fuzzing loop

Memory Safety Vulnerabilities

Protocol Vulnerabilities

✓ Reachable

Triggered by logical attacks

✓ Reachable
✓ Detectable

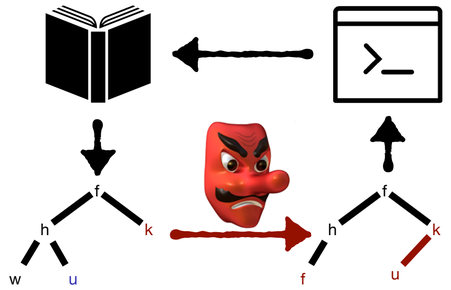
↯ Heartbleed

↯ Skip

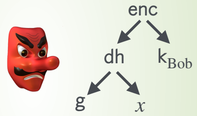
Design-level

Bit-level Fuzzing

DY Fuzzing [S&P'24]



DY Formal Verification



↯ CVE-2022-42905

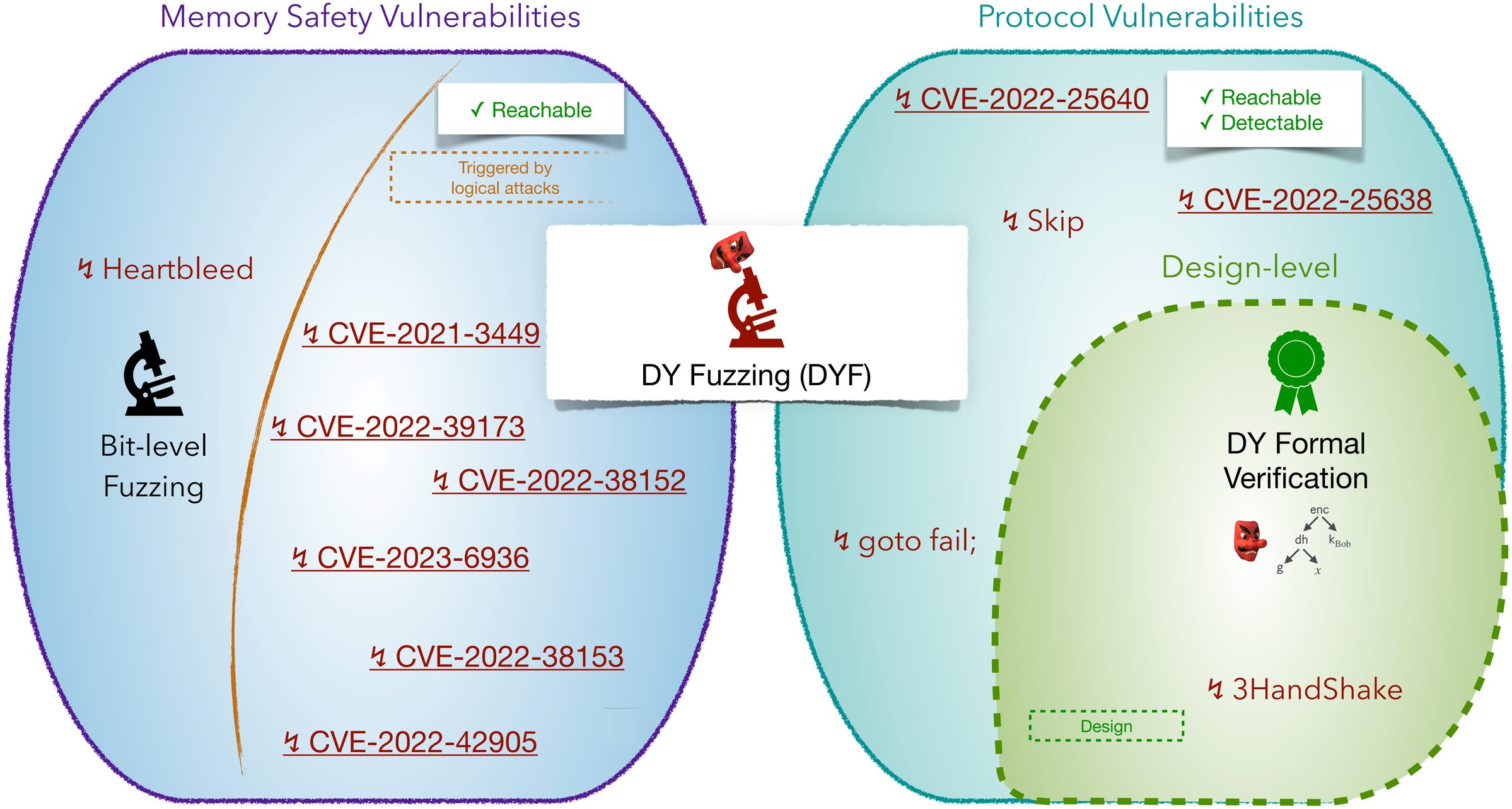
↯ 3HandShake

Design

(e.g. buf

on bypass)

DY Fuzzer = DY attacker in a fuzzing loop


Memory Safety Vulnerabilities
Protocol Vulnerabilities
✓ Reachable
Triggered by logical attacks
↯ Heartbleed
Bit-level Fuzzing
↯ CVE-2021-3449
↯ CVE-2022-39173
↯ CVE-2022-38152
↯ CVE-2023-6936
↯ CVE-2022-38153
↯ CVE-2022-42905
DY Fuzzing (DYF)
↯ CVE-2022-25640
✓ Reachable
✓ Detectable
↯ CVE-2022-25638
↯ Skip
Design-level
DY Formal Verification
enc
dh
k_{Bob}
g
x
↯ goto fail;
↯ 3HandShake
Design
(e.g. buffer-overflow)
(e.g. authentication bypass)

DY Fuzzing Design: Traces and Mapper

Trace: expresses one possible DY 👺 behavior = 1 input for DY Fuzzing

$\texttt{out}(a_1, x_1).\texttt{in}(a_2, R_2).\texttt{out}\ldots$ where $a_i \in \mathcal{A}$ (agent), $R_i \in \mathcal{T}$ (*attacker term*), $x_i \in \mathcal{X}$ (*attacker knowledge*)

DY Fuzzing Design: Traces and Mapper

Trace: expresses one possible DY 👺 behavior = 1 input for DY Fuzzing

$\texttt{out}(a_1, x_1).\texttt{in}(a_2, R_2).\texttt{out}\ldots$ where $a_i \in \mathcal{A}$ (agent), $R_i \in \mathcal{T}$ (*attacker term*), $x_i \in \mathcal{X}$ (*attacker knowledge*)

Mapper: PUTs expect bitstrings $[u8]$ rather than terms: *DY terms ↔ bitstrings*

DY Fuzzing Design: Traces and Mapper

Trace: expresses one possible DY 👺 behavior = 1 input for DY Fuzzing

`out`(a_1, x_1).`in`(a_2, R_2).`out`... where $a_i \in \mathcal{A}$ (agent), $R_i \in \mathcal{T}$ (*attacker term*), $x_i \in \mathcal{X}$ (*attacker knowledge*)

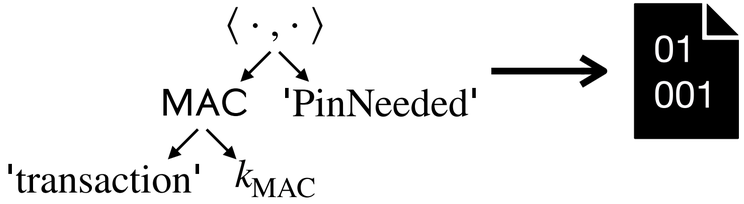
Mapper: PUTs expect bitstrings $[u8]$ rather than terms: *DY terms ↔ bitstrings*

- *Concretization:* for the current attacker state $\phi = \{x_1 \mapsto m_1, \ldots\}, x_i \in \mathcal{X}, m_i \in [\text{u8}]$

  $[\![\text{f}]\!] : [\text{u8}]^i \to [\text{u8}]$ // $[\![\text{MAC}]\!]$:= `openssl.crypto.hmac`

  $[\![\cdot]\!]_\phi : \mathcal{T} \to [\text{u8}]$ // $[\![MAC(t_1, t_2)]\!]_\phi$ = `openssl.crypto.hmac`$([\![t_1]\!]_\phi, [\![t_2]\!]_\phi)$

  // $[\![x]\!]_\phi = x\phi$



DY Fuzzing Design: Traces and Mapper

Trace: expresses one possible DY 😈 behavior = 1 input for DY Fuzzing

$\texttt{out}(a_1, x_1).\texttt{in}(a_2, R_2).\texttt{out}\ldots$ where $a_i \in \mathcal{A}$ (agent), $R_i \in \mathcal{T}$ (*attacker term*), $x_i \in \mathcal{X}$ (*attacker knowledge*)

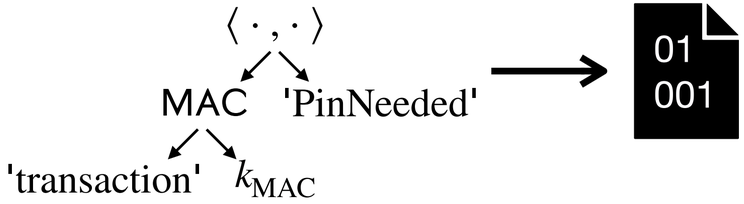
Mapper: PUTs expect bitstrings $[u8]$ rather than terms: *DY terms ↔ bitstrings*

- *Concretization:* for the current attacker state $\phi = \{x_1 \mapsto m_1, \ldots\}, x_i \in \mathcal{X}, m_i \in [\text{u8}]$

  $[\![\text{f}]\!] : [\text{u8}]^i \to [\text{u8}]$ // $[\![\text{MAC}]\!] := \texttt{openssl.crypto.hmac}$

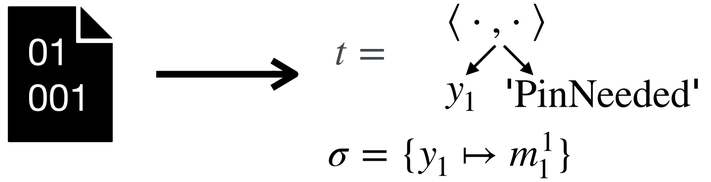
  $[\![\cdot]\!]_\phi : \mathcal{T} \to [\text{u8}]$ // $[\![MAC(t_1, t_2)]\!]_\phi = \texttt{openssl.crypto.hmac}([\![t_1]\!]_\phi, [\![t_2]\!]_\phi)$

  // $[\![x]\!]_\phi = x\phi$



- *Partial parsing*: may not fully parse bitstrings (best effort)

  $\lceil \cdot \rceil : [\text{u8}] \to \mathcal{T} \times (\mathcal{X} \mapsto [\text{u8}])$



