

DY Fuzzing Reunites with HAVOC

Lucca Hirschi, Inria, France

July 7, 2026 @ StANP Workshop



Scope: Crypto Protocol Implementation

Crypto
Primitives

ECDHE, RSA, AES GCM, SHA384
ElGamal encryption, SHA1

Protocol
Design

TLS, SSH, Signal, Wifi, Bluetooth,
e-voting protocols

**Protocol
Implementation**

OpenSSL, WolfSSL, BoringSSL,
libSSH, wolfSSH, Helios

Hardware

CPU, FPGA, TEE, etc.

Scope: Crypto Protocol Implementation

Crypto
Primitives

ECDHE, RSA, AES GCM, SHA384
ElGamal encryption, SHA1

Protocol
Design

TLS, SSH, Signal, Wifi, Bluetooth,
e-voting protocols

Verification through Formal methods:
Formal Analysis in Dolev-Yao Models



**Protocol
Implementation**

OpenSSL, WolfSSL, BoringSSL,
libSSH, wolfSSH, Helios

Hardware

CPU, FPGA, TEE, etc.

Scope: Crypto Protocol Implementation

Crypto
Primitives

ECDHE, RSA, AES GCM, SHA384
ElGamal encryption, SHA1

Protocol
Design

TLS, SSH, Signal, Wifi, Bluetooth,
e-voting protocols

**Protocol
Implementation**

OpenSSL, WolfSSL, BoringSSL,
libSSH, wolfSSH, Helios

Hardware

CPU, FPGA, TEE, etc.

Verification through Formal methods:
Formal Analysis in Dolev-Yao Models



Testing through Fuzzing:
E.g., Dolev-Yao Model-Guided Fuzzing



Scope: Crypto Protocol Implementation

Crypto
Primitives

ECDHE, RSA, AES GCM, SHA384
ElGamal encryption, SHA1

Protocol
Design

TLS, SSH, Signal, Wifi, Bluetooth,
e-voting protocols

**Protocol
Implementation**

OpenSSL, WolfSSL, BoringSSL,
libSSH, wolfSSH, Helios

Hardware

CPU, FPGA, TEE, etc.

Verification through Formal methods:
Formal Analysis in Dolev-Yao Models



Testing through Fuzzing:
E.g., Dolev-Yao Model-Guided Fuzzing



Dolev-Yao Fuzzing



DY Fuzzing: Formal Dolev-Yao Models Meet Cryptographic Protocol Fuzz Testing

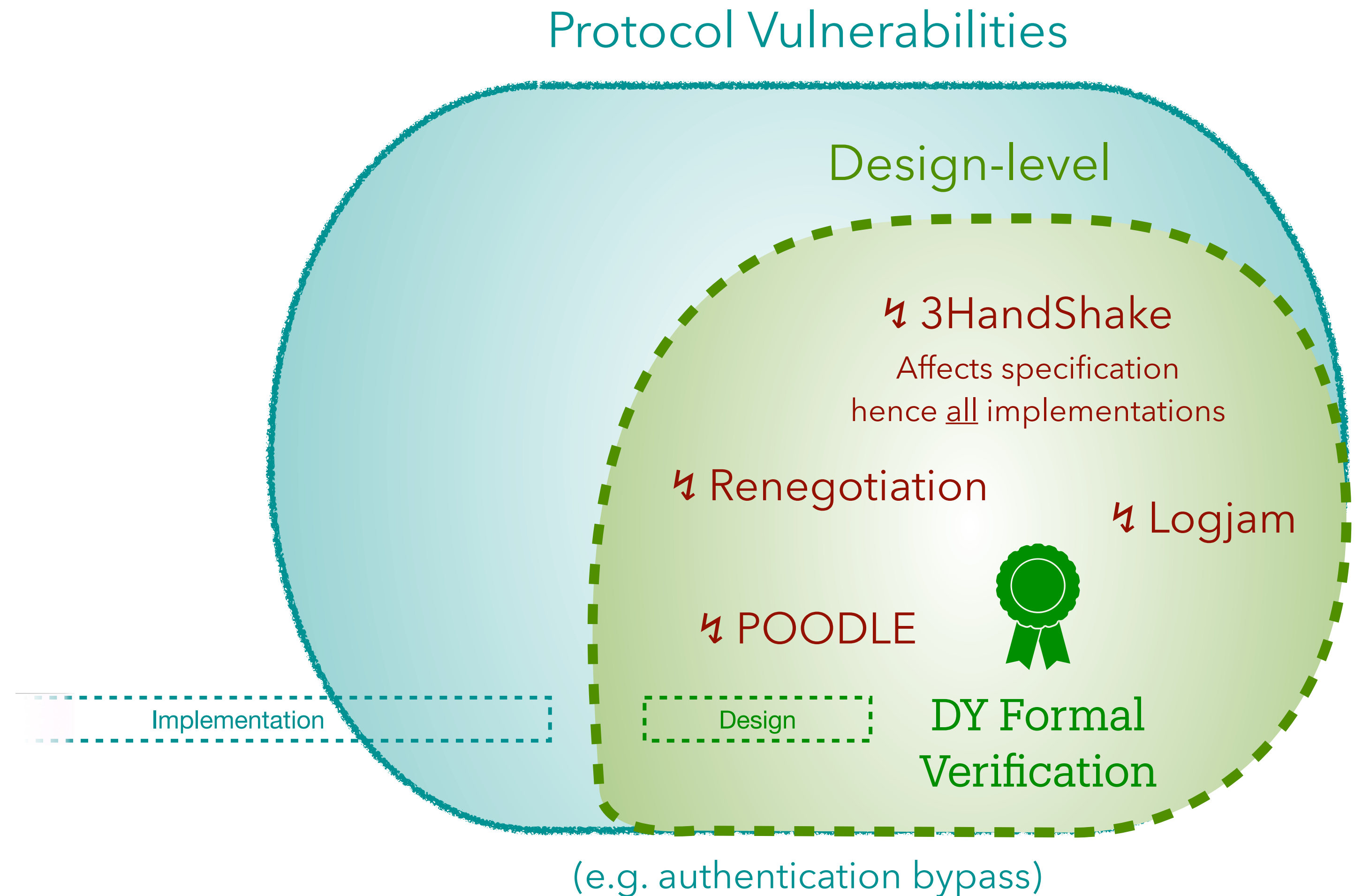
Max Ammann
*Independent Researcher &
Trail of Bits*
max@maxammann.org

Lucca Hirschi
*Inria Nancy Grand-Est
Université de Lorraine, LORIA, France*
lucca.hirschi@inria.fr

Steve Kremer
*Inria Nancy Grand-Est
Université de Lorraine, LORIA, France*
steve.kremer@inria.fr

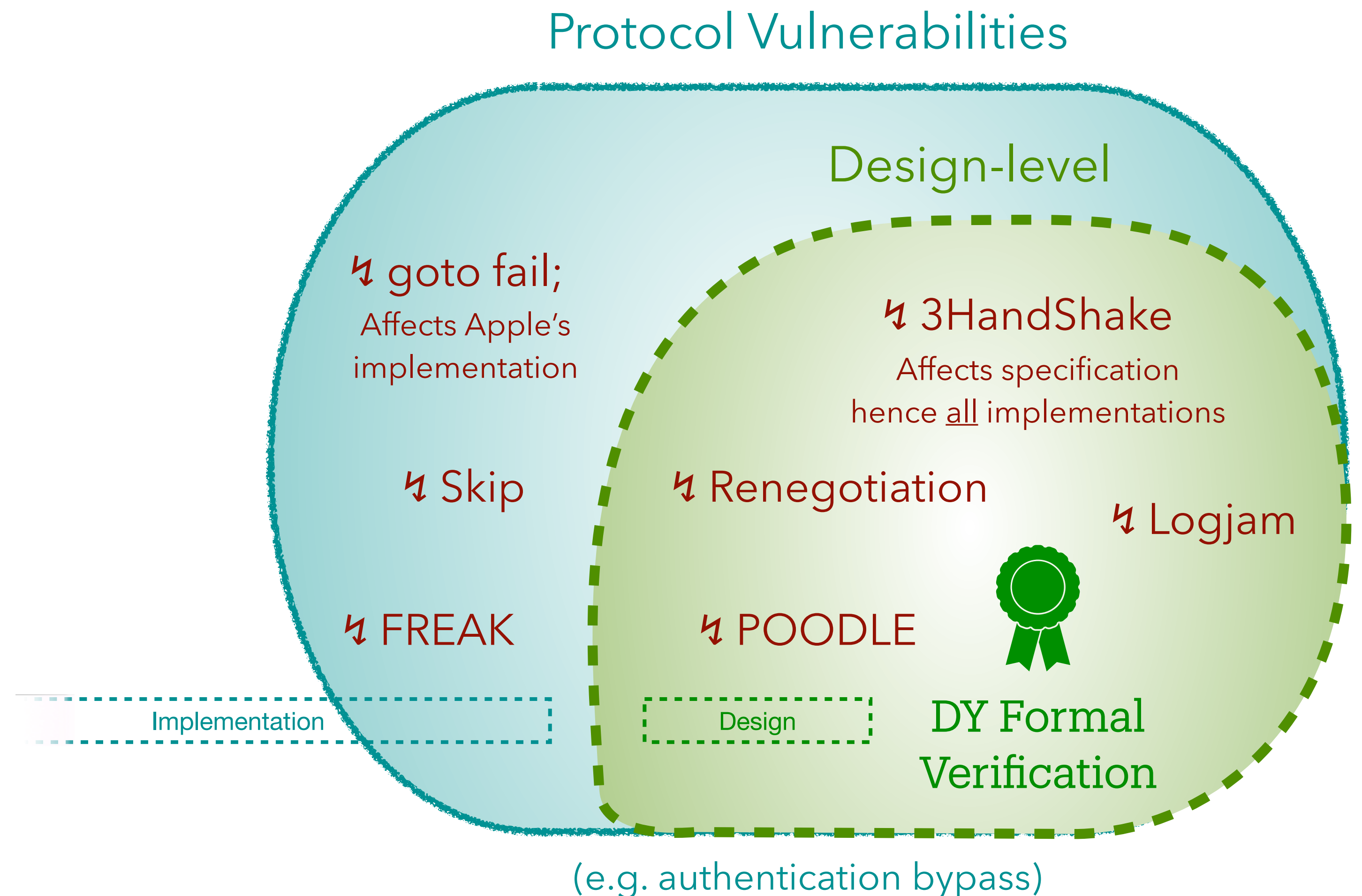
Scope of Symbolic Verification

Illustrated on some attacks ⚡ on TLS (2019-2022)  <https://>



Scope of Symbolic Verification

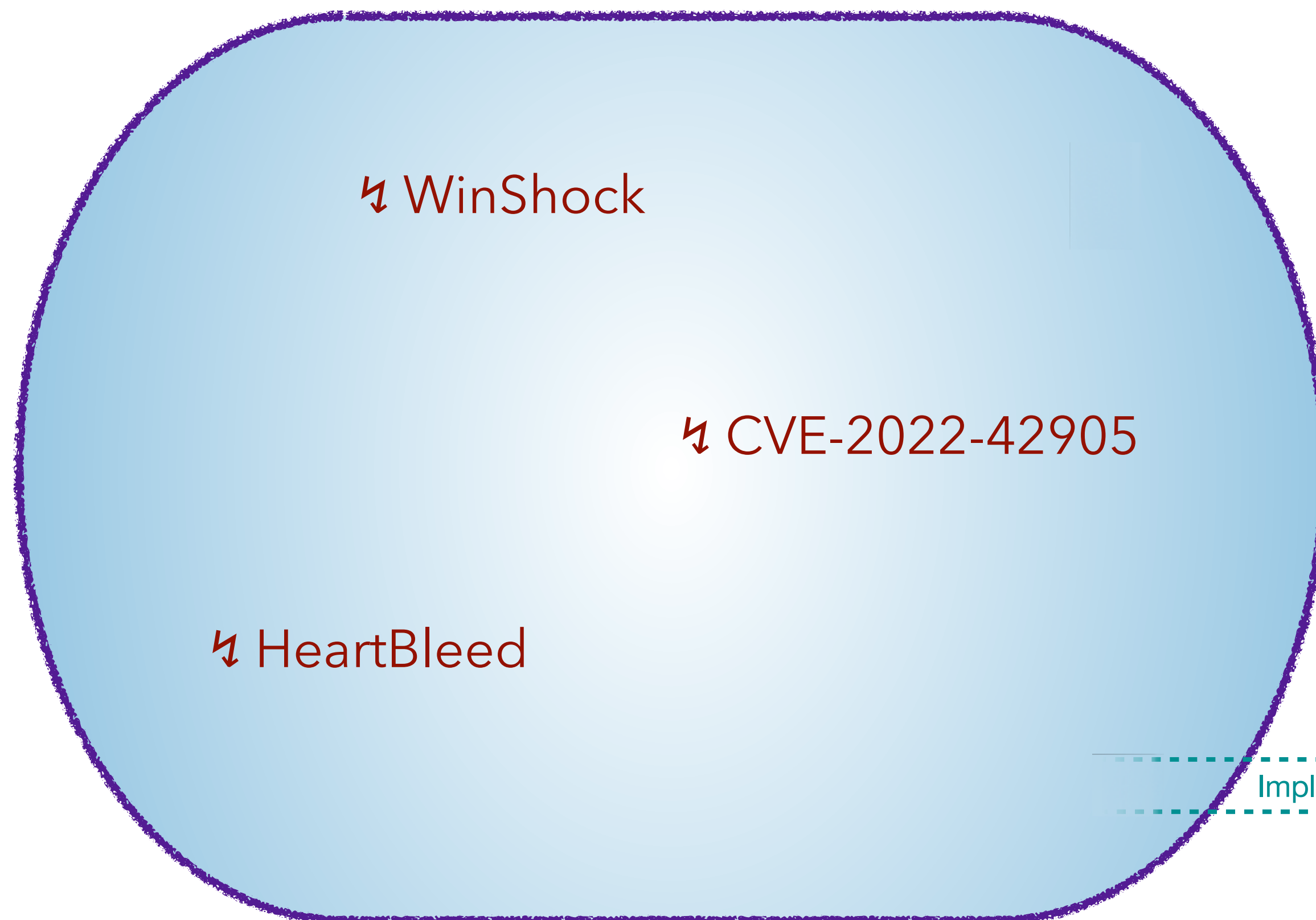
Illustrated on some attacks ⚡ on TLS (2019-2022)  <https://>



Scope of Symbolic Verification

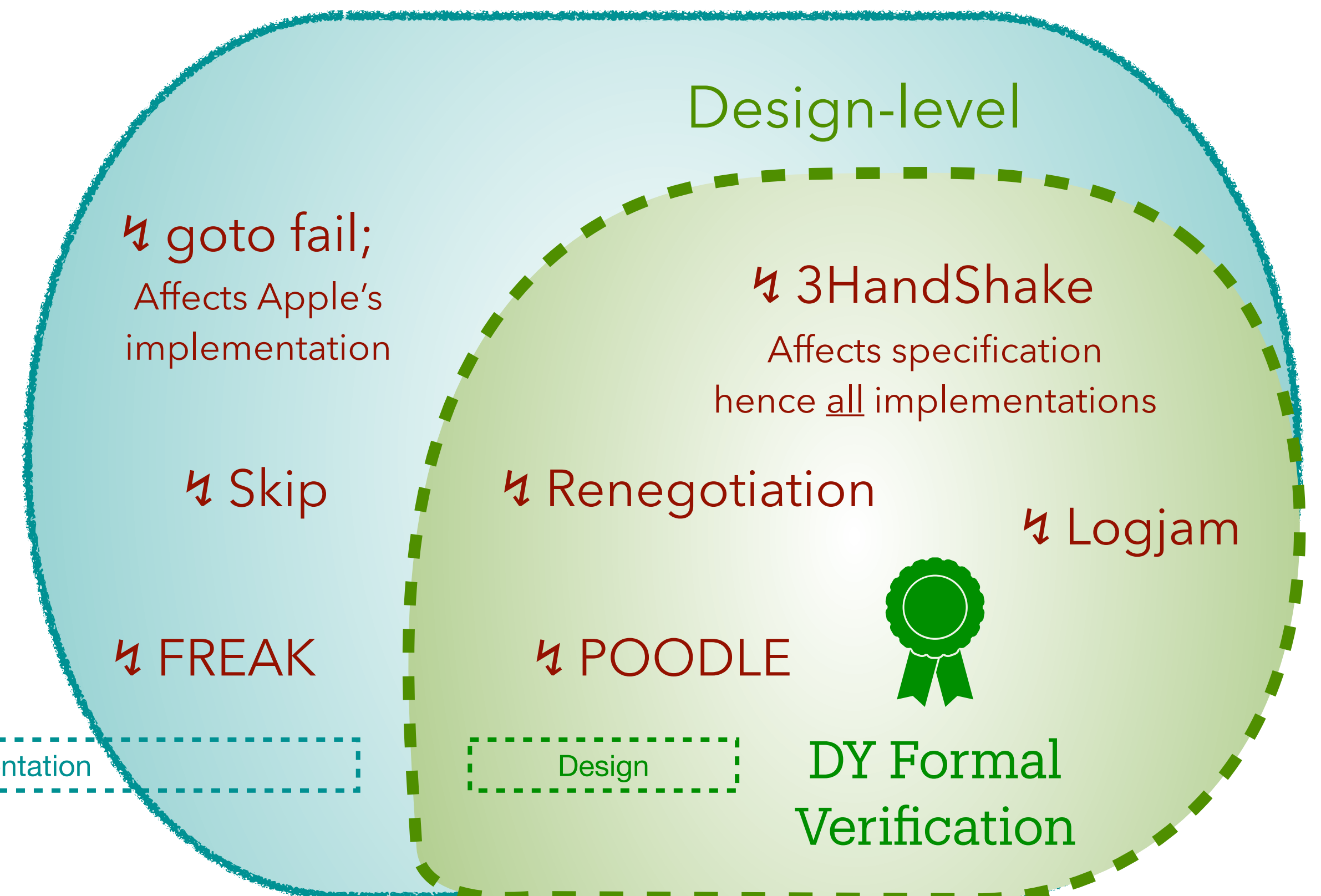
Illustrated on some attacks ↱ on TLS (2019-2022)  <https://>

Memory Safety Vulnerabilities



(e.g. buffer-overflow)

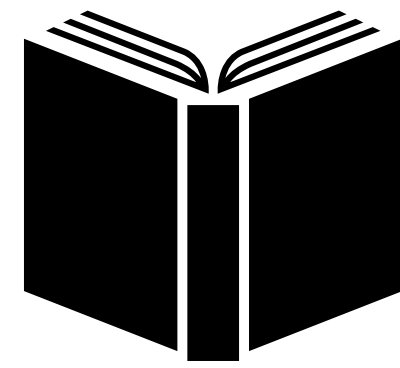
Protocol Vulnerabilities



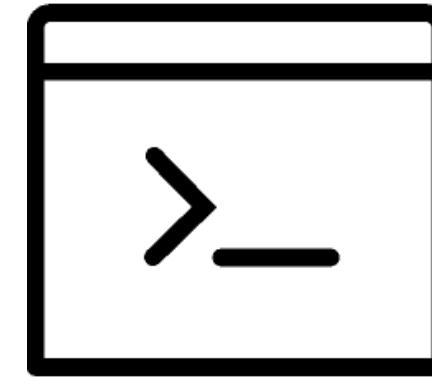
(e.g. authentication bypass)

Bit-level Fuzzing (AFL-like)

Corpus of test-cases

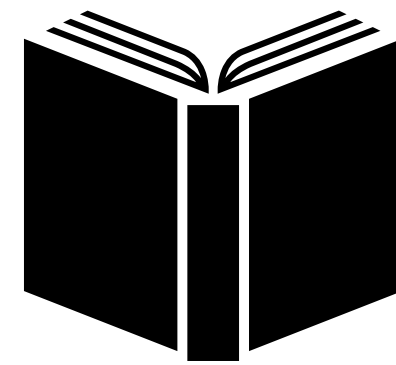


Program Under Test (PUT)

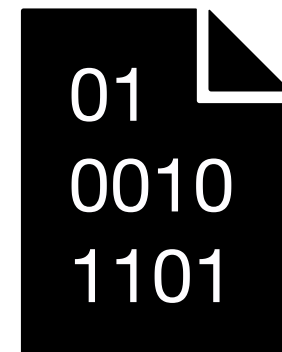


Bit-level Fuzzing (AFL-like)

Corpus of test-cases

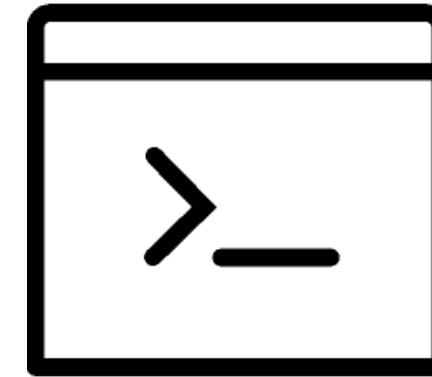


Pick a test-case

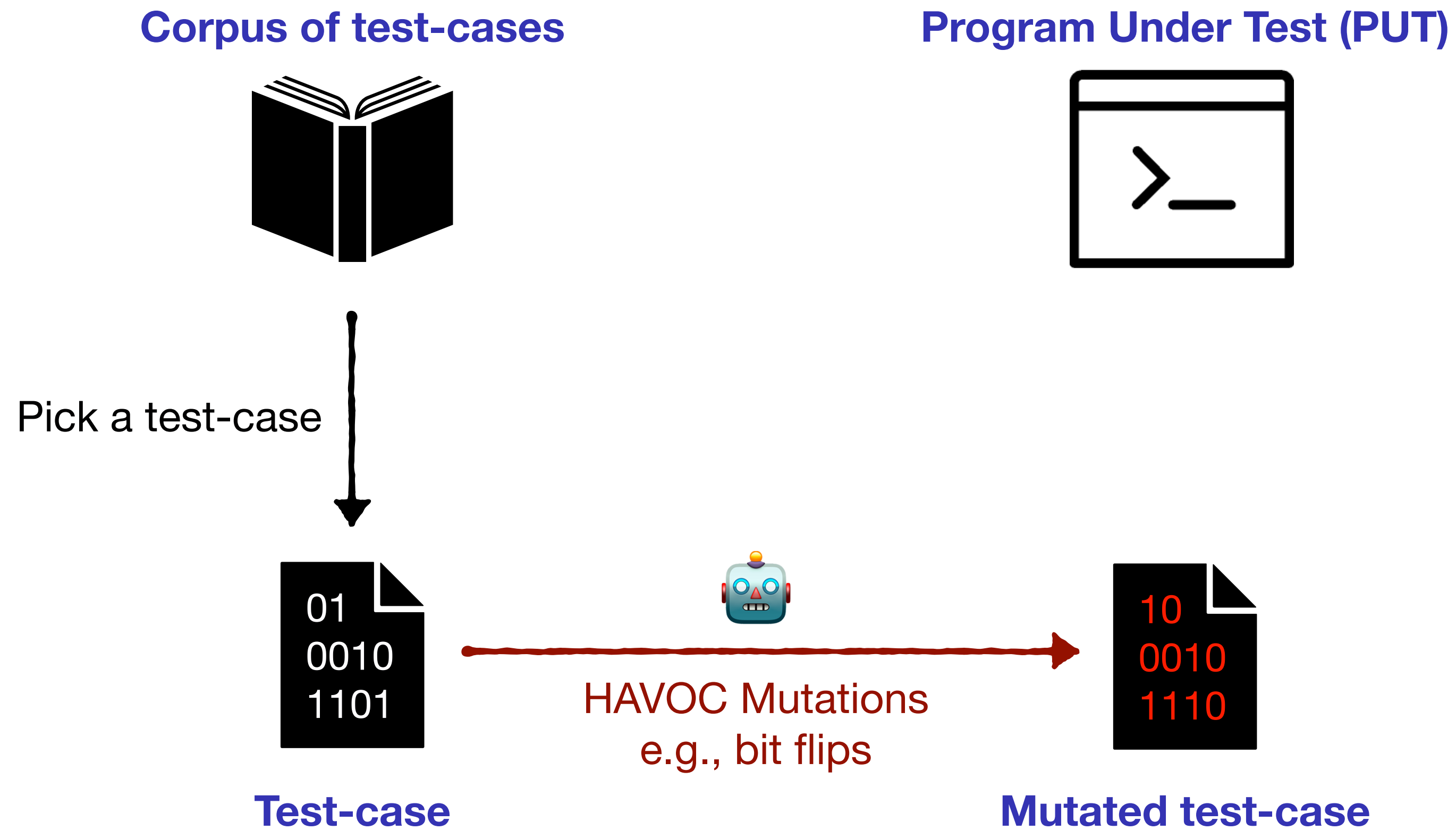


Test-case

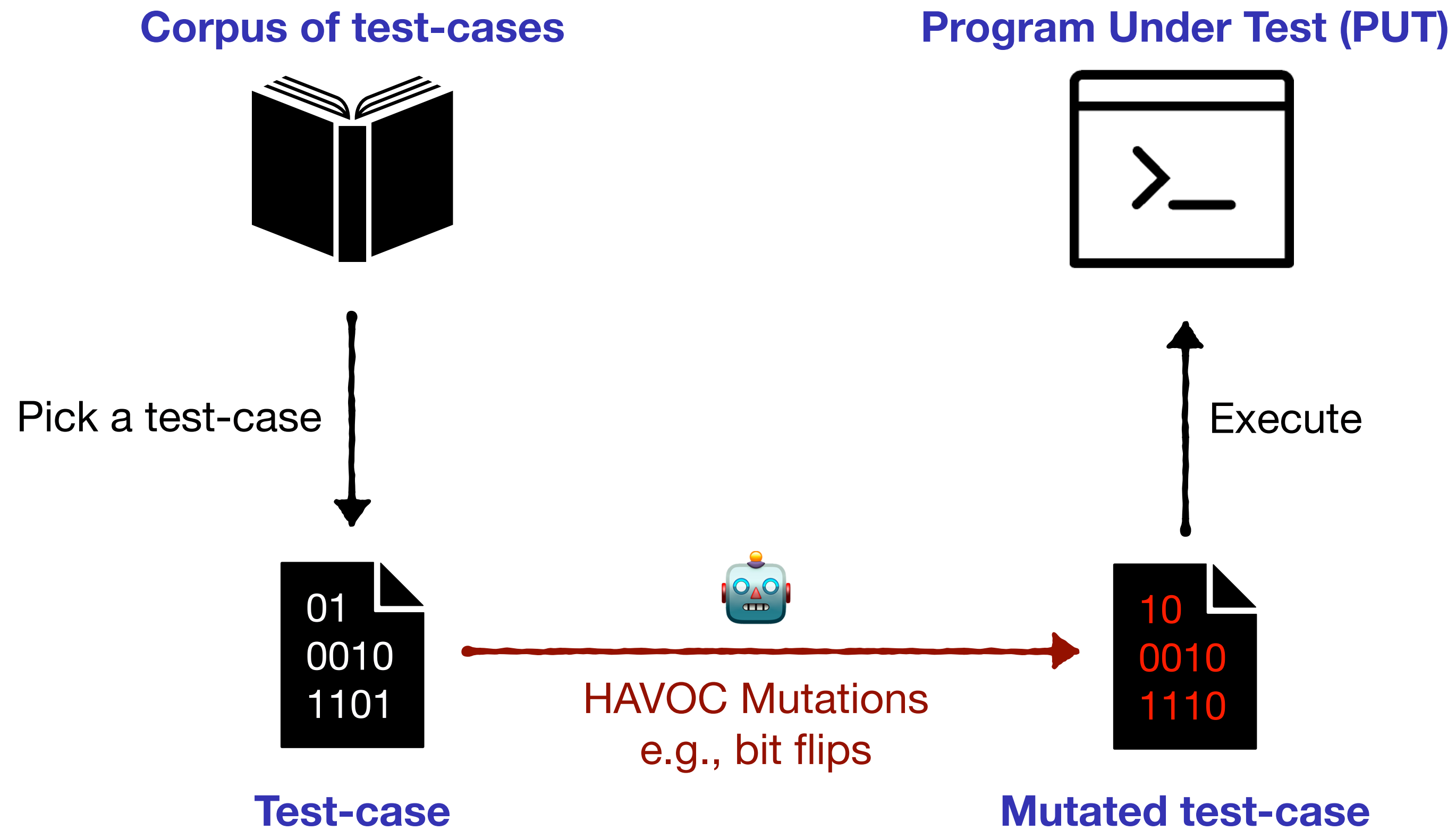
Program Under Test (PUT)



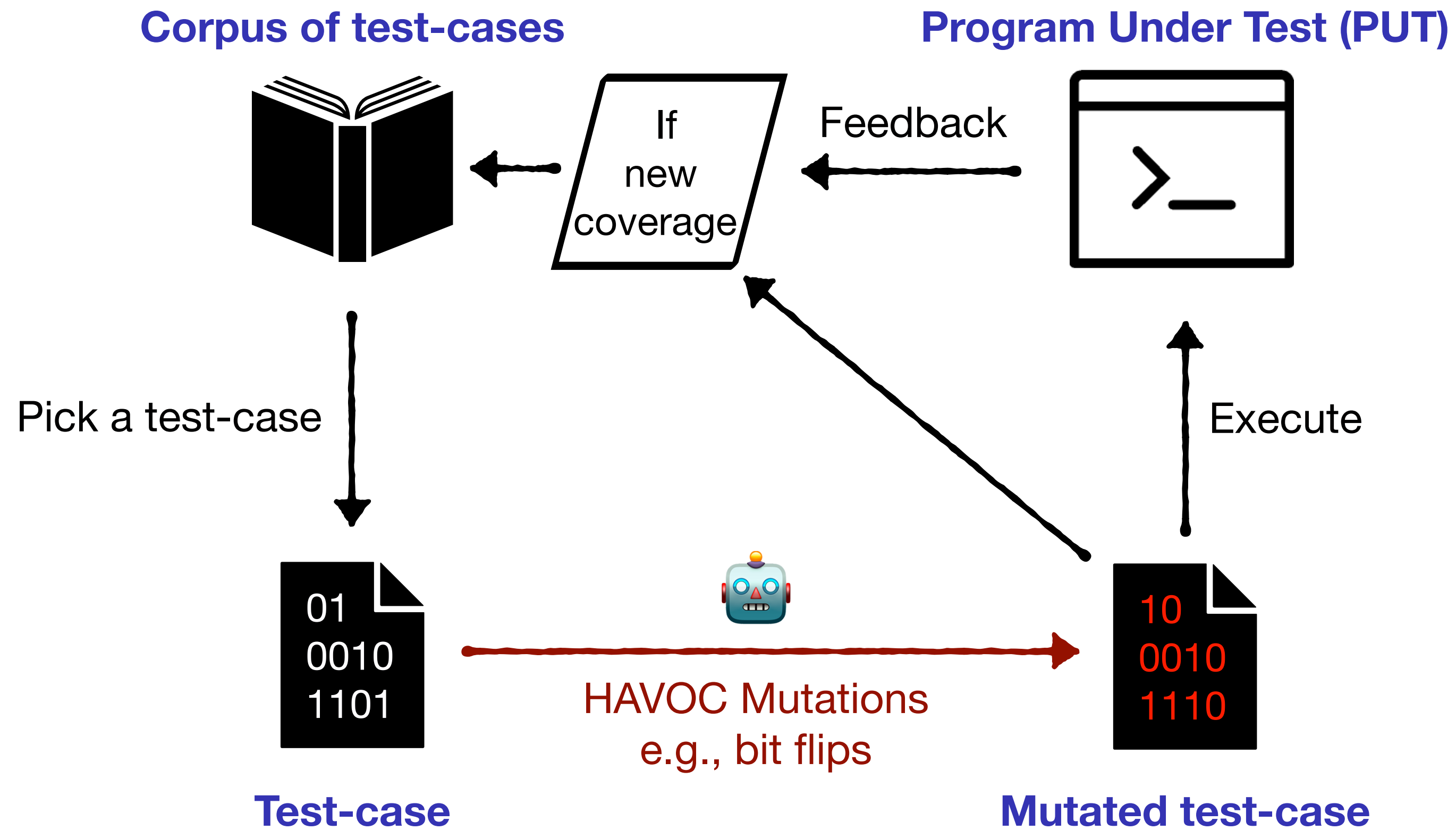
Bit-level Fuzzing (AFL-like)



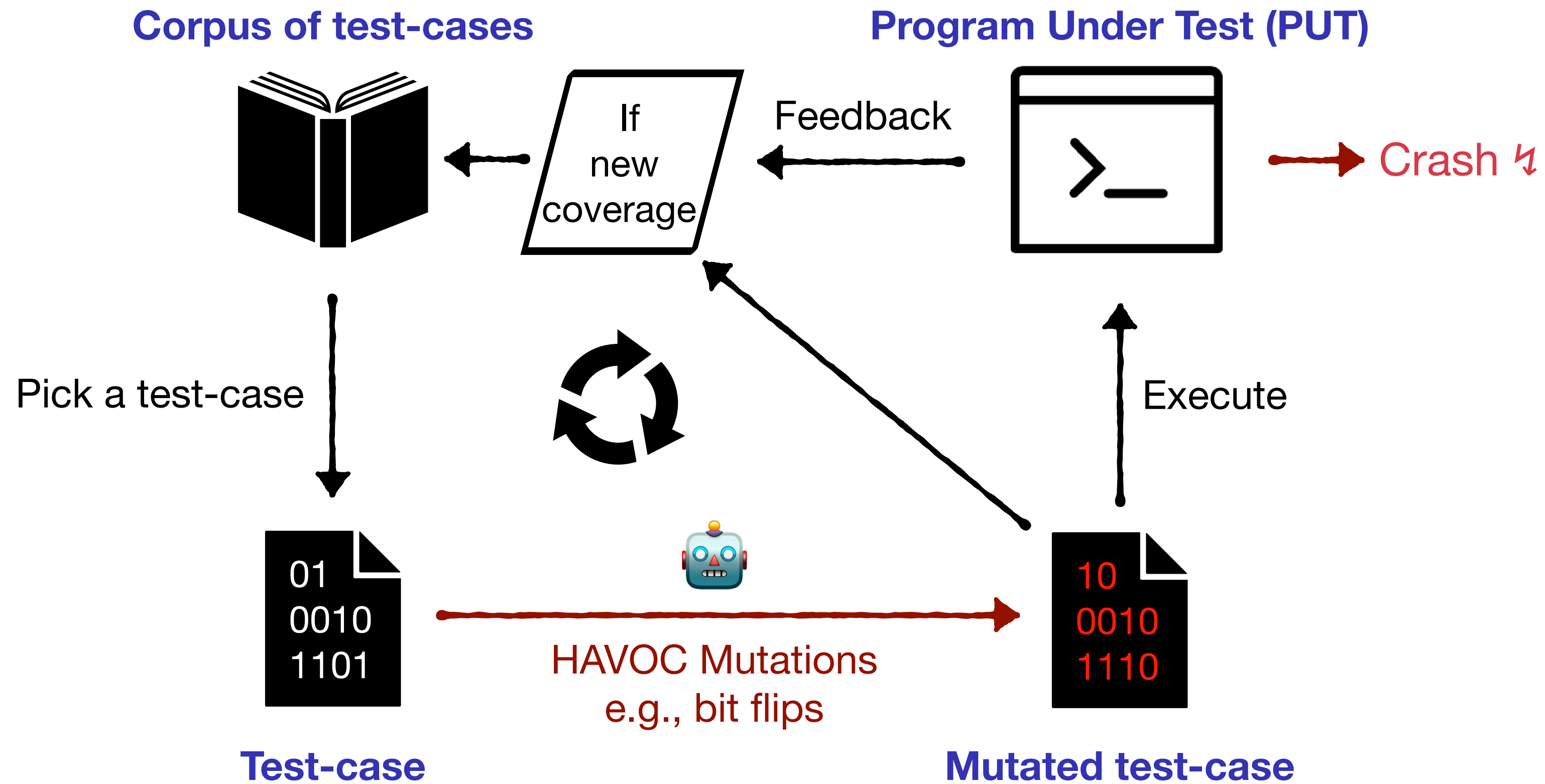
Bit-level Fuzzing (AFL-like)



Bit-level Fuzzing (AFL-like)



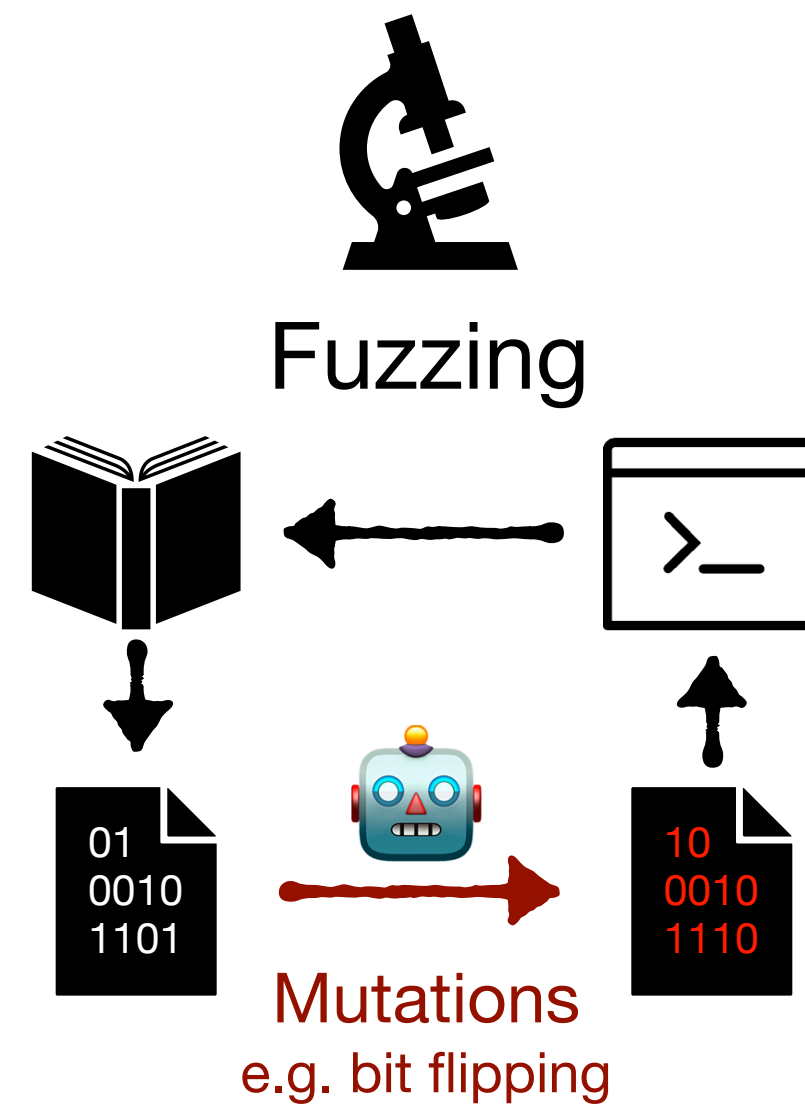
Bit-level Fuzzing (AFL-like)



Bit-level Fuzzing (AFL-like)