DY Fuzzing Design: Fuzzing Loop

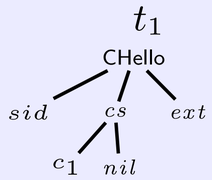
Corpus

DY traces: $T, T', \ldots$

Seed corpus (DSL): $T_0, T_0', \ldots$



Mutations



$T = \mathsf{in}(s, t_1) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$

schedule T

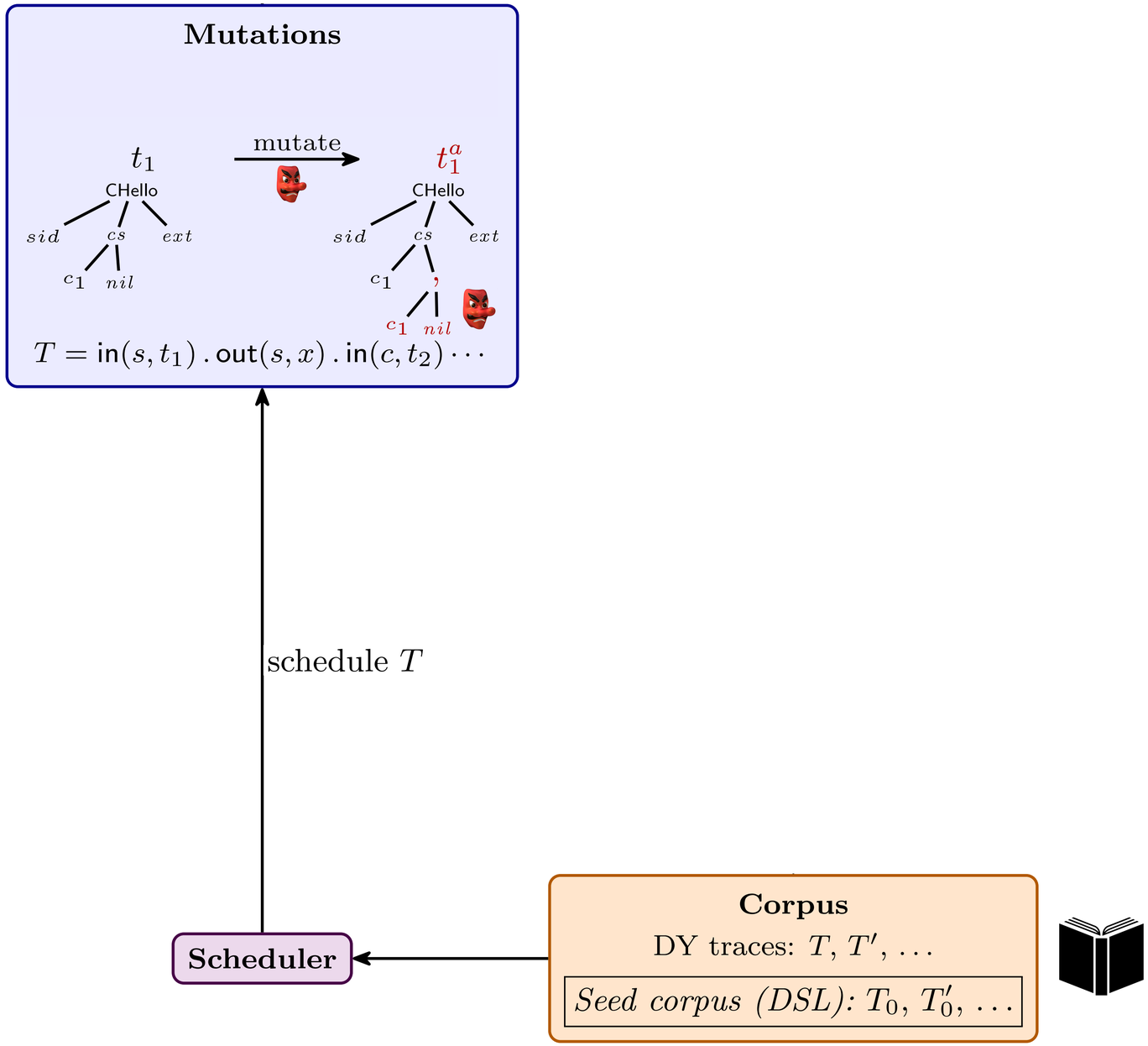
Scheduler

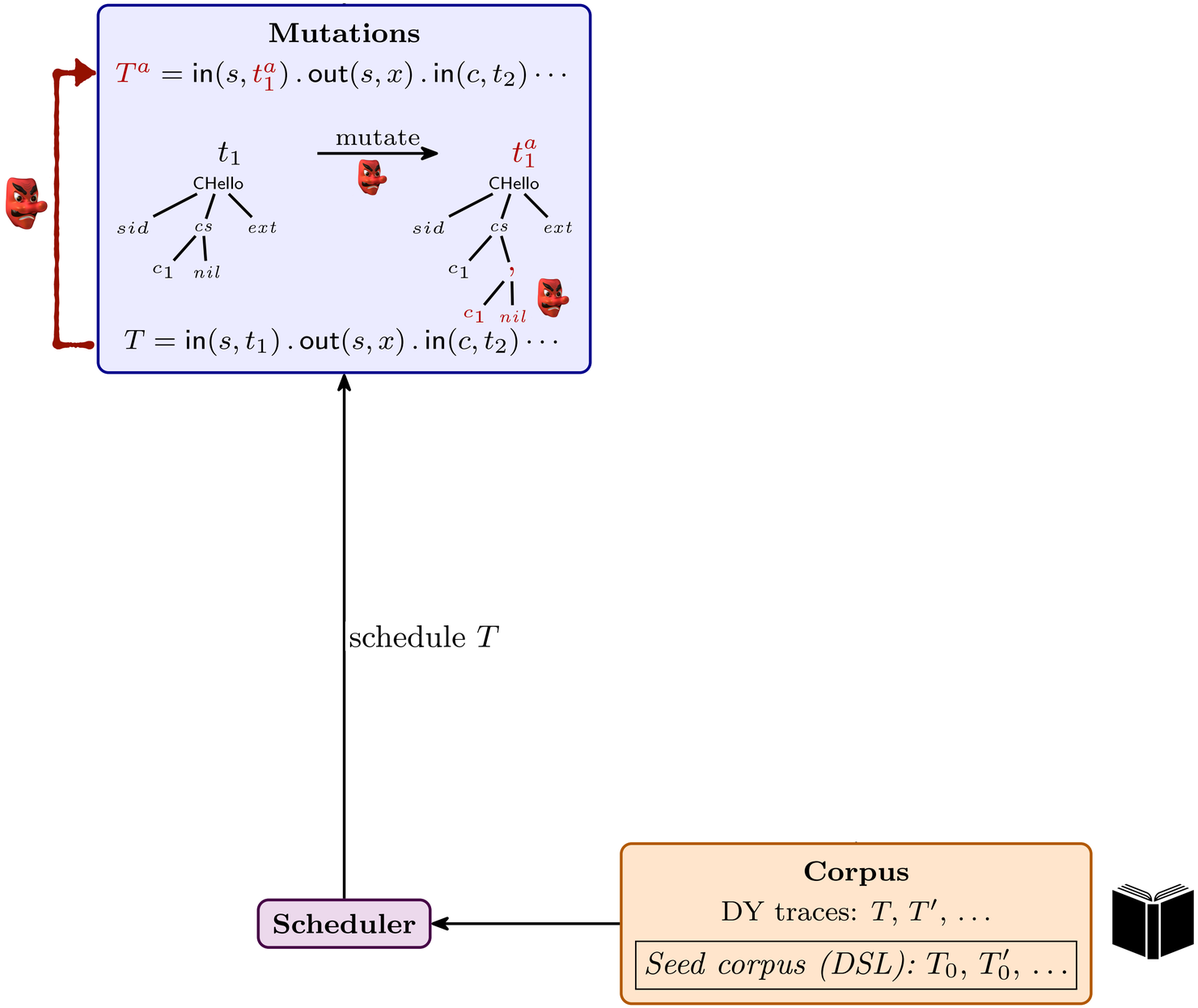
Corpus

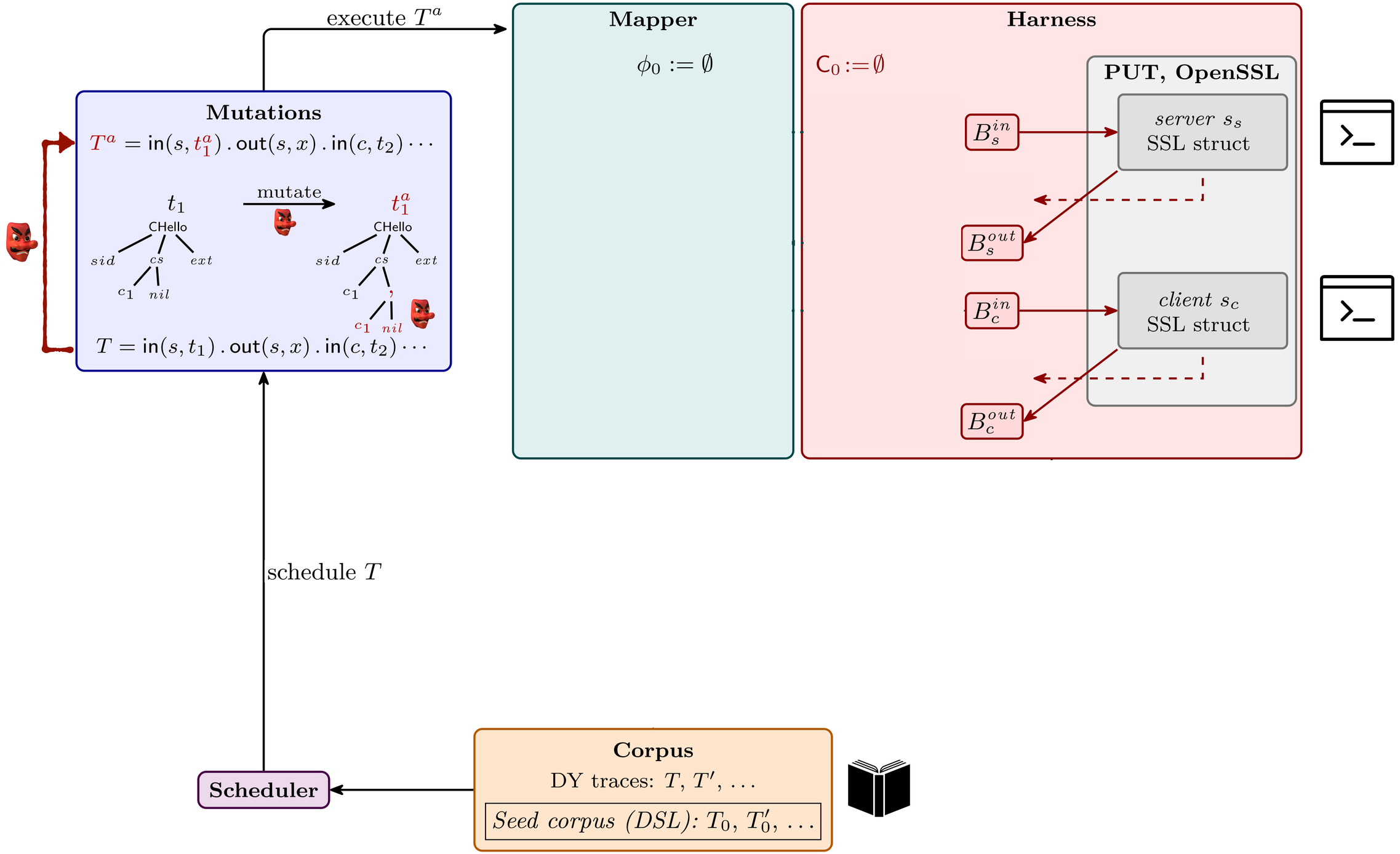
DY traces: T, T', ...

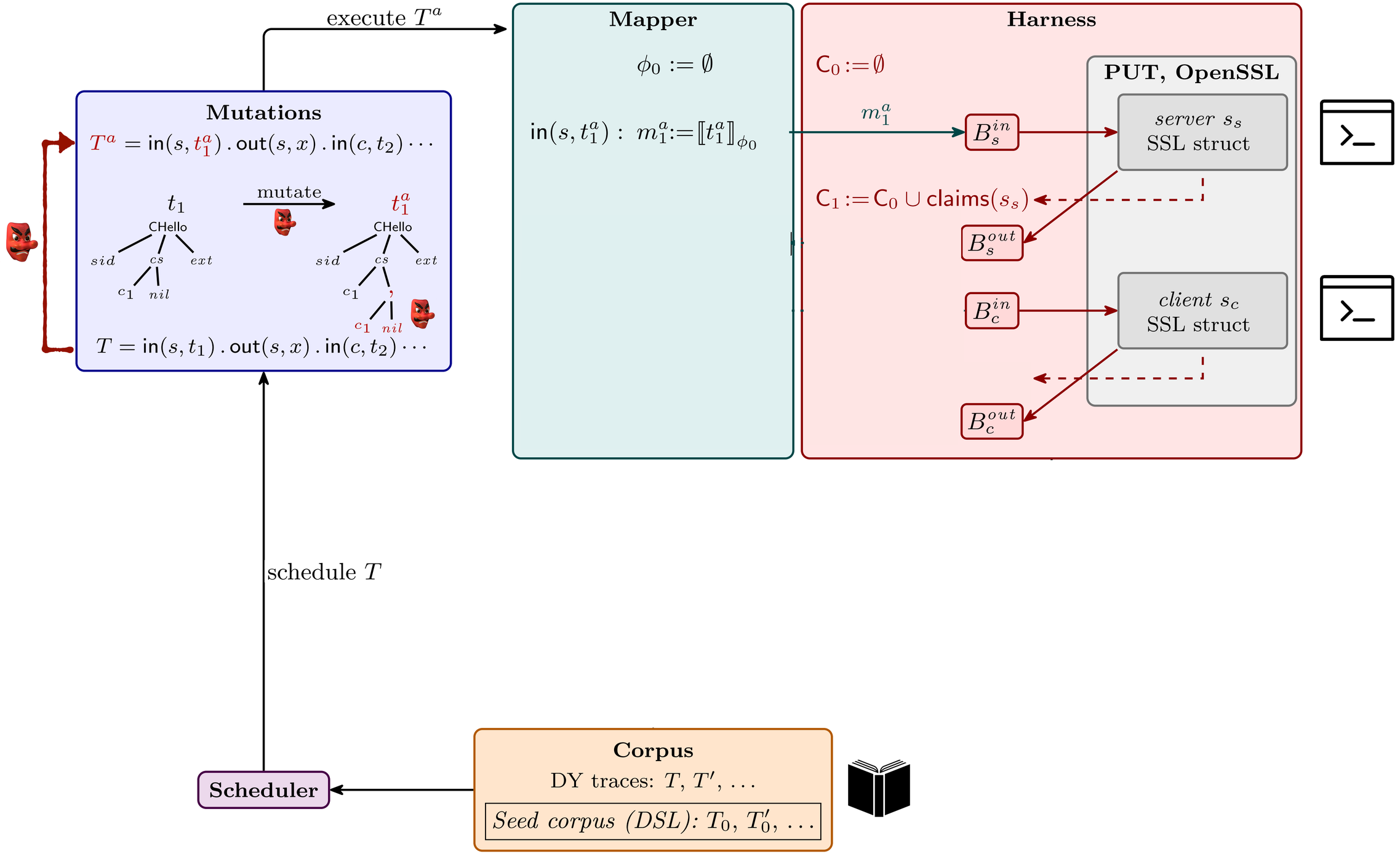
Seed corpus (DSL): T_0, T_0', ...

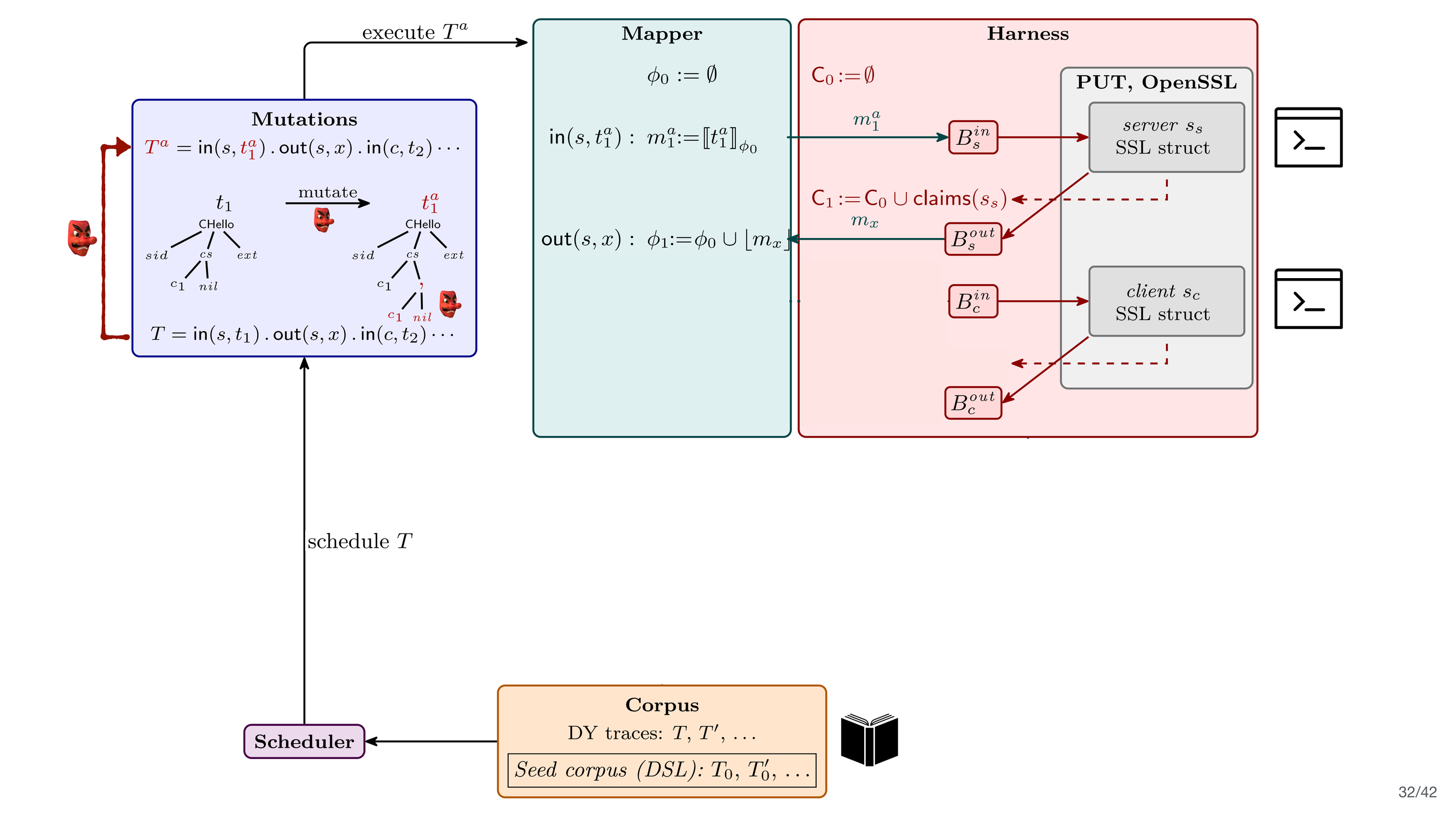


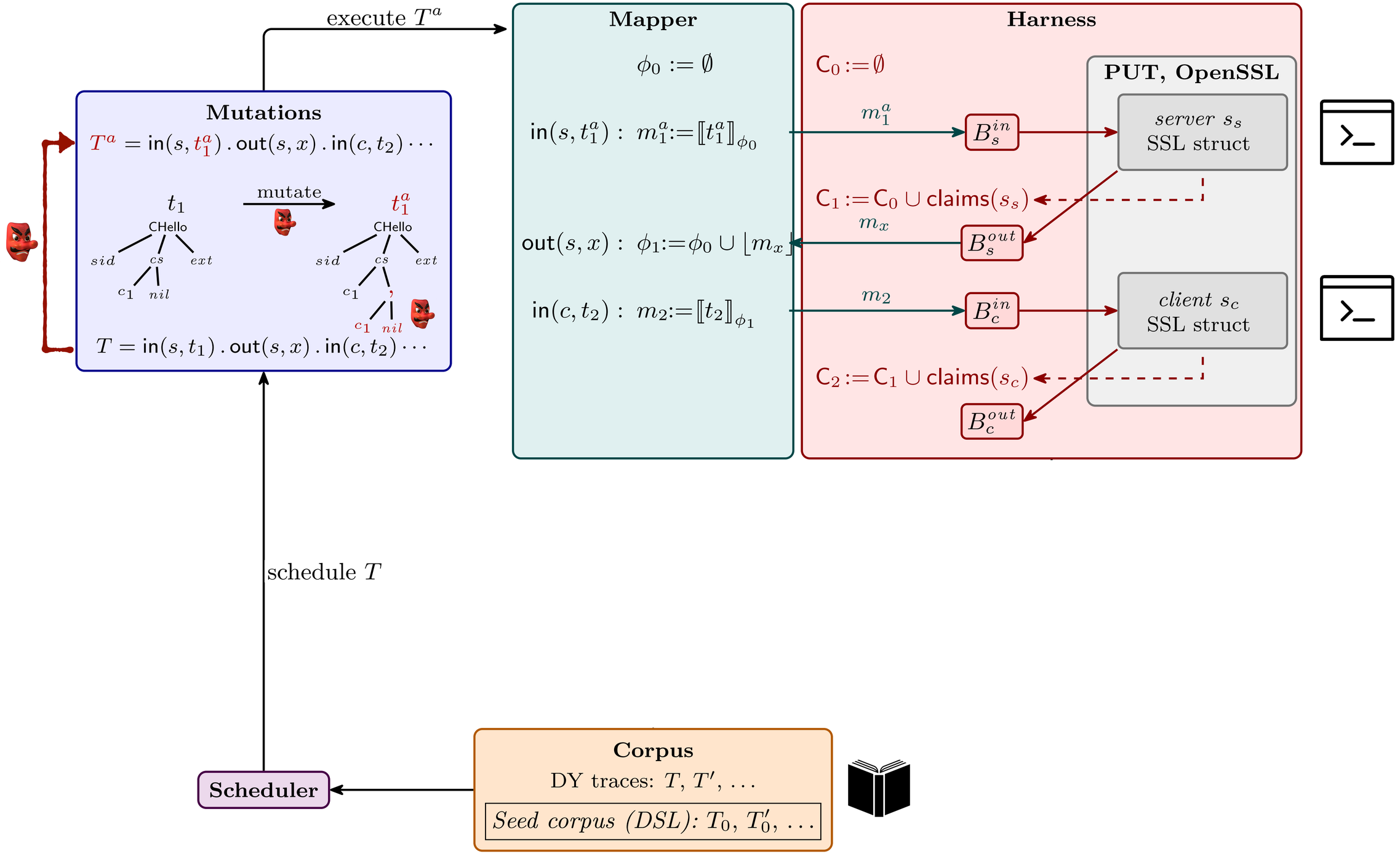

Mutations
t_1
CHello
sid
cs
ext
c_1
nil
mutate
t_1^a
CHello
sid
cs
ext
c_1
,
c_1
nil
$T = \mathsf{in}(s, t_1) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
schedule T
Scheduler
Corpus
DY traces: T, T', ...
Seed corpus (DSL): T_0, T_0', ...

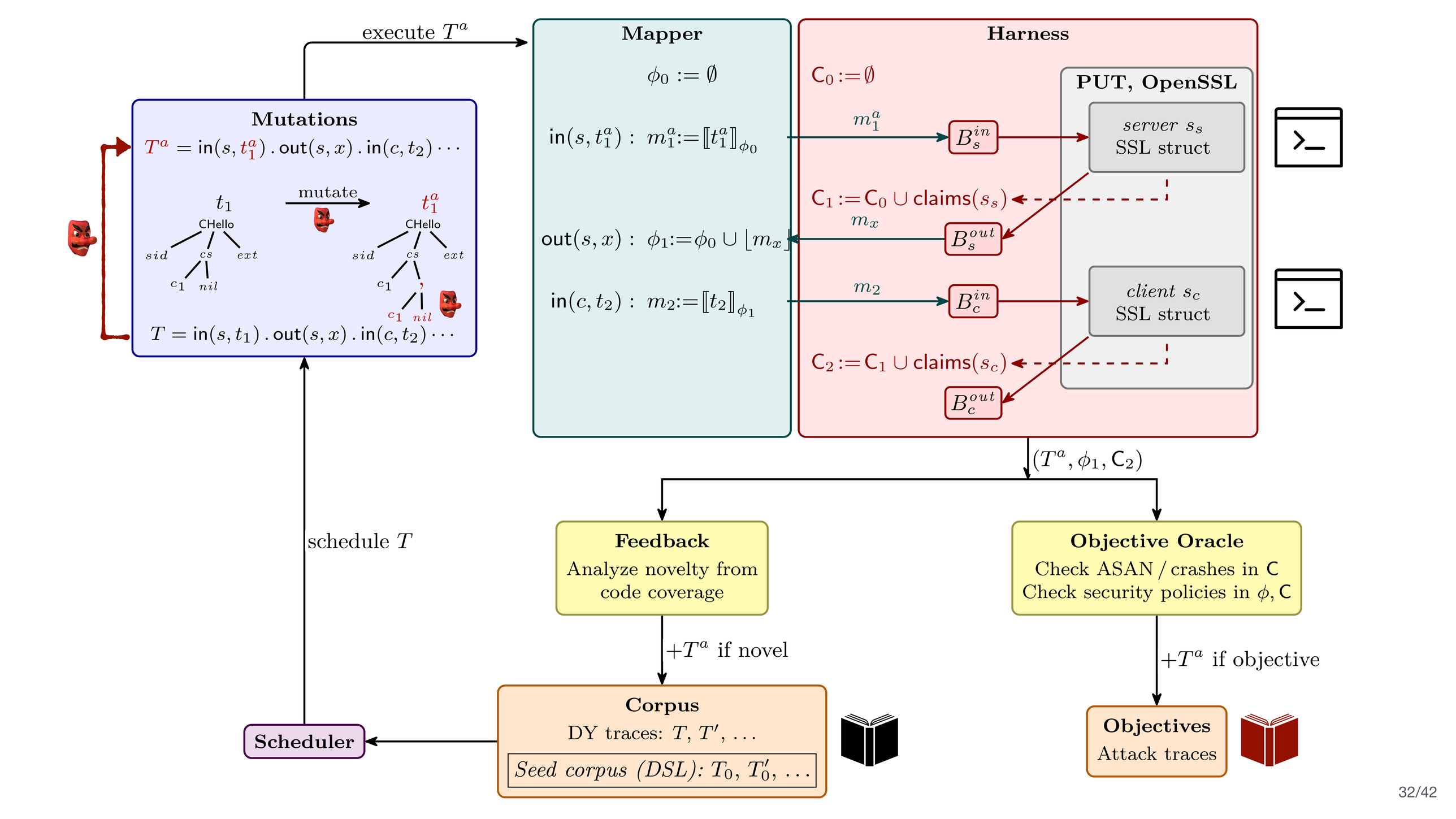

Mutations
$T^a = \mathsf{in}(s, t_1^a) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
t_1
CHello
sid
cs
ext
c_1
nil
mutate
t_1^a
CHello
sid
cs
ext
c_1
,
c_1
nil
$T = \mathsf{in}(s, t_1) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
schedule T
Scheduler
Corpus
DY traces: T, T', ...
Seed corpus (DSL): T_0, T_0', ...