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

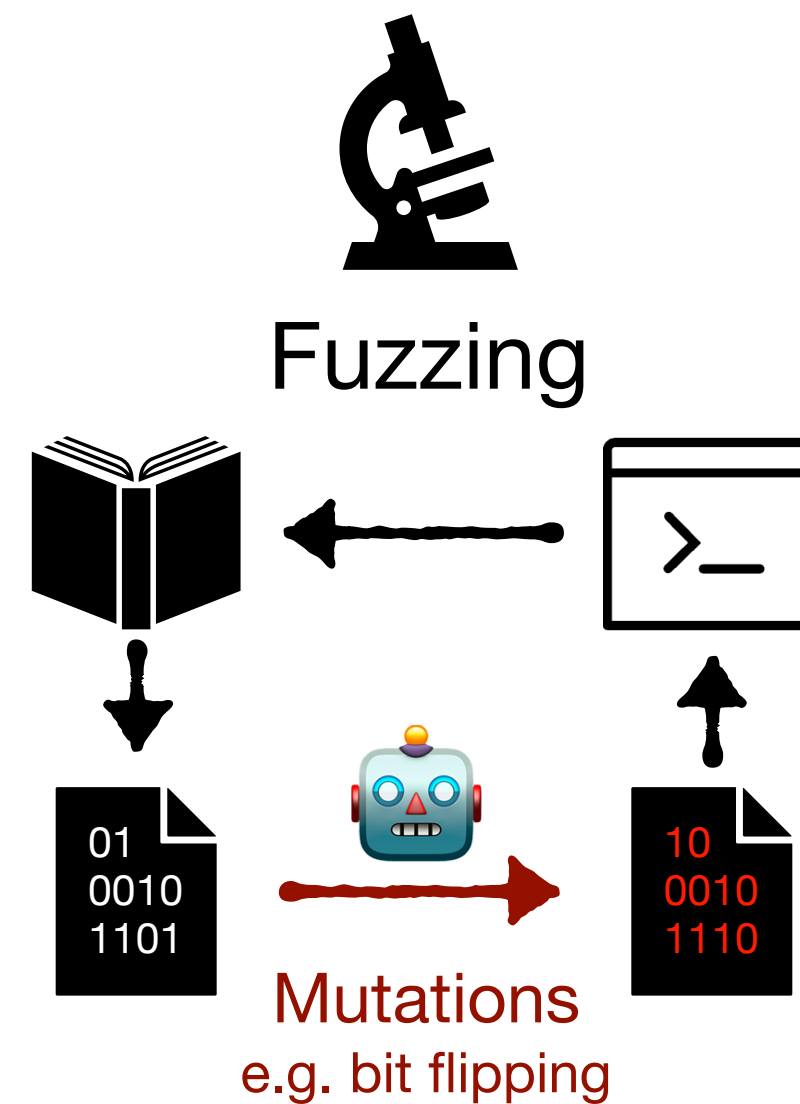
    if (SSL_do_handshake(server) == 1) {
        /* Keep reading data */
        uint8_t tmp[1024];
        for (;;) {
            if (SSL_read(server, tmp) <= 0) {
                break;
            }
        }
    }
    return 0;
}
```



Bit-level Fuzzing (AFL-like)

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

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



Cryptographic protocol uses cryptographic primitives

Bit-level mutations won't interact well with these!

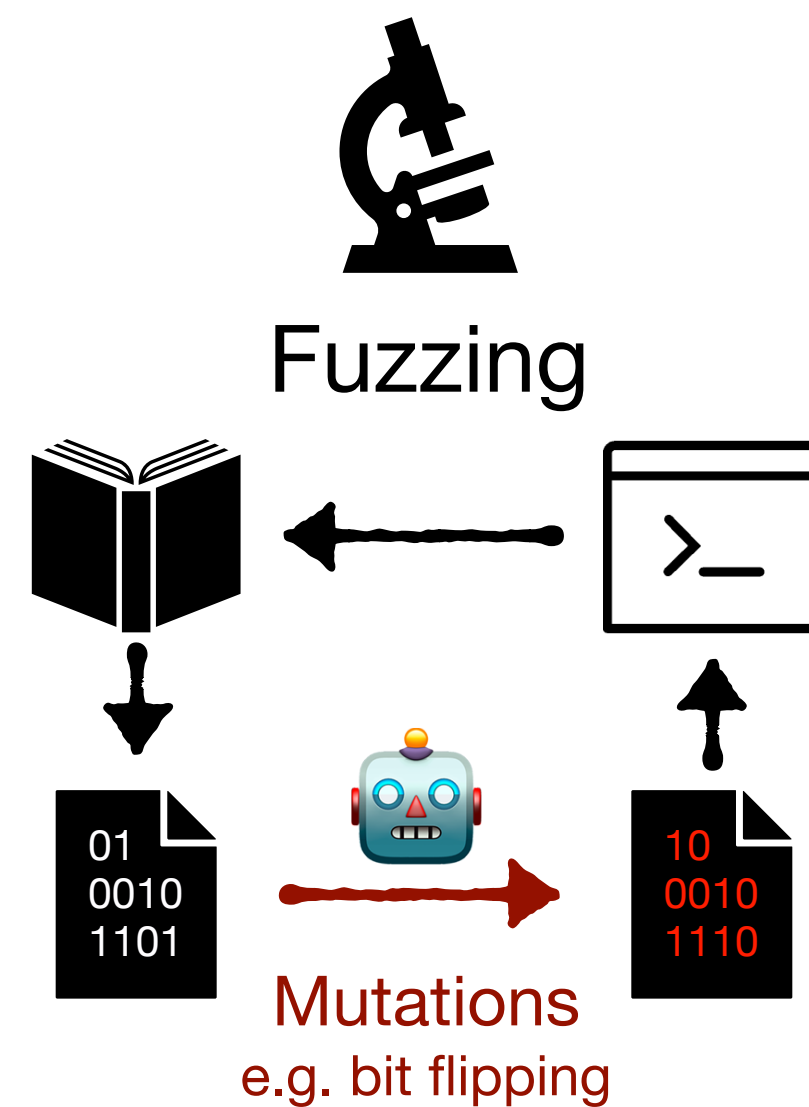
Cryptographic protocols are interactive programs

Bit-level: 1 test-case = 1 bitstring

Bit-level Fuzzing (AFL-like)

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

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



Cryptographic protocol uses cryptographic primitives

Bit-level mutations won't interact well with these!

Cryptographic protocols are interactive programs

Bit-level: 1 test-case = 1 bitstring

✗ Fuzzing Limitation 1:
Bitstring-level mutations only

No structural message/message flow modification
e.g. negligible probability of computing cryptographic and other structural modifications through bit-level mutations

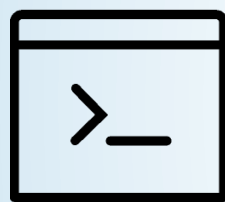
👉 Reachability problem

Memory Safety Vulnerabilities

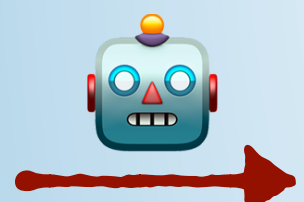
⚡ HeartBleed



Fuzzing



01
0010
1101



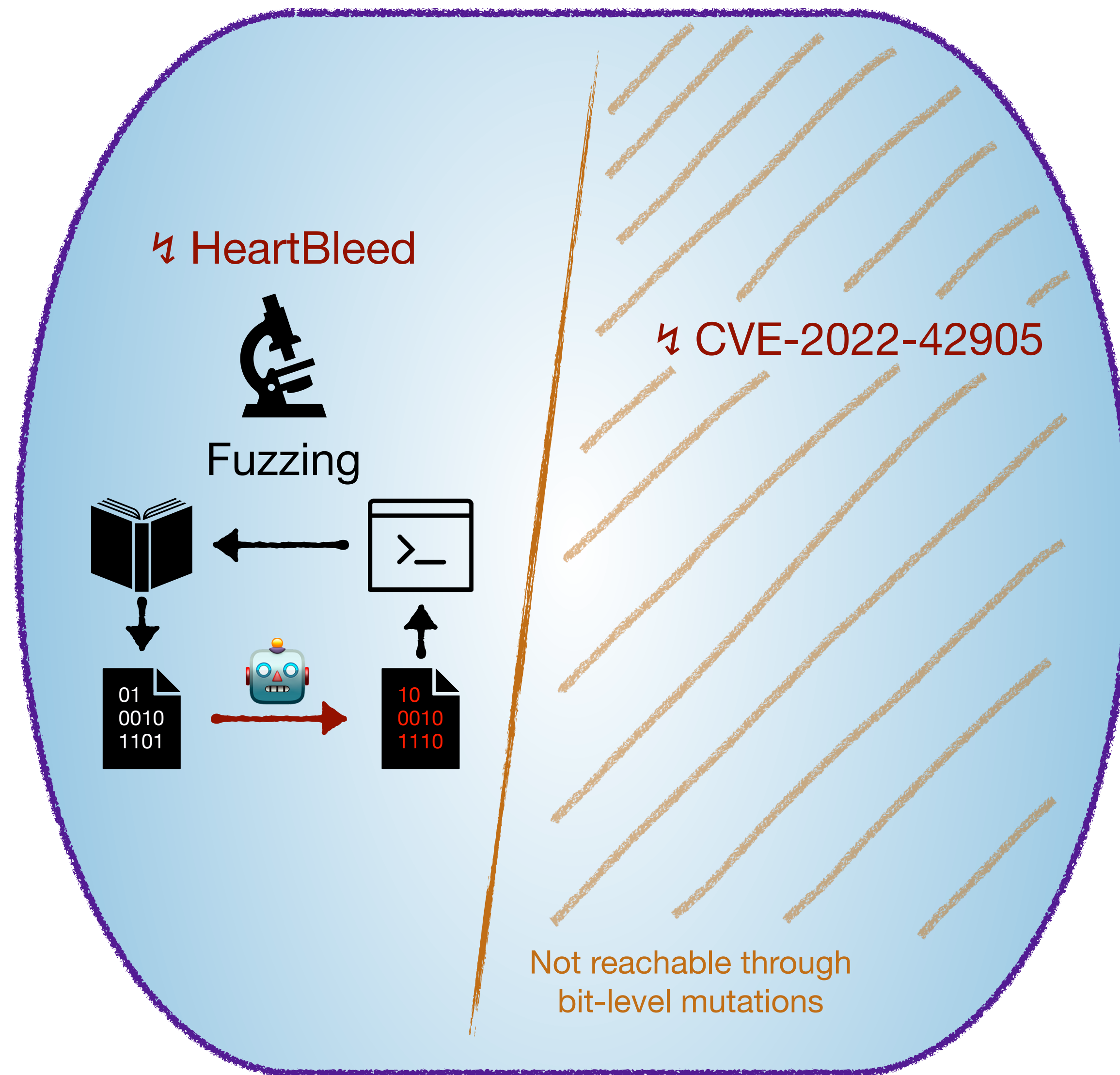
10
0010
1110

⚡ CVE-2022-42905

Not reachable through
bit-level mutations

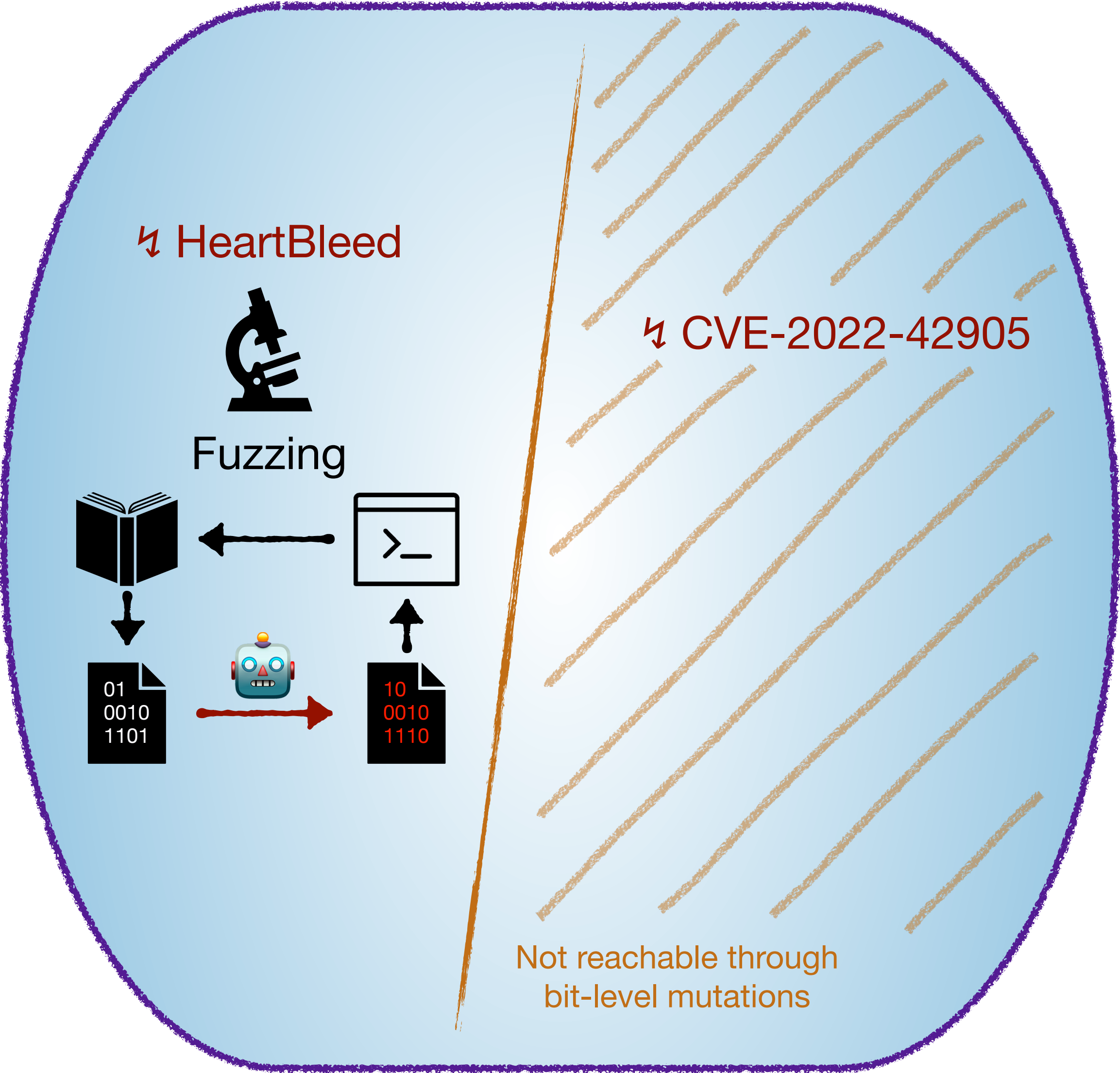
(e.g., buffer-overflow)

Memory Safety Vulnerabilities



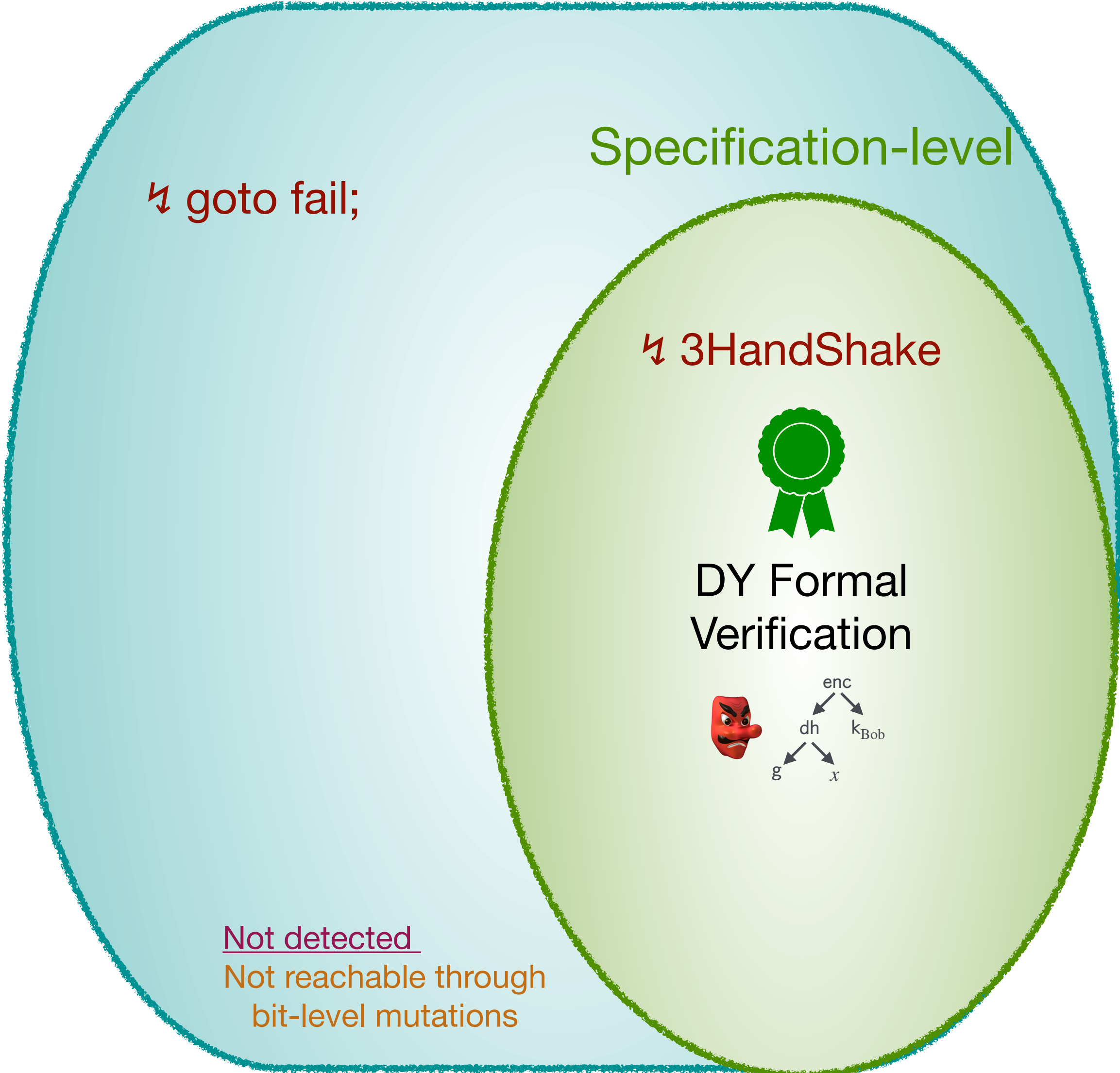
(e.g., buffer-overflow)

Memory Safety Vulnerabilities



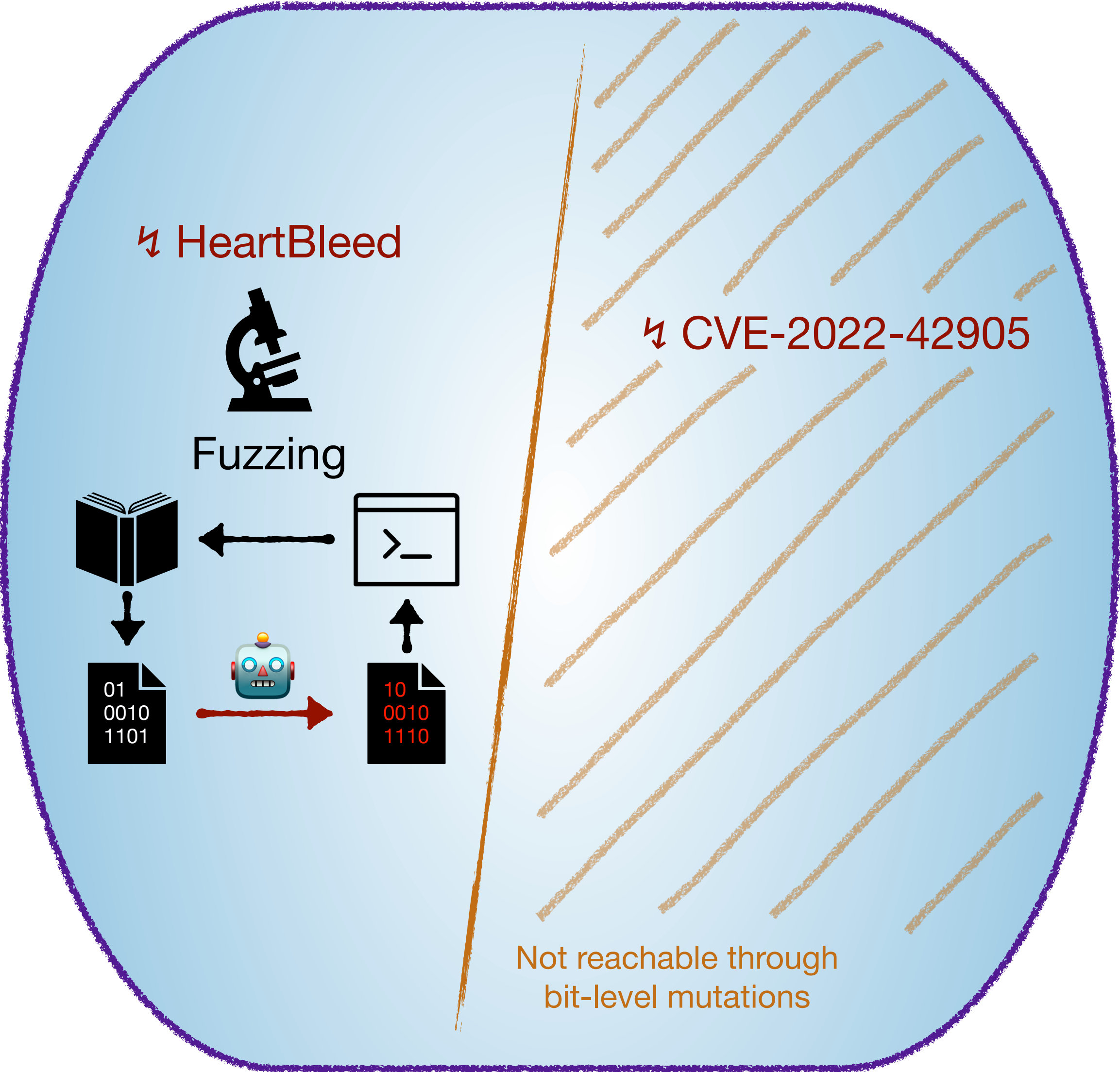
(e.g., buffer-overflow)

Protocol Vulnerabilities

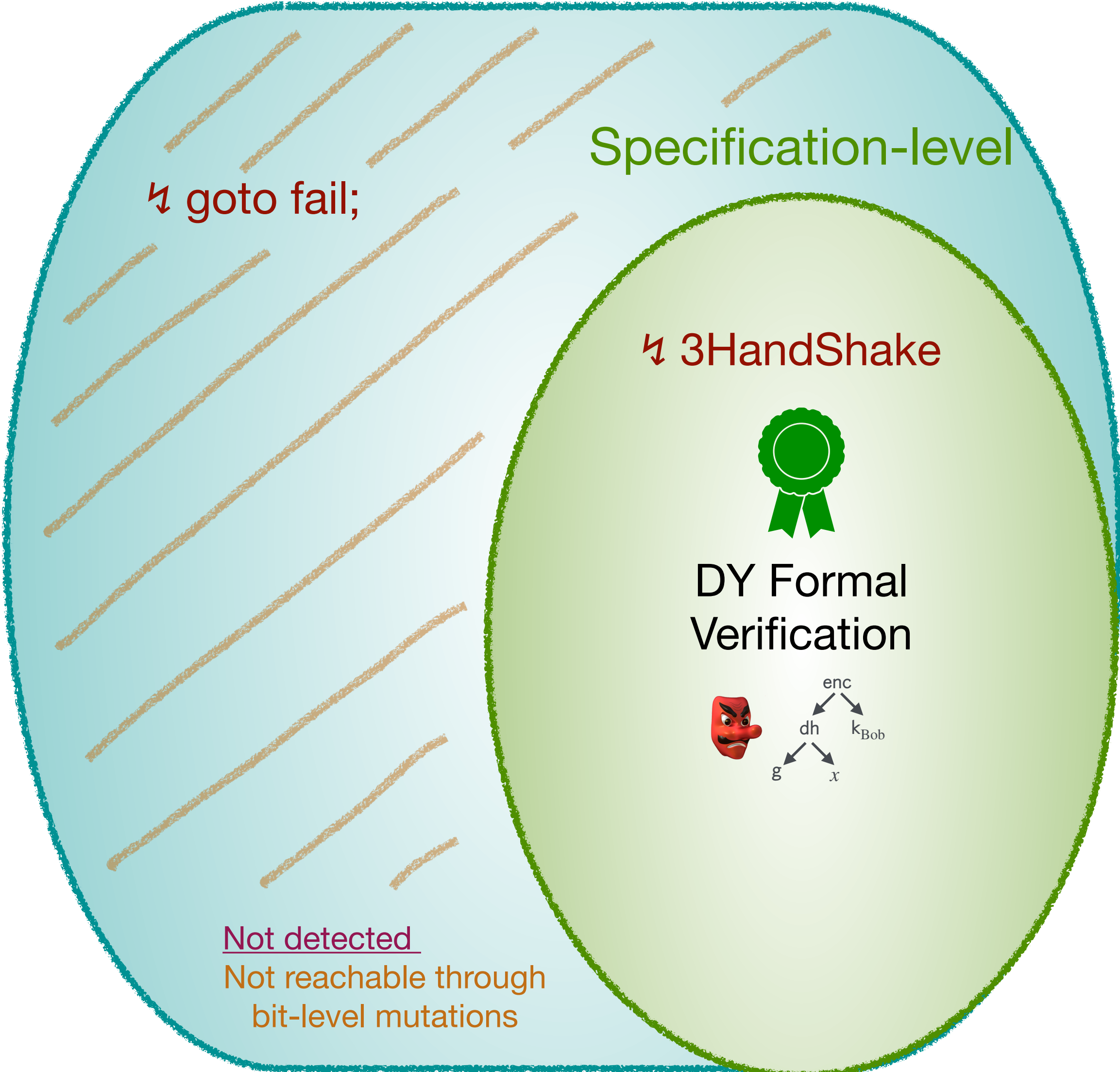


(e.g., authentication bypass)

Memory Safety Vulnerabilities



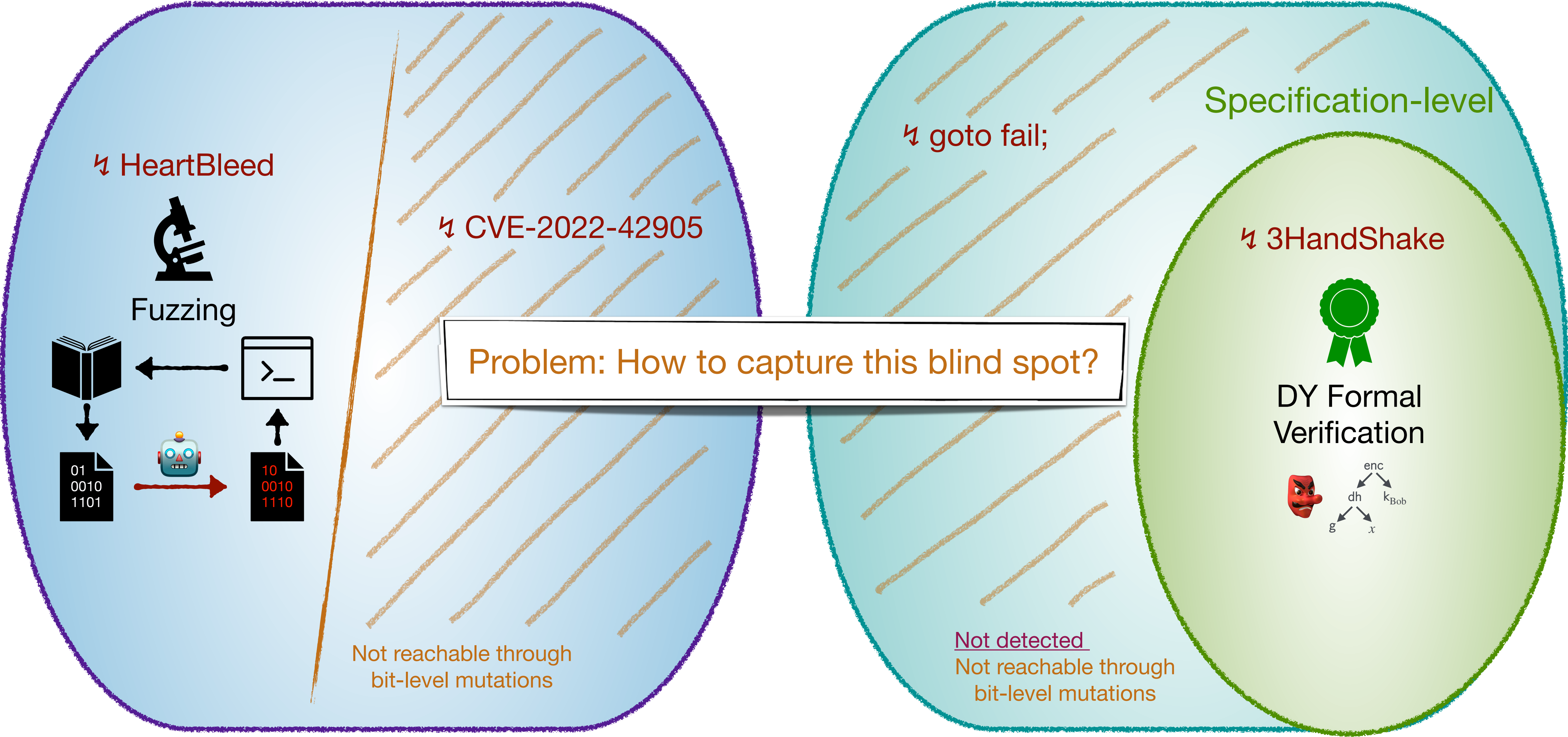
Protocol Vulnerabilities



✗ Fuzzing Limitation 2: inability to detect protocol vulnerabilities ➡ Detection problem

Memory Safety Vulnerabilities

Protocol Vulnerabilities



✗ Fuzzing Limitation 2: inability to detect protocol vulnerabilities ➡ Detection problem

Securing Cryptographic Protocol Fuzz Implementations

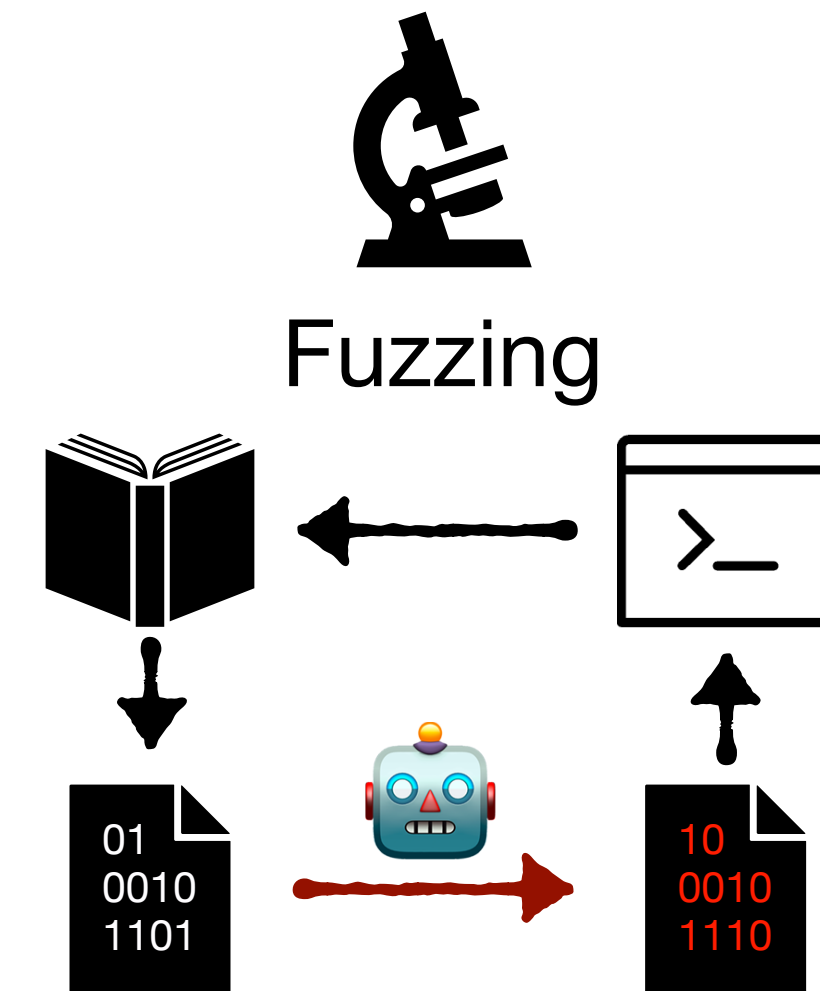
The Blind Spot

✗ Fuzzing Limitation 1: Reachability
Bitstring-level mutations only

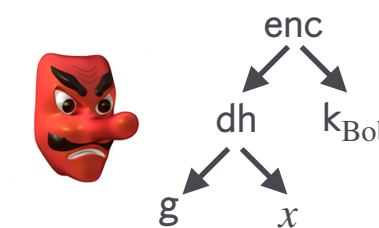
✗ Fuzzing Limitation 2: Detection
Inability to detect protocol vulnerabilities

(both limitations similarly apply to State-Machine-based Fuzzers)

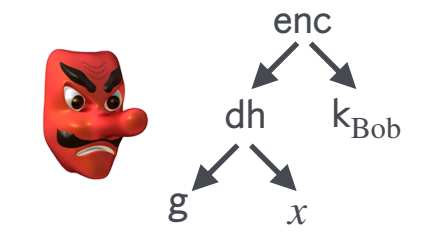
✗ DY Verification Limitation: Spec. only
No guarantee on implementation



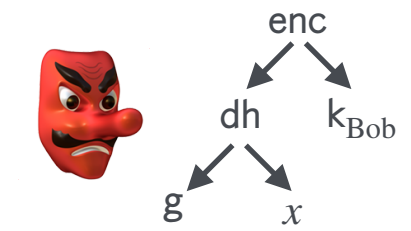

DY Formal
Verification



DY Message and Attacker Model



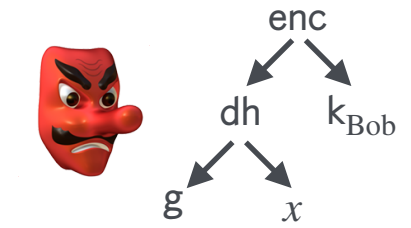
DY Message and Attacker Model



- A signature is a set $\mathcal{F}$ of *function symbols* with arity

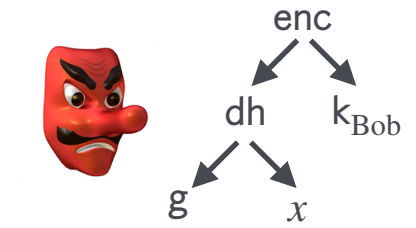
$$\mathcal{F} = \{\text{enc}[2], \text{dec}[2], \text{hash}[1], \text{dh}[2], \text{g}[0], \text{k}_{\text{Bob}}[0], \dots\}$$

DY Message and Attacker Model



- A signature is a set $\mathcal{F}$ of *function symbols* with arity $\mathcal{F} = \{\text{enc}[2], \text{dec}[2], \text{hash}[1], \text{dh}[2], \text{g}[0], \text{k}_{\text{Bob}}[0], \dots\}$
- The set of terms $\mathcal{T}$ is inductively defined as *variables* from $\mathcal{X}$ and by application of *function symbols* on *terms*

DY Message and Attacker Model

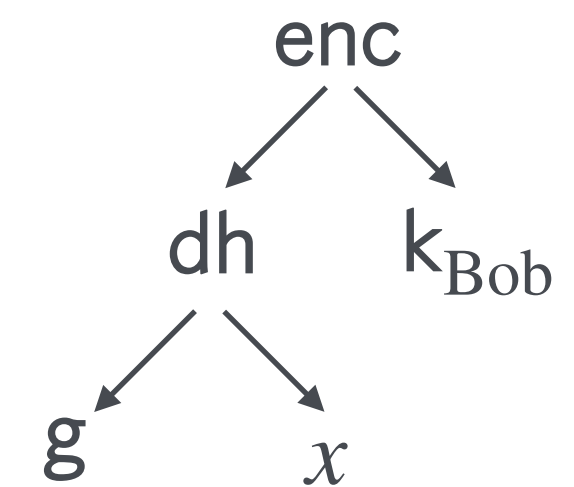


- A signature is a set $\mathcal{F}$ of *function symbols* with arity

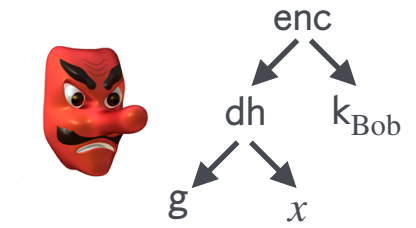
$$\mathcal{F} = \{\text{enc}[2], \text{dec}[2], \text{hash}[1], \text{dh}[2], g[0], k_{\text{Bob}}[0], \dots\}$$

- The set of terms $\mathcal{T}$ is inductively defined as *variables* from $\mathcal{X}$ and by application of *function symbols* on *terms*

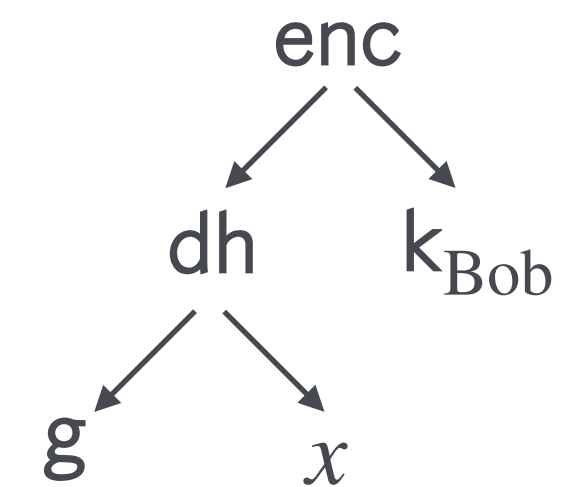
$$\text{enc}(\text{dh}(g, x), k_{\text{Bob}}) \in \mathcal{T}$$



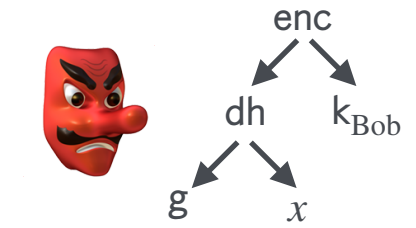
DY Message and Attacker Model



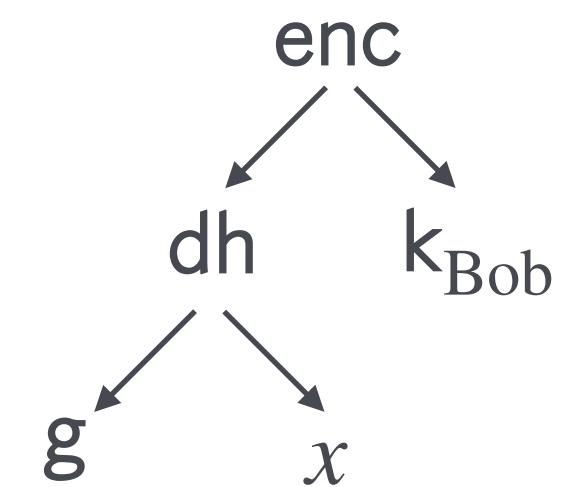
- A signature is a set $\mathcal{F}$ of *function symbols* with arity $\mathcal{F} = \{\text{enc}[2], \text{dec}[2], \text{hash}[1], \text{dh}[2], \text{g}[0], \text{k}_{\text{Bob}}[0], \dots\}$
- The set of terms $\mathcal{T}$ is inductively defined as *variables* from $\mathcal{X}$ and by application of *function symbols* on *terms*
 $\text{enc}(\text{dh}(\text{g}, x), \text{k}_{\text{Bob}}) \in \mathcal{T}$
- A trace is a list of actions (*attacker's interaction* with agents in $\mathcal{A}$)
 - **in**(a, R) // for $a \in \mathcal{A}$ (agent) and $R \in \mathcal{T}$ (*attacker term*)
 - **out**(a, x) // for $a \in \mathcal{A}$ (agent) and $x \in \mathcal{X}$ (*attacker knowledge*)



DY Message and Attacker Model



- A signature is a set $\mathcal{F}$ of *function symbols* with arity $\mathcal{F} = \{\text{enc}[2], \text{dec}[2], \text{hash}[1], \text{dh}[2], \text{g}[0], \text{k}_{\text{Bob}}[0], \dots\}$
- The set of terms $\mathcal{T}$ is inductively defined as *variables* from $\mathcal{X}$ and by application of *function symbols* on *terms*
 $\text{enc}(\text{dh}(\text{g}, x), \text{k}_{\text{Bob}}) \in \mathcal{T}$
- A trace is a list of actions (*attacker's interaction* with agents in $\mathcal{A}$)
 - **in**(a, R) // for $a \in \mathcal{A}$ (agent) and $R \in \mathcal{T}$ (*attacker term*)
 - **out**(a, x) // for $a \in \mathcal{A}$ (agent) and $x \in \mathcal{X}$ (*attacker knowledge*)



$\mathcal{A} = \{\text{client}, \text{server}\}$

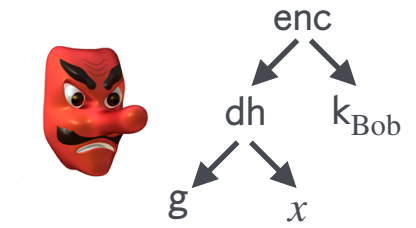
client

Attacker 

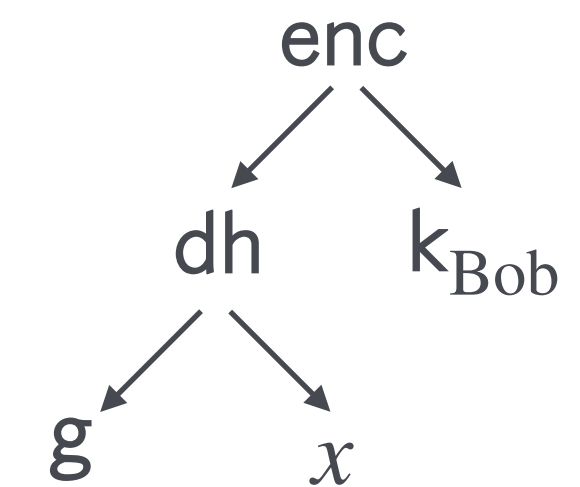
server



DY Message and Attacker Model

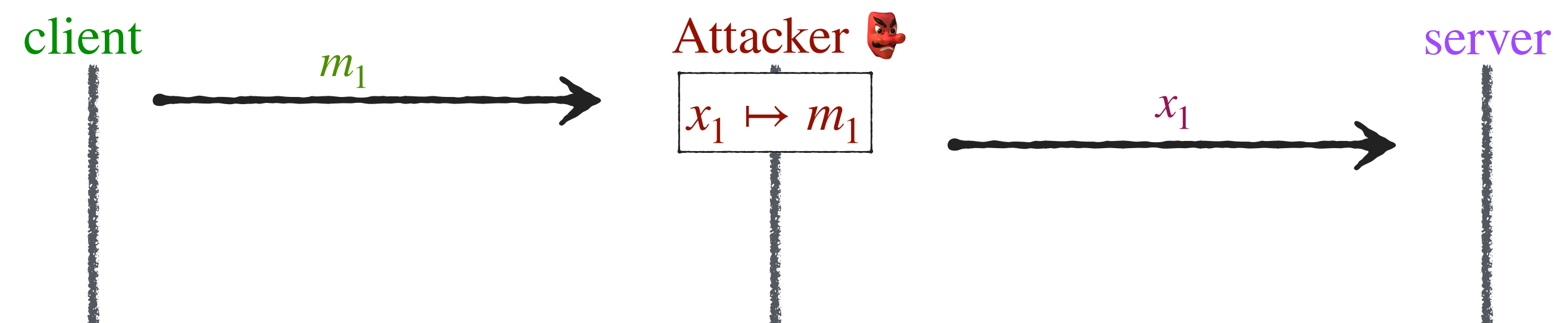


- A signature is a set $\mathcal{F}$ of *function symbols* with arity $\mathcal{F} = \{\text{enc}[2], \text{dec}[2], \text{hash}[1], \text{dh}[2], \text{g}[0], \text{k}_{\text{Bob}}[0], \dots\}$
- The set of terms $\mathcal{T}$ is inductively defined as *variables* from $\mathcal{X}$ and by application of *function symbols* on *terms*
 $\text{enc}(\text{dh}(\text{g}, x), \text{k}_{\text{Bob}}) \in \mathcal{T}$
- A trace is a list of actions (*attacker's interaction* with agents in $\mathcal{A}$)
 - **in**(a, R) // for $a \in \mathcal{A}$ (agent) and $R \in \mathcal{T}$ (*attacker term*)
 - **out**(a, x) // for $a \in \mathcal{A}$ (agent) and $x \in \mathcal{X}$ (*attacker knowledge*)

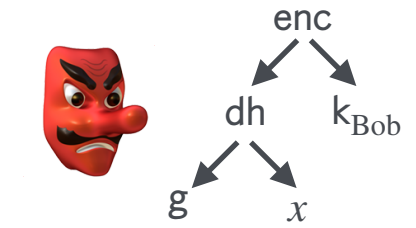


$\mathcal{A} = \{\text{client}, \text{server}\}$

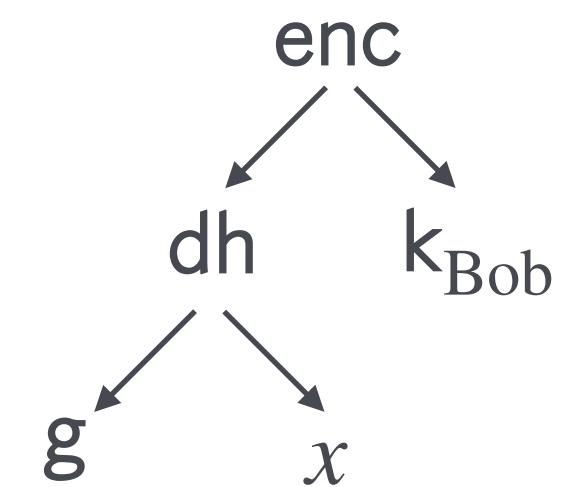
$\text{tr} = \text{out}(\text{client}, x_1).$
 $\text{in}(\text{server}, x_1).$



DY Message and Attacker Model

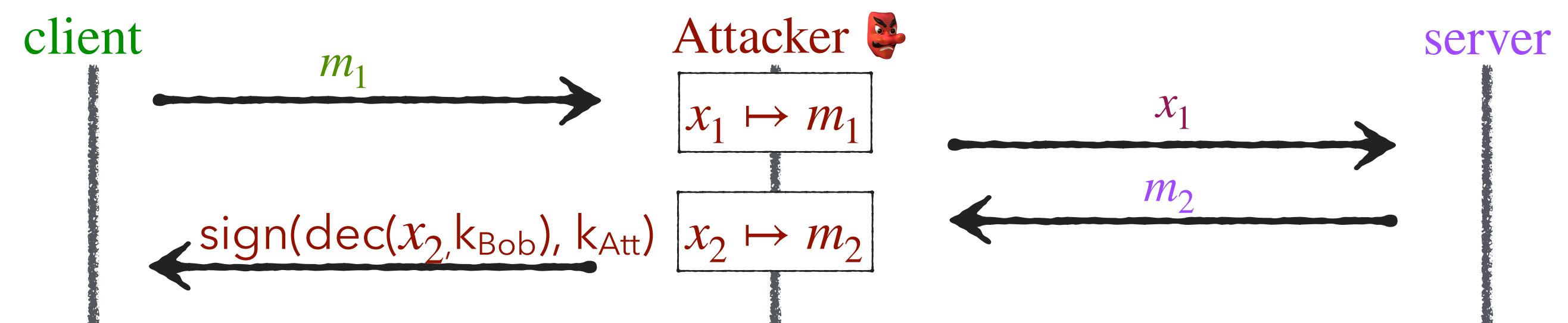


- A signature is a set $\mathcal{F}$ of *function symbols* with arity $\mathcal{F} = \{\text{enc}[2], \text{dec}[2], \text{hash}[1], \text{dh}[2], \text{g}[0], \text{k}_{\text{Bob}}[0], \dots\}$
- The set of terms $\mathcal{T}$ is inductively defined as *variables* from $\mathcal{X}$ and by application of *function symbols* on *terms*
 $\text{enc}(\text{dh}(\text{g}, x), \text{k}_{\text{Bob}}) \in \mathcal{T}$
- A trace is a list of actions (attacker's interaction with agents in $\mathcal{A}$)
 - **in**(a, R) // for $a \in \mathcal{A}$ (agent) and $R \in \mathcal{T}$ (attacker term)
 - **out**(a, x) // for $a \in \mathcal{A}$ (agent) and $x \in \mathcal{X}$ (attacker knowledge)

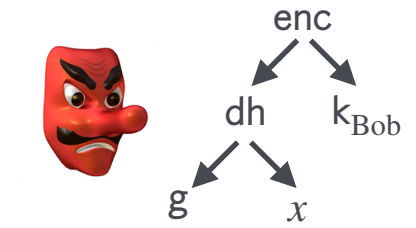


$\mathcal{A} = \{\text{client}, \text{server}\}$

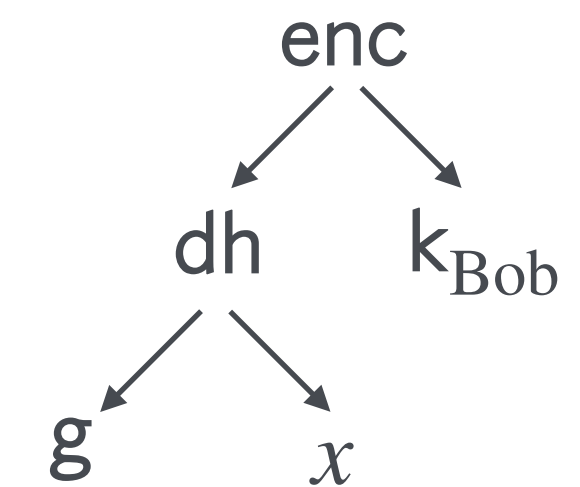
tr = **out**(client, x_1).
 in(server, x_1).
 out(server, x_2).
 in(client, $\text{sign}(\text{dec}(x_2, \text{k}_{\text{Bob}}), \text{k}_{\text{Att}})$)



DY Message and Attacker Model

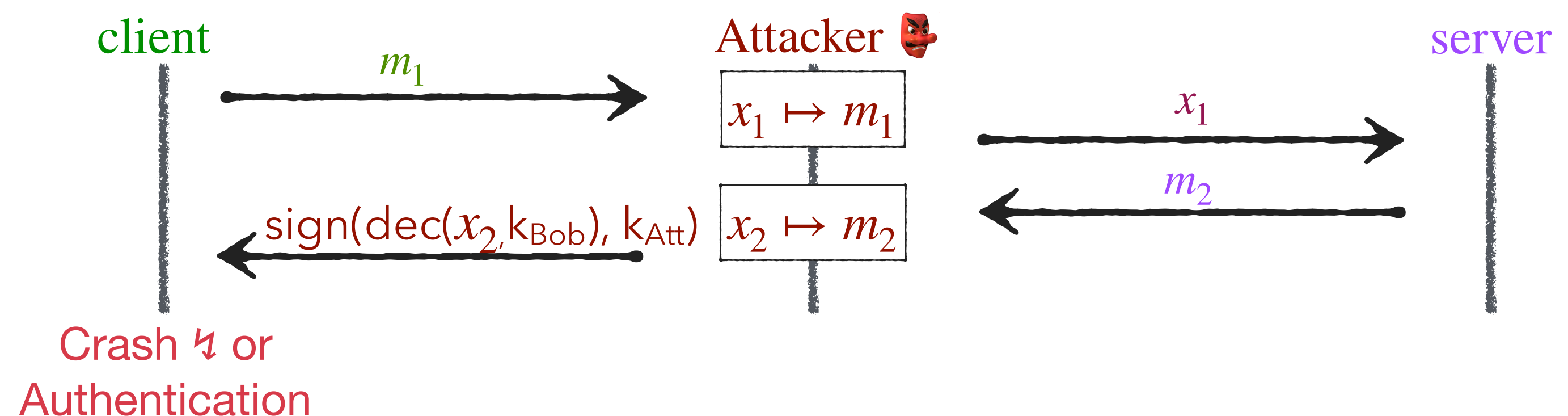


- A signature is a set $\mathcal{F}$ of *function symbols* with arity $\mathcal{F} = \{\text{enc}[2], \text{dec}[2], \text{hash}[1], \text{dh}[2], g[0], k_{\text{Bob}}[0], \dots\}$
- The set of terms $\mathcal{T}$ is inductively defined as *variables* from $\mathcal{X}$ and by application of *function symbols* on *terms*
 $\text{enc}(\text{dh}(g, x), k_{\text{Bob}}) \in \mathcal{T}$
- A trace is a list of actions (attacker's interaction with agents in $\mathcal{A}$)
 - **in**(a, R) // for $a \in \mathcal{A}$ (agent) and $R \in \mathcal{T}$ (attacker term)
 - **out**(a, x) // for $a \in \mathcal{A}$ (agent) and $x \in \mathcal{X}$ (attacker knowledge)

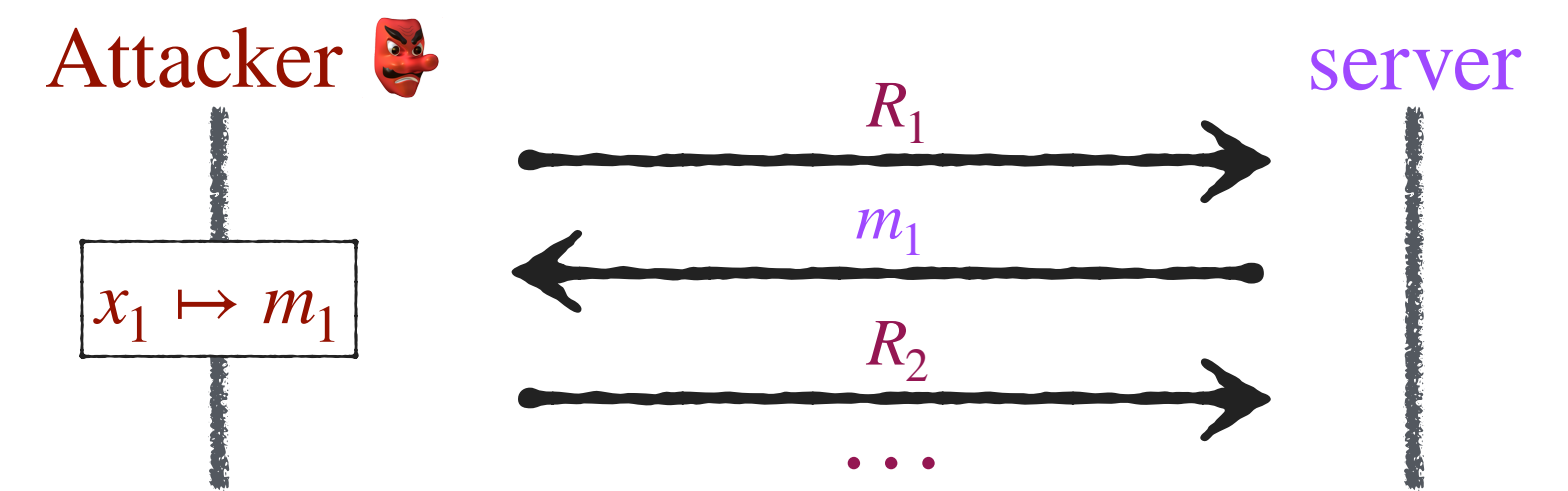


$\mathcal{A} = \{\text{client}, \text{server}\}$

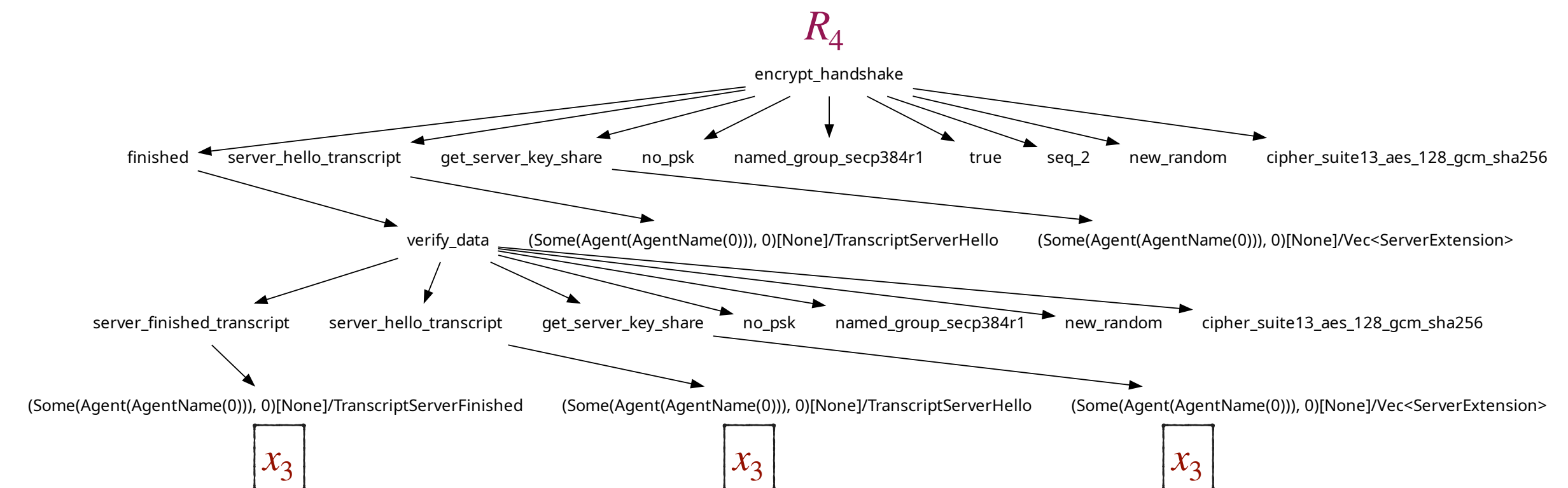
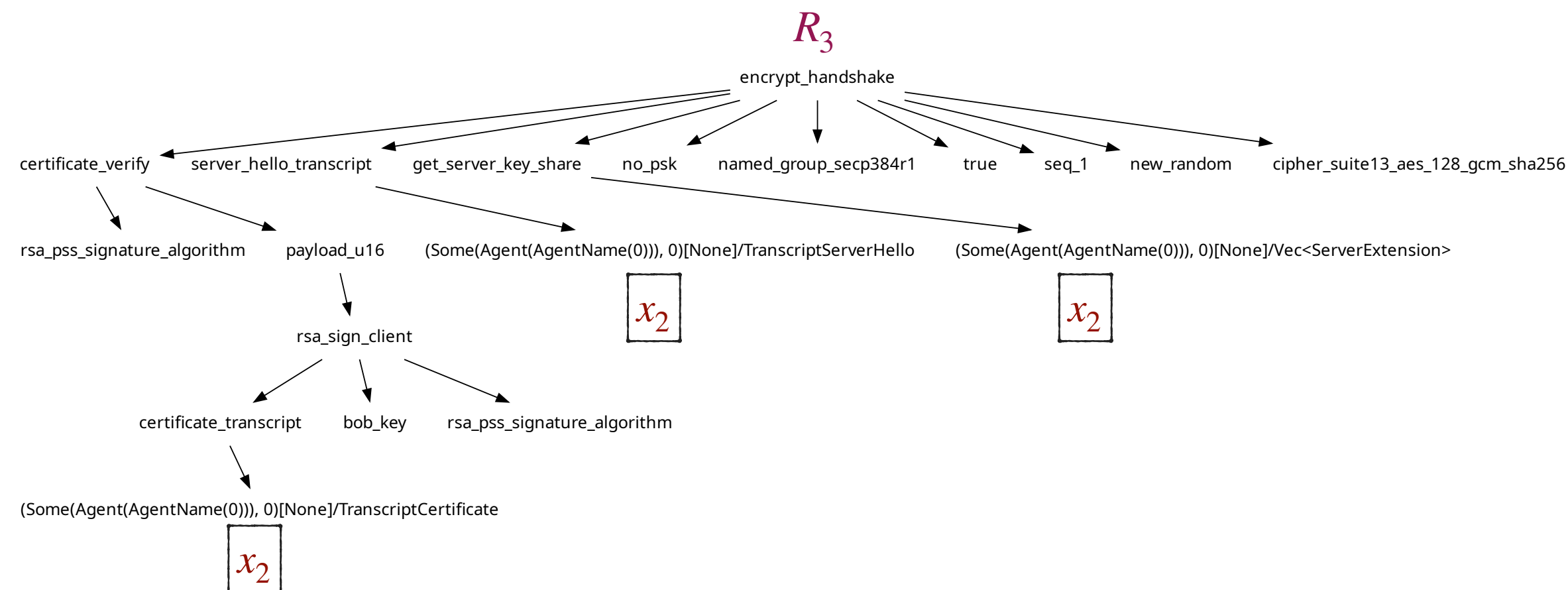
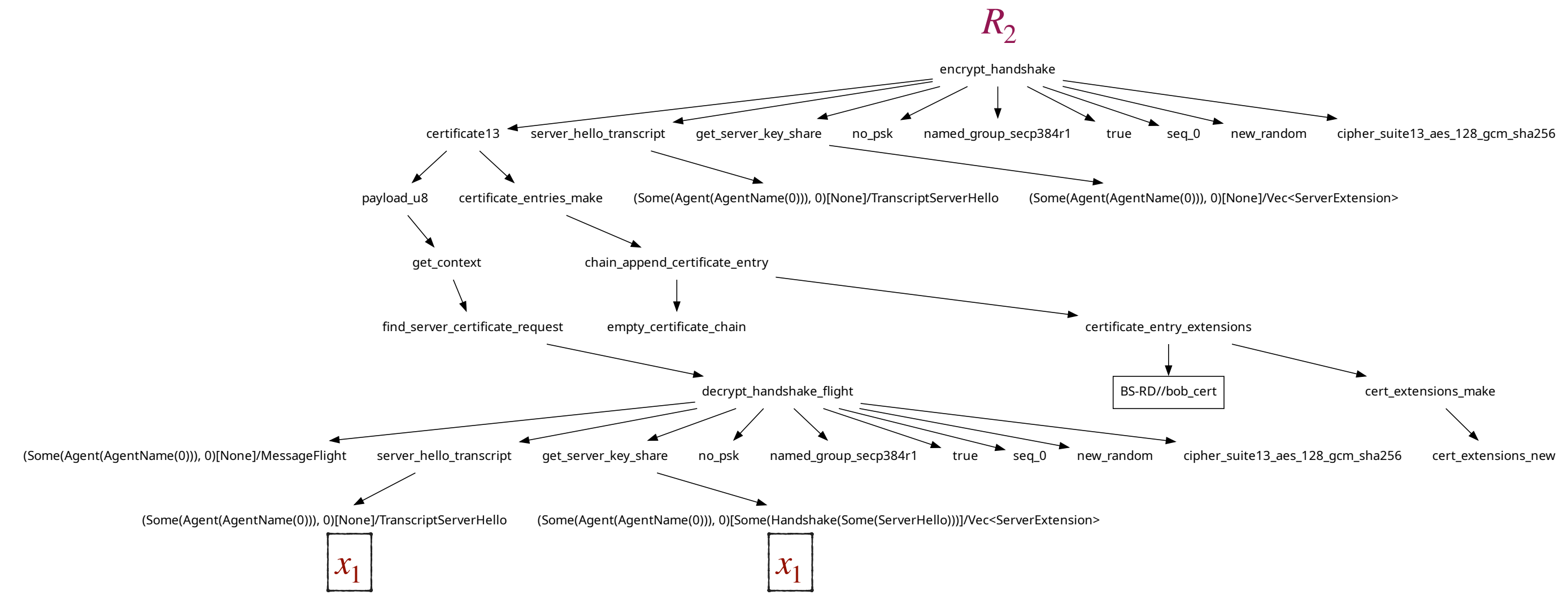
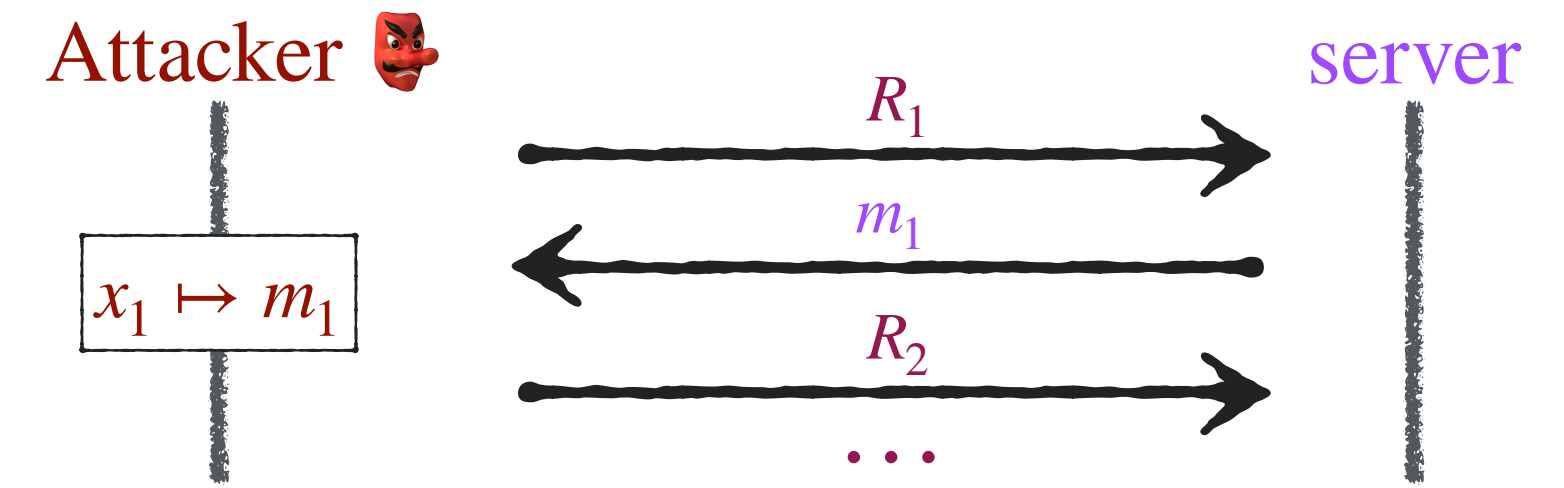
tr = **out**(client, x_1).
 in(server, x_1).
 out(server, x_2).
 in(client, $\text{sign}(\text{dec}(x_2, k_{\text{Bob}}), k_{\text{Att}})$)



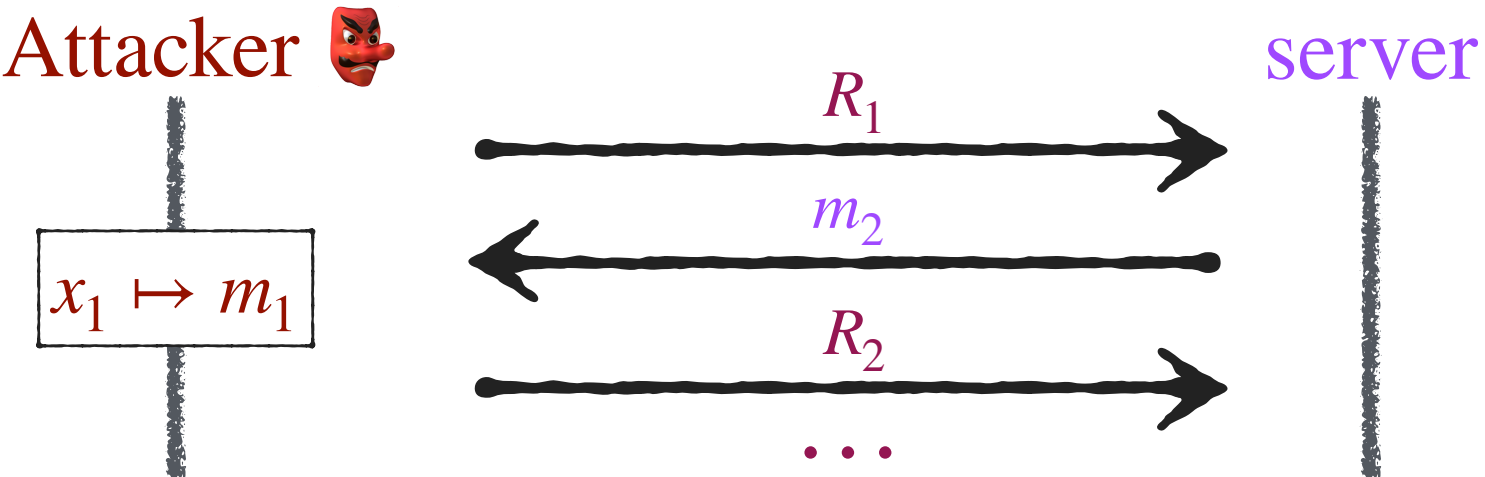
TLS Trace Example



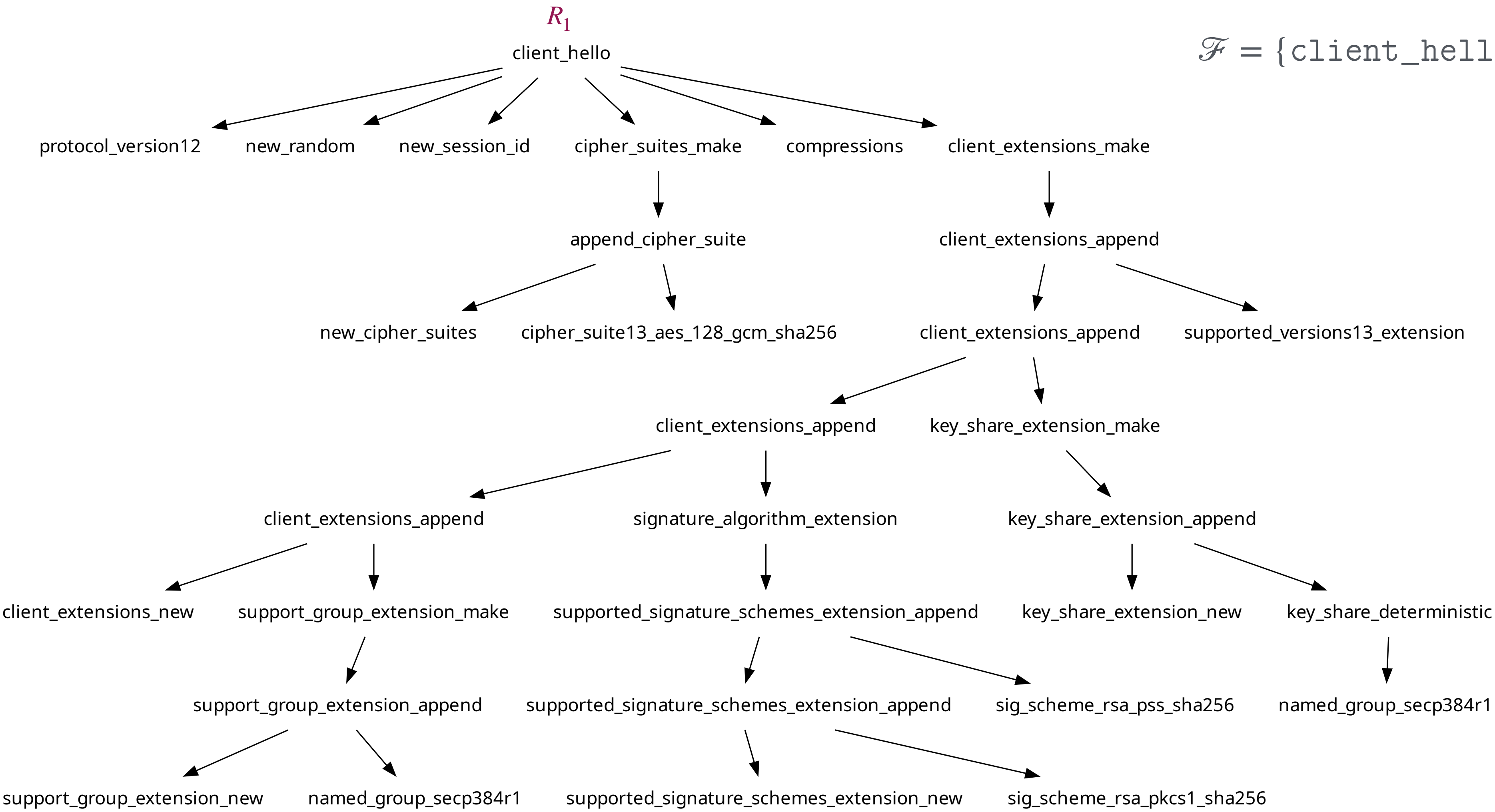
TLS Trace Example



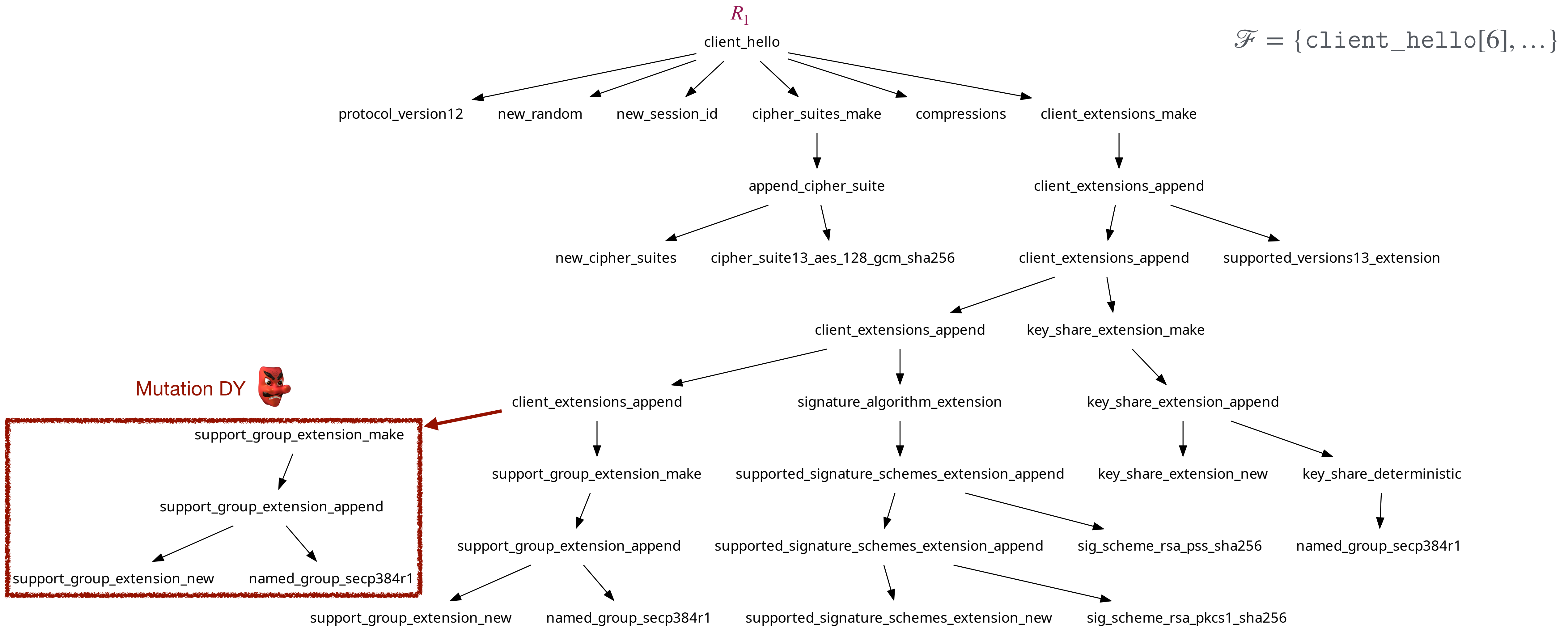
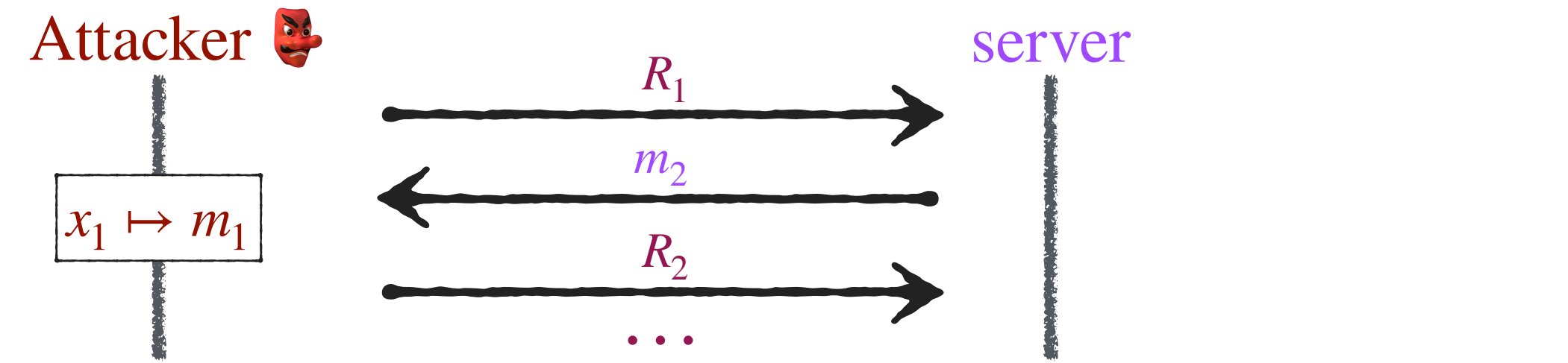
TLS Trace Example



$\mathcal{F} = \{\text{client_hello}[6], \dots\}$

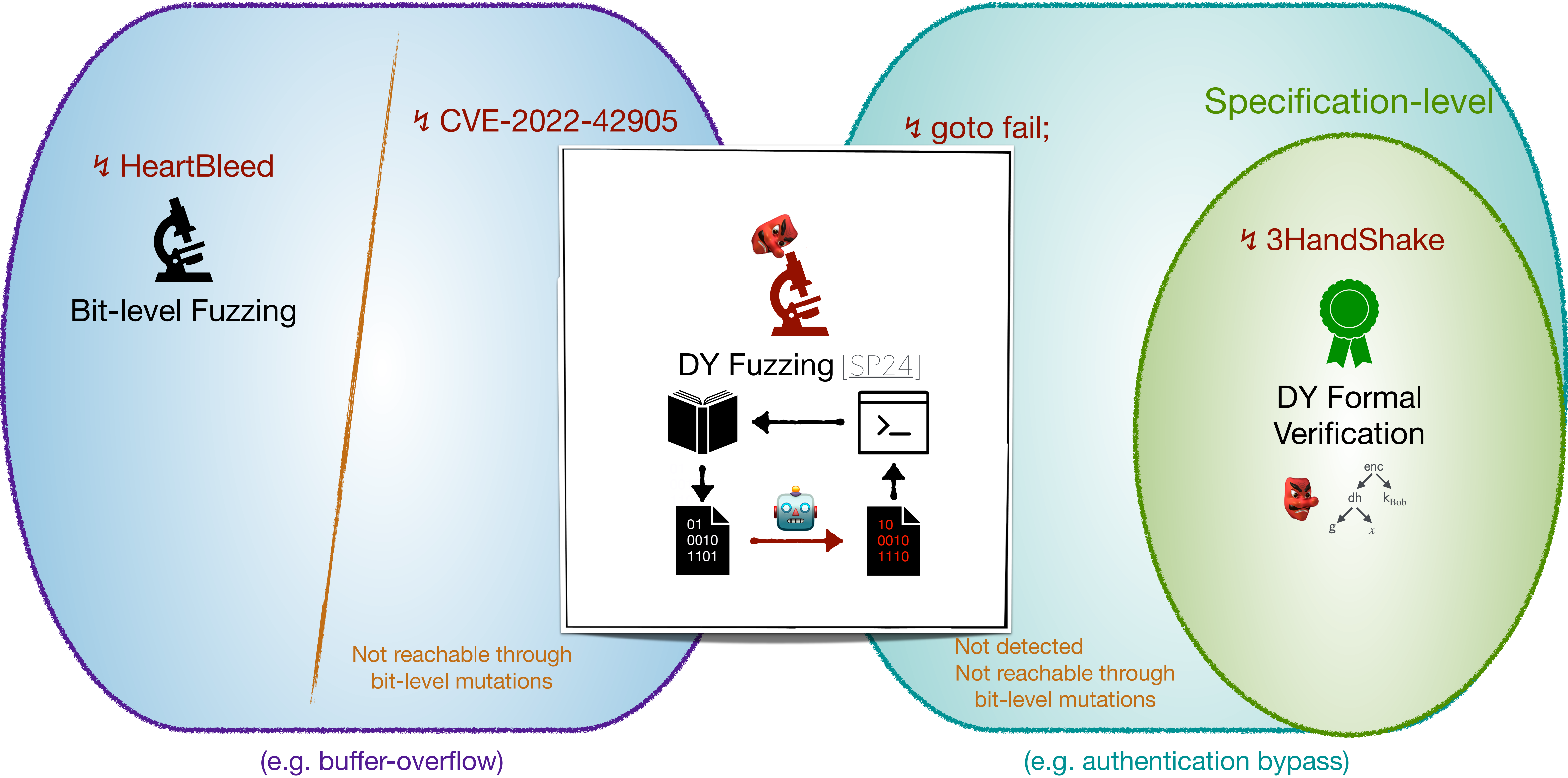


TLS Trace Example



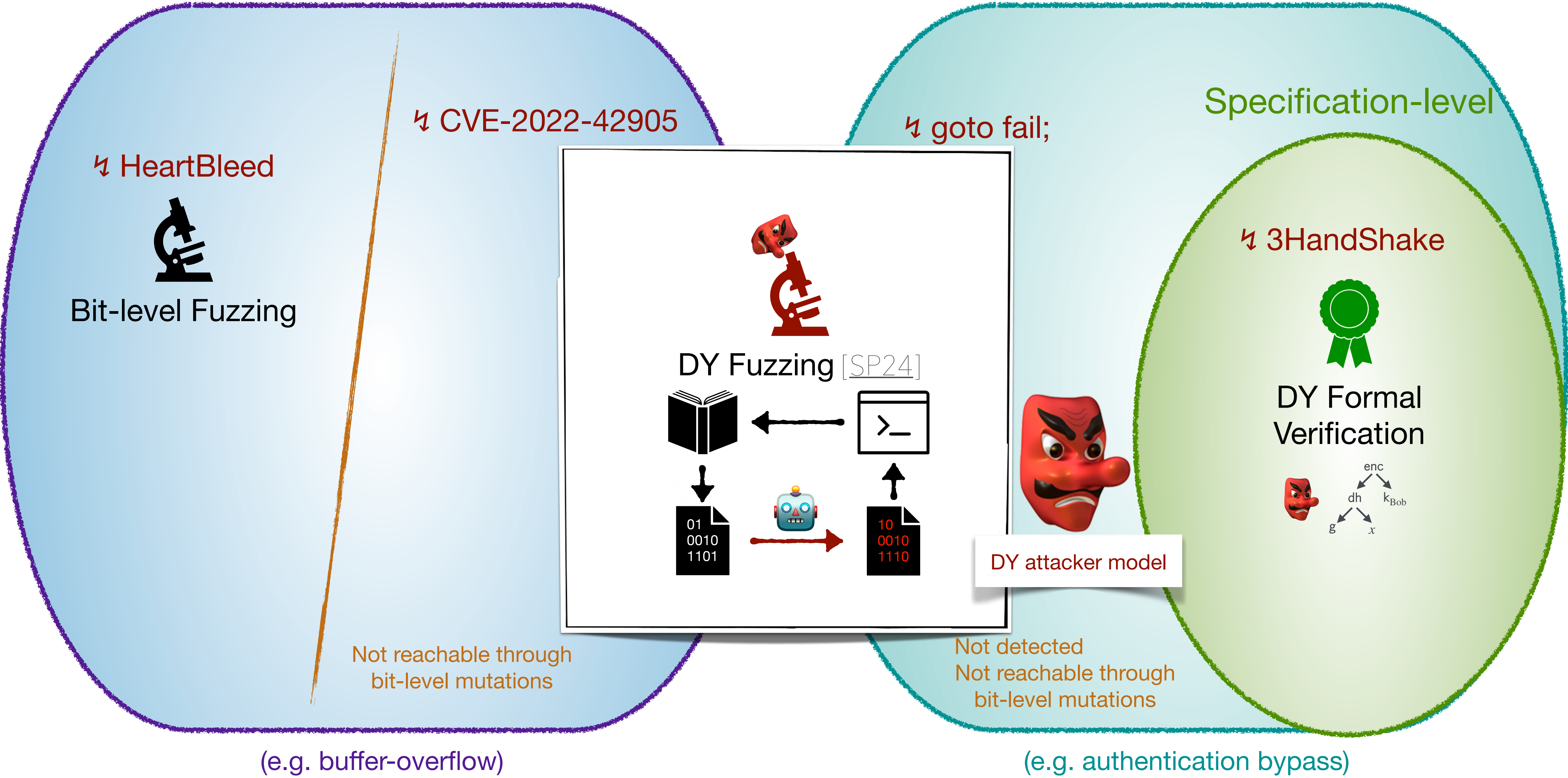
Memory Safety Vulnerabilities

Protocol Vulnerabilities



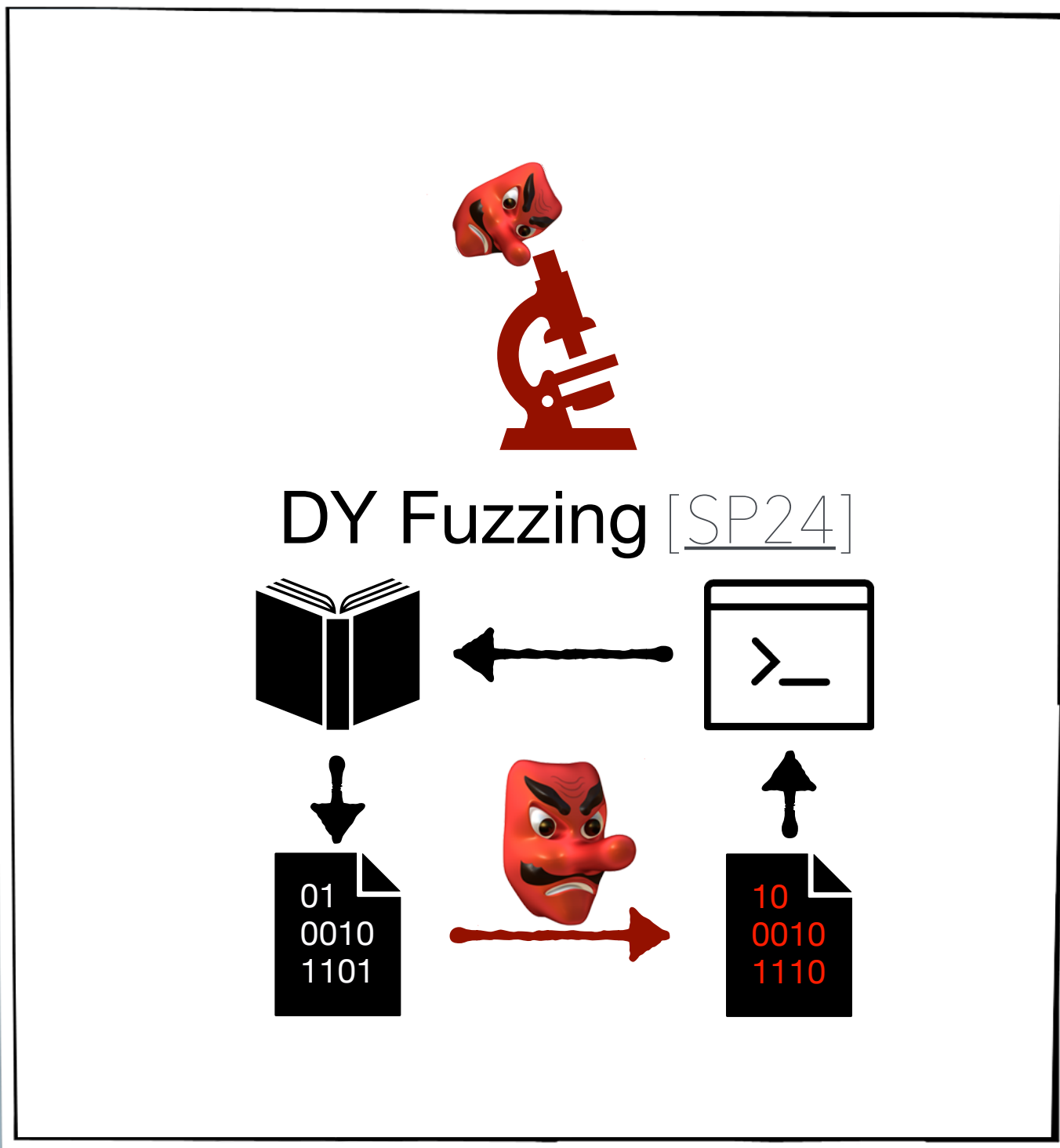
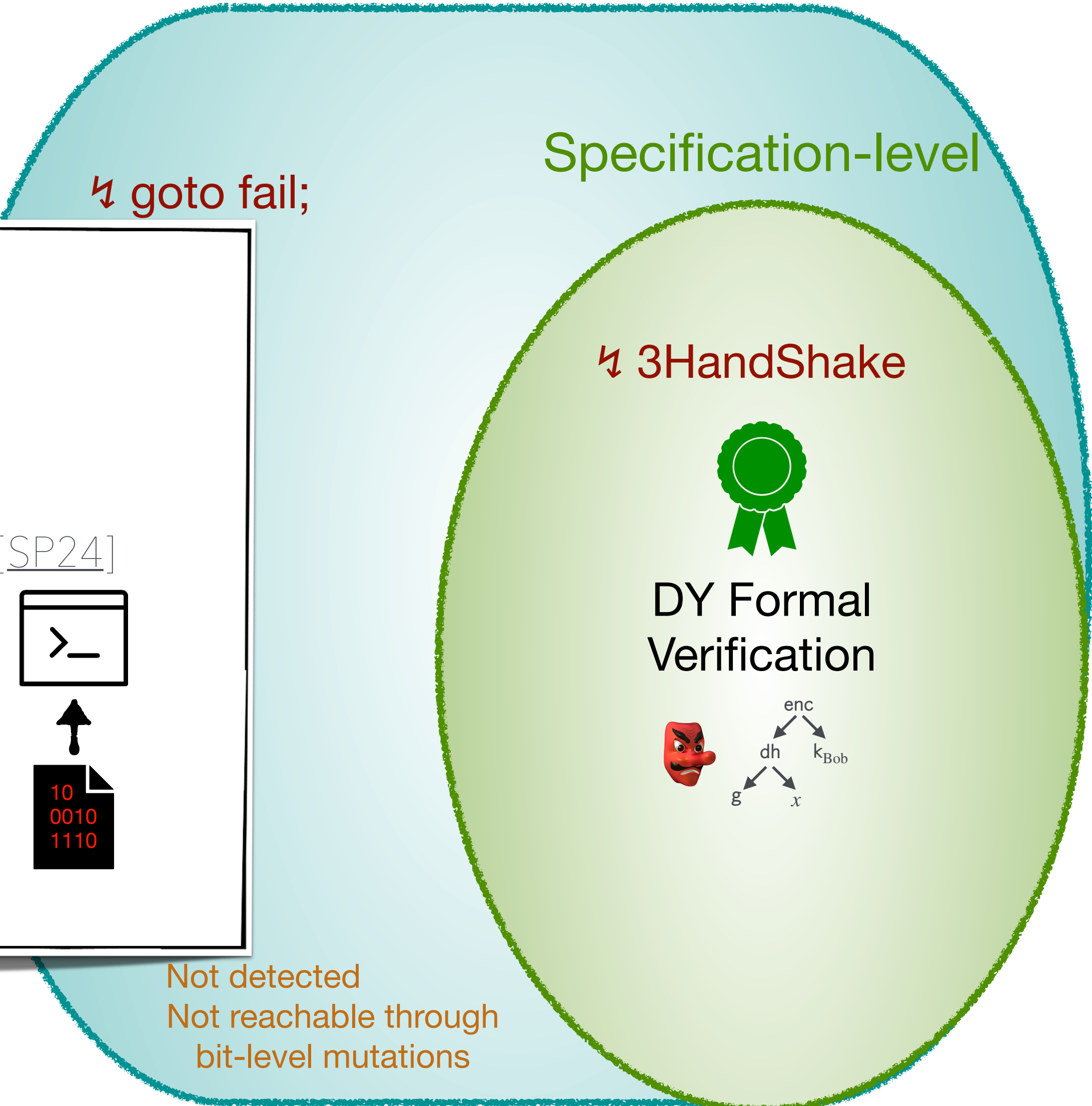
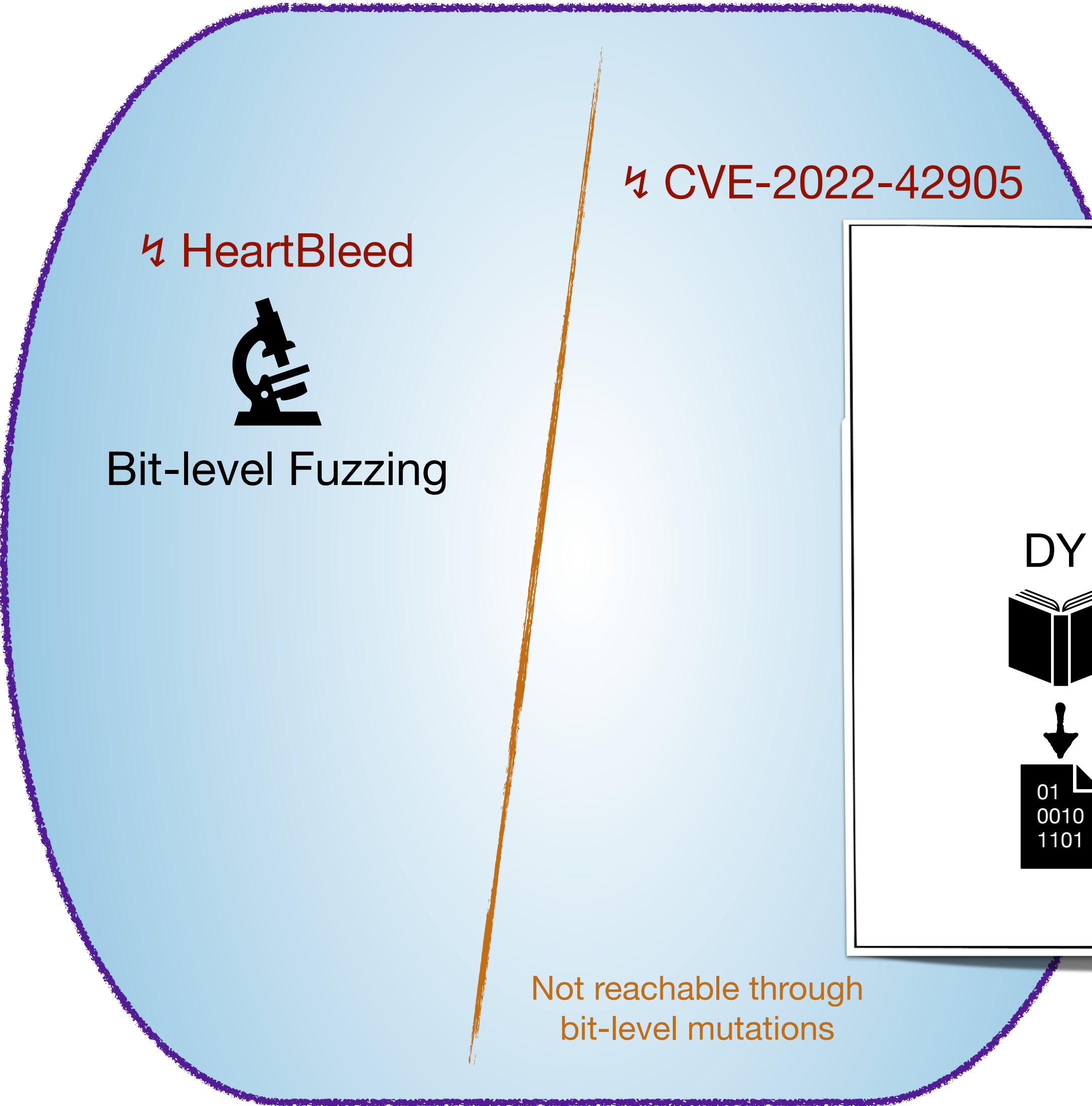
Memory Safety Vulnerabilities

Protocol Vulnerabilities



Memory Safety Vulnerabilities

Protocol Vulnerabilities



(e.g. buffer-overflow)

(e.g. authentication bypass)

Memory Safety Vulnerabilities

Protocol Vulnerabilities

⚡ HeartBleed



Bit-level Fuzzing

⚡ CVE-2022-42905

Not reachable through bit-level mutations

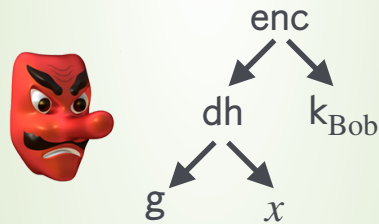
⚡ goto fail;

Specification-level

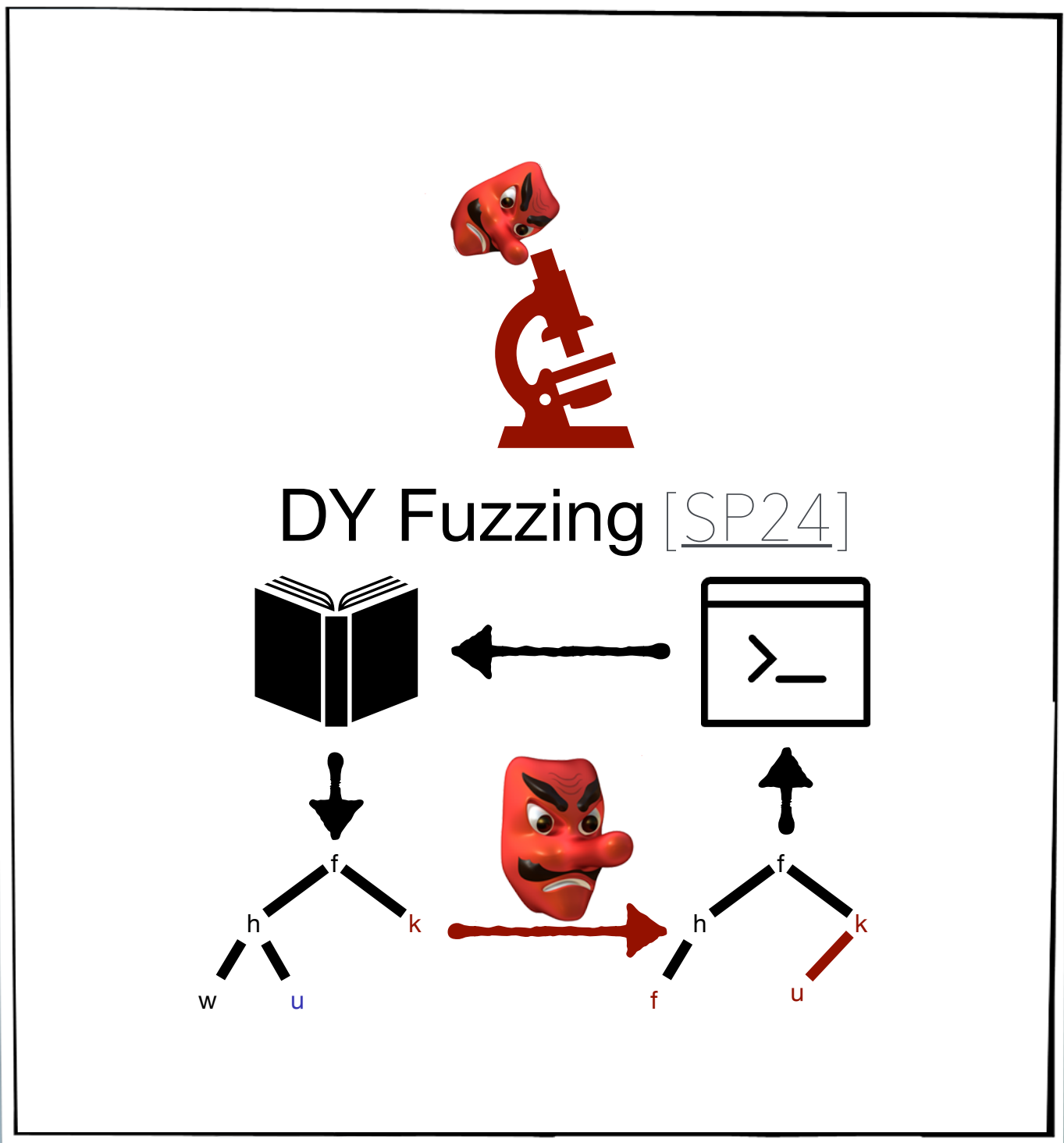
⚡ 3HandShake



DY Formal Verification



Not detected
Not reachable through bit-level mutations



(e.g. buffer-overflow)

(e.g. authentication bypass)

Memory Safety Vulnerabilities

Protocol Vulnerabilities

⚡ HeartBleed



Bit-level Fuzzing

⚡ CVE-2022-42905

Not reachable through bit-level mutations

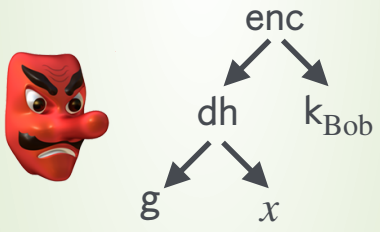
⚡ goto fail;

Specification-level

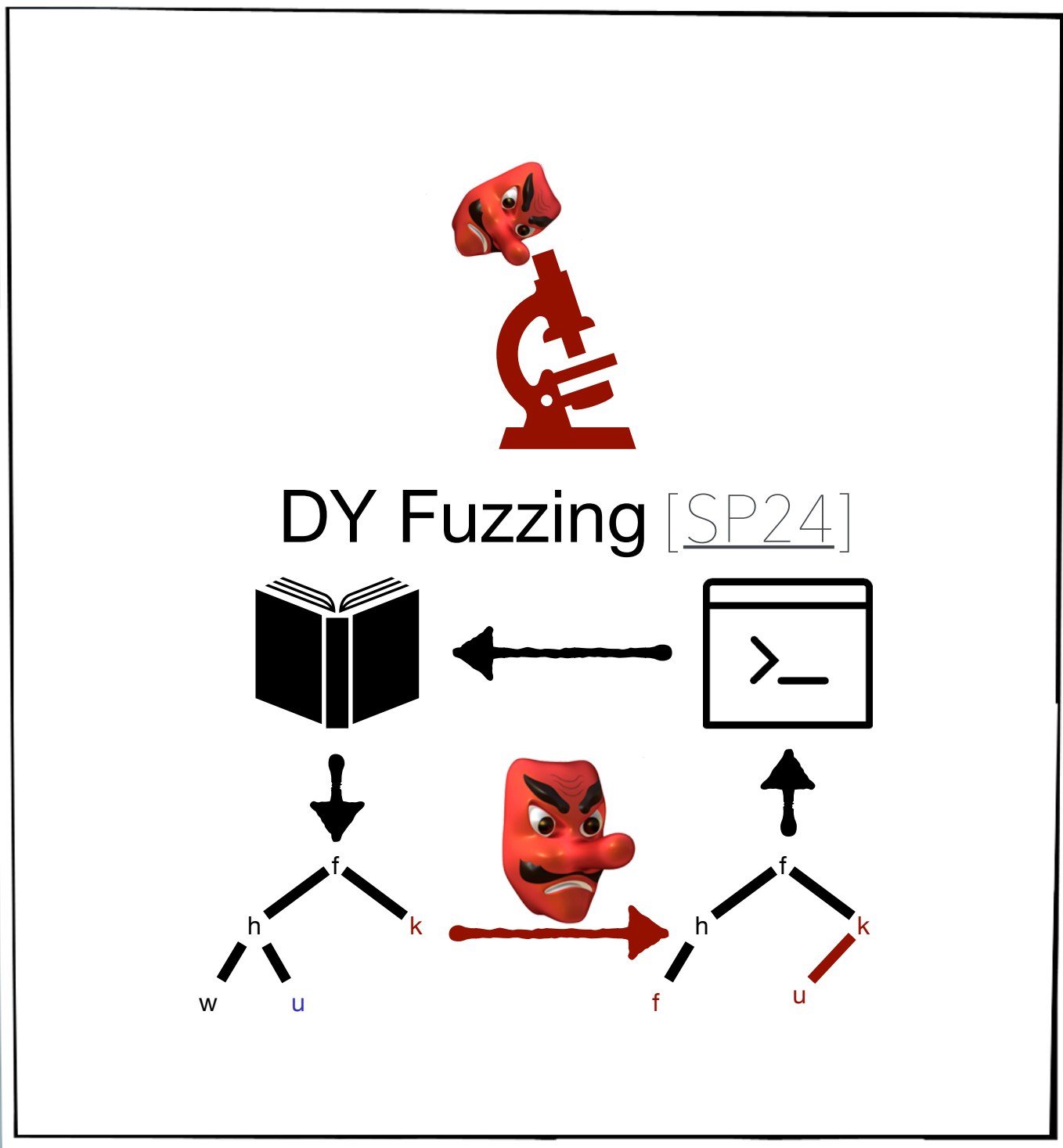
⚡ 3HandShake



DY Formal Verification



Not detected
Not reachable through bit-level mutations



(e.g. buffer-overflow)

(e.g. authentication bypass)

Memory Safety Vulnerabilities

Protocol Vulnerabilities

⚡ CVE-2022-42905

⚡ HeartBleed



Bit-level Fuzzing

Not reachable through bit-level mutations

⚡ goto fail;

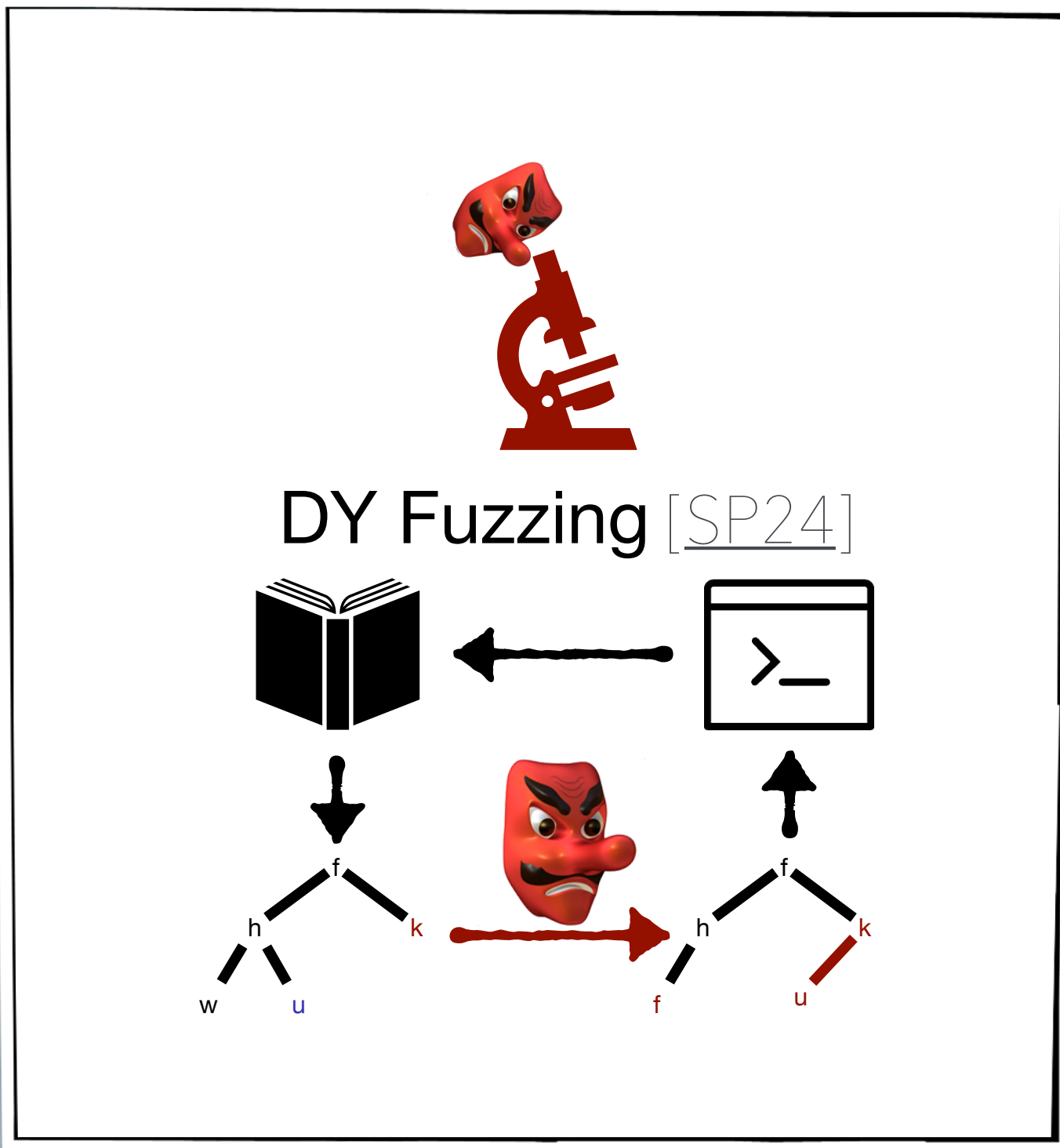
Specification-level

⚡ 3HandShake



DY Formal Verification

Not detected
Not reachable through bit-level mutations



(e.g. buffer

DY Fuzzer = DY attacker  in a fuzzing loop

on bypass)

Mapper and Harness

Mapper and Harness

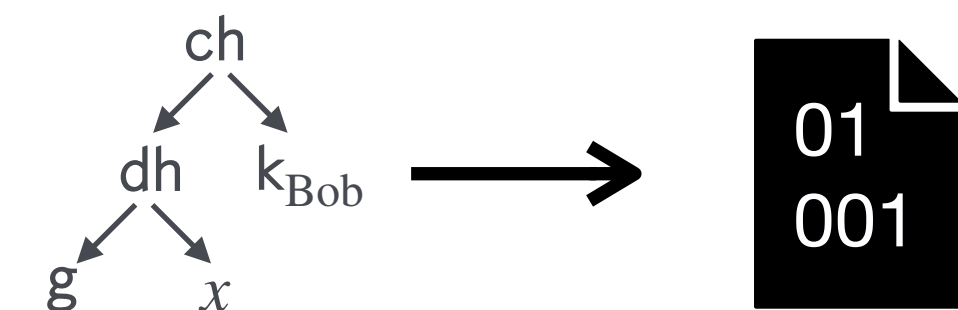
Mapper: terms are just an abstraction, we need *DY terms* $\leftrightarrow$ *bitstrings*

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

$\llbracket f \rrbracket : [\text{u8}]^i \rightarrow [\text{u8}]$ $// \llbracket \text{enc} \rrbracket := \text{openssl.crypto.aes}$

$\llbracket \cdot \rrbracket_\phi : \mathcal{T} \rightarrow [\text{u8}]$ $// \llbracket \text{enc}(t_1, t_2) \rrbracket_\phi = \text{openssl.crypto.aes}(\llbracket t_1 \rrbracket_\phi, \llbracket t_2 \rrbracket_\phi)$

$// \llbracket x \rrbracket_\phi = x\phi$



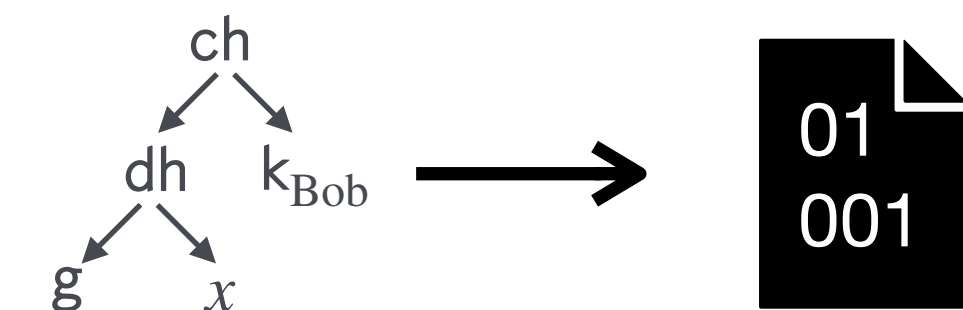
Mapper and Harness

Mapper: terms are just an abstraction, we need *DY terms* $\leftrightarrow$ *bitstrings*

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

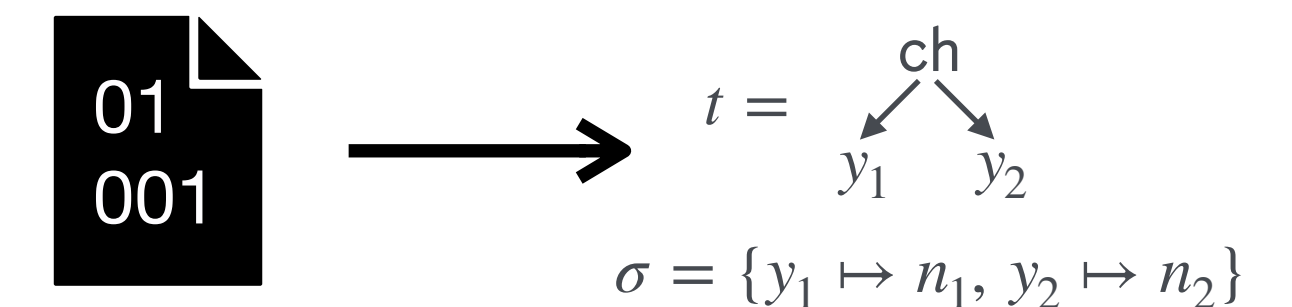
$\llbracket f \rrbracket : [\text{u8}]^i \rightarrow [\text{u8}]$ // $\llbracket \text{enc} \rrbracket := \text{openssl.crypto.aes}$

$\llbracket \cdot \rrbracket_\phi : \mathcal{T} \rightarrow [\text{u8}]$ // $\llbracket \text{enc}(t_1, t_2) \rrbracket_\phi = \text{openssl.crypto.aes}(\llbracket t_1 \rrbracket_\phi, \llbracket t_2 \rrbracket_\phi)$
 // $\llbracket x \rrbracket_\phi = x\phi$



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

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



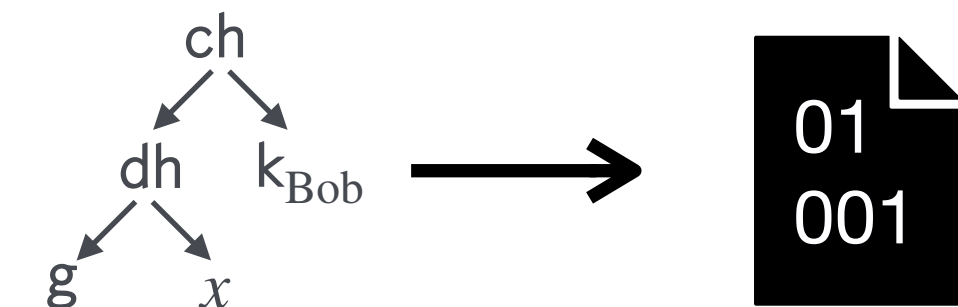
Mapper and Harness

Mapper: terms are just an abstraction, we need *DY terms* $\leftrightarrow$ *bitstrings*

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

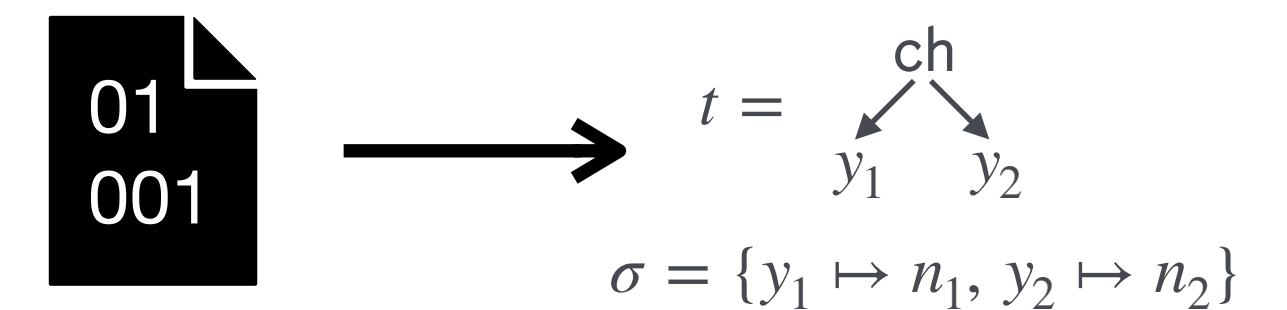
$\llbracket f \rrbracket : [\text{u8}]^i \rightarrow [\text{u8}]$ // $\llbracket \text{enc} \rrbracket := \text{openssl.crypto.aes}$

$\llbracket \cdot \rrbracket_\phi : \mathcal{T} \rightarrow [\text{u8}]$ // $\llbracket \text{enc}(t_1, t_2) \rrbracket_\phi = \text{openssl.crypto.aes}(\llbracket t_1 \rrbracket_\phi, \llbracket t_2 \rrbracket_\phi)$
 // $\llbracket x \rrbracket_\phi = x\phi$



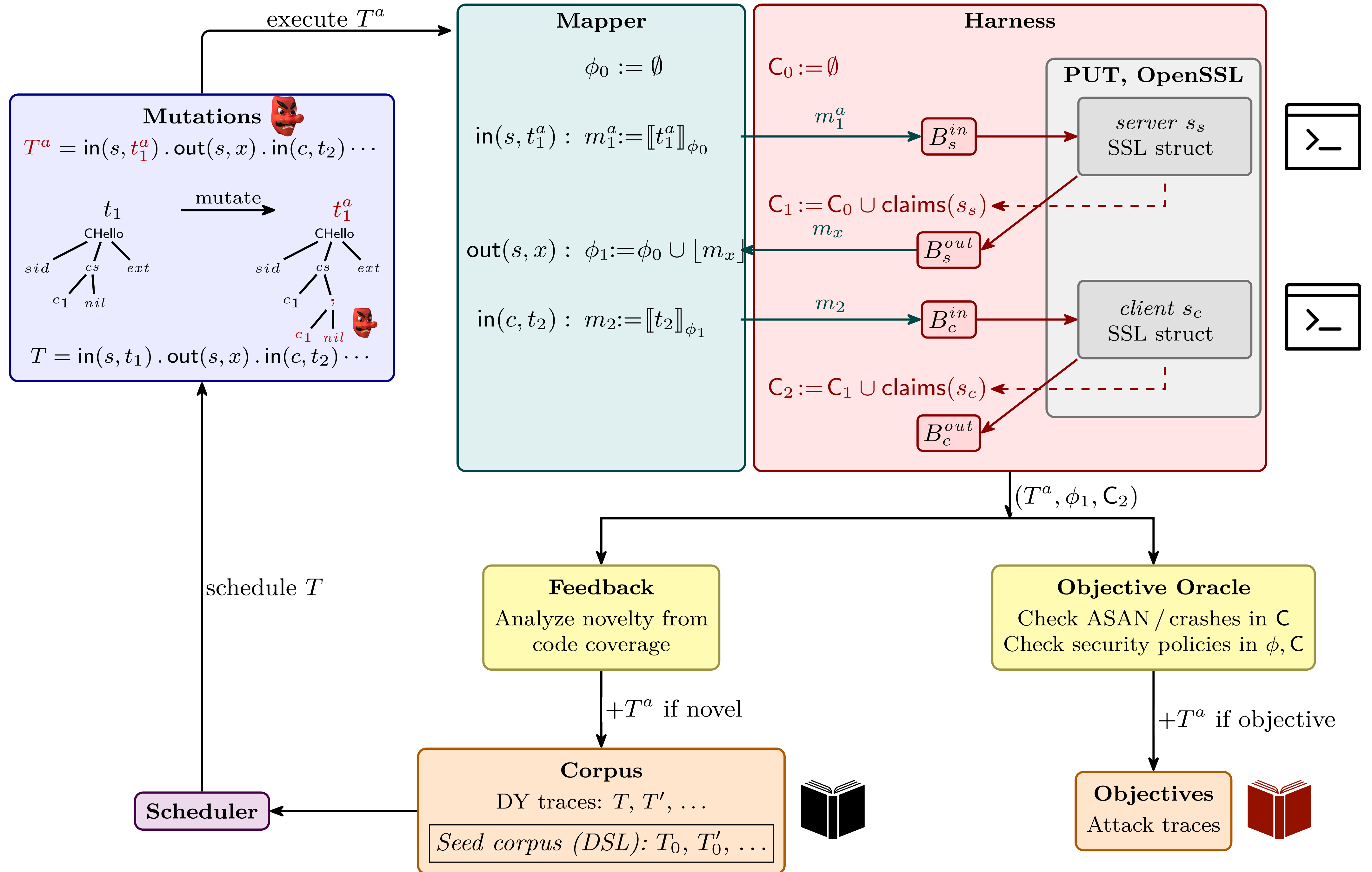
- **Partial parsing**: may not fully parse bitstrings (best effort)

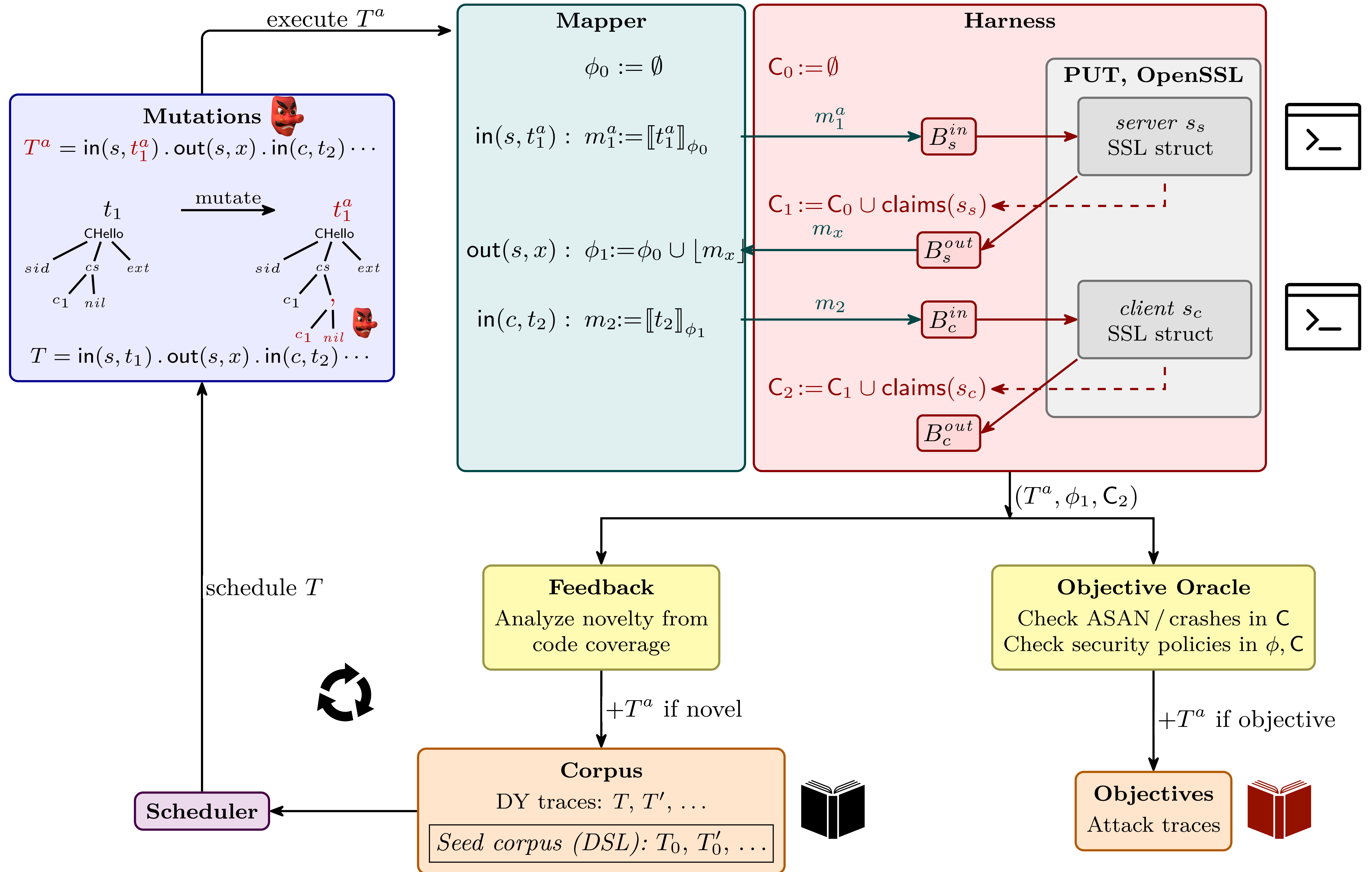
$\lceil \cdot \rceil : [\text{u8}] \rightarrow \mathcal{T} \times (\mathcal{X} \mapsto [\text{u8}])$



Harness: interact with the Program Under Test & gather Claims (log of PUT internal states)

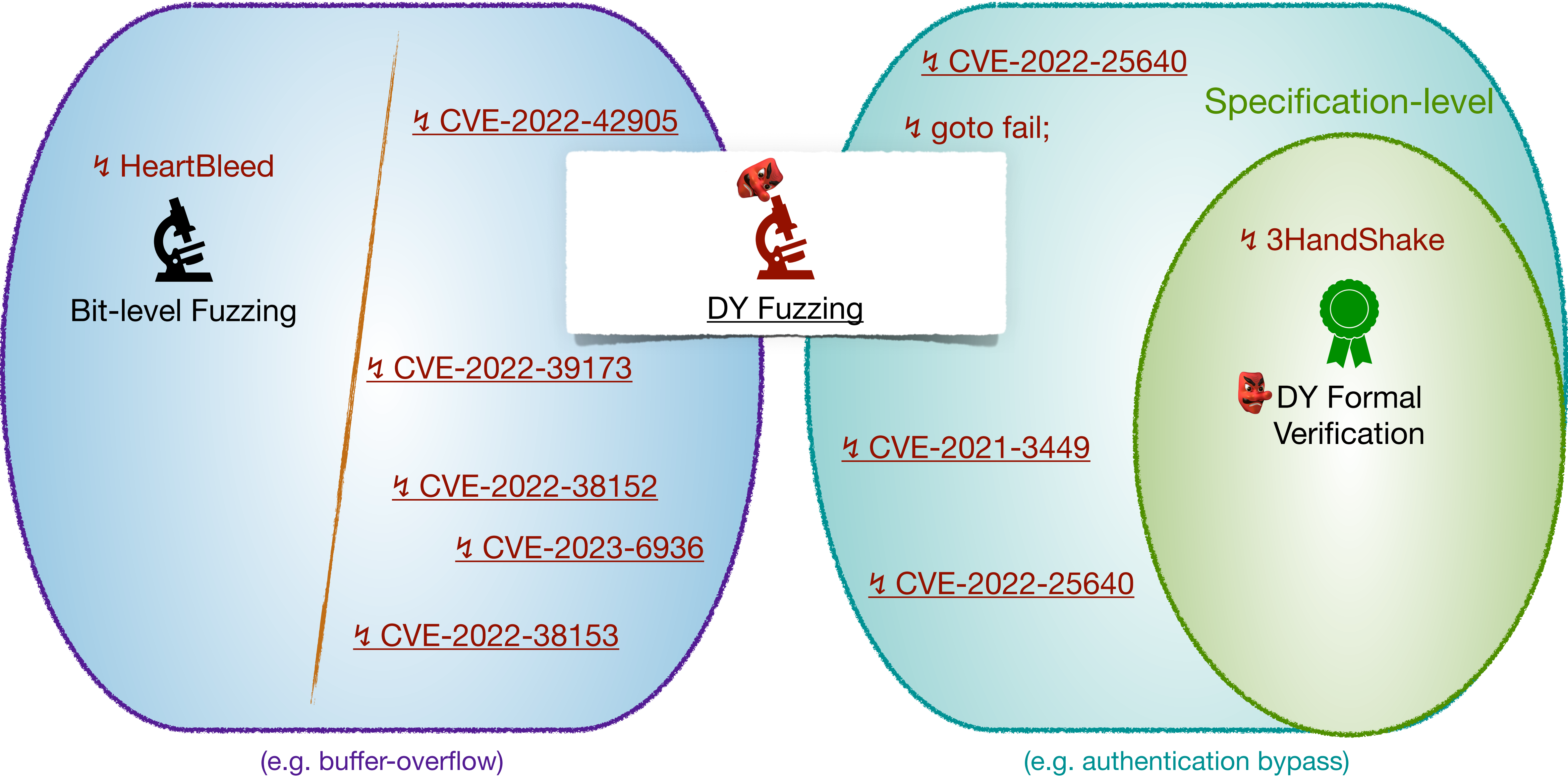
- **out**(*a*, *x*) : receives $m_x \in [\text{u8}]$ from *a*,
 computes $(t, \sigma) = \lceil m_x \rceil$,
 add all nodes of $t\sigma$ to ϕ , i.e., **attacker's Knowledges** (not only roots $\rightarrow$ ease access)
- **in**(*a*, *R*) : computes $m_R = \llbracket R \rrbracket_\phi \in [\text{u8}]$, sends it to *a*



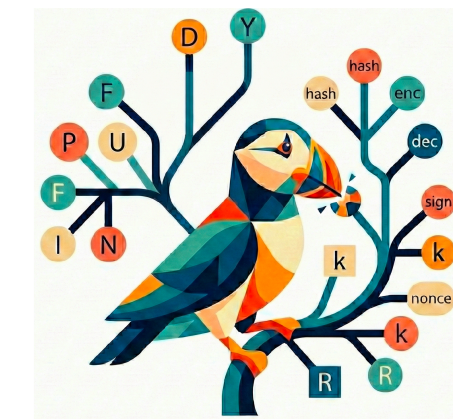


Memory Safety Vulnerabilities

Protocol Vulnerabilities



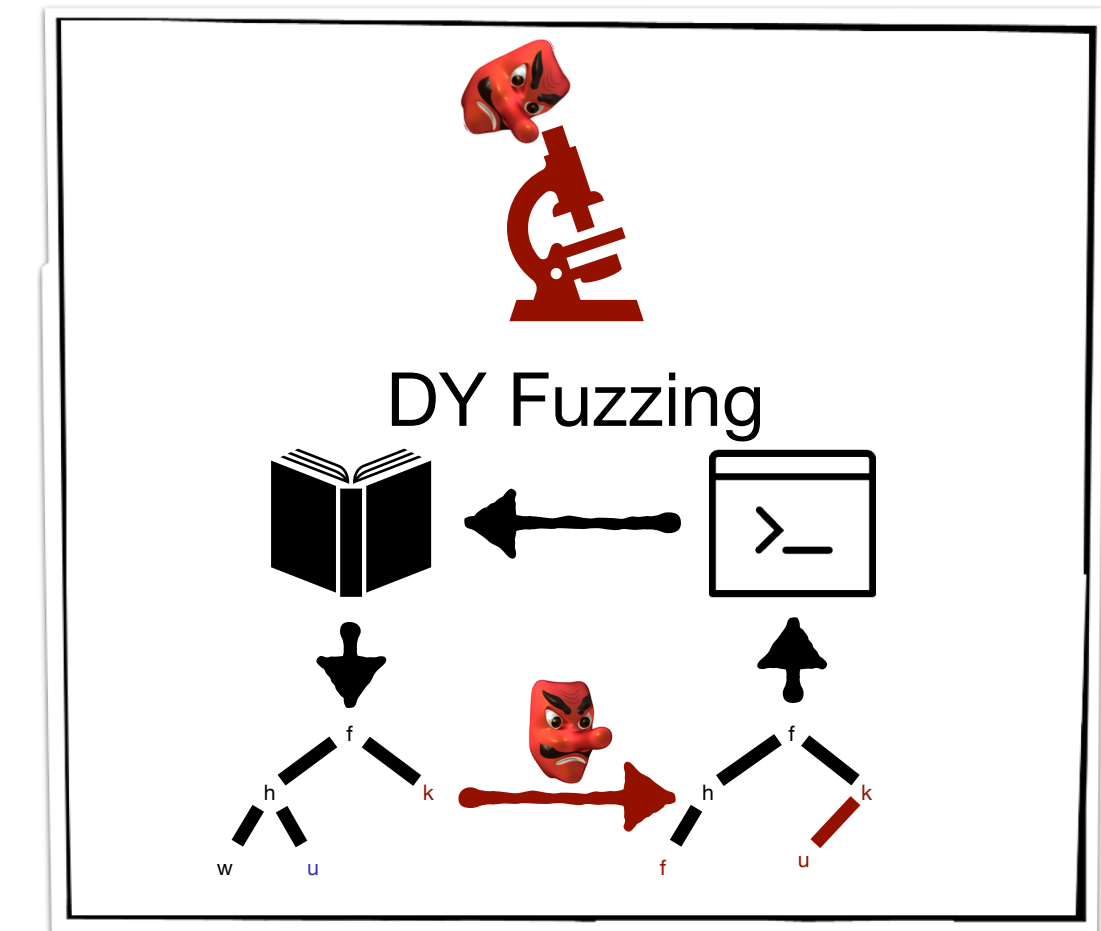
DY Fuzzer = DY attacker in a fuzzing loop



Checkout the puffin website:
<https://tlspuffin.github.io>

We implemented **puffin**: DY fuzzer library + instantiation for protocols: e.g., **tlspuffin**

- ✓ **Open-source**, 28k Rust LoC, built on top of **LibAFL**
- ✓ **Modular**: new protocols (beyond TLS) and PUTs can be added to **puffin**
- ✓ **Current Protocol Implementations**:
 - TLS: Open/Wolf/Libre/BoringSSL
 - SSH: OpenSSH, WolfSSH (SSH) // to be extended
 - OPC-UA: open62541 // to be extended
- ✓ Bench + Evaluation + 11 CVEs (**5 new CVEs**, acknowledged and patched)

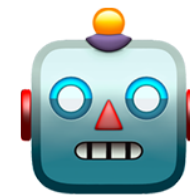


CVE ID	CVSS	Type	New?	Target
2021-3449	5.9	Server DoS, M	✗	OpenSSL
2022-25638	6.5	Auth. Bypass, P	✗	WolfSSL
2022-25640	7.5	Auth. Bypass, P	✗	WolfSSL
2022-38152	7.5	Client DoS, M	✓	WolfSSL
2022-38153	5.9	Server DoS, M	✓	WolfSSL
2022-39173	7.5	Server DoS, M	✓	WolfSSL
2022-42905	9.1	Info. Leak, M	✓	WolfSSL
2023-6936	5.3	Info. Leak, M	✓	WolfSSL
2023-3724	9.1	Weak key	✗	WolfSSL
2023-6937	5.3	Confidenti. Viola.	✗	WolfSSL
2024-5814	5.3	Downgrade attack	✗	WolfSSL

Combining HAVOC with DY Fuzzing



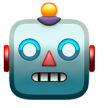
+



WIP...

Problem: Reconciling Structured and Bit-Level Fuzzing

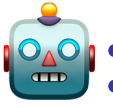
Problem: Reconciling Structured and Bit-Level Fuzzing

Bit-level (HAVOC) : bit-flips / byte-increment / ... on raw bytes

✓ **Escape the message grammar** (and well-types messages)

➡ inconsistent lengths, header/payload, malformed nesting, invalid bytes under crypto, ...

Problem: Reconciling Structured and Bit-Level Fuzzing

Bit-level (HAVOC) : bit-flips / byte-increment / ... on raw bytes

✓ **Escape the message grammar** (and well-types messages)

➔ inconsistent lengths, header/payload, malformed nesting, invalid bytes under crypto, ...

✗ **Discards structure** by design

➔ re-interpreting/parsing mutated bytes as structured objects **defeat the whole purpose**

Problem: Reconciling Structured and Bit-Level Fuzzing

Bit-level (HAVOC) 🤖: bit-flips / byte-increment / ... on raw bytes

- ✓ **Escape the message grammar** (and well-types messages)
 - ➡ inconsistent lengths, header/payload, malformed nesting, invalid bytes under crypto, ...
- ✗ **Discards structure** by design
 - ➡ re-interpreting/parsing mutated bytes as structured objects **defeat the whole purpose**

Structured (DYF) 🤡: structure-aware mutations (term rewriting)

- ✓ Remain in message grammar/well-typed but **escape protocol logic / state machine**
 - ➡ capture DY attacker's capabilities and all the logical attacks it can mount

Problem: Reconciling Structured and Bit-Level Fuzzing

Bit-level (HAVOC) 🤖: bit-flips / byte-increment / ... on raw bytes

- ✓ **Escape the message grammar** (and well-types messages)
 - ➡ inconsistent lengths, header/payload, malformed nesting, invalid bytes under crypto, ...
- ✗ **Discards structure** by design
 - ➡ re-interpreting/parsing mutated bytes as structured objects **defeat the whole purpose**

Structured (DYF) 🤡: structure-aware mutations (term rewriting)

- ✓ Remain in message grammar/well-typed but **escape protocol logic / state machine**
 - ➡ capture DY attacker's capabilities and all the logical attacks it can mount
- ✗ Manipulates **structured data only**
 - ➡ **Unable** to operate on **raw bytes**

Problem: Reconciling Structured and Bit-Level Fuzzing

Bit-level (HAVOC) 🤖: bit-flips / byte-increment / ... on raw bytes

- ✓ **Escape the message grammar** (and well-types messages)
 - ➔ inconsistent lengths, header/payload, malformed nesting, invalid bytes under crypto, ...
- ✗ **Discards structure** by design
 - ➔ re-interpreting/parsing mutated bytes as structured objects **defeat the whole purpose**

Structured (DYF) 😡: structure-aware mutations (term rewriting)

- ✓ Remain in message grammar/well-typed but **escape protocol logic / state machine**
 - ➔ capture DY attacker's capabilities and all the logical attacks it can mount
- ✗ Manipulates **structured data only**
 - ➔ **Unable** to operate on **raw bytes**

Combining both? 🤖+😡

- ✓ DYF on messages as terms + HAVOC on message encodings >> better than each in isolation
 - ➔ **Escapes both message grammar and protocol logic / state machine**

Problem: Reconciling Structured and Bit-Level Fuzzing

Bit-level (HAVOC) 🤖: bit-flips / byte-increment / ... on raw bytes

- ✓ **Escape the message grammar** (and well-types messages)
 - ➔ inconsistent lengths, header/payload, malformed nesting, invalid bytes under crypto, ...
- ✗ **Discards structure** by design
 - ➔ re-interpreting/parsing mutated bytes as structured objects **defeat the whole purpose**

Structured (DYF) 😡: structure-aware mutations (term rewriting)

- ✓ Remain in message grammar/well-typed but **escape protocol logic / state machine**
 - ➔ capture DY attacker's capabilities and all the logical attacks it can mount
- ✗ Manipulates **structured data only**
 - ➔ **Unable** to operate on **raw bytes**

Combining both? 🤖+😡

- ✓ DYF on messages as terms + HAVOC on message encodings >> better than each in isolation
 - ➔ **Escapes both message grammar and protocol logic / state machine**

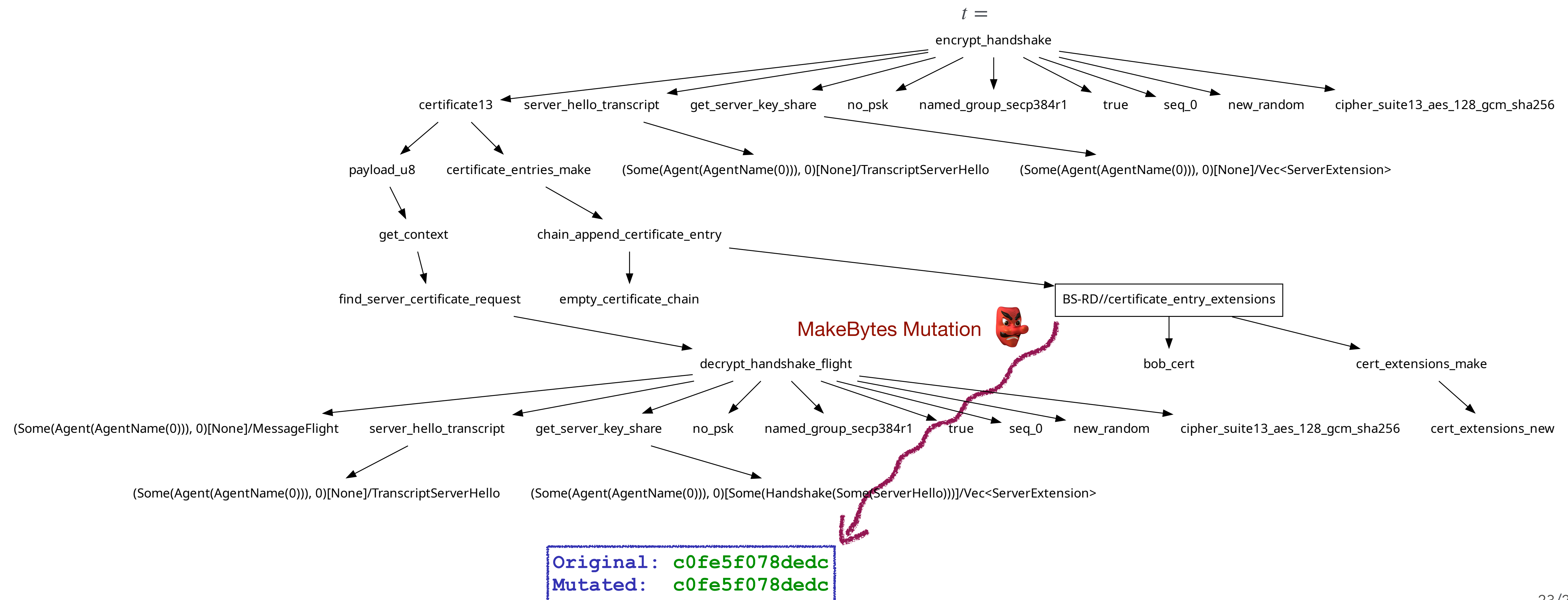
!? Both approaches appear irreconcilable: one requires structure, the other smashes it

Terms with payloads (bytes)

- Generic design: trace keeps its term structure, but any sub-term can be **frozen into the bytes** it evaluates to and **bit-mutated thereafter**. **No re-parsing**.

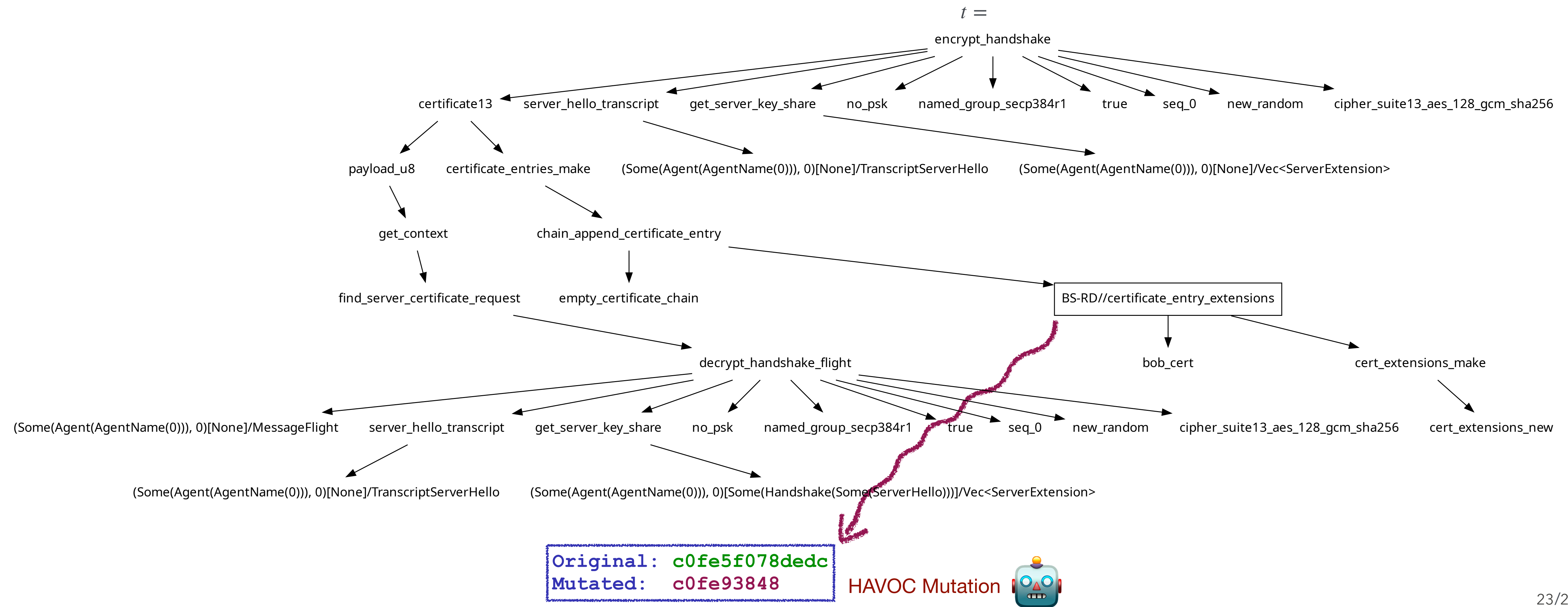
Terms with payloads (bytes)

- Generic design: trace keeps its term structure, but any sub-term can be **frozen into the bytes** it evaluates to and **bit-mutated thereafter**. **No re-parsing**.

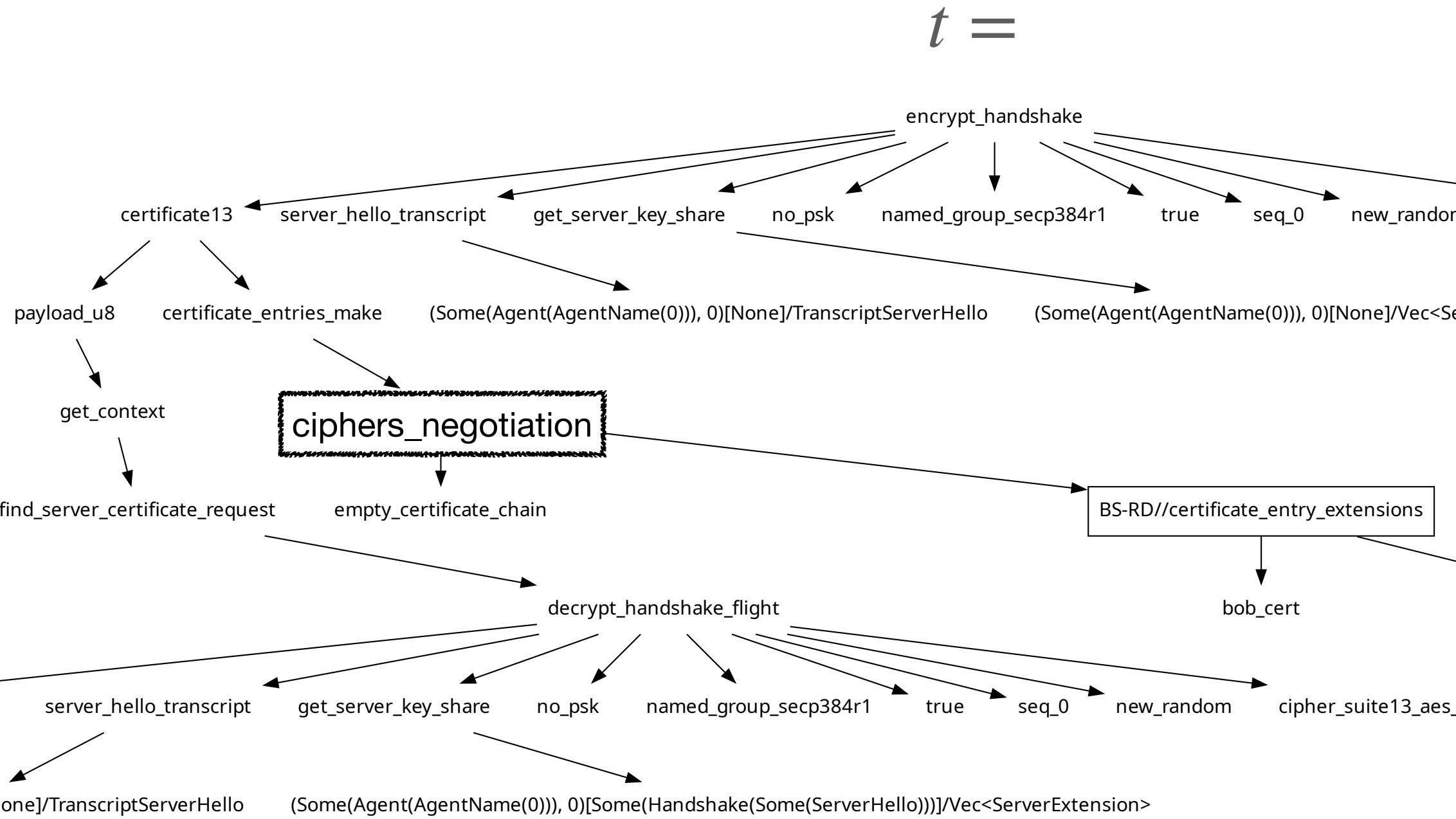


Terms with payloads (bytes)

- Generic design: trace keeps its term structure, but any sub-term can be **frozen into the bytes** it evaluates to and **bit-mutated thereafter**. **No re-parsing**.

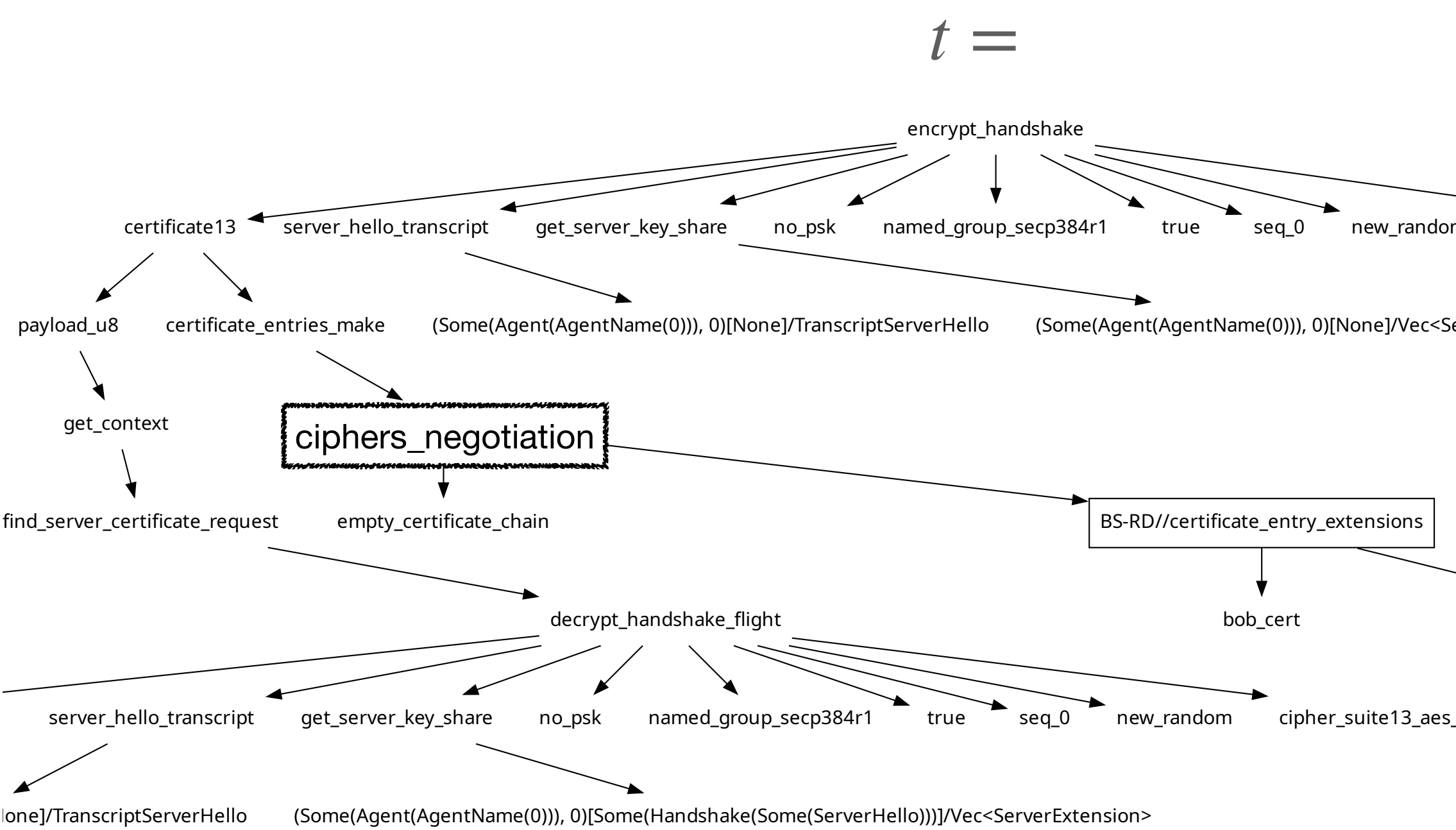


Evaluation of terms with payloads



Evaluation of terms with payloads

Straw man solutions



Evaluation of terms with payloads

Straw man solutions

1. Whole term $\rightarrow$ bytes

✗ Loses DY entirely

$t =$

Original: **c0fe5f078dedc**
Mutated: **c0fe93848**

Evaluation of terms with payloads

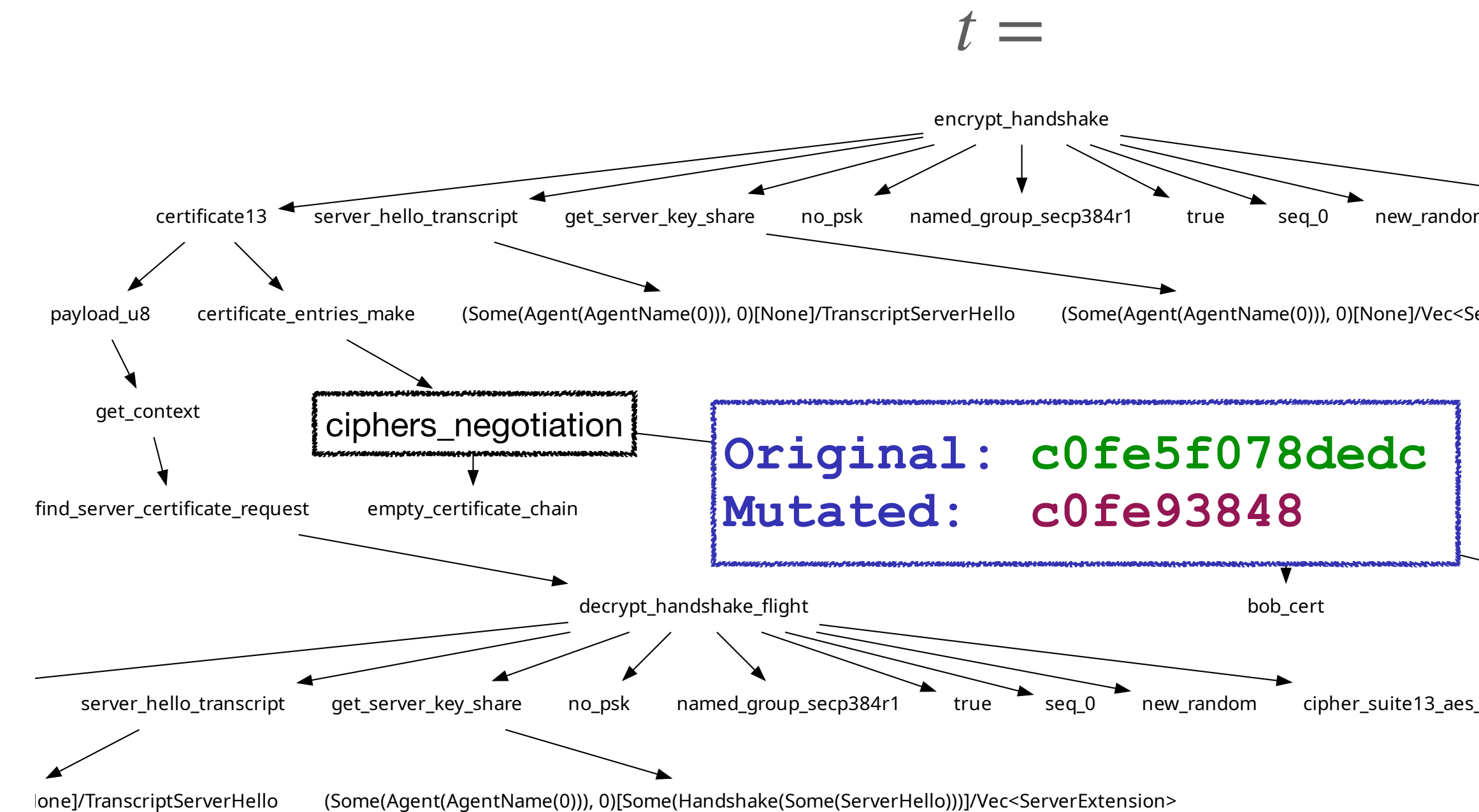
Straw man solutions

1. Whole term $\rightarrow$ bytes

✗ Loses DY entirely

2. Re-parse **c0fe93...** and always re-apply the parent function

✗ Fails exactly when HAVOC succeeds at grammar escaping



Evaluation of terms with payloads

Straw man solutions

1. Whole term $\rightarrow$ bytes

✗ Loses DY entirely

2. Re-parse `c0fe93...` and always re-apply the parent function

✗ Fails exactly when HAVOC succeeds at grammar escaping

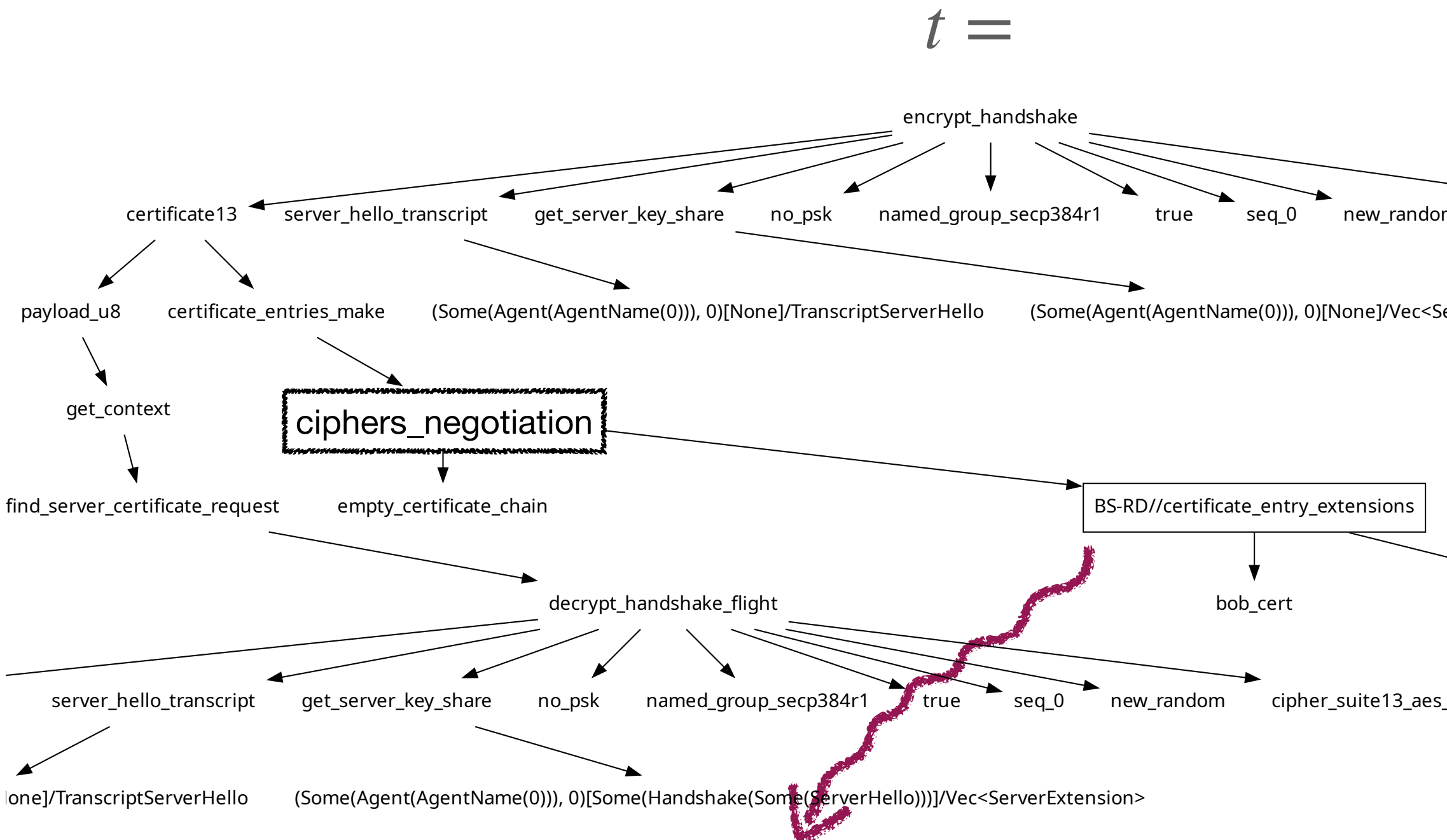
3. Locate `c0fe5f...` in $\llbracket t \rrbracket_\phi$ and replaces by `c0fe93...`

✗ Opaque functions (e.g. encryption) `c0fe5f...` $\notin \llbracket t \rrbracket_\phi$

✗ No unique match: original bytes can be `0` or even be empty

✗ Different-length mutation: terms above have inconsistent lengths (buffer $\neq$ announced length). **Wanted!**

But needs to be controlled: otherwise encryption may not work, client hello might be rejected, etc.



Original: `c0fe5f078dedc`

Mutated: `c0fe93848`

$\llbracket t \rrbracket_\phi =$

6d883ddff3f5c2308687440269d47e37
d932f4d6b60941669676768a34ea18b7
aac0fe5f078dedc5dc87a65aa4063789
0f666c388d4d2c7ca15bf533605a0e90
a7e33bb47623b49715ca1c6b53972cac

Evaluation of terms with payloads

Straw man solutions

1. Whole term $\rightarrow$ bytes

✗ Loses DY entirely

2. Re-parse `c0fe93...` and always re-apply the parent function

✗ Fails exactly when HAVOC succeeds at grammar escaping

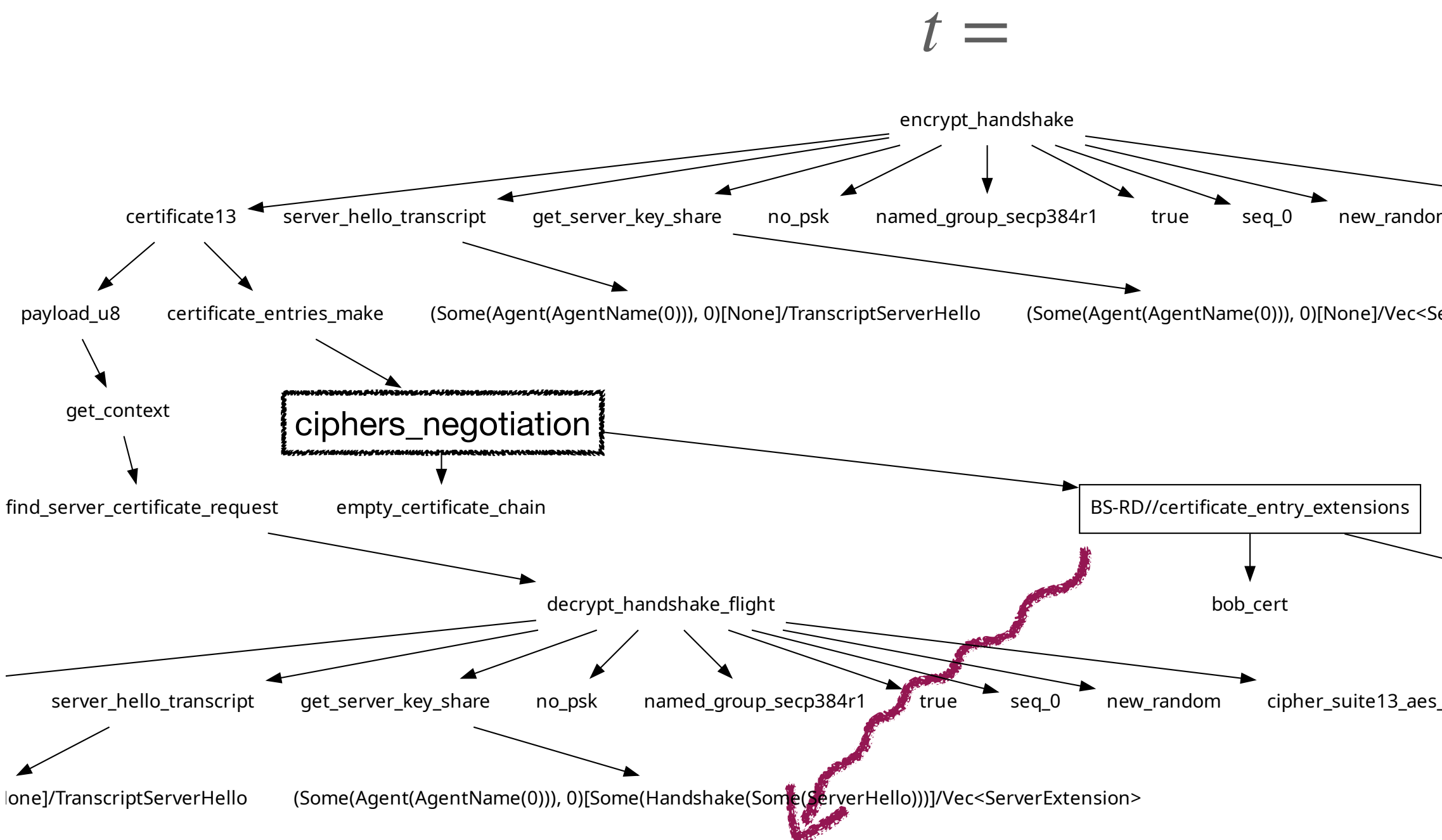
3. Locate `c0fe5f...` in $\llbracket t \rrbracket_\phi$ and replaces by `c0fe93...`

✗ Opaque functions (e.g. encryption) `c0fe5f...` $\notin \llbracket t \rrbracket_\phi$

✗ No unique match: original bytes can be `0` or even be empty

✗ Different-length mutation: terms above have inconsistent lengths (buffer $\neq$ announced length). **Wanted!**

But needs to be controlled: otherwise encryption may not work, client hello might be rejected, etc.

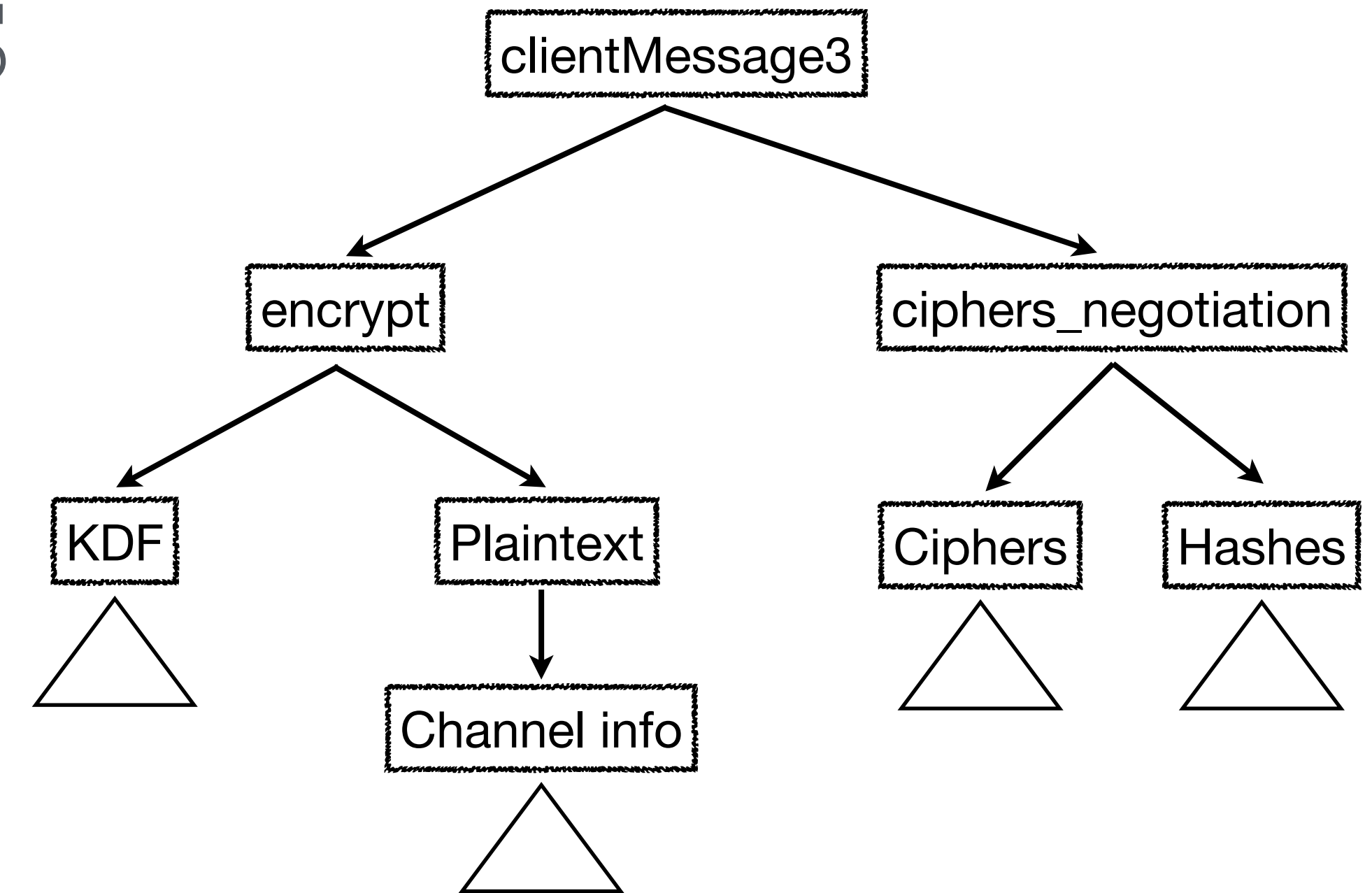


$\llbracket t \rrbracket_\phi =$

6d883ddff3f5c2308687440269d47e37
d932f4d6b60941669676768a34ea18b7
aac0fe938485dc87a65aa4063789
0f666c388d4d2c7ca15bf533605a0e90
a7e33bb47623b49715ca1c6b53972cac

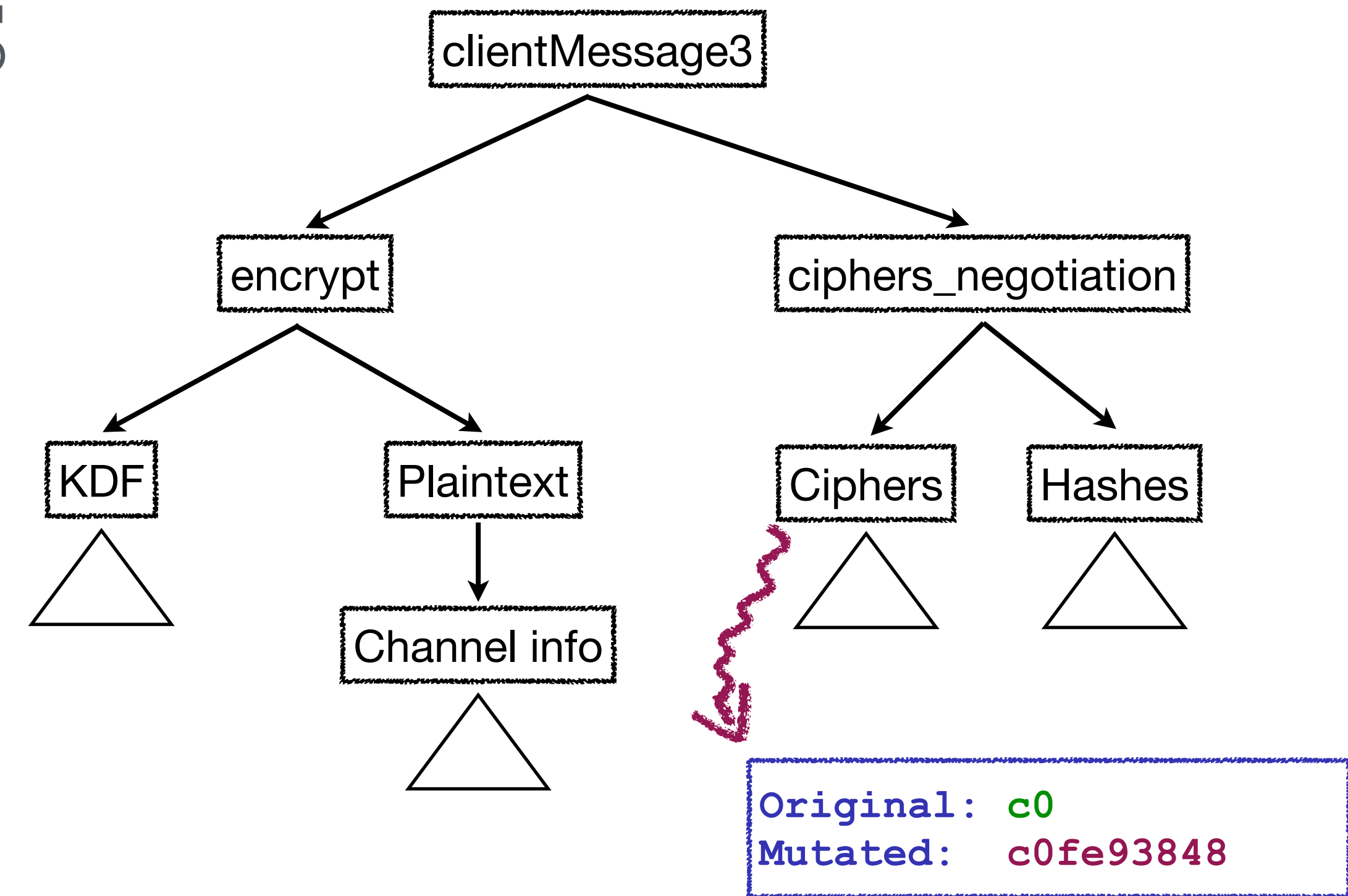
Evaluation: Key Ingredients

- **Locate**: walk the term's path down cached encodings
→ at each level, child bytes = unique substring of parent bytes (disambiguate by position) → narrower and narrower search window until hitting the location
- **Splice**: overwrite that found span with the mutated payload and keep track of shift deltas (different sizes)
- **Eval(t)**: recurse each sub-term bottom-up; its payload bubbles up; collected payloads spliced into the encoding at the root (locate+splice)
- **Read Flag**  put at some terms: short-circuit – value is parsed after payload replacements (absorbs all payloads in sub-terms)
- **Opaque** boundary: no more payloads bubbling up when crossing opaque functions!



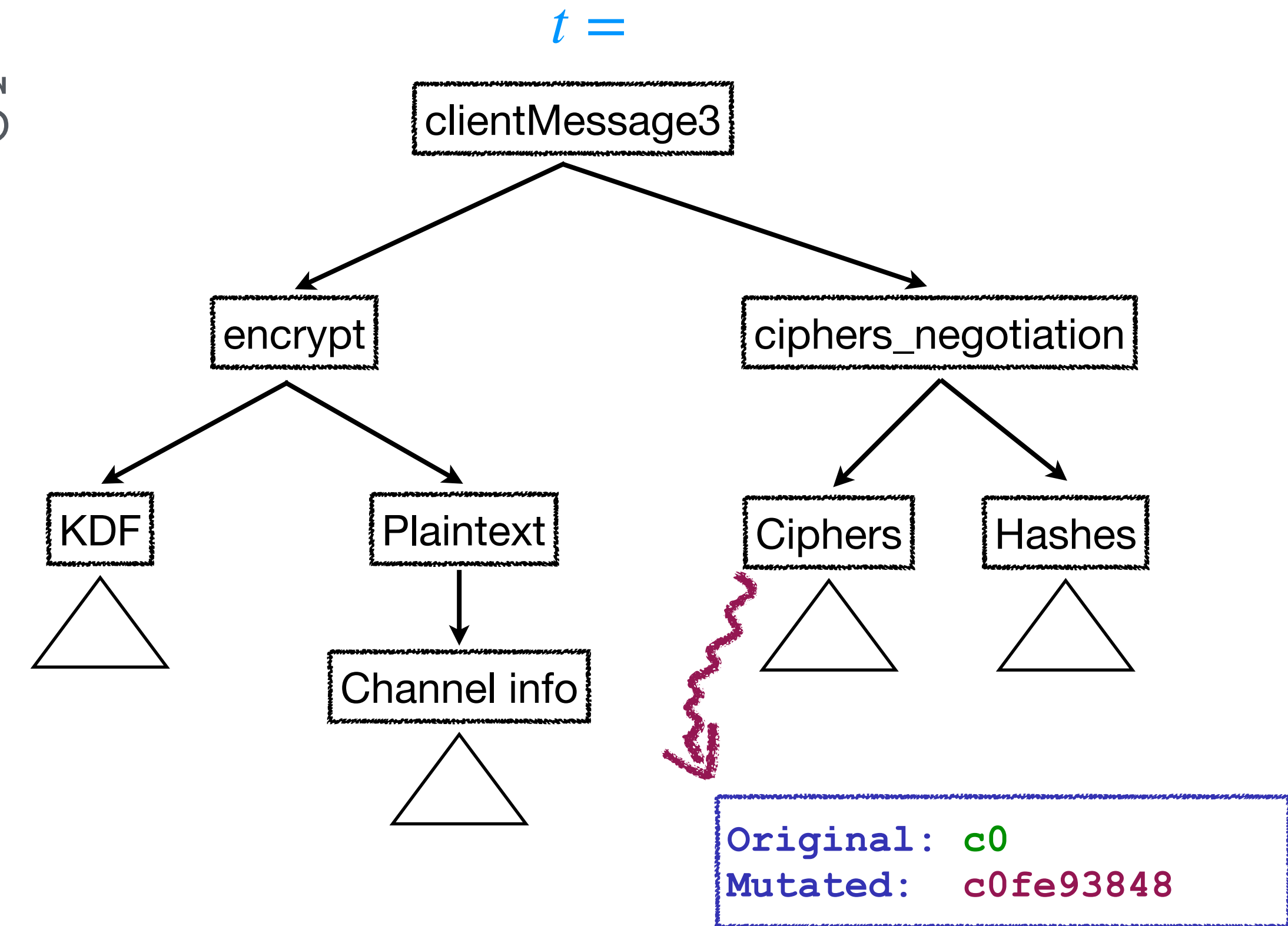
Evaluation: Key Ingredients

- **Locate**: walk the term's path down cached encodings
→ at each level, child bytes = unique substring of parent bytes (disambiguate by position) → narrower and narrower search window until hitting the location
- **Splice**: overwrite that found span with the mutated payload and keep track of shift deltas (different sizes)
- **Eval(t)**: recurse each sub-term bottom-up; its payload bubbles up; collected payloads spliced into the encoding at the root (locate+splice)
- **Read Flag**  put at some terms: short-circuit – value is parsed after payload replacements (absorbs all payloads in sub-terms)
- **Opaque** boundary: no more payloads bubbling up when crossing opaque functions!



Evaluation: Key Ingredients

- **Locate**: walk the term's path down cached encodings
→ at each level, child bytes = unique substring of parent bytes (disambiguate by position) → narrower and narrower search window until hitting the location
- **Splice**: overwrite that found span with the mutated payload and keep track of shift deltas (different sizes)
- **Eval(t)**: recurse each sub-term bottom-up; its payload bubbles up; collected payloads spliced into the encoding at the root (locate+splice)
- **Read Flag**  put at some terms: short-circuit – value is parsed after payload replacements (absorbs all payloads in sub-terms)
- **Opaque** boundary: no more payloads bubbling up when crossing opaque functions!

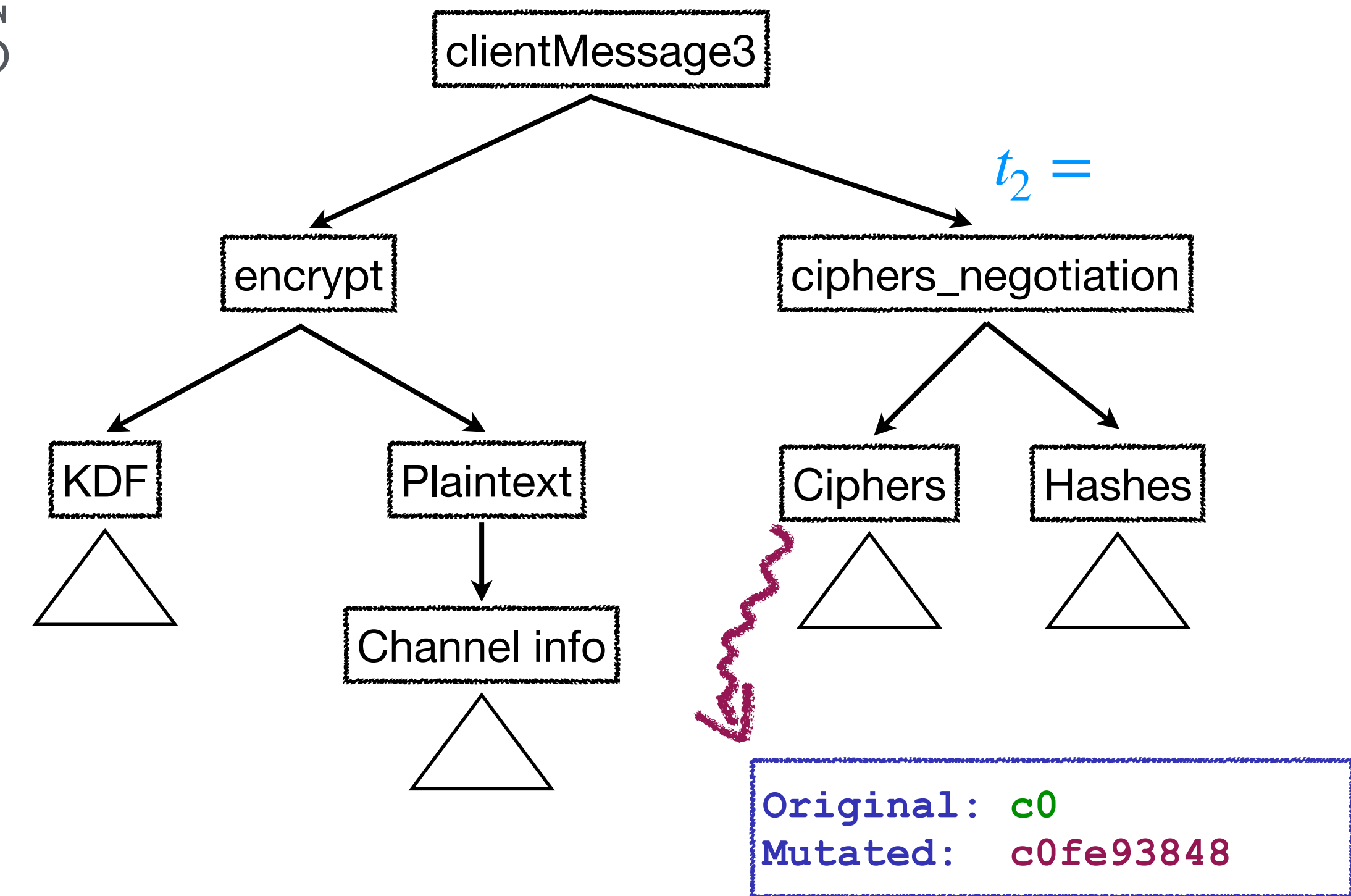


$\llbracket t \rrbracket_\phi =$