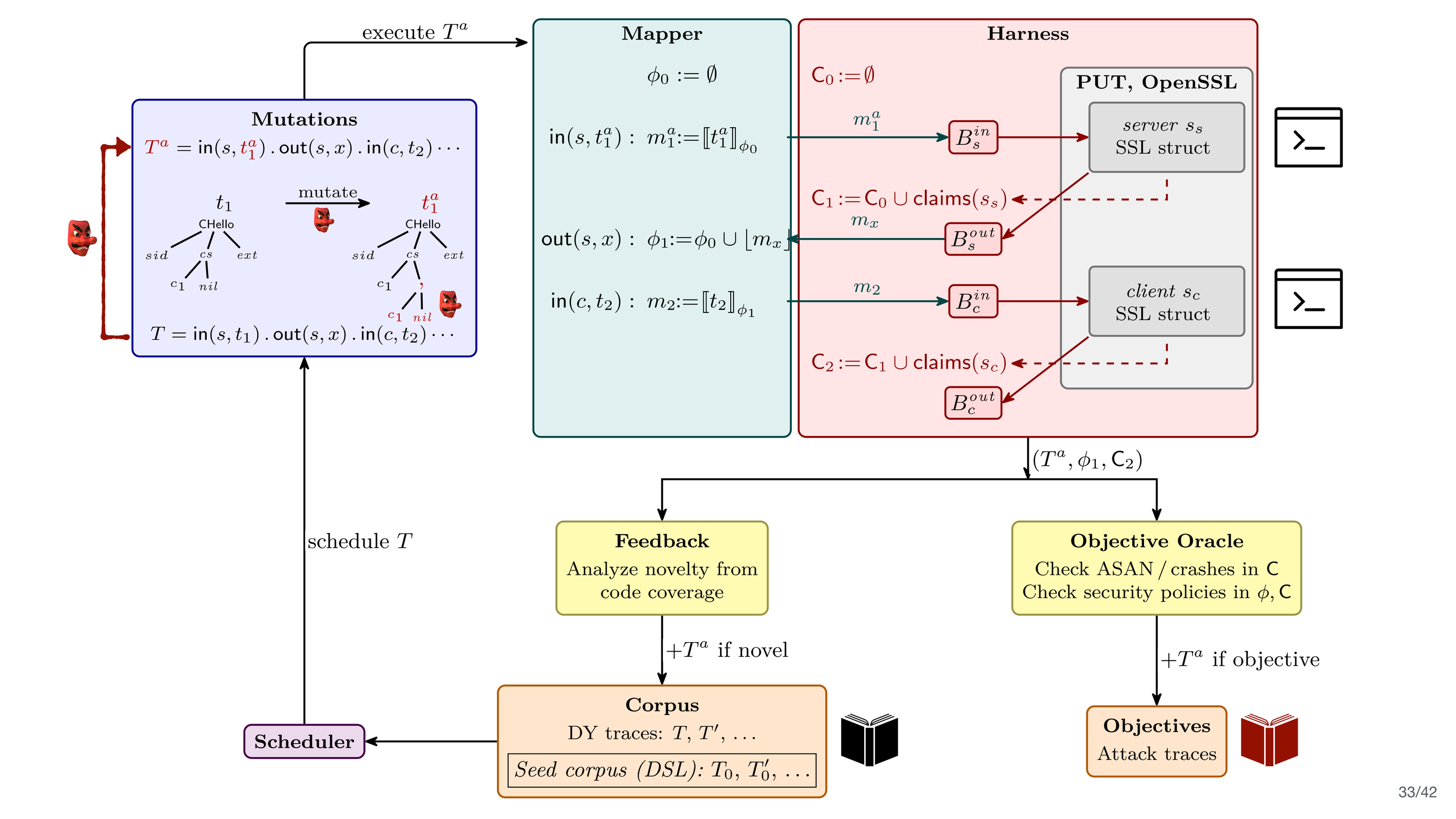

execute T^a
Mapper
$\phi_0 := \emptyset$
Harness
$C_0 := \emptyset$
PUT, OpenSSL
server s_s
SSL struct
client s_c
SSL struct
B_s^{in}
B_s^{out}
B_c^{in}
B_c^{out}
Mutations
$T^a = \mathsf{in}(s, t_1^a) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
t_1
CHello
sid
cs
ext
c_1
nil
mutate
t_1^a
CHello
sid
cs
ext
c_1
,
c_1
nil
$T = \mathsf{in}(s, t_1) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
schedule T
Scheduler
Corpus
DY traces: T, T', ...
Seed corpus (DSL): T_0, T_0', ...

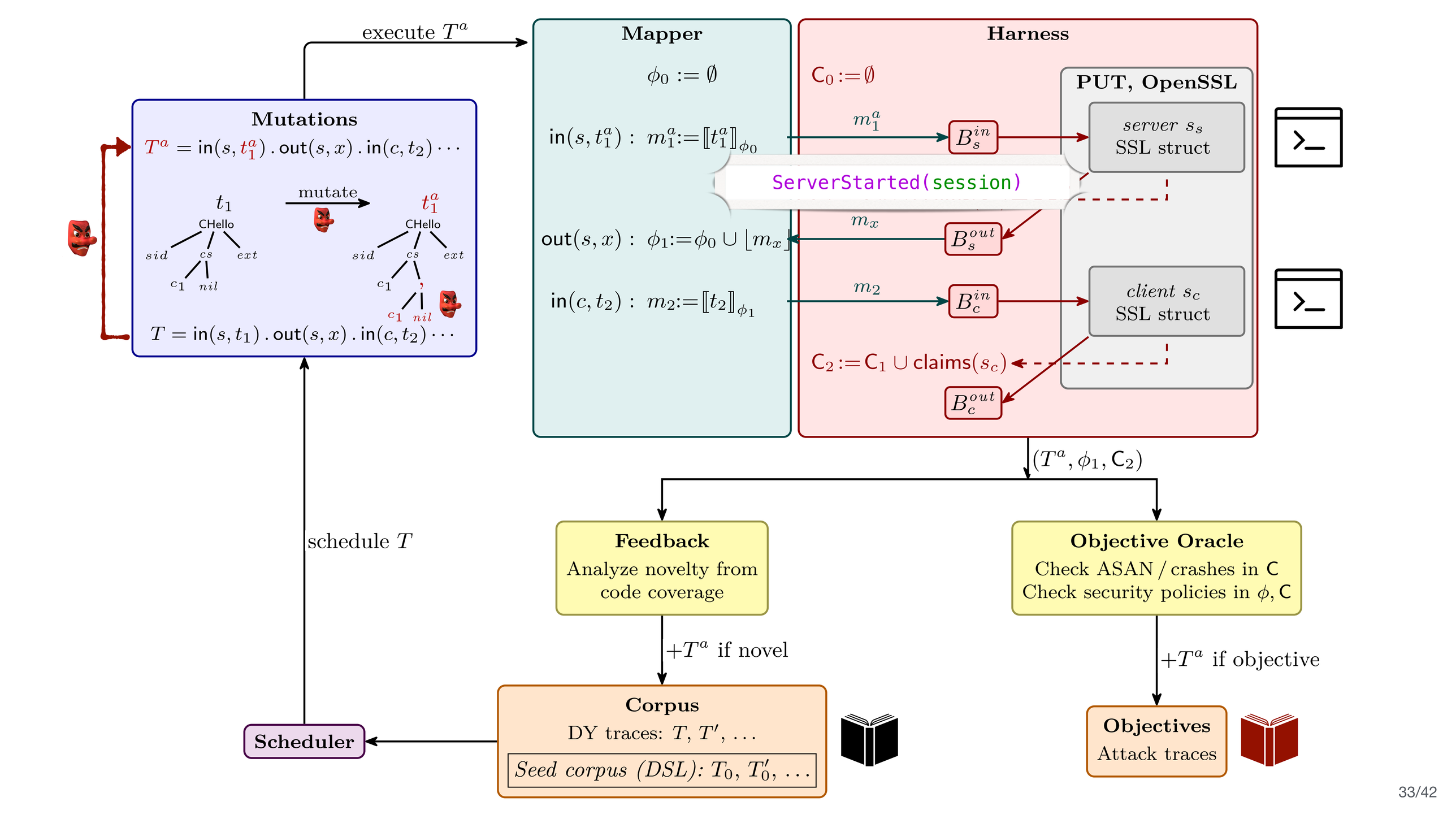

execute T^a
Mapper
$\phi_0 := \emptyset$
$\mathsf{in}(s, t_1^a) : \; m_1^a := [\![t_1^a]\!]_{\phi_0}$
Harness
$\mathsf{C}_0 := \emptyset$
m_1^a
B_s^{in}
PUT, OpenSSL
server s_s
SSL struct
$\mathsf{C}_1 := \mathsf{C}_0 \cup \mathsf{claims}(s_s)$
B_s^{out}
B_c^{in}
client s_c
SSL struct
B_c^{out}
Mutations
$T^a = \mathsf{in}(s, t_1^a) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
t_1
CHello
sid
cs
ext
c_1
nil
mutate
t_1^a
CHello
sid
cs
ext
c_1
,
c_1
nil
$T = \mathsf{in}(s, t_1) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
schedule T
Scheduler
Corpus
DY traces: T, T', ...
Seed corpus (DSL): T_0, T_0', ...

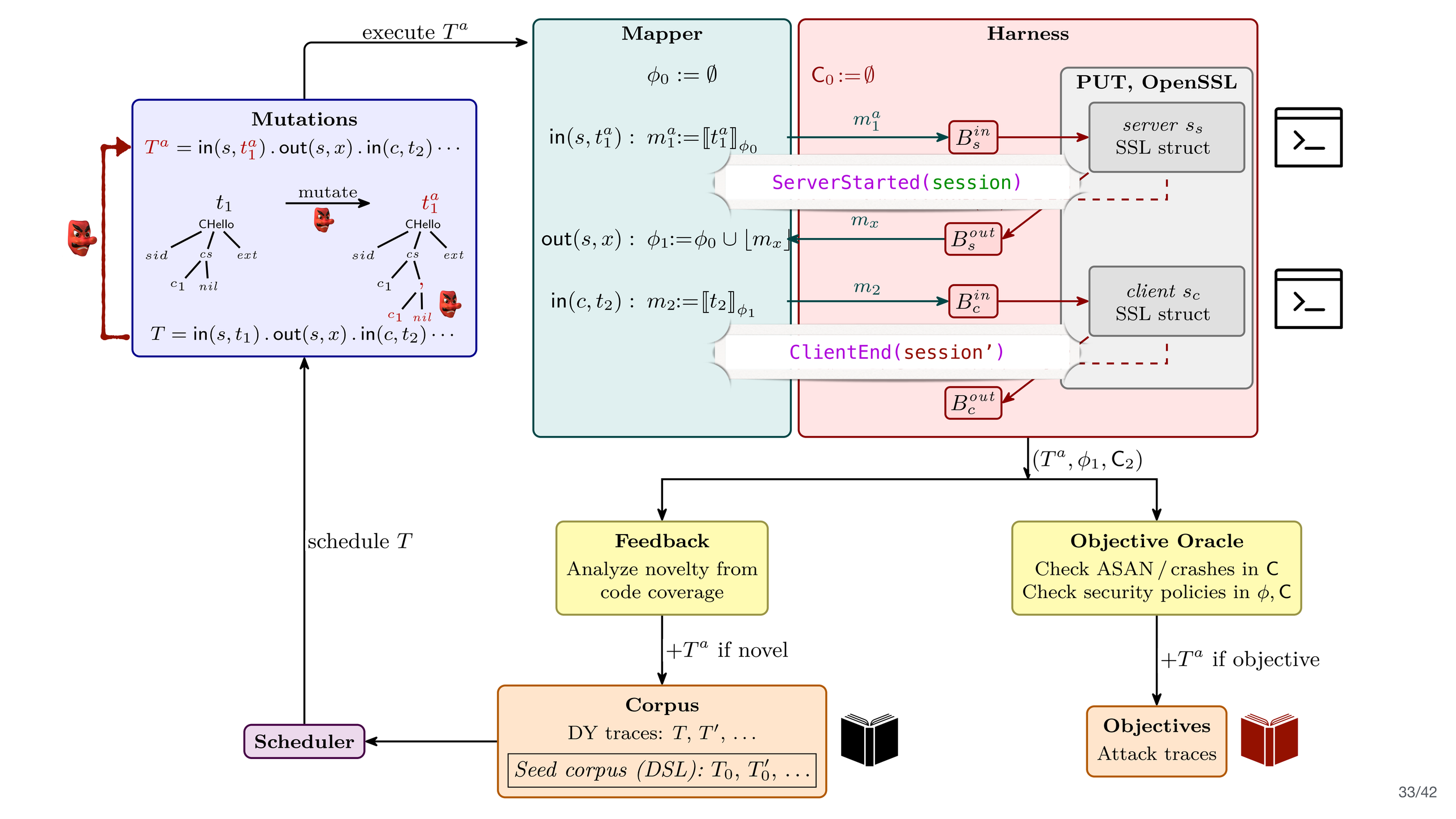

execute T^a
Mapper
$\phi_0 := \emptyset$
$\mathsf{in}(s, t_1^a)$: $m_1^a := [\![t_1^a]\!]_{\phi_0}$
$\mathsf{out}(s, x)$: $\phi_1 := \phi_0 \cup \lfloor m_x \rfloor$
Harness
$\mathsf{C}_0 := \emptyset$
m_1^a
B_s^{in}
PUT, OpenSSL
server s_s
SSL struct
$\mathsf{C}_1 := \mathsf{C}_0 \cup \mathsf{claims}(s_s)$
m_x
B_s^{out}
B_c^{in}
client s_c
SSL struct
B_c^{out}
Mutations
$T^a = \mathsf{in}(s, t_1^a) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
t_1
CHello
sid
cs
ext
c_1
nil
mutate
t_1^a
CHello
sid
cs
ext
c_1
,
c_1
nil
$T = \mathsf{in}(s, t_1) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
schedule T
Scheduler
Corpus
DY traces: T, T', ...
Seed corpus (DSL): T_0, T_0', ...

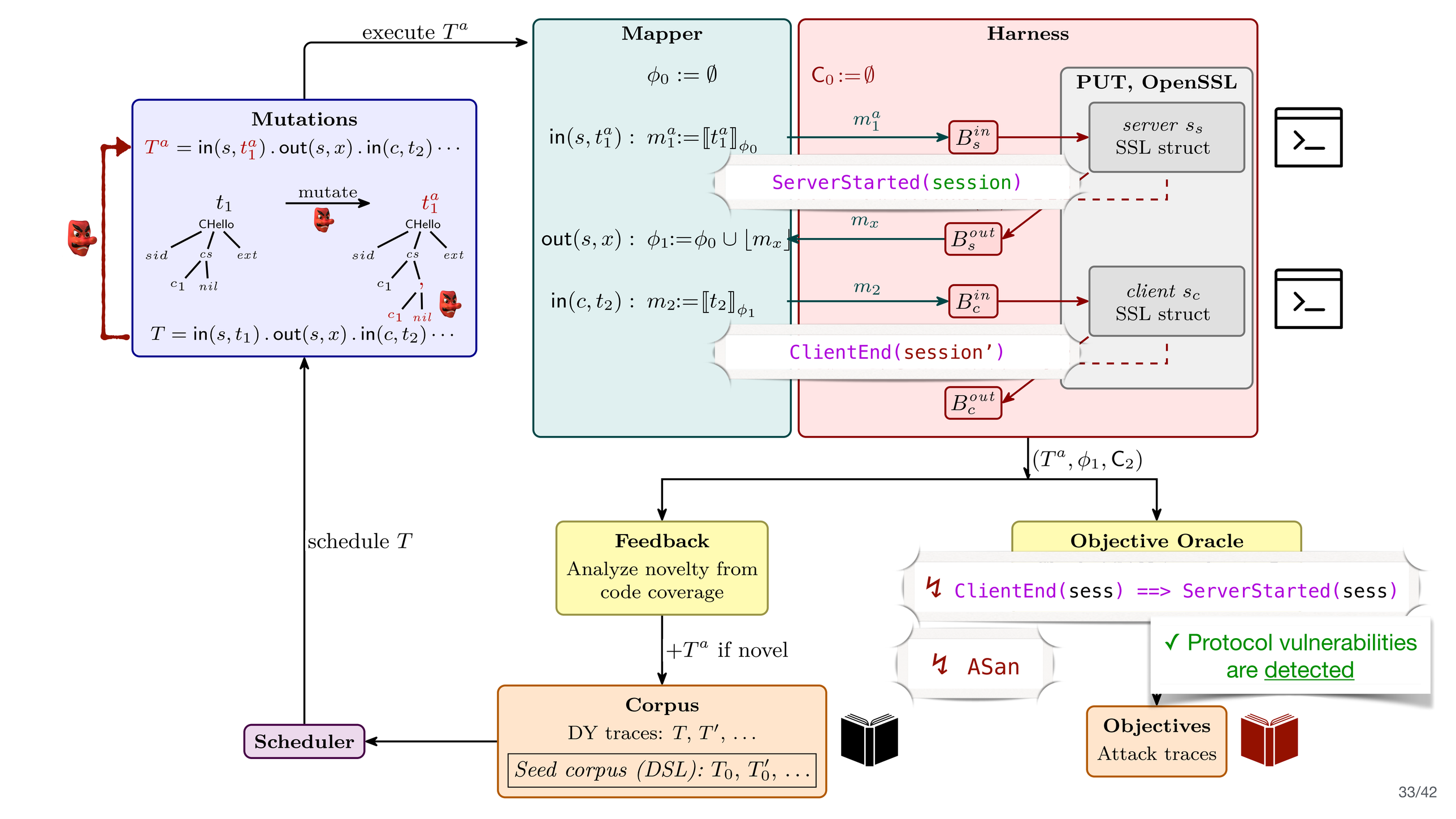

execute T^a
Mapper
$\phi_0 := \emptyset$
$\mathsf{in}(s, t_1^a)$: $m_1^a := [\![t_1^a]\!]_{\phi_0}$
$\mathsf{out}(s, x)$: $\phi_1 := \phi_0 \cup \lfloor m_x \rfloor$
$\mathsf{in}(c, t_2)$: $m_2 := [\![t_2]\!]_{\phi_1}$
Harness
$\mathsf{C}_0 := \emptyset$
m_1^a
B_s^{in}
PUT, OpenSSL
server s_s
SSL struct
$\mathsf{C}_1 := \mathsf{C}_0 \cup \mathsf{claims}(s_s)$
m_x
B_s^{out}
m_2
B_c^{in}
client s_c
SSL struct
$\mathsf{C}_2 := \mathsf{C}_1 \cup \mathsf{claims}(s_c)$
B_c^{out}
Mutations
$T^a = \mathsf{in}(s, t_1^a) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
t_1
CHello
sid
cs
ext
c_1
nil
mutate
t_1^a
CHello
sid
cs
ext
c_1
,
c_1
nil
$T = \mathsf{in}(s, t_1) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
schedule T
Scheduler
Corpus
DY traces: T, T', ...
Seed corpus (DSL): T_0, T_0', ...

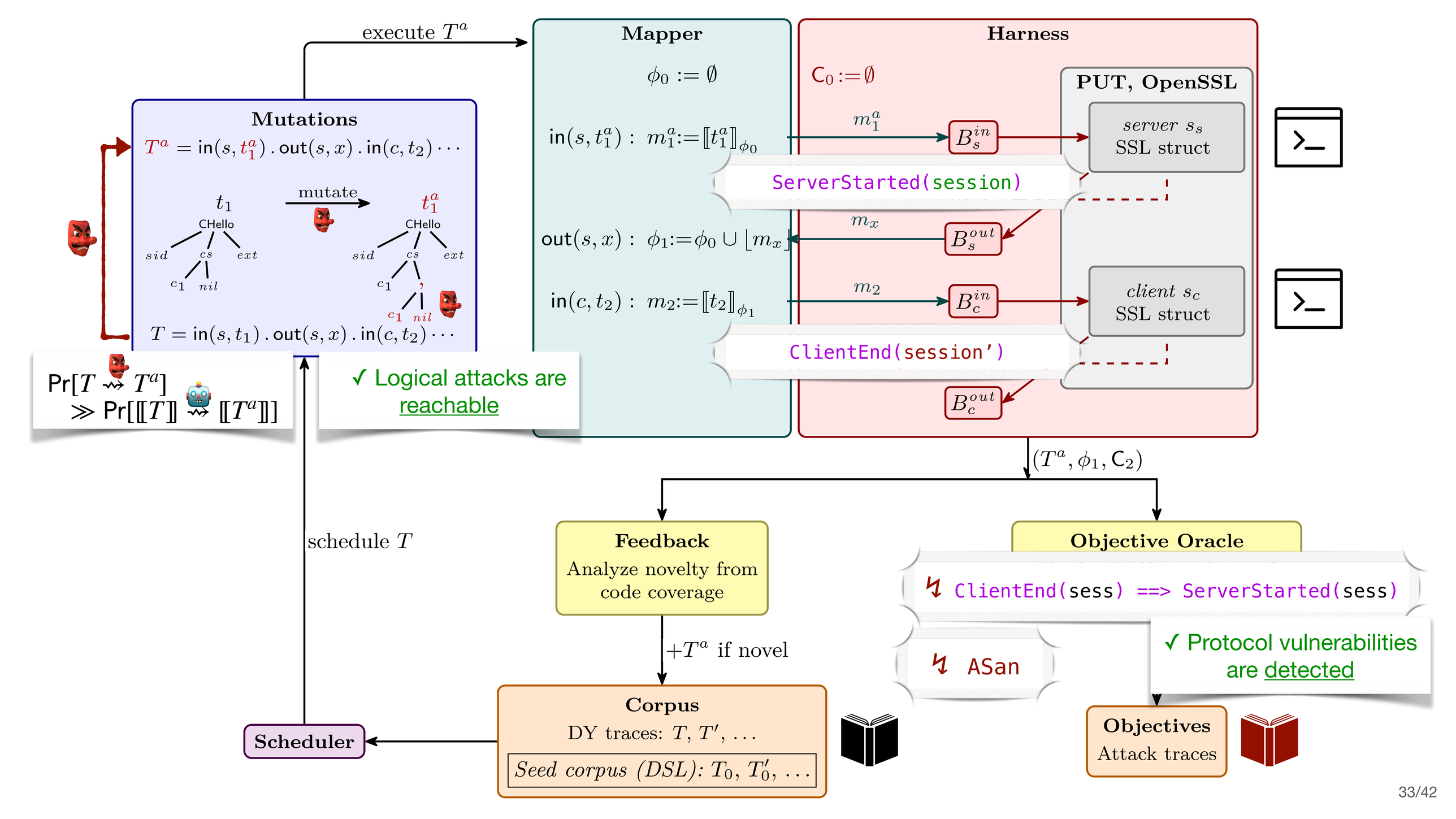

execute T^a
Mapper
$\phi_0 := \emptyset$
$\mathsf{in}(s, t_1^a)$: $m_1^a := [\![t_1^a]\!]_{\phi_0}$
$\mathsf{out}(s, x)$: $\phi_1 := \phi_0 \cup \lfloor m_x \rfloor$
$\mathsf{in}(c, t_2)$: $m_2 := [\![t_2]\!]_{\phi_1}$
Harness
$\mathsf{C}_0 := \emptyset$
m_1^a
B_s^{in}
PUT, OpenSSL
server s_s
SSL struct
$\mathsf{C}_1 := \mathsf{C}_0 \cup \mathsf{claims}(s_s)$
m_x
B_s^{out}
m_2
B_c^{in}
client s_c
SSL struct
$\mathsf{C}_2 := \mathsf{C}_1 \cup \mathsf{claims}(s_c)$
B_c^{out}
Mutations
$T^a = \mathsf{in}(s, t_1^a) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
t_1
CHello
sid
cs
ext
c_1
nil
mutate
t_1^a
CHello
sid
cs
ext
c_1
,
c_1
nil
$T = \mathsf{in}(s, t_1) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
$(T^a, \phi_1, \mathsf{C}_2)$
Feedback
Analyze novelty from code coverage
Objective Oracle
Check ASAN / crashes in C
Check security policies in ϕ, C
$+T^a$ if novel
$+T^a$ if objective
Corpus
DY traces: T, T', ...
Seed corpus (DSL): T_0, T_0', ...
Objectives
Attack traces
Scheduler
schedule T


execute T^a
Mapper
$\phi_0 := \emptyset$
$\mathsf{in}(s, t_1^a)$: $m_1^a := [\![t_1^a]\!]_{\phi_0}$
$\mathsf{out}(s, x)$: $\phi_1 := \phi_0 \cup \lfloor m_x \rfloor$
$\mathsf{in}(c, t_2)$: $m_2 := [\![t_2]\!]_{\phi_1}$
Harness
$\mathsf{C}_0 := \emptyset$
m_1^a
B_s^{in}
PUT, OpenSSL
server s_s
SSL struct
$\mathsf{C}_1 := \mathsf{C}_0 \cup \mathsf{claims}(s_s)$
m_x
B_s^{out}
m_2
B_c^{in}
client s_c
SSL struct
$\mathsf{C}_2 := \mathsf{C}_1 \cup \mathsf{claims}(s_c)$
B_c^{out}
Mutations
$T^a = \mathsf{in}(s, t_1^a) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
t_1
mutate
t_1^a
CHello
sid
cs
ext
c_1
nil
$T = \mathsf{in}(s, t_1) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
$(T^a, \phi_1, \mathsf{C}_2)$
Feedback
Analyze novelty from code coverage
Objective Oracle
Check ASAN / crashes in C
Check security policies in ϕ, C
$+T^a$ if novel
$+T^a$ if objective
schedule T
Corpus
DY traces: T, T', ...
Seed corpus (DSL): T_0, T_0', ...
Objectives
Attack traces
Scheduler


execute T^a
Mutations
$T^a = \mathsf{in}(s, t_1^a) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
t_1
CHello
sid cs ext
c_1 nil
mutate
t_1^a
CHello
sid cs ext
c_1 ,
c_1 nil
$T = \mathsf{in}(s, t_1) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
Mapper
$\phi_0 := \emptyset$
$\mathsf{in}(s, t_1^a)$: $m_1^a := [\![t_1^a]\!]_{\phi_0}$
$\mathsf{out}(s, x)$: $\phi_1 := \phi_0 \cup \lfloor m_x \rfloor$
$\mathsf{in}(c, t_2)$: $m_2 := [\![t_2]\!]_{\phi_1}$
Harness
$\mathsf{C}_0 := \emptyset$
PUT, OpenSSL
m_1^a
B_s^{in}
server s_s
SSL struct
ServerStarted(session)
m_x
B_s^{out}
m_2
B_c^{in}
client s_c
SSL struct
$\mathsf{C}_2 := \mathsf{C}_1 \cup \mathsf{claims}(s_c)$
B_c^{out}
$(T^a, \phi_1, \mathsf{C}_2)$
Feedback
Analyze novelty from code coverage
$+T^a$ if novel
Objective Oracle
Check ASAN / crashes in C
Check security policies in ϕ, C
$+T^a$ if objective
Corpus
DY traces: T, T', ...
Seed corpus (DSL): T_0, T_0', ...
Objectives
Attack traces
Scheduler
schedule T


execute T^a
Mutations
$T^a = \mathsf{in}(s, t_1^a) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
t_1
CHello
sid cs ext
c_1 nil
mutate
t_1^a
CHello
sid cs ext
c_1 ,
c_1 nil
$T = \mathsf{in}(s, t_1) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
Mapper
$\phi_0 := \emptyset$
$\mathsf{in}(s, t_1^a)$: $m_1^a := [\![t_1^a]\!]_{\phi_0}$
$\mathsf{out}(s, x)$: $\phi_1 := \phi_0 \cup \lfloor m_x \rfloor$
$\mathsf{in}(c, t_2)$: $m_2 := [\![t_2]\!]_{\phi_1}$
Harness
$\mathsf{C}_0 := \emptyset$
PUT, OpenSSL
m_1^a
B_s^{in}
server s_s
SSL struct
ServerStarted(session)
m_x
B_s^{out}
m_2
B_c^{in}
client s_c
SSL struct
ClientEnd(session')
B_c^{out}
$(T^a, \phi_1, \mathsf{C}_2)$
schedule T
Feedback
Analyze novelty from code coverage
$+T^a$ if novel
Objective Oracle
Check ASAN / crashes in C
Check security policies in ϕ, C
$+T^a$ if objective
Scheduler
Corpus
DY traces: T, T', ...
Seed corpus (DSL): T_0, T_0', ...
Objectives
Attack traces


execute T^a
Mutations
$T^a = \mathsf{in}(s, t_1^a) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
t_1
CHello
sid cs ext
c_1 nil
mutate
t_1^a
CHello
sid cs ext
c_1 ,
c_1 nil
$T = \mathsf{in}(s, t_1) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
Mapper
$\phi_0 := \emptyset$
$\mathsf{in}(s, t_1^a)$: $m_1^a := [\![t_1^a]\!]_{\phi_0}$
$\mathsf{out}(s, x)$: $\phi_1 := \phi_0 \cup \lfloor m_x \rfloor$
$\mathsf{in}(c, t_2)$: $m_2 := [\![t_2]\!]_{\phi_1}$
Harness
$\mathsf{C}_0 := \emptyset$
PUT, OpenSSL
m_1^a
B_s^{in}
server s_s
SSL struct
ServerStarted(session)
m_x
B_s^{out}
m_2
B_c^{in}
client s_c
SSL struct
ClientEnd(session')
B_c^{out}
$(T^a, \phi_1, \mathsf{C}_2)$
Feedback
Analyze novelty from code coverage
$+T^a$ if novel
Objective Oracle
ClientEnd(sess) ==> ServerStarted(sess)
ASan
✓ Protocol vulnerabilities are detected
Corpus
DY traces: T, T', ...
Seed corpus (DSL): T_0, T_0', ...
Objectives
Attack traces
Scheduler
schedule T


execute T^a
Mapper
$\phi_0 := \emptyset$
$\mathsf{in}(s, t_1^a)$: $m_1^a := [\![t_1^a]\!]_{\phi_0}$
$\mathsf{out}(s, x)$: $\phi_1 := \phi_0 \cup \lfloor m_x \rfloor$
$\mathsf{in}(c, t_2)$: $m_2 := [\![t_2]\!]_{\phi_1}$
Harness
$\mathsf{C}_0 := \emptyset$
PUT, OpenSSL
m_1^a
B_s^{in}
server s_s
SSL struct
ServerStarted(session)
m_x
B_s^{out}
m_2
B_c^{in}
client s_c
SSL struct
ClientEnd(session')
B_c^{out}
Mutations
$T^a = \mathsf{in}(s, t_1^a) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
mutate
t_1
CHello
sid
cs
ext
c_1
nil
t_1^a
CHello
sid
cs
ext
c_1
,
c_1
nil
$T = \mathsf{in}(s, t_1) \,.\, \mathsf{out}(s, x) \,.\, \mathsf{in}(c, t_2) \cdots$
$\Pr[T \rightsquigarrow T^a] \gg \Pr[[\![T]\!] \rightsquigarrow [\![T^a]\!]]$
✓ Logical attacks are reachable
$(T^a, \phi_1, \mathsf{C}_2)$
Feedback
Analyze novelty from code coverage
$+T^a$ if novel
Objective Oracle
ClientEnd(sess) ==> ServerStarted(sess)
ASan
✓ Protocol vulnerabilities are detected
Objectives
Attack traces
schedule T
Scheduler
Corpus
DY traces: T, T', ...
Seed corpus (DSL): T_0, T_0', ...

Differential DYF [S&P'27] With T. Gouville (PhD: 2023-2027)

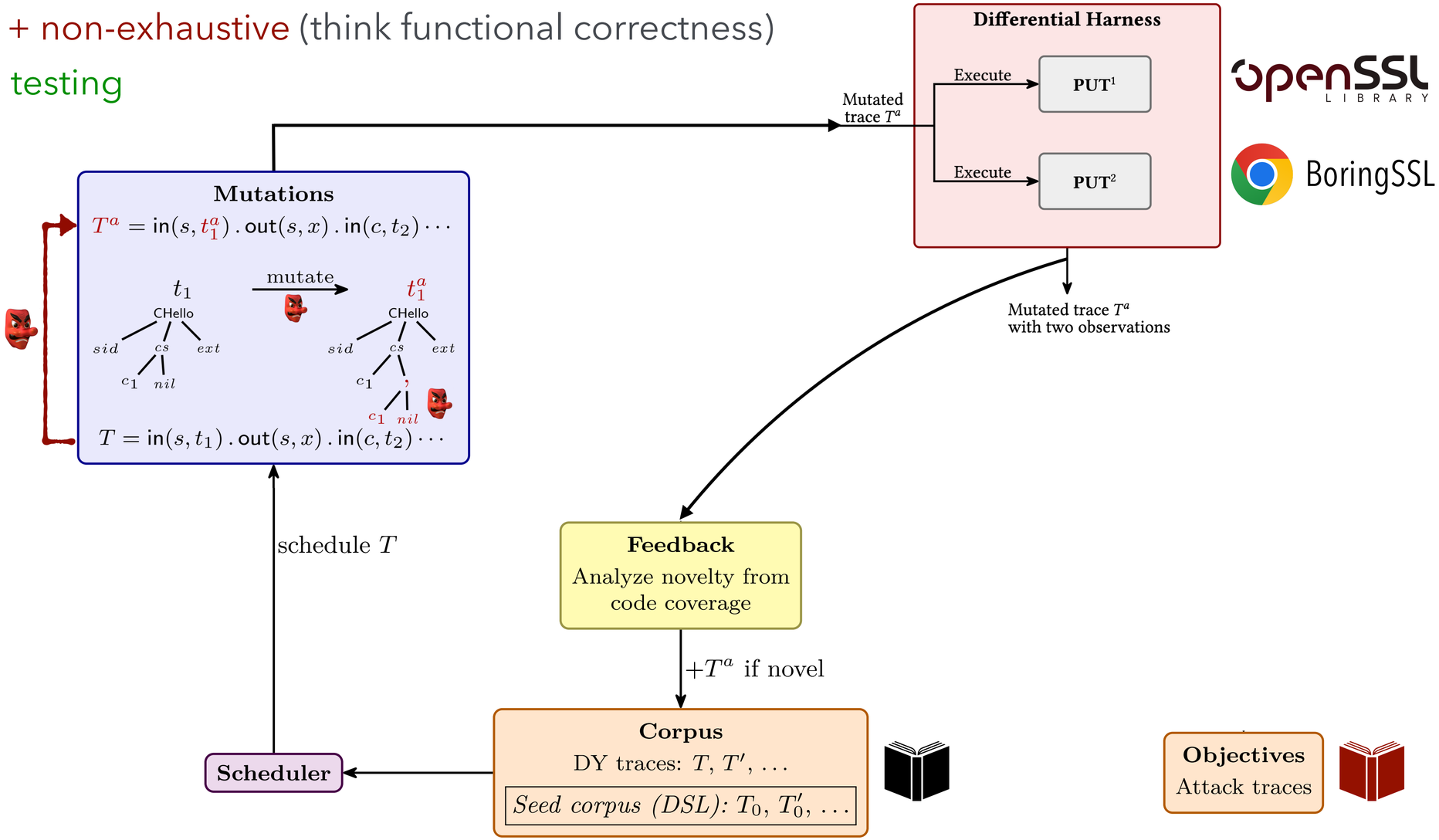
- Objective oracle: requires security properties & claims
- Can be costly + non-exhaustive (think functional correctness)