```
6d883ddff3f5c2308687440269d47e37
d932f4d6b60941669676768a34ea18b7
aac0fe5f078dedc5dc87a65aa4063789
0f666c388d4d2c7ca15bf533605a0e90
a7e33bb47623b49715ca1c6b539724c0
```

Evaluation: Key Ingredients

- **Locate**: walk the term's path down cached encodings
→ at each level, child bytes = unique substring of parent bytes (disambiguate by position) → narrower and narrower search window until hitting the location
- **Splice**: overwrite that found span with the mutated payload and keep track of shift deltas (different sizes)
- **Eval(t)**: recurse each sub-term bottom-up; its payload bubbles up; collected payloads spliced into the encoding at the root (locate+splice)
- **Read Flag**  put at some terms: short-circuit – value is parsed after payload replacements (absorbs all payloads in sub-terms)
- **Opaque** boundary: no more payloads bubbling up when crossing opaque functions!

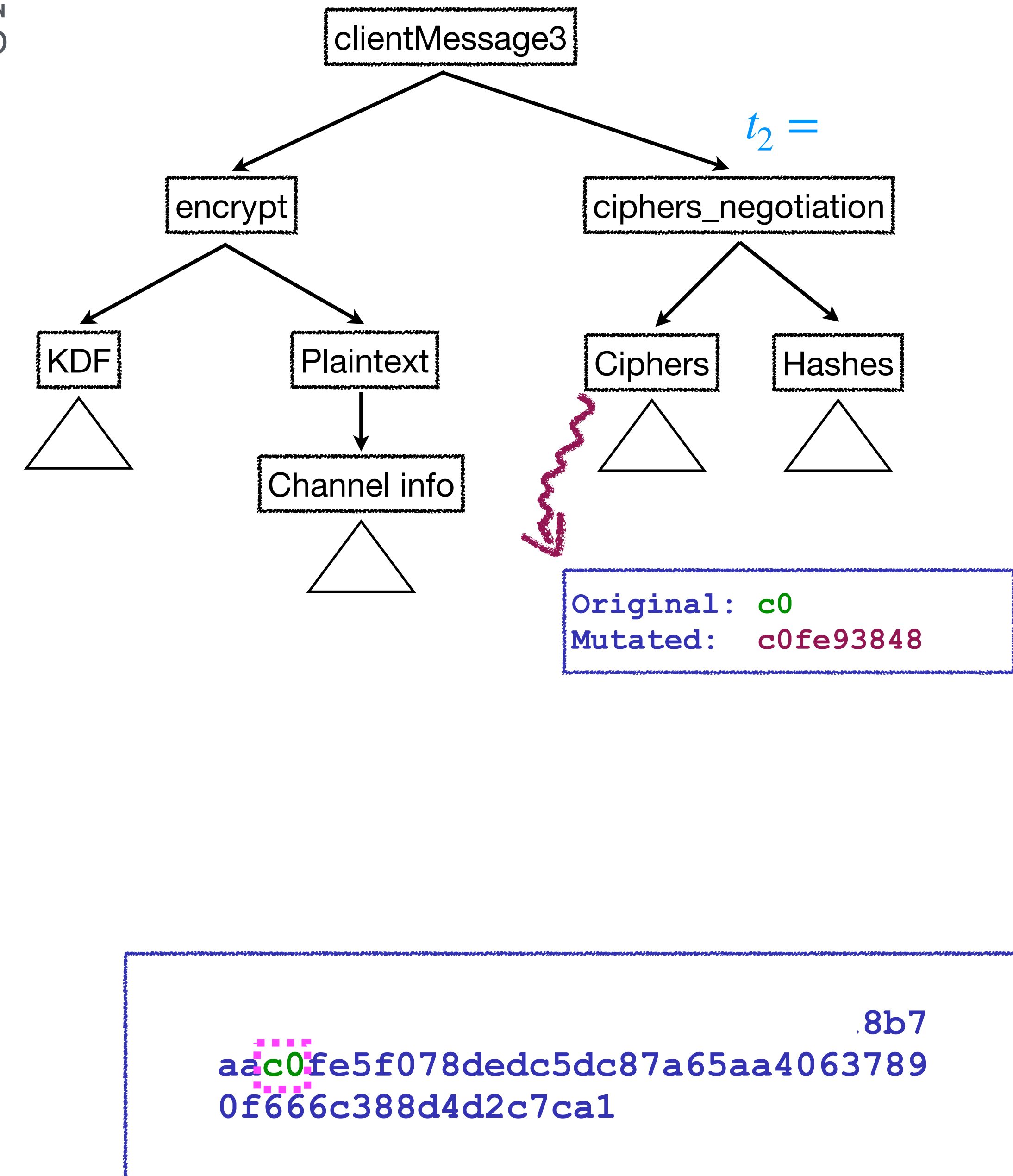


$\llbracket t_2 \rrbracket_\phi =$

aac0fe5f078dedc5dc87a65aa40637898b7
0f666c388d4d2c7ca1

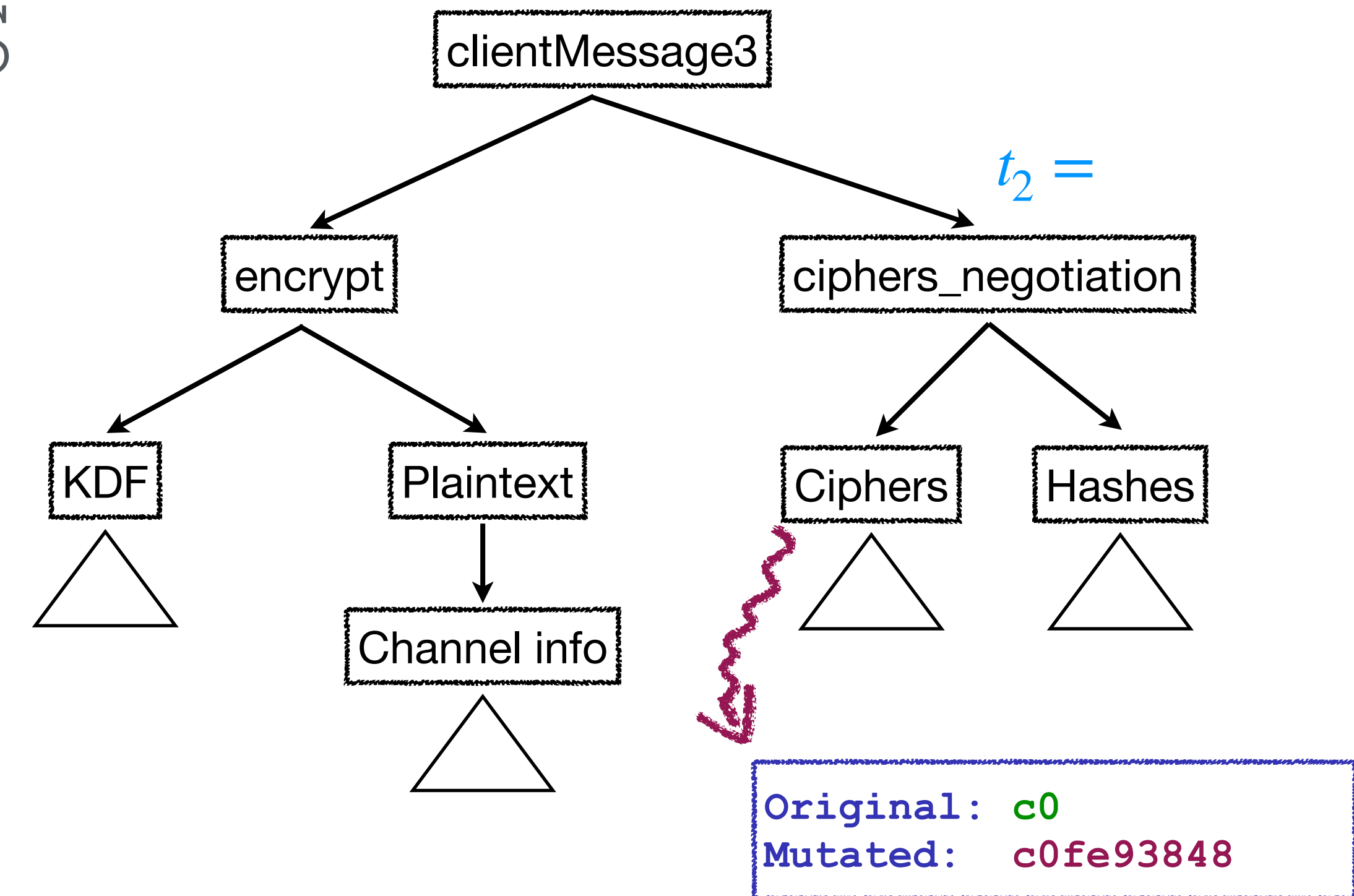
Evaluation: Key Ingredients

- **Locate**: walk the term's path down cached encodings
→ at each level, child bytes = unique substring of parent bytes (disambiguate by position) → narrower and narrower search window until hitting the location
- **Splice**: overwrite that found span with the mutated payload and keep track of shift deltas (different sizes)
- **Eval(t)**: recurse each sub-term bottom-up; its payload bubbles up; collected payloads spliced into the encoding at the root (locate+splice)
- **Read Flag**  put at some terms: short-circuit – value is parsed after payload replacements (absorbs all payloads in sub-terms)
- **Opaque** boundary: no more payloads bubbling up when crossing opaque functions!



Evaluation: Key Ingredients

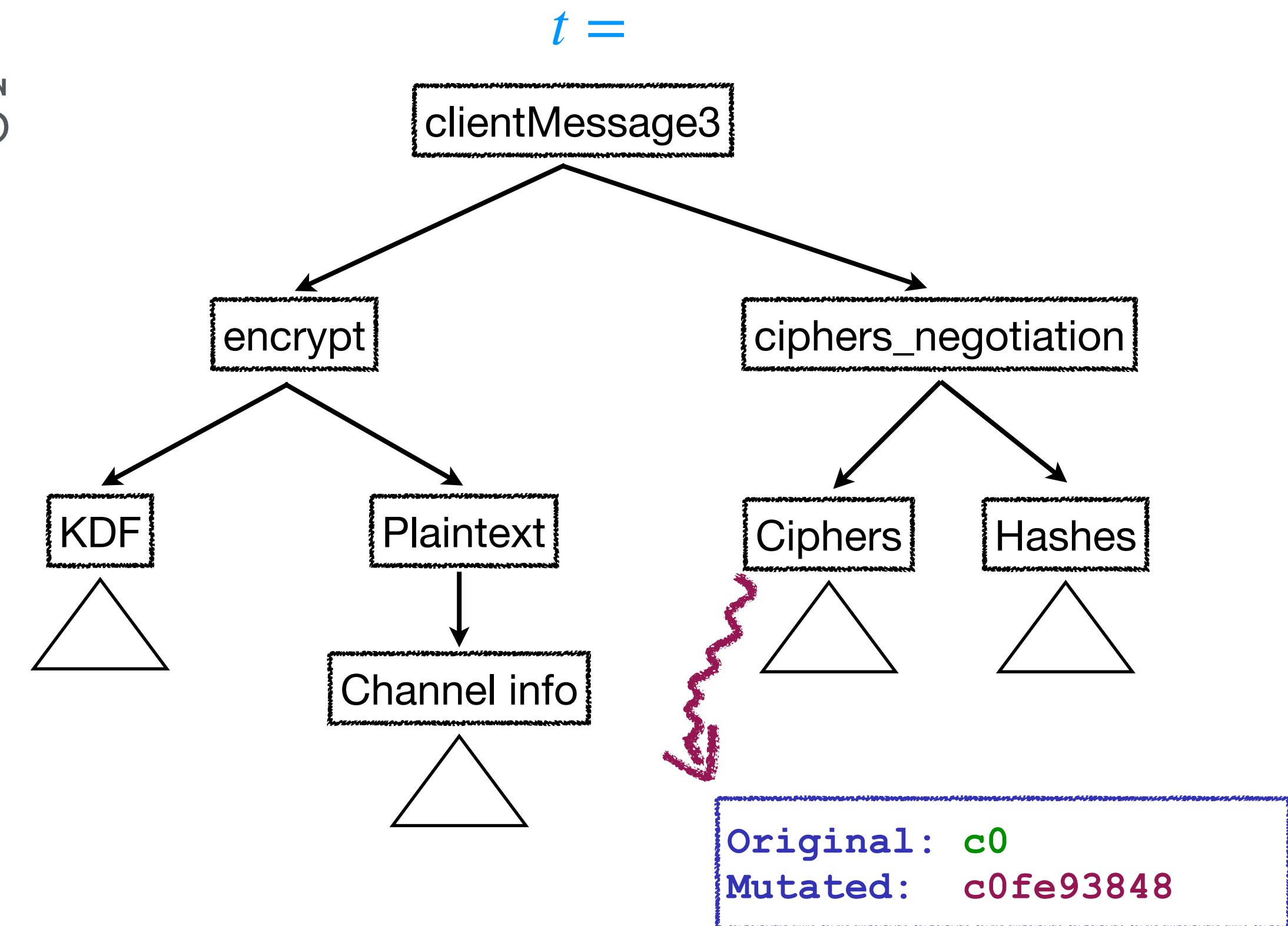
- **Locate**: walk the term's path down cached encodings
→ at each level, child bytes = unique substring of parent bytes (disambiguate by position) → narrower and narrower search window until hitting the location
- **Splice**: overwrite that found span with the mutated payload and keep track of shift deltas (different sizes)
- **Eval(t)**: recurse each sub-term bottom-up; its payload bubbles up; collected payloads spliced into the encoding at the root (locate+splice)
- **Read Flag**  put at some terms: short-circuit – value is parsed after payload replacements (absorbs all payloads in sub-terms)
- **Opaque** boundary: no more payloads bubbling up when crossing opaque functions!