  ☞ Differential testing

Differential DYF [S&P'27] With T. Gouville (PhD: 2023-2027)

- Objective oracle: requires security properties & claims
- Can be costly + non-exhaustive (think functional correctness)

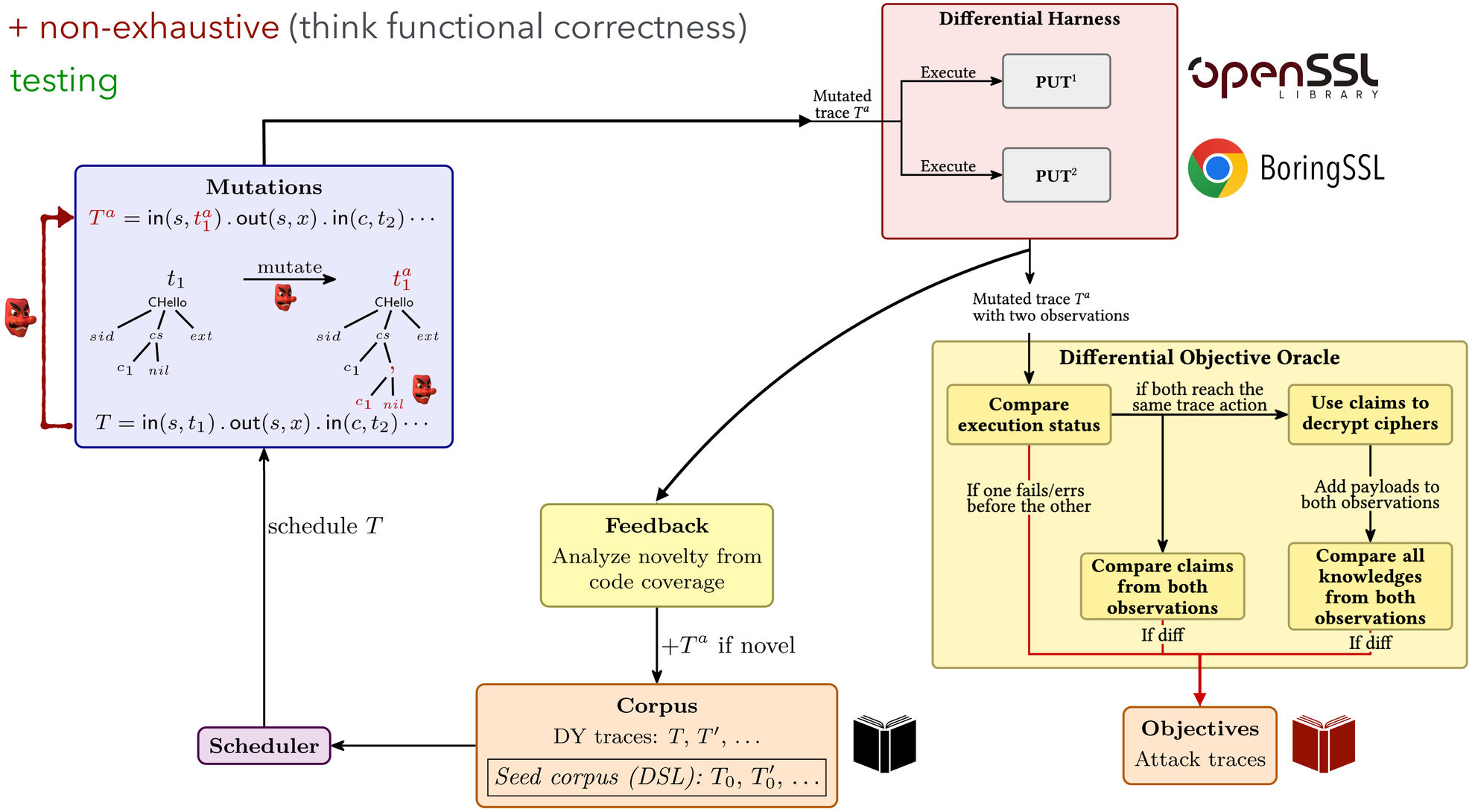
  ☞ Differential testing



Differential DYF [S&P'27] With T. Gouville (PhD: 2023-2027)

- Objective oracle: requires security properties & claims
- Can be costly + non-exhaustive (think functional correctness)

  ☞ Differential testing



Implementation — Puffin: modular DY fuzzers

- Terms
- Objective oracle: diff and properties
- Domain-Specific Language for Traces
- Mutations expressing DY 👺
- Fuzzing loop, CLI with tooling

Puffin

Protocol-agnostic, once and for all

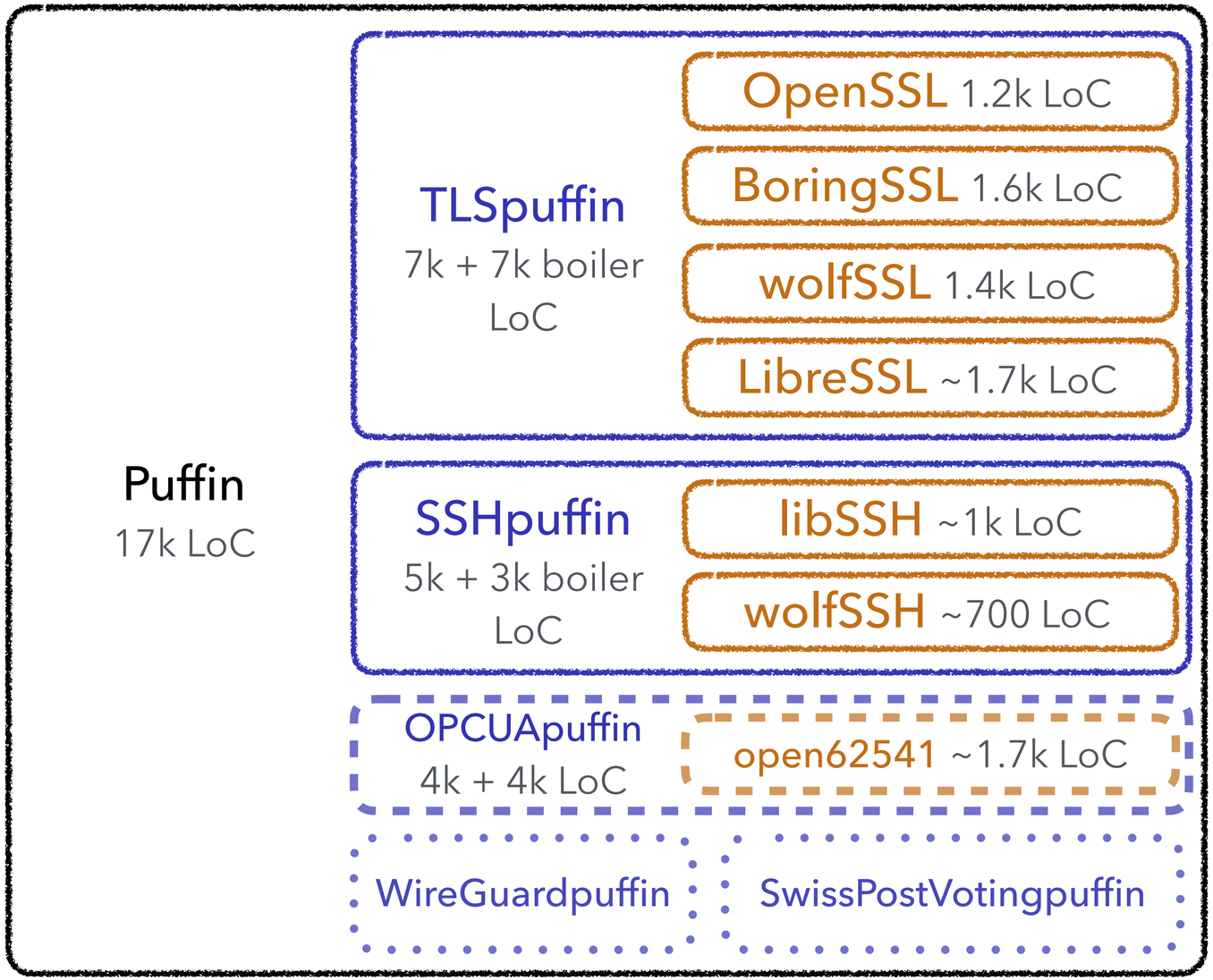
Implementation — Puffin: modular DY fuzzers



Puffin

Protocol-agnostic, once and for all

Implementation — Puffin: modular DY fuzzers



- Open-source library (on top of LibAFL) and family of fuzzers
- In-memory buffers, embarrassingly parallel, fast (700 execs/s/core)
- Differential mode and tool-assisted bug triaging [S&P'27]
- More protocols and vendors to come…

Puffin fuzzers – ~60k LoC

https://tlspuffin.github.io

Puffin Evaluation

Puffin Evaluation

a) **Finding logical attacks: puffin vs. other fuzzers?**

- ☞ Criteria-based selection: 3 fuzzers / 3+5 CVEs
- ☞ puffin found all (in min/h); all missed by others
- ☞ also missed by Google's ossl-fuzz
- ☞ puffin found 5 new CVEs

Puffin Evaluation

a) Finding logical attacks: puffin vs. other fuzzers?

- ☞ Criteria-based selection: 3 fuzzers / 3+5 CVEs
- ☞ puffin found all (in min/h); all missed by others
- ☞ also missed by Google's ossl-fuzz
- ☞ puffin found 5 new CVEs

CVE ID	CVSS	Target	New	Type	Oracle	AFL-like?	DYF?
2021-3449	5.9	OpenSSL	✗	Server DoS	ASan	✗	✓
2022-25638	6.5	wolfSSL	✗	Auth. Bypass	DY or Diff	✗	✓
2022-25640	7.5	wolfSSL	✗	Auth. Bypass	DY or Diff	✗	✓
2022-38152	7.5	wolfSSL	✓	Server DoS	ASan	✗	✓
2022-38153	5.9	wolfSSL	✓	Client DoS	ASan	✗	✓
2022-39173	7.5	wolfSSL	✓	Server DoS	ASan	✗	✓
2022-42905	9.1	wolfSSL	✓	Info. Leak	ASan	✗	✓
2023-6936	5.3	wolfSSL	✓	Info. Leak	ASan	✗	✓

Puffin Evaluation

a) Finding logical attacks: puffin vs. other fuzzers?

- ☞ Criteria-based selection: 3 fuzzers / 3+5 CVEs
- ☞ puffin found all (in min/h); all missed by others
- ☞ also missed by Google's ossl-fuzz
- ☞ puffin found 5 new CVEs

b) Insights from code coverage comparison against other SoA fuzzers?

On par but:

- ☞ Advertising vs. using features/crypto
- ☞ Code coverage gets exhausted (attack≈seed)
- ☞ Poor evaluation and feedback metric

CVE ID	CVSS	Target	New	Type	Oracle	AFL-like?	DYF?
2021-3449	5.9	OpenSSL	✗	Server DoS	ASan	✗	✓
2022-25638	6.5	wolfSSL	✗	Auth. Bypass	DY or Diff	✗	✓
2022-25640	7.5	wolfSSL	✗	Auth. Bypass	DY or Diff	✗	✓
2022-38152	7.5	wolfSSL	✓	Server DoS	ASan	✗	✓
2022-38153	5.9	wolfSSL	✓	Client DoS	ASan	✗	✓
2022-39173	7.5	wolfSSL	✓	Server DoS	ASan	✗	✓
2022-42905	9.1	wolfSSL	✓	Info. Leak	ASan	✗	✓
2023-6936	5.3	wolfSSL	✓	Info. Leak	ASan	✗	✓

Puffin Evaluation

a) Finding logical attacks: puffin vs. other fuzzers?

- ☞ Criteria-based selection: 3 fuzzers / 3+5 CVEs
- ☞ puffin found all (in min/h); all missed by others
- ☞ also missed by Google's ossl-fuzz
- ☞ puffin found 5 new CVEs

CVE ID	CVSS	Target	New	Type	Oracle	AFL-like?	DYF?
2021-3449	5.9	OpenSSL	✗	Server DoS	ASan	✗	✓
2022-25638	6.5	wolfSSL	✗	Auth. Bypass	DY or Diff	✗	✓
2022-25640	7.5	wolfSSL	✗	Auth. Bypass	DY or Diff	✗	✓
2022-38152	7.5	wolfSSL	✓	Server DoS	ASan	✗	✓
2022-38153	5.9	wolfSSL	✓	Client DoS	ASan	✗	✓
2022-39173	7.5	wolfSSL	✓	Server DoS	ASan	✗	✓
2022-42905	9.1	wolfSSL	✓	Info. Leak	ASan	✗	✓
2023-6936	5.3	wolfSSL	✓	Info. Leak	ASan	✗	✓

b) Insights from code coverage comparison against other SoA fuzzers?

On par but:

- ☞ Advertising vs. using features/crypto
- ☞ Code coverage gets exhausted (attack≈seed)
- ☞ Poor evaluation and feedback metric

c) Can Differential DYF detect vulnerabilities beyond the reach of DYF?

Puffin Evaluation

a) Finding logical attacks: puffin vs. other fuzzers?

- ☞ Criteria-based selection: 3 fuzzers / 3+5 CVEs
- ☞ puffin found all (in min/h); all missed by others
- ☞ also missed by Google's ossl-fuzz
- ☞ puffin found 5 new CVEs

b) Insights from code coverage comparison against other SoA fuzzers?

On par but:

- ☞ Advertising vs. using features/crypto
- ☞ Code coverage gets exhausted (attack≈seed)
- ☞ Poor evaluation and feedback metric

CVE ID	CVSS	Target	New	Type	Oracle	AFL-like?	DYF?	DDYF?
2021-3449	5.9	OpenSSL	✗	Server DoS	ASan	✗	✓	-
2022-25638	6.5	wolfSSL	✗	Auth. Bypass	DY or Diff	✗	✓	✓
2022-25640	7.5	wolfSSL	✗	Auth. Bypass	DY or Diff	✗	✓	✓
2022-38152	7.5	wolfSSL	✓	Server DoS	ASan	✗	✓	-
2022-38153	5.9	wolfSSL	✓	Client DoS	ASan	✗	✓	-
2022-39173	7.5	wolfSSL	✓	Server DoS	ASan	✗	✓	-
2022-42905	9.1	wolfSSL	✓	Info. Leak	ASan	✗	✓	-
2023-6936	5.3	wolfSSL	✓	Info. Leak	ASan	✗	✓	-
2024-5814	5.3	wolfSSL	✗	Downgrade	DY or Diff	✗	✓	✓
2023-3724	9.1	wolfSSL	✗	Weak key	Diff	✗	✗	✓
2023-6937	5.3	wolfSSL	✗	Confidentiality	Diff	✗	✗	✓

c) Can Differential DYF detect vulnerabilities beyond the reach of DYF?

- ☞ Yes, confirmed with 2 CVEs + First approach to find "secrecy"-style logical attacks through fuzzing
- ☞ 52 new acknowledged compliance bugs (RFC violations) in Open/Boring/Wolf/SSL and lib/wolf/SSH
- ☞ 4 new security-related bugs in BoringSSL, WolfSSL, LibreSSL

Perspectives



Formal Verification Perspectives



Verifiers improved a lot ⇾ analyze whole standards

⇾ verification gets overly **complex** and **interactive**

Ex: OPC UA: 15 person-month and some goals remain open

Ex: comprehensive 5G AKA privacy verification remains open

Formal Verification Perspectives



Verifiers improved a lot ⇾ analyze whole standards

⇾ verification gets overly **complex** and **interactive**

Ex: OPC UA: 15 person-month and some goals remain open

Ex: comprehensive 5G AKA privacy verification remains open

Multi-objective verification and **lattice framework:**

⇾ opens new opportunities for optimizing proofs and addressing verification open problems

1. Reuse as helpers: If goal G1 is verified and doesn't prune goal G2, we should verify G2 assuming G1
2. Composite argument for proof goals:
   ✗ Verify goal G1 (e.g. authentication) and the very costly goal G2 (e.g. privacy)
   ✓ Prove, once and for all, Inva. $\wedge$ G1 $\Longrightarrow$ G2. Verify Inva. (cheap) and G1 assuming Inva. (simpler)
3. Design proof goals, lattice construction, and exploration strategies accordingly

⇾ & discover and measure security & privacy limits (e.g., privacy loss?)

Formal Verification Perspectives



Verifiers improved a lot ⇾ analyze whole standards

⇾ verification gets overly **complex** and **interactive**

Ex: OPC UA: 15 person-month and some goals remain open

Ex: comprehensive 5G AKA privacy verification remains open

Multi-objective verification and **lattice framework:**

⇾ opens new opportunities for optimizing proofs and addressing verification open problems

1. Reuse as helpers: If goal G1 is verified and doesn't prune goal G2, we should verify G2 assuming G1
2. Composite argument for proof goals:
   ✗ Verify goal G1 (e.g. authentication) and the very costly goal G2 (e.g. privacy)
   ✓ Prove, once and for all, Inva. $\wedge$ G1 $\Longrightarrow$ G2. Verify Inva. (cheap) and G1 assuming Inva. (simpler)
3. Design proof goals, lattice construction, and exploration strategies accordingly

⇾ & discover and measure security & privacy limits (e.g., privacy loss?)

Tackle large-scale analysis of future real-world protocols

- Address verifiers/modeling limitations that really matter in practice (this feedback loop has been productive so far)
- Improve standards, come up with recommendations and lessons for the future

E-Voting Perspectives

Academic Solutions

E-Voting Perspectives

Academic Solutions

Real-World Deployments, Constraints, and Regulation

E-Voting Perspectives

Academic Solutions



Real-World Deployments, Constraints, and Regulation

1. Dialogue about technical doctrine/recommendations (ANSSI, CNIL)
2. Integrate operational constraints into protocols (vote tagging, lawful quorum, etc.)

E-Voting Perspectives

Academic Solutions



Real-World Deployments, Constraints, and Regulation



1. Dialogue about technical doctrine/recommendations (ANSSI, CNIL)
2. Integrate operational constraints into protocols (vote tagging, lawful quorum, etc.)
3. Hybrid-Secure E-Voting
   PhD: A. Outrey

E-Voting Perspectives

CNIL.

ANSSI

MINISTÈRE DE L'EUROPE ET DES AFFAIRES ÉTRANGÈRES

SWISS POST

Academic Solutions

Real-World Deployments, Constraints, and Regulation

1. Dialogue about technical doctrine/recommendations (ANSSI, CNIL)
2. Integrate operational constraints into protocols (vote tagging, lawful quorum, etc.)

3. Hybrid-Secure E-Voting
   PhD: A. Outrey

4. Formal analysis
5. Dolev-Yao Fuzzing
   PhD: B. Voisin

Dolev-Yao Fuzzing Perspectives



Puffin and DY fuzzing

- **Feedback**: identified and quantified limitations of code coverage ↛ needs DY-aware feedback
- **Mapper**: towards exhaustiveness 👺 and easing protocol integration. With T. Gouville (PhD)
- **Reconciling DY and bit-level fuzzing**: latter smashes message structure needed by the former

Dolev-Yao Fuzzing Perspectives



Puffin and DY fuzzing

- **Feedback**: identified and quantified limitations of code coverage ↛ needs DY-aware feedback
- **Mapper**: towards exhaustiveness  and easing protocol integration. With T. Gouville (PhD)
- **Reconciling DY and bit-level fuzzing**: latter smashes message structure needed by the former

Long Haul: Connecting DY Verification ↔ DY Fuzzing (with a protocol model & verifiers)

- **Objective Oracle**: deduction ↛ confidentiality, behavioral equivalence ↛ privacy (DDYF?), model compliance ↛ functional correctness
- Model **extraction** and **refinement** by fuzzer-provided functional correctness counter-examples
- Use **verifiers** to synthesize attack candidates that **steer** the fuzzer
- **Coupling** formal models and implementations with dual executions ↛ open broad perspectives

🔮 LLM: Existential crisis, what's beyond the hype peak?

Formal methods: LLMs 🔮 are unreliable ⇾ let 🔮 write machine-checked proofs 🏅

⇾ Still requires foundations, what to prove, and well-scoped and sound models (hard part)

🔮 LLM: Existential crisis, what's beyond the hype peak?

Formal methods: LLMs 🔮 are unreliable ⇾ let 🔮 write machine-checked proofs 🏅
⇾ Still requires foundations, what to prove, and well-scoped and sound models (hard part)

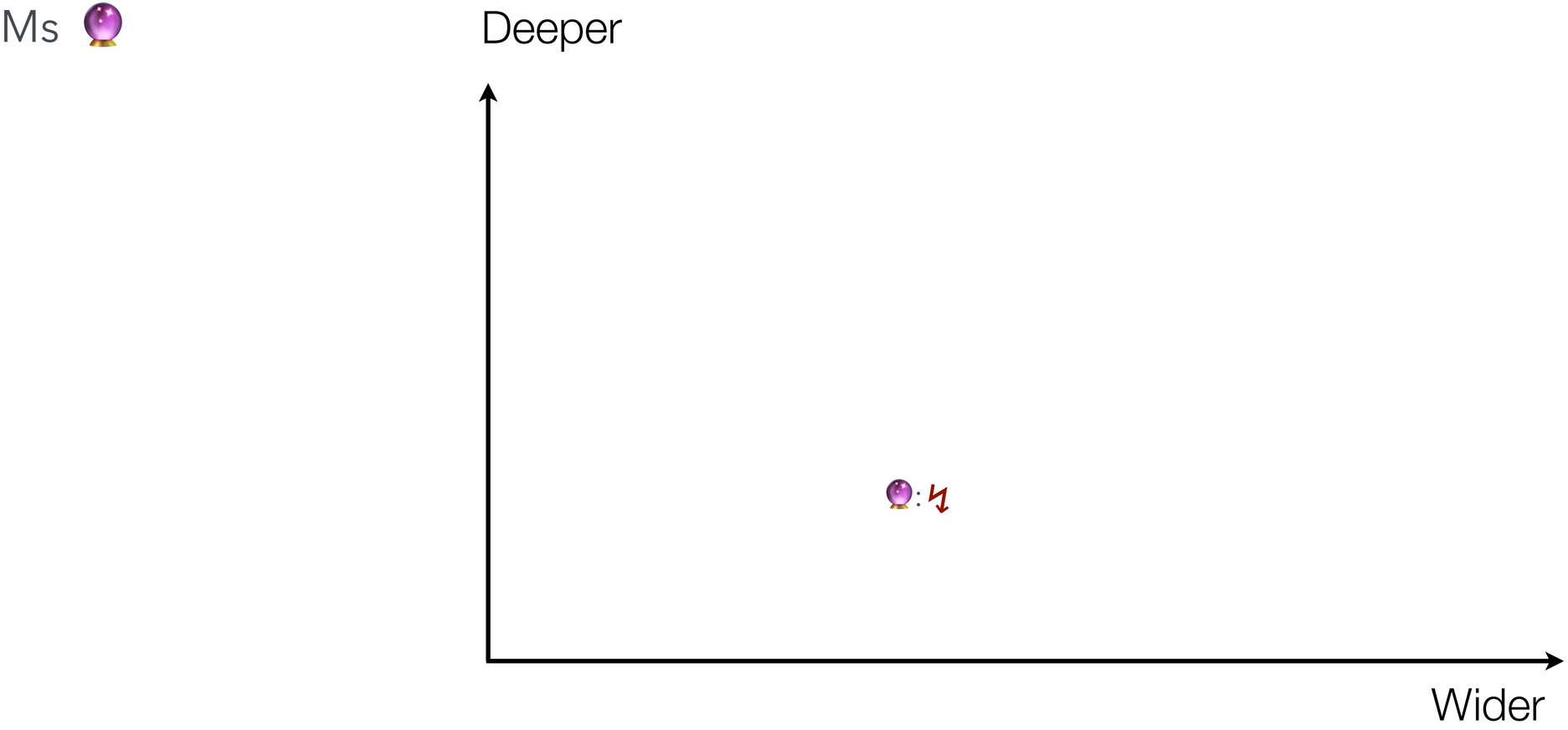
Fuzzing:

🔮 LLM: Existential crisis, what's beyond the hype peak?

Formal methods: LLMs 🔮 are unreliable ⇾ let 🔮 write machine-checked proofs 🏅
⇾ **Still** requires foundations, what to prove, and well-scoped and sound models (hard part)

Fuzzing:

🔮 • Attack finding by LLMs 🔮 💸

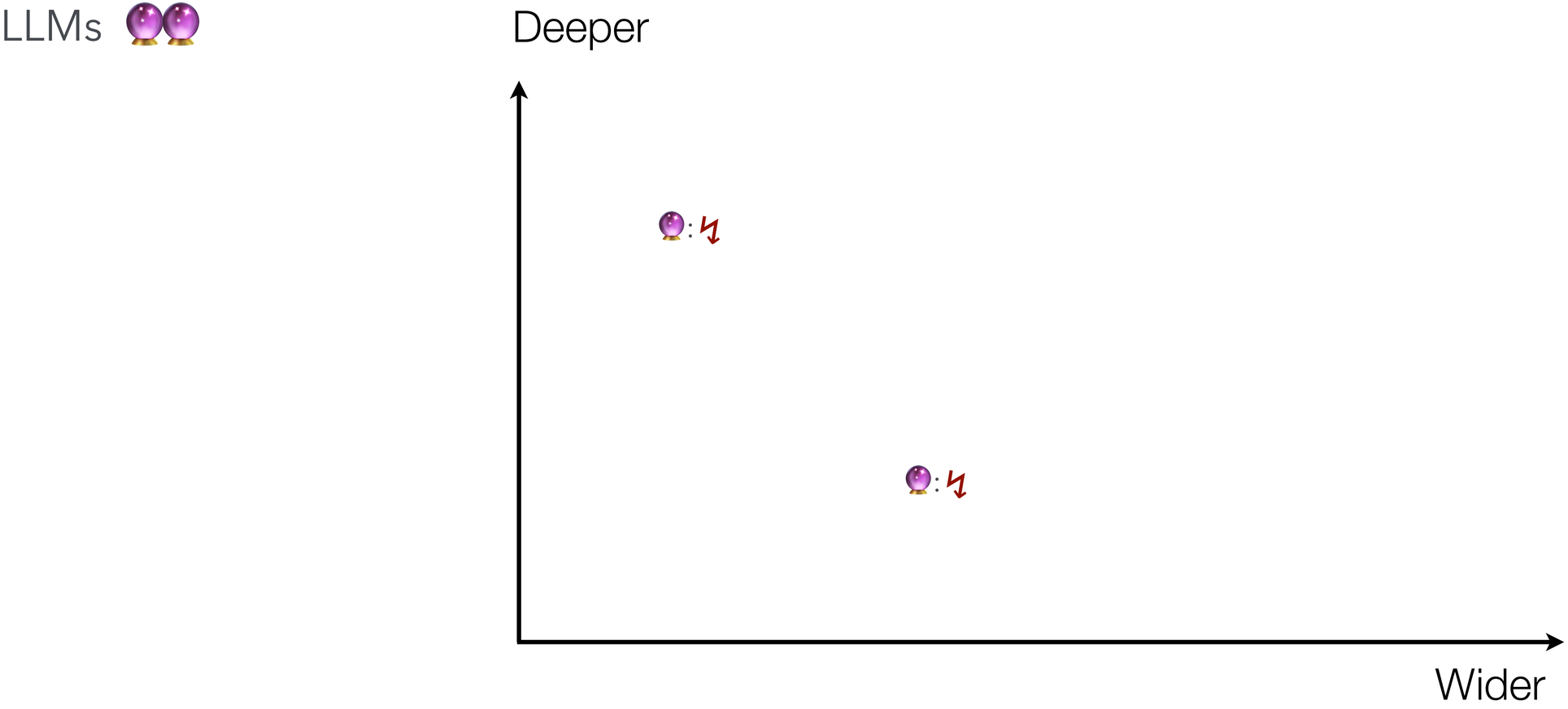


LLM: Existential crisis, what's beyond the hype peak?

Formal methods: LLMs 🔮 are unreliable ↛ let 🔮 write machine-checked proofs 🏅
↛ **Still** requires foundations, what to prove, and well-scoped and sound models (hard part)

Fuzzing:

🔮 • Attack finding by LLMs 💵🔮🔮

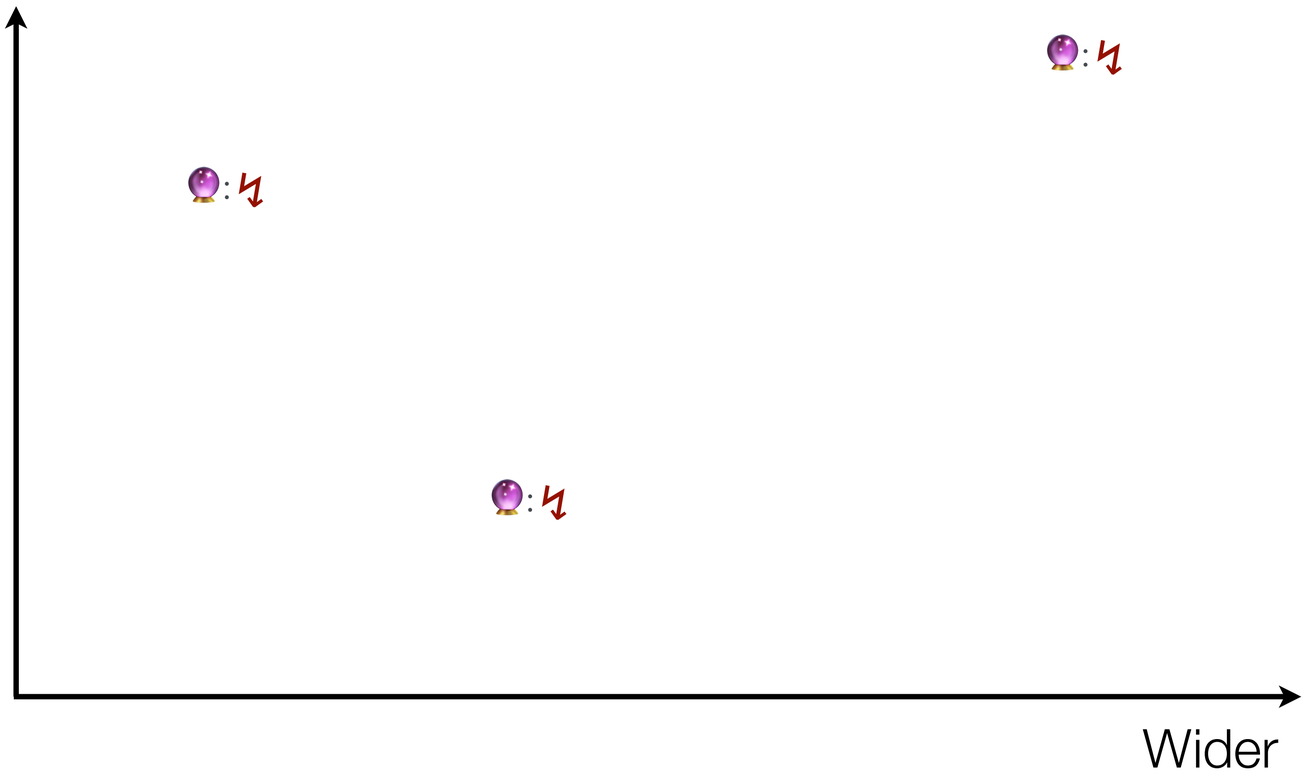


🔮 LLM: Existential crisis, what's beyond the hype peak?

Formal methods: LLMs 🔮 are unreliable ⇾ let 🔮 write machine-checked proofs 🏅
⇾ **Still** requires foundations, what to prove, and well-scoped and sound models (hard part)

Fuzzing:

🔮 • Attack finding by LLMs 

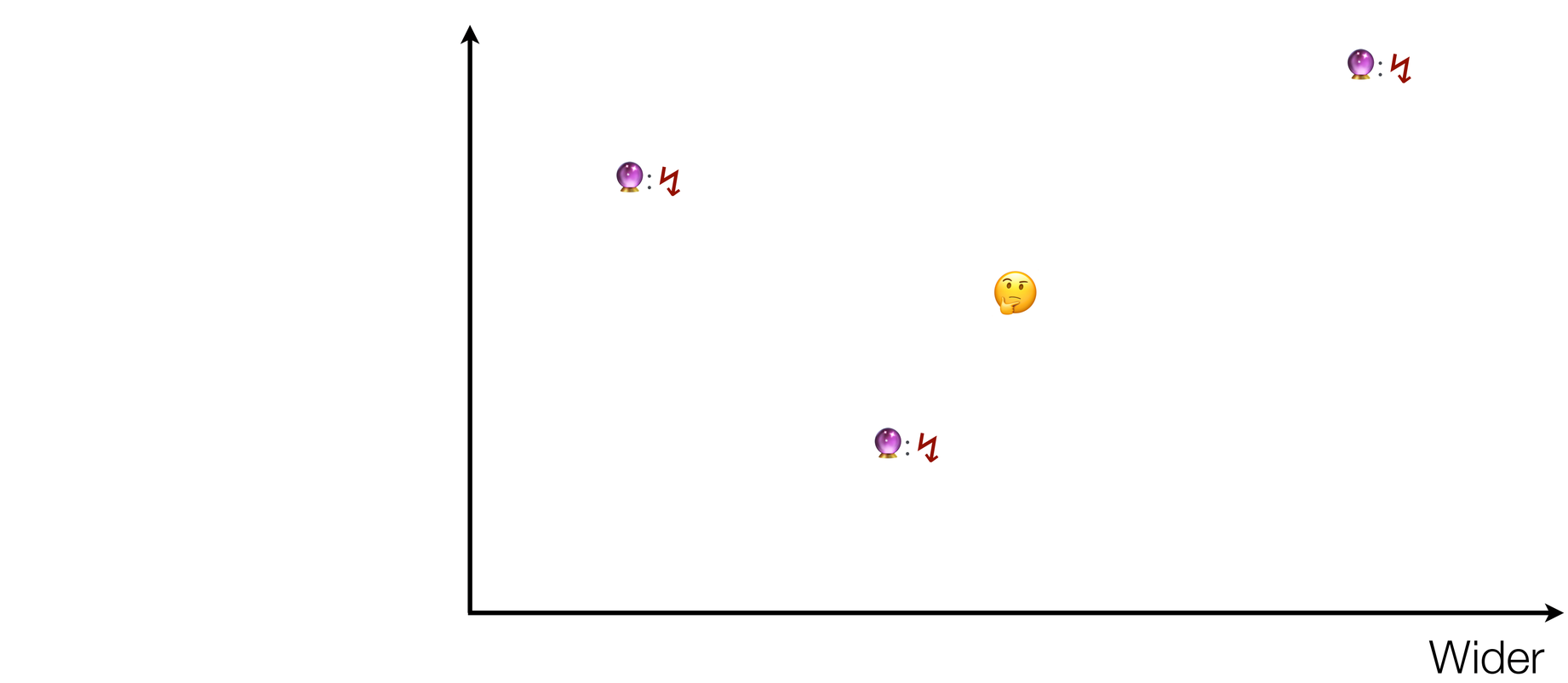


🔮 LLM: Existential crisis, what's beyond the hype peak?