```
6d883ddff3f5c2308687440269d47e37
d932f4d6b60941669676768a34ea18b7
aac0fe5f078dedc5dc87a65aa4063789
0f666c388d4d2c7ca15bf533605a0e90
a7e33bb47623b49715ca1c6b539724c0
```

Evaluation: Key Ingredients

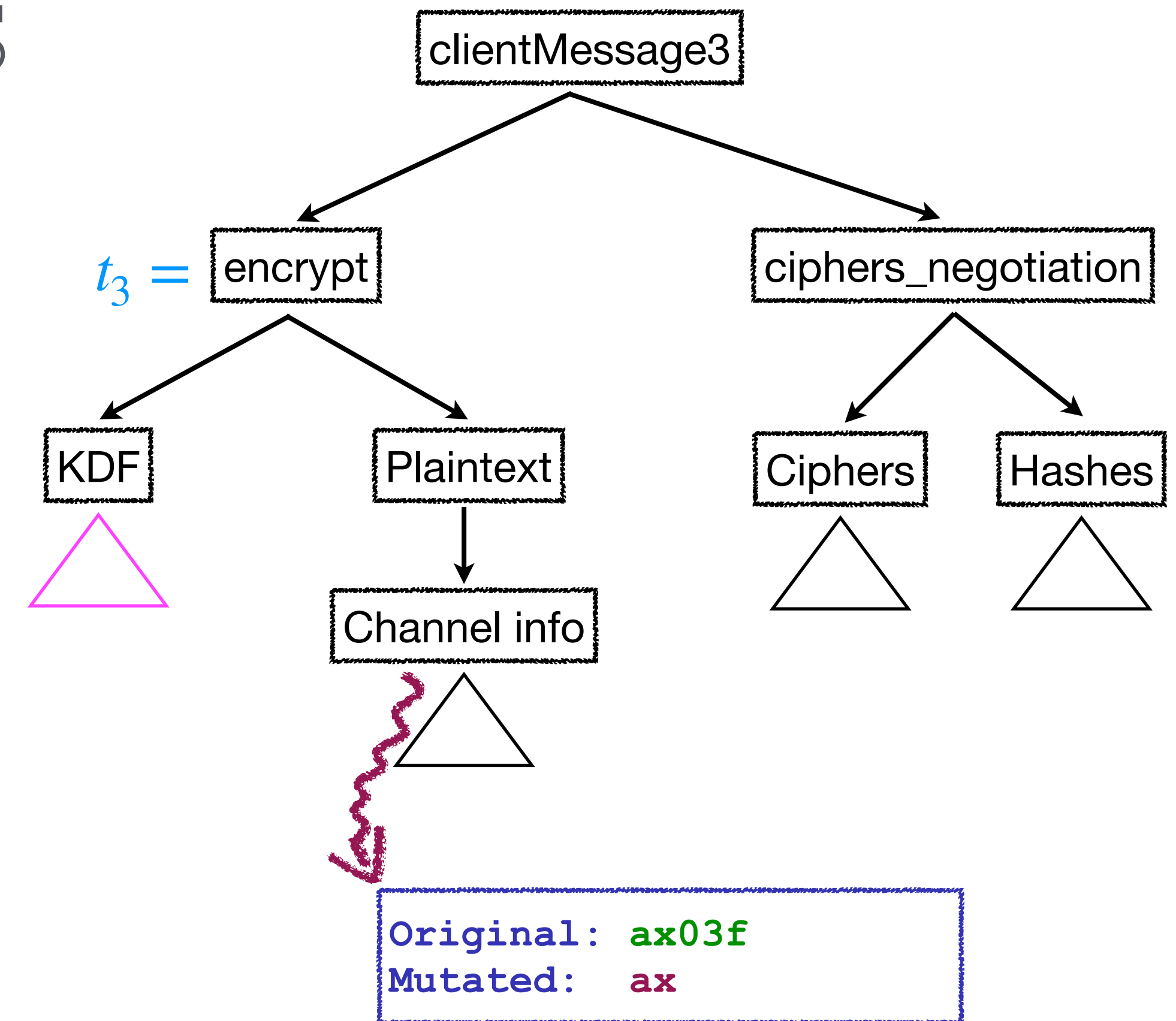
- **Locate**: walk the term's path down cached encodings
→ at each level, child bytes = unique substring of parent bytes (disambiguate by position) → narrower and narrower search window until hitting the location
- **Splice**: overwrite that found span with the mutated payload and keep track of shift deltas (different sizes)
- **Eval(t)**: recurse each sub-term bottom-up; its payload bubbles up; collected payloads spliced into the encoding at the root (locate+splice)
- **Read Flag**  put at some terms: short-circuit – value is parsed after payload replacements (absorbs all payloads in sub-terms)
- **Opaque** boundary: no more payloads bubbling up when crossing opaque functions!



```
6d883ddff3f5c2308687440269d47e37
d932f4d6b60941669676768a34ea18b7
aac0fe93848fe5f078dedc5dc87a65aa4063789
0f666c388d4d2c7ca15bf533605a0e90
a7e33bb47623b49715ca1c6b539724c0
```

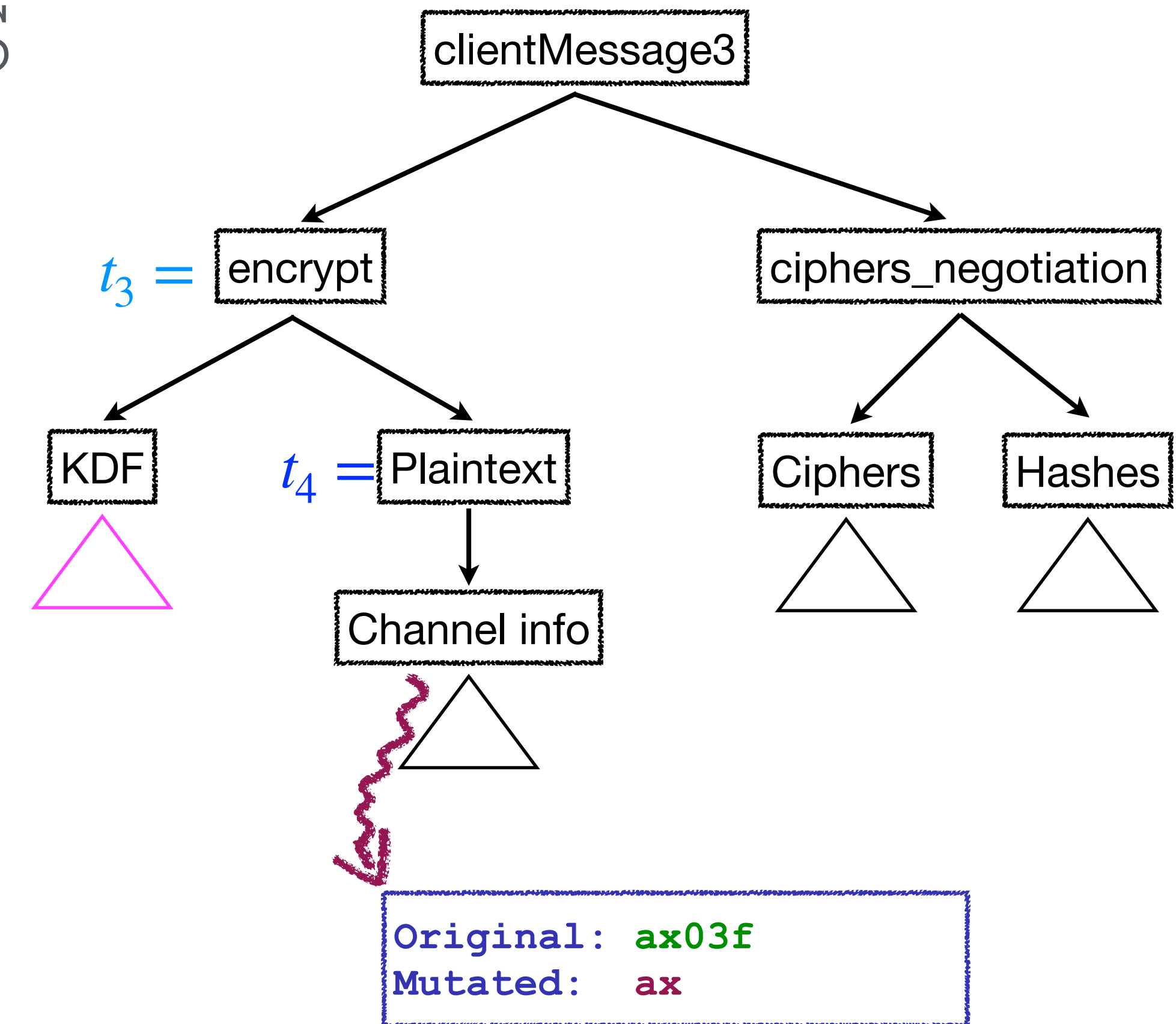

Evaluation: Key Ingredients

- **Locate**: walk the term's path down cached encodings
→ at each level, child bytes = unique substring of parent bytes (disambiguate by position) → narrower and narrower search window until hitting the location
- **Splice**: overwrite that found span with the mutated payload and keep track of shift deltas (different sizes)
- **Eval(t)**: recurse each sub-term bottom-up; its payload bubbles up; collected payloads spliced into the encoding at the root (locate+splice)
- **Read Flag**  put at some terms: short-circuit – value is parsed after payload replacements (absorbs all payloads in sub-terms)
- **Opaque** boundary: no more payloads bubbling up when crossing opaque functions!



Evaluation: Key Ingredients

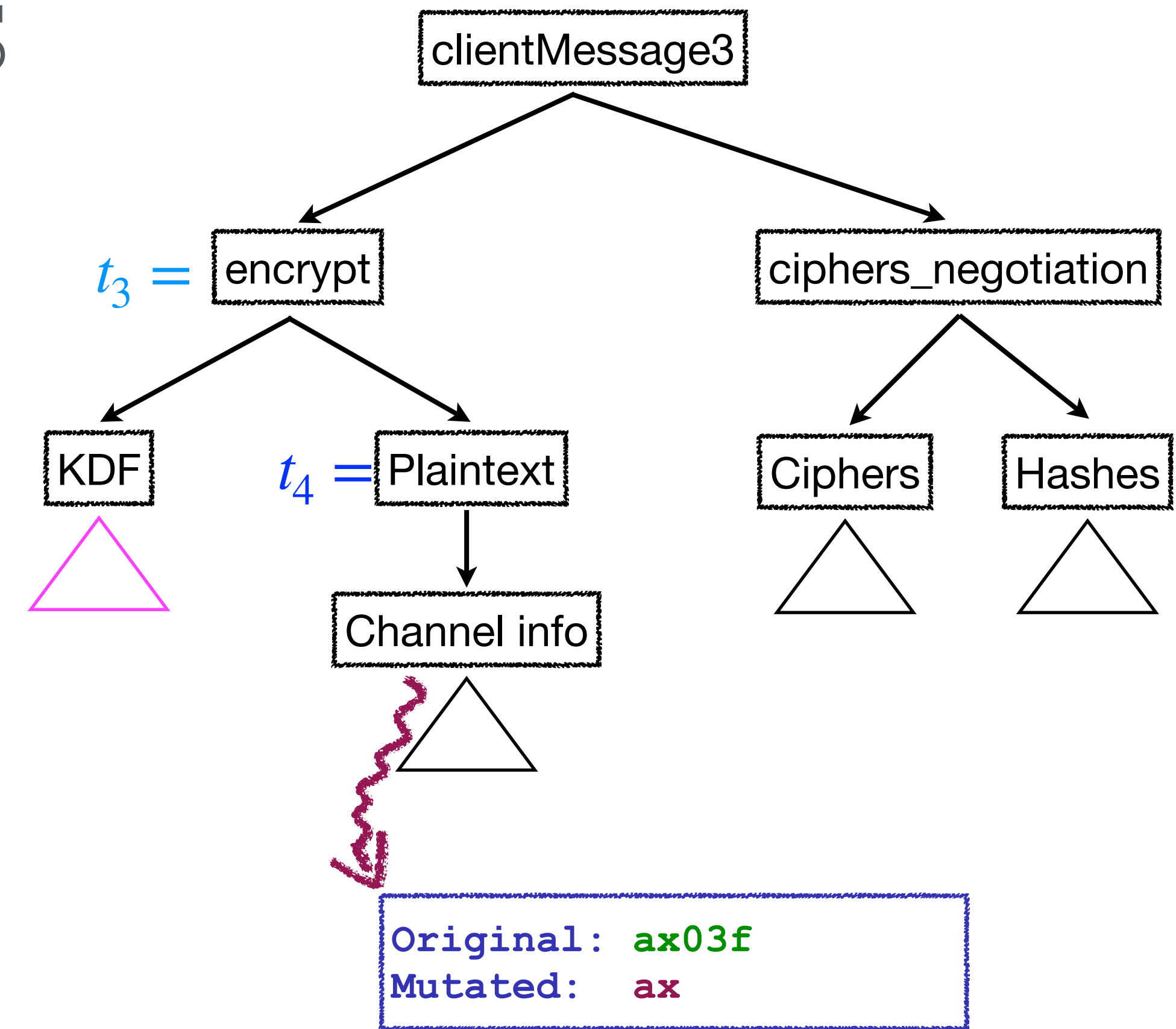
- **Locate**: walk the term's path down cached encodings
→ at each level, child bytes = unique substring of parent bytes (disambiguate by position) → narrower and narrower search window until hitting the location
- **Splice**: overwrite that found span with the mutated payload and keep track of shift deltas (different sizes)
- **Eval(t)**: recurse each sub-term bottom-up; its payload bubbles up; collected payloads spliced into the encoding at the root (locate+splice)
- **Read Flag**  put at some terms: short-circuit – value is parsed after payload replacements (absorbs all payloads in sub-terms)
- **Opaque** boundary: no more payloads bubbling up when crossing opaque functions!



$\text{eval_replace}(t_3, \phi) =$
 $\text{replace}(\text{openssl.crypto.aes}(\text{eval}(\dots),$
 $\text{eval}(t_4, \phi))$

Evaluation: Key Ingredients

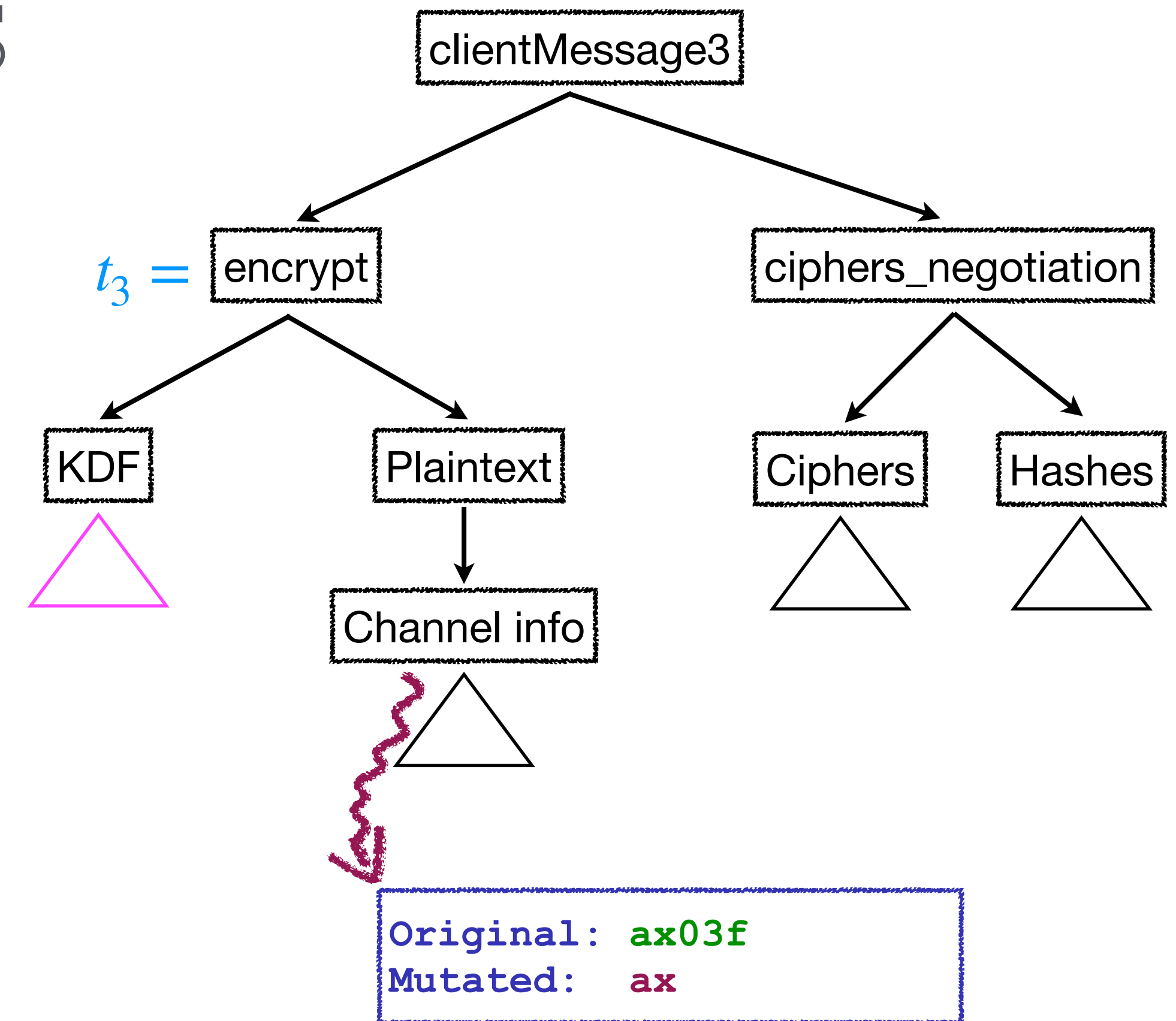
- **Locate**: walk the term's path down cached encodings
→ at each level, child bytes = unique substring of parent bytes (disambiguate by position) → narrower and narrower search window until hitting the location
- **Splice**: overwrite that found span with the mutated payload and keep track of shift deltas (different sizes)
- **Eval(t)**: recurse each sub-term bottom-up; its payload bubbles up; collected payloads spliced into the encoding at the root (locate+splice)
- **Read Flag**  put at some terms: short-circuit – value is parsed after payload replacements (absorbs all payloads in sub-terms)
- **Opaque** boundary: no more payloads bubbling up when crossing opaque functions!