Formal methods: LLMs 🔮 are unreliable ⇾ let 🔮 write machine-checked proofs 🏅
⇾ **Still** requires foundations, what to prove, and well-scoped and sound models (hard part)

Fuzzing:

🔮 • Attack finding by LLMs 💸💸💸🏭🔥
🔮🔮🔮🔮🔮🔮🔮🔮



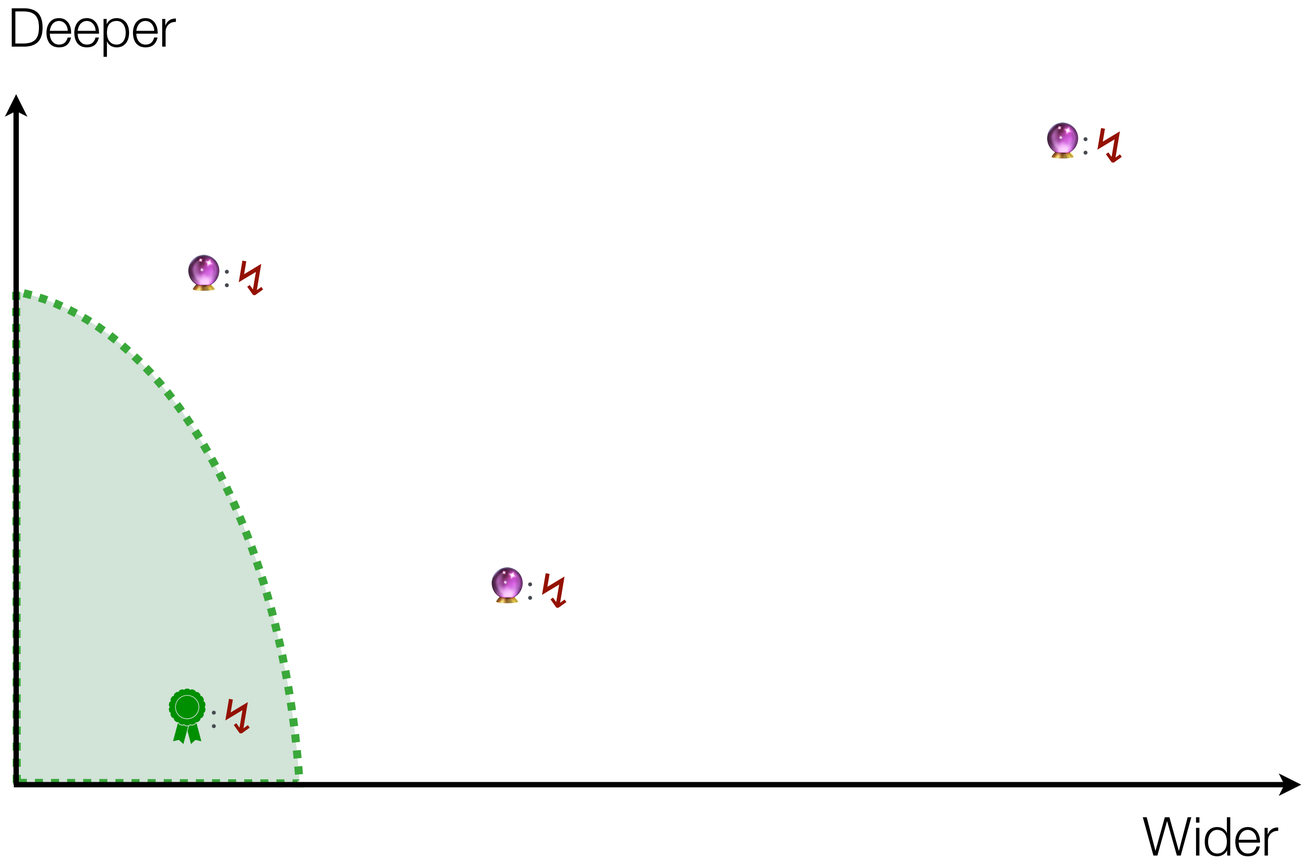
LLM: Existential crisis, what's beyond the hype peak?

Formal methods: LLMs 🔮 are unreliable ⇾ let 🔮 write machine-checked proofs 🏅
⇾ **Still** requires foundations, what to prove, and well-scoped and sound models (hard part)

Fuzzing:



- 🔮 • Attack finding by LLMs
- 🏅 • Formally prove (possibly with 🔮)



LLM: Existential crisis, what's beyond the hype peak?

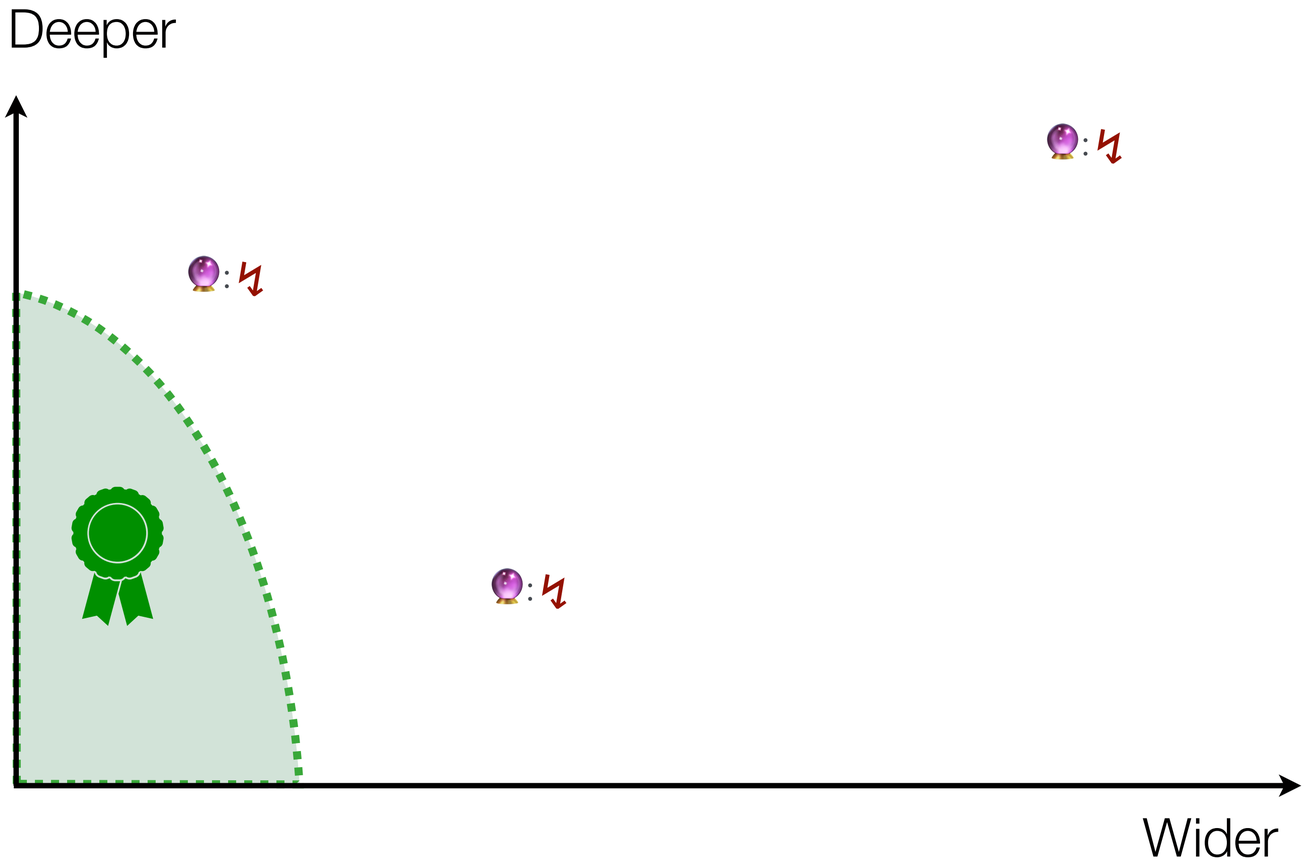
Formal methods: LLMs 🔮 are unreliable ⇾ let 🔮 write machine-checked proofs 🏅
⇾ **Still** requires foundations, what to prove, and well-scoped and sound models (hard part)

Fuzzing:



🔮 • Attack finding by LLMs

🏅 • Formally prove (possibly with 🔮)



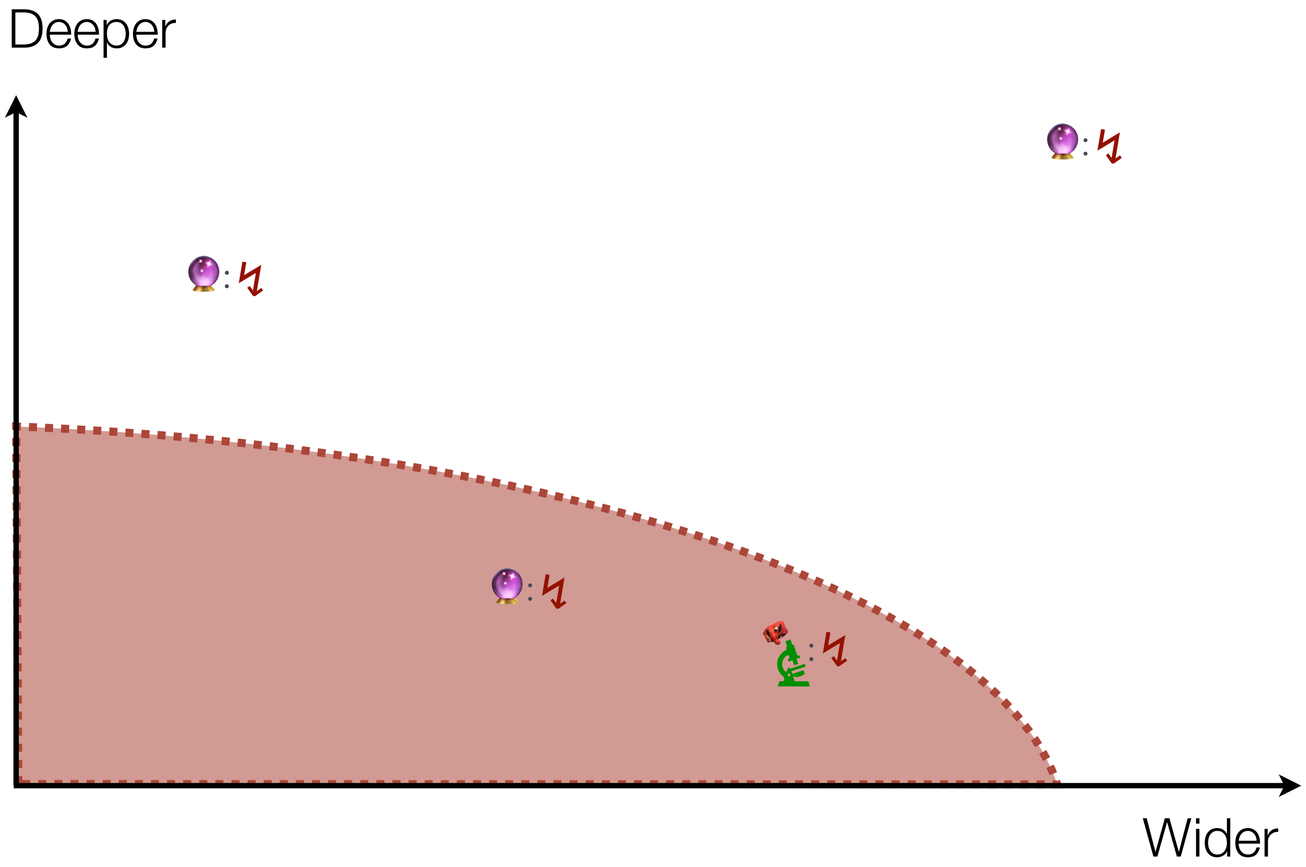
LLM: Existential crisis, what's beyond the hype peak?

Formal methods: LLMs 🔮 are unreliable ⇾ let 🔮 write machine-checked proofs 🏅
⇾ **Still** requires foundations, what to prove, and well-scoped and sound models (hard part)

Fuzzing:



- 🔮 • Attack finding by LLMs
- 🏅 • Formally prove (possibly with 🔮)
- 🔬 • Fuzzing



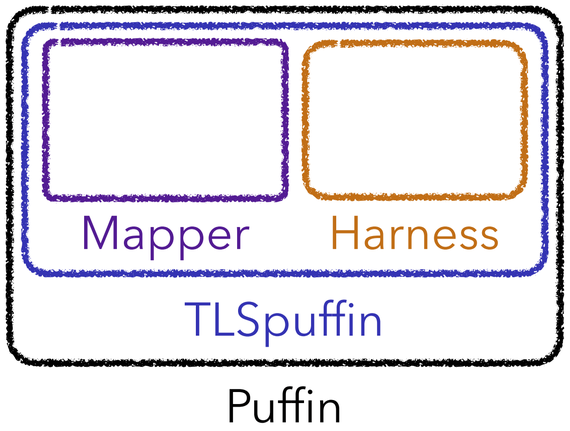
LLM: Existential crisis, what's beyond the hype peak?

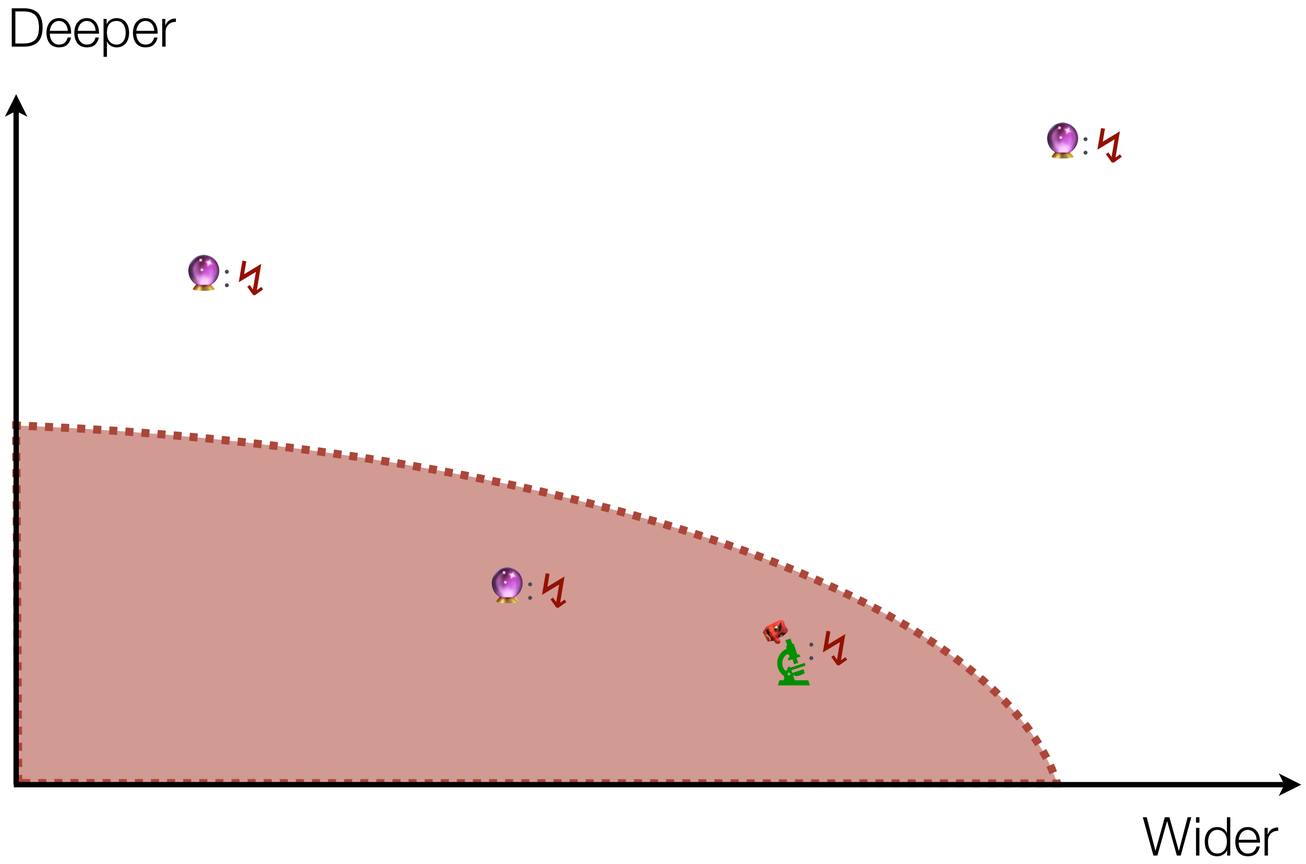
Formal methods: LLMs 🔮 are unreliable ↠ let 🔮 write machine-checked proofs 🏅
↠ **Still** requires foundations, what to prove, and well-scoped and sound models (hard part)

Fuzzing:



- 🔮 • Attack finding by LLMs
- 🏅 • Formally prove (possibly with 🔮)
- 🔬 • Fuzzing





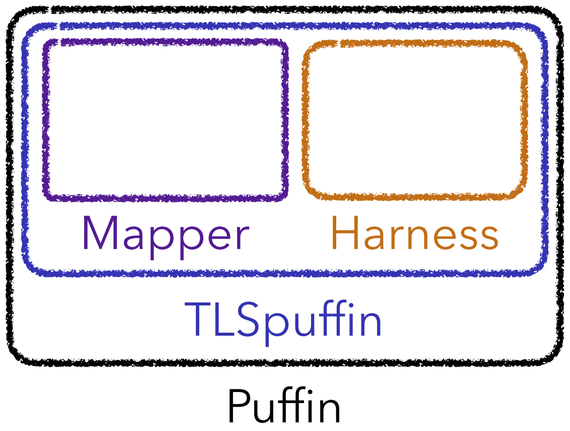
LLM: Existential crisis, what's beyond the hype peak?

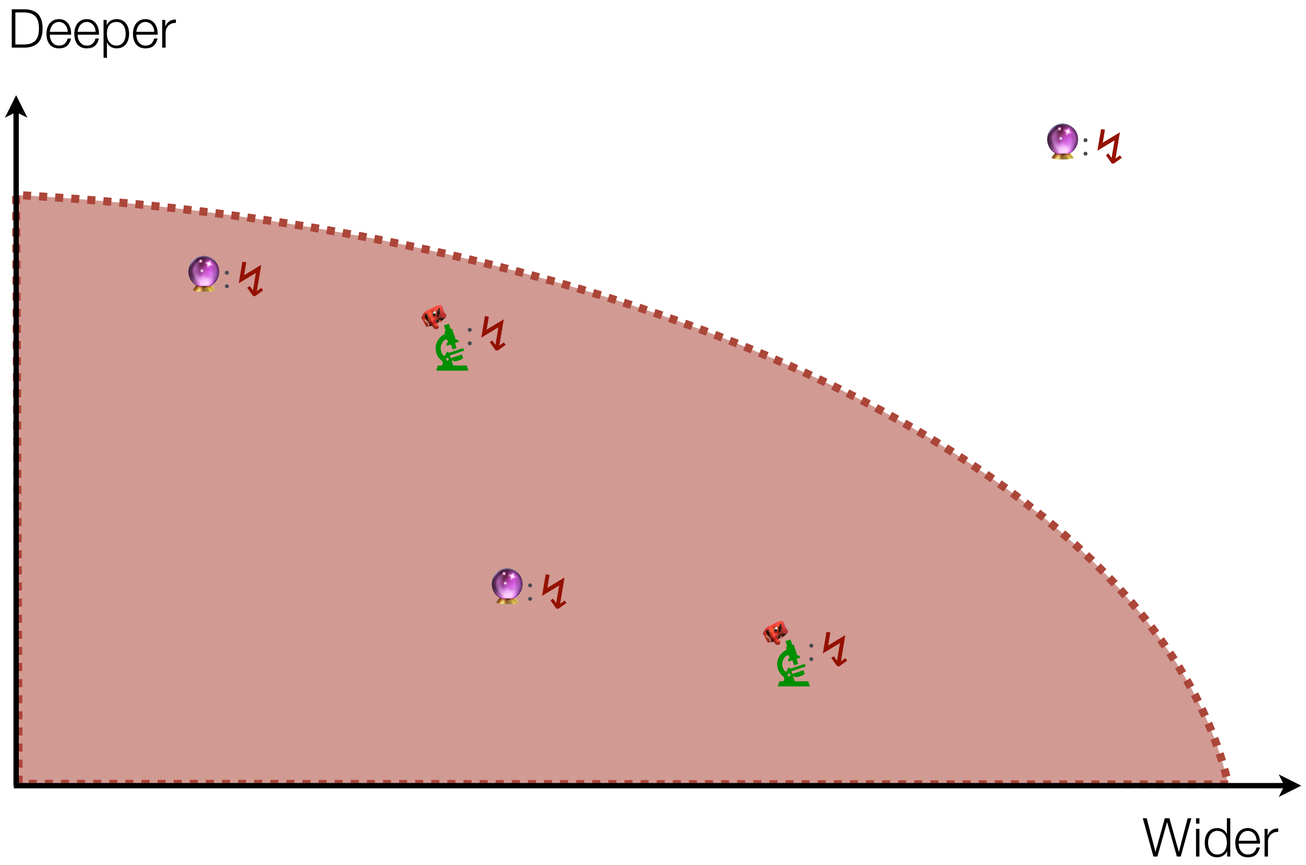
Formal methods: LLMs 🔮 are unreliable ↠ let 🔮 write machine-checked proofs 🏅
↠ **Still** requires foundations, what to prove, and well-scoped and sound models (hard part)

Fuzzing:



- 🔮 • Attack finding by LLMs
- 🏅 • Formally prove (possibly with 🔮)
- 🔬 • Fuzzing
- 🔬 • Fuzzing with 🔮





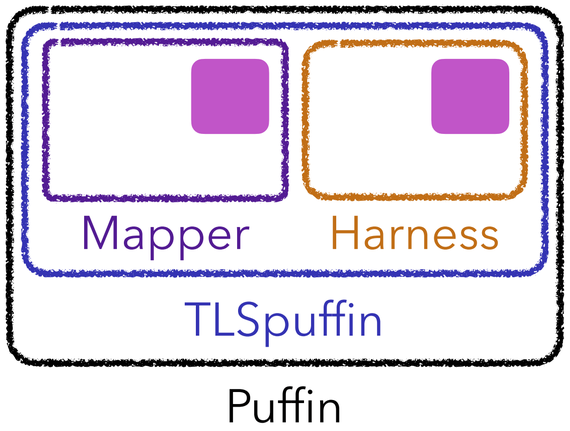
LLM: Existential crisis, what's beyond the hype peak?

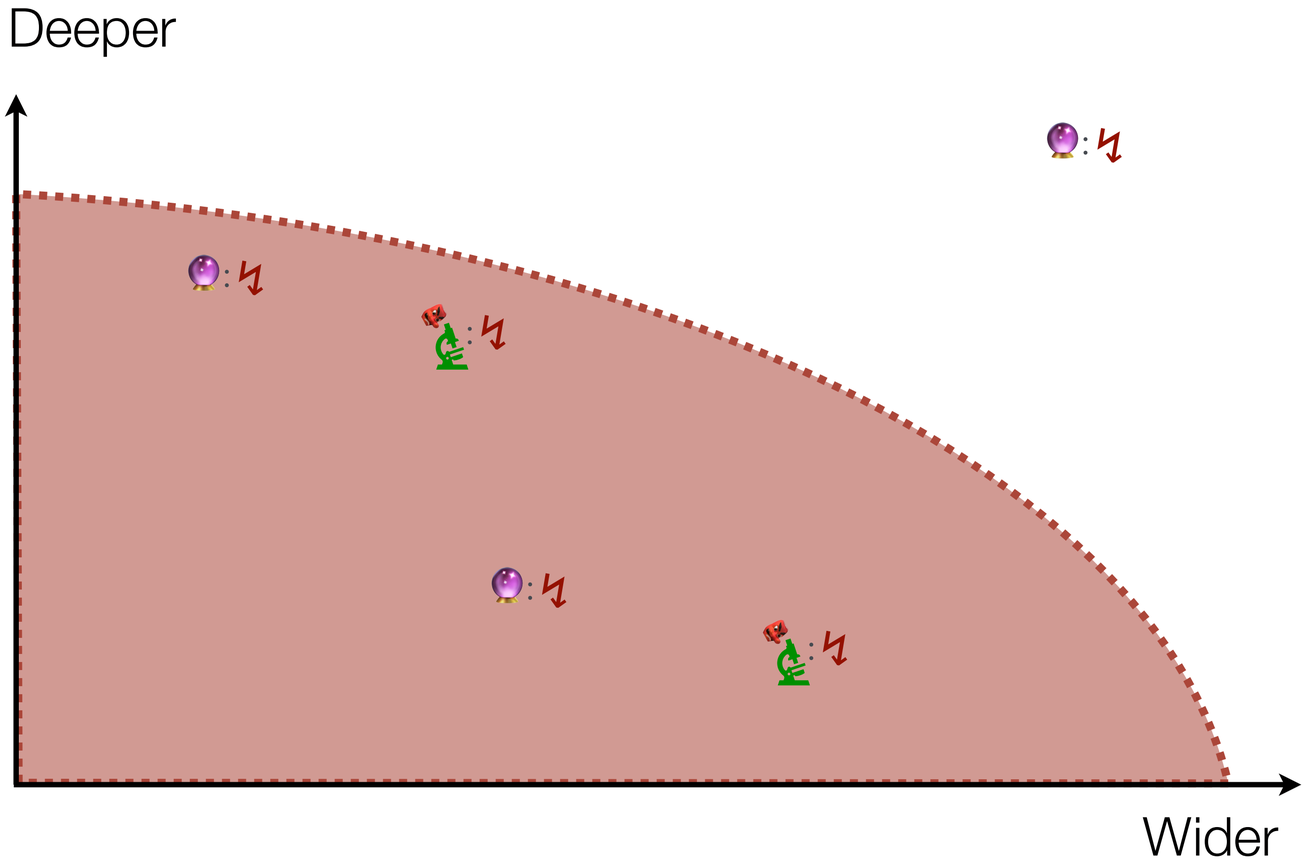
Formal methods: LLMs 🔮 are unreliable ⇾ let 🔮 write machine-checked proofs 🏅
⇾ **Still** requires foundations, what to prove, and well-scoped and sound models (hard part)

Fuzzing:



- 🔮 • Attack finding by LLMs
- 🏅 • Formally prove (possibly with 🔮)
- 🔬 • Fuzzing
- • Fuzzing with 🔮





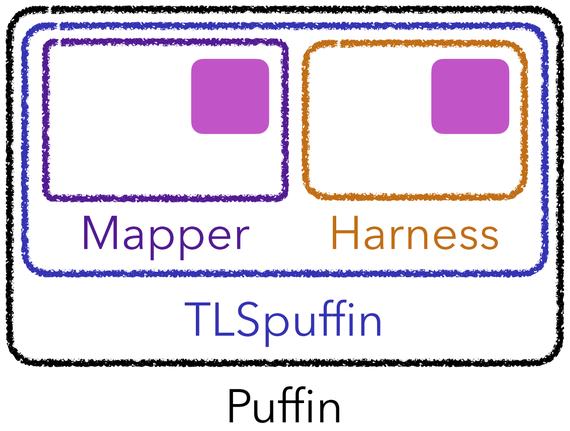
LLM: Existential crisis, what's beyond the hype peak?

Formal methods: LLMs 🔮 are unreliable ⇾ let 🔮 write machine-checked proofs 🏅
⇾ **Still** requires foundations, what to prove, and well-scoped and sound models (hard part)

Fuzzing:



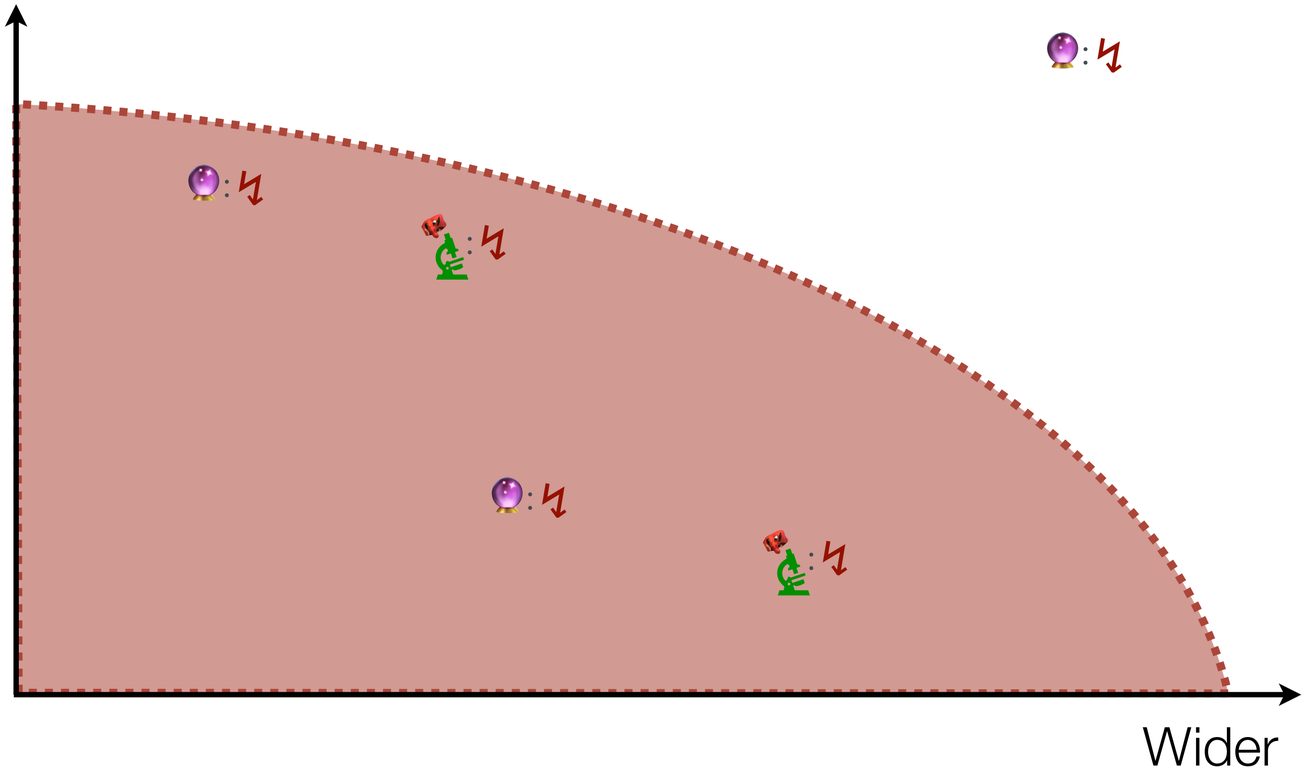
- 🔮 • Attack finding by LLMs
- 🏅 • Formally prove (possibly with 🔮)
- 🔬 • Fuzzing
- 🔬 • Fuzzing with 🔮



→ Objectives

↓ Triaging by 🔮

Bugs/CVEs



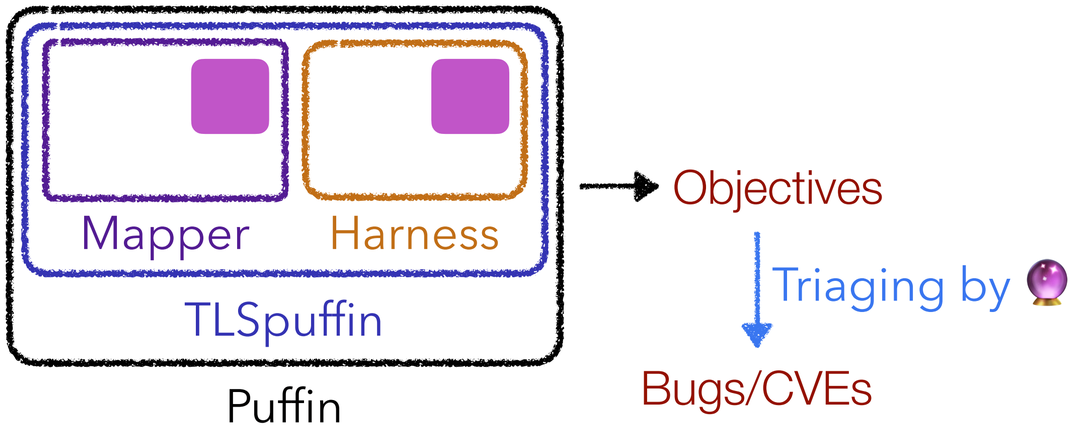
LLM: Existential crisis, what's beyond the hype peak?

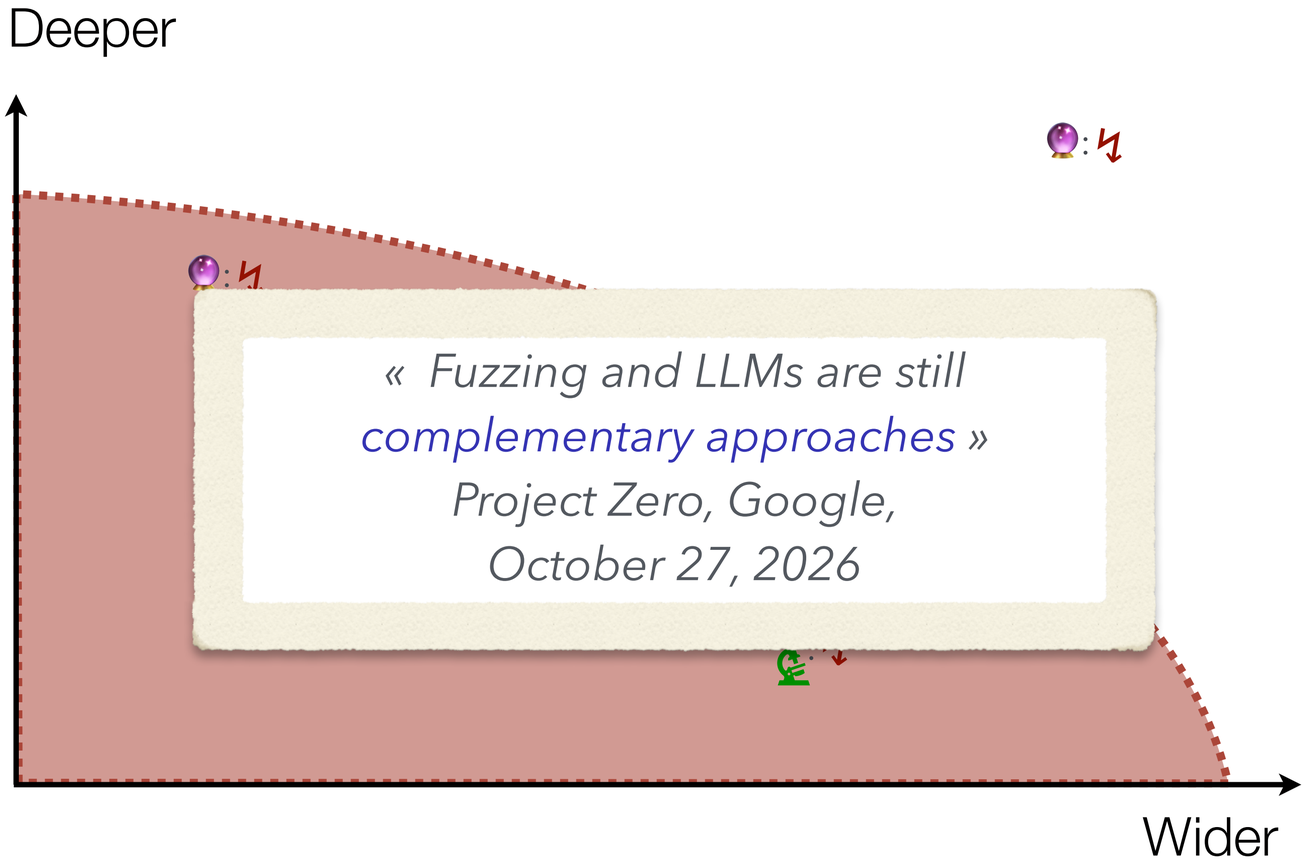
Formal methods: LLMs 🔮 are unreliable ⇾ let 🔮 write machine-checked proofs 🏅
⇾ **Still** requires foundations, what to prove, and well-scoped and sound models (hard part)

Fuzzing:



- 🔮 • Attack finding by LLMs
- 🏅 • Formally prove (possibly with 🔮)
- 🔬 • Fuzzing
- 🔬 • Fuzzing with 🔮





Science is collaborative! Thanks to my co-authors:

▶ **Max Ammann**, TU Munich, Germany & Trail of Bits, USA ▶ **David Baelde**, LSV, ENS Paris-Saclay, France ▶ **David Basin**, ETH Zürich, Switzerland ▶ **Ravishankar Borgaonkar**, TU Berlin, Germany & SINTEF Digital, Norway & NTNU, Norway ▶ **Vincent Cheval**, Inria Paris, France ▶ **Véronique Cortier**, LORIA, CNRS, France ▶ **Cas Cremers**, CISPA, Germany & University of Oxford, UK ▶ **Alexander Dax**, CISPA, Germany ▶ **Alexandre Debant**, LORIA, Inria Nancy, France ▶ **Stéphanie Delaune**, IRISA, Université de Rennes, France ▶ **Vincent Diemunsch**, ANSSI & LORIA, Inria Nancy, France ▶ **Jannik Dreier**, LORIA, Université de Lorraine, France ▶ **Guillaume Girol**, École Polytechnique, France & ETH Zürich, Switzerland & CEA, List, Université Paris-Saclay, France ▶ **Anselme Goetschmann**, LORIA, Inria Nancy, France ▶ **Tom Gouville** LORIA, Inria Nancy, France ▶ **Dennis Jackson**, University of Oxford, UK ▶ **Charlie Jacomme**, Inria Paris, France ▶ **Steve Kremer**, LORIA, Inria Nancy, France ▶ **Piers O'Hanlon**, University of Oxford, UK ▶ **Shinjo Park**, TU Berlin, Germany ▶ Saša **Radomirović**, ETH Zürich, Switzerland & University of Dundee, UK ▶ **Ralf Sasse**, ETH Zürich, Switzerland ▶ **Lara Schmid**, ETH Zürich, Switzerland ▶ **Altaf Shaik**, TU Berlin, Germany ▶ **Vincent Stettler**, ETH Zürich, Switzerland

And thanks to the engineers who worked with us:

▶ **Nataël Baffou** ▶ **Olivier Demengeon** ▶ **Michael Mera** — Inria, LORIA, France