```
eval_replace( $t_3$ ,  $\phi$ ) =
    replace(openssl.crypto.aes( eval(...),
                                eval( $t_4$ ,  $\phi$ ))
            )
```

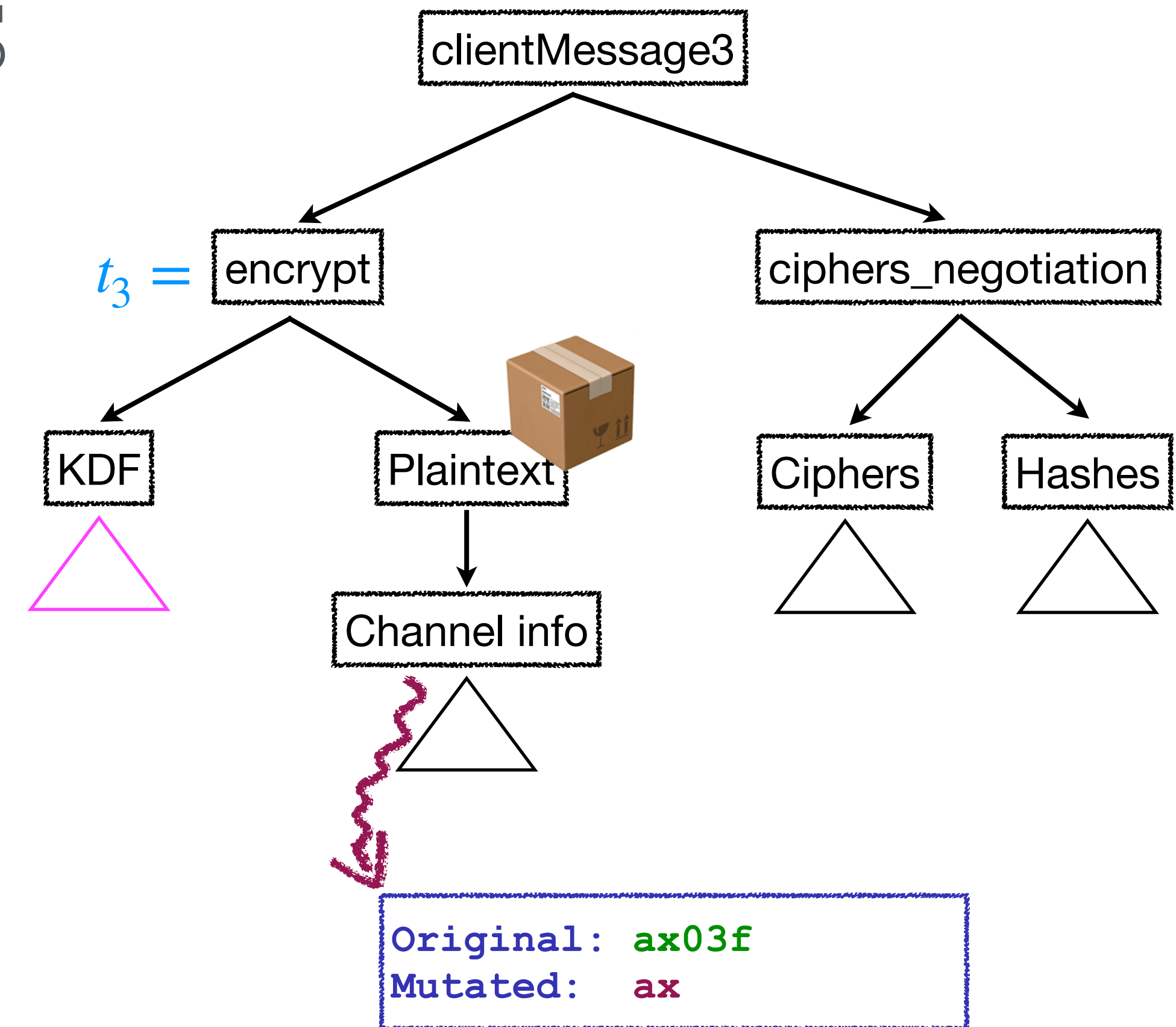

Evaluation: Key Ingredients

- **Locate**: walk the term's path down cached encodings
→ at each level, child bytes = unique substring of parent bytes (disambiguate by position) → narrower and narrower search window until hitting the location
- **Splice**: overwrite that found span with the mutated payload and keep track of shift deltas (different sizes)
- **Eval(t)**: recurse each sub-term bottom-up; its payload bubbles up; collected payloads spliced into the encoding at the root (locate+splice)
- **Read Flag**  put at some terms: short-circuit – value is parsed after payload replacements (absorbs all payloads in sub-terms)
- **Opaque** boundary: no more payloads bubbling up when crossing opaque functions!



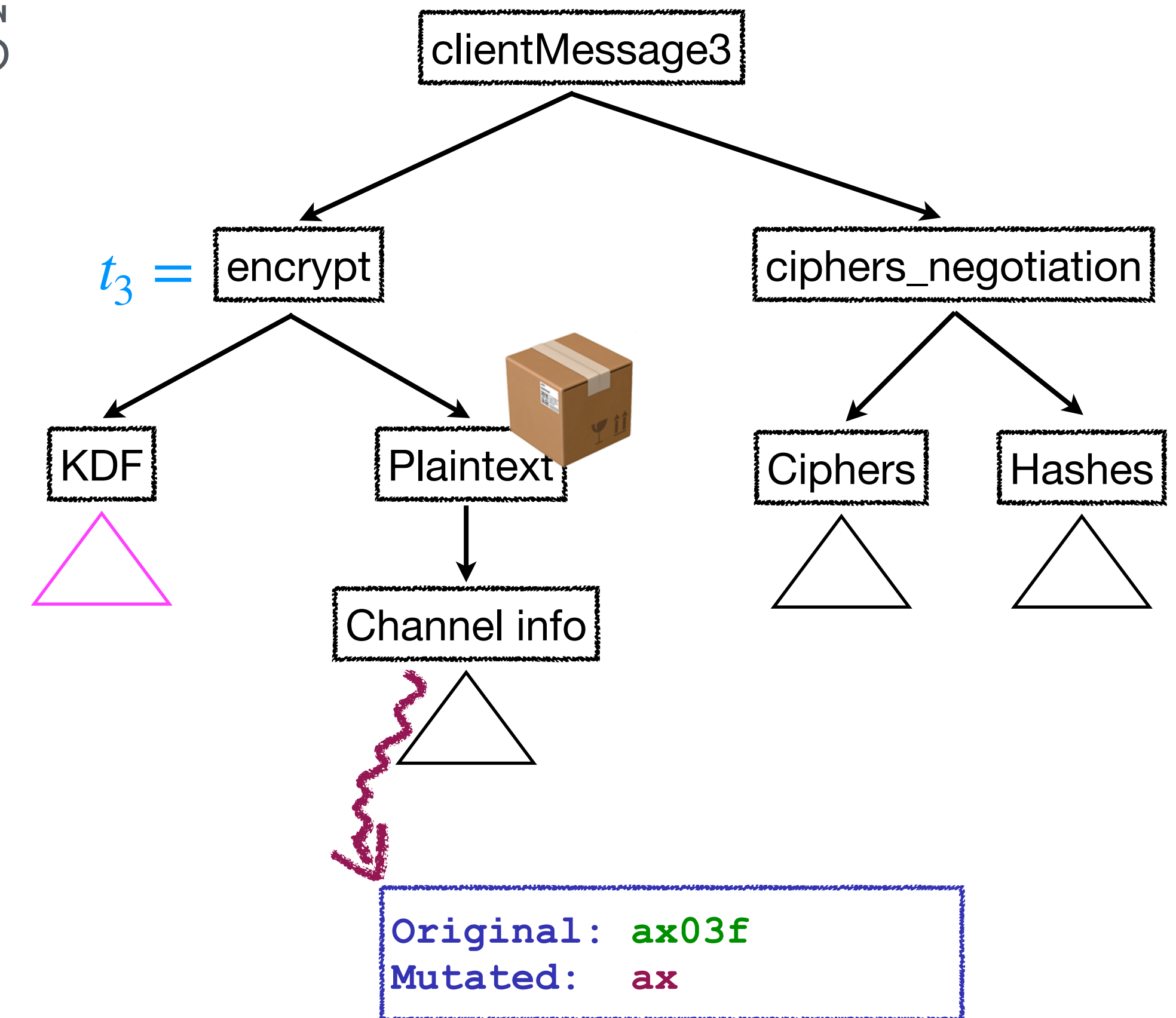
Evaluation: Key Ingredients

- **Locate**: walk the term's path down cached encodings
→ at each level, child bytes = unique substring of parent bytes (disambiguate by position) → narrower and narrower search window until hitting the location
- **Splice**: overwrite that found span with the mutated payload and keep track of shift deltas (different sizes)
- **Eval(t)**: recurse each sub-term bottom-up; its payload bubbles up; collected payloads spliced into the encoding at the root (locate+splice)
- **Read Flag**  put at some terms: short-circuit – value is parsed after payload replacements (absorbs all payloads in sub-terms)
- **Opaque** boundary: no more payloads bubbling up when crossing opaque functions!



Evaluation: Key Ingredients

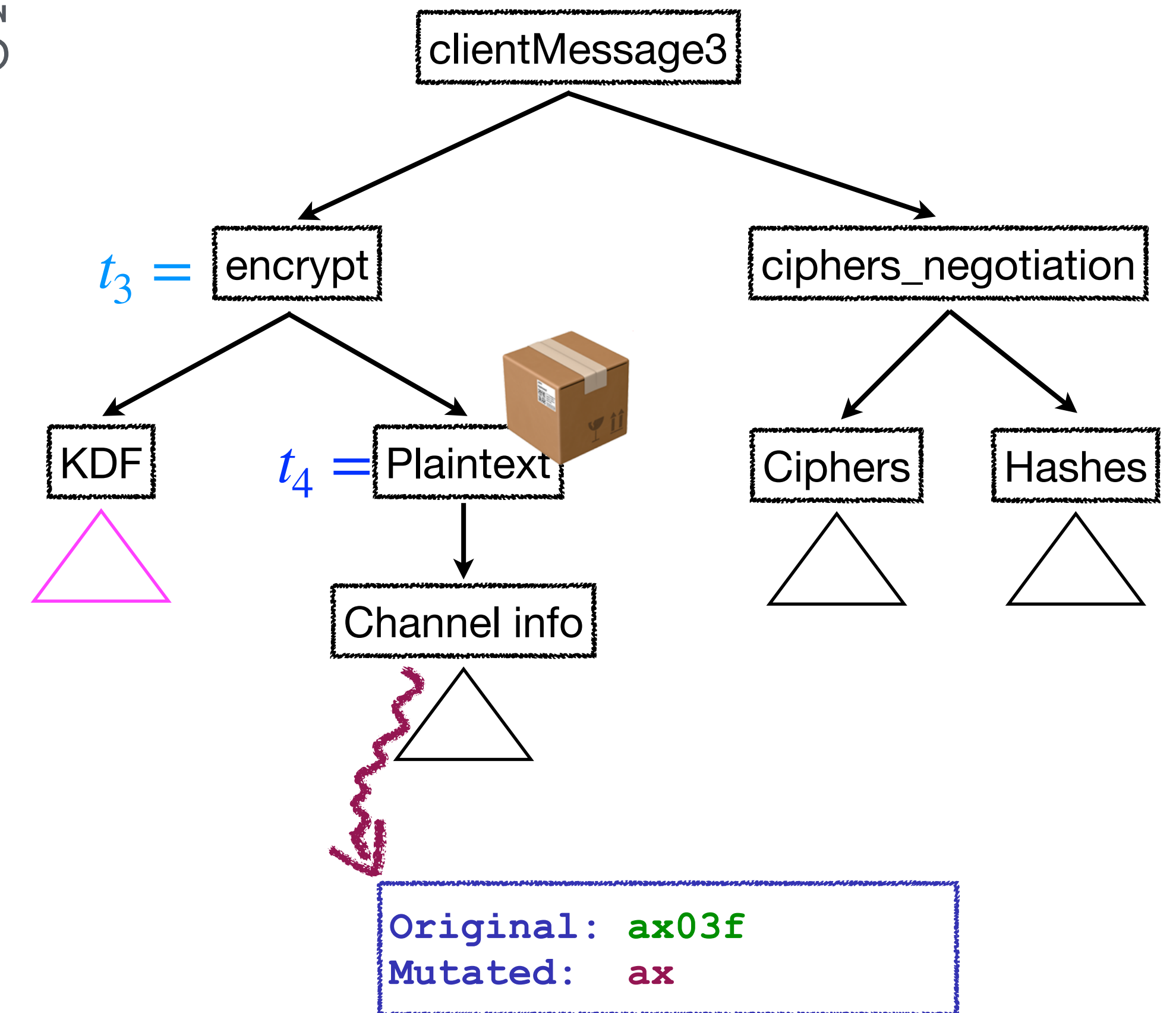
- **Locate**: walk the term's path down cached encodings
→ at each level, child bytes = unique substring of parent bytes (disambiguate by position) → narrower and narrower search window until hitting the location
- **Splice**: overwrite that found span with the mutated payload and keep track of shift deltas (different sizes)
- **Eval(t)**: recurse each sub-term bottom-up; its payload bubbles up; collected payloads spliced into the encoding at the root (locate+splice)
- **Read Flag**  put at some terms: short-circuit – value is parsed after payload replacements (absorbs all payloads in sub-terms)
- **Opaque** boundary: no more payloads bubbling up when crossing opaque functions!



$\text{eval_replace}(t_3, \phi) =$
`openssl.crypto.aes( eval(...),`
`)`

Evaluation: Key Ingredients

- **Locate**: walk the term's path down cached encodings
→ at each level, child bytes = unique substring of parent bytes (disambiguate by position) → narrower and narrower search window until hitting the location
- **Splice**: overwrite that found span with the mutated payload and keep track of shift deltas (different sizes)
- **Eval(t)**: recurse each sub-term bottom-up; its payload bubbles up; collected payloads spliced into the encoding at the root (locate+splice)
- **Read Flag**  put at some terms: short-circuit – value is parsed after payload replacements (absorbs all payloads in sub-terms)
- **Opaque** boundary: no more payloads bubbling up when crossing opaque functions!



$\text{eval_replace}(t_3, \phi) =$
 $\text{openssl.crypto.aes}(\text{eval}(\dots),$
 $\quad \llbracket \text{parse} \text{  (\text{eval_replace}(t_4, \phi)) \rrbracket \rrbracket$
 $\quad)$

WIP Evaluation and Results

WIP Evaluation and Results

New bug class DY cannot express

WIP Evaluation and Results

New bug class DY cannot express

- HAVOC+DY rediscovered **Heartbleed** in minutes. Pure DY never reaches it. Lying length → root

WIP Evaluation and Results

New bug class DY cannot express

- HAVOC+DY rediscovered **Heartbleed** in minutes. Pure DY never reaches it. Lying length → root
- **New security bug** 🤡: WolfSSL suffers from a certificate malleability / agreement violation (under responsible disclosure). Found in ~15 min. Impacts consumer like ircd-hybrid
Requires: (i) HAVOC under encryption, (ii) lying about length in cert. but NOT in cert. wrapper, **and** (iii) DY Objective Oracle capabilities (checking agreement properties) 

WIP Evaluation and Results

New bug class DY cannot express

- HAVOC+DY rediscovered **Heartbleed** in minutes. Pure DY never reaches it. Lying length → root
- **New security bug** 🐞: WolfSSL suffers from a certificate malleability / agreement violation (under responsible disclosure). Found in ~15 min. Impacts consumer like ircd-hybrid
Requires: (i) HAVOC under encryption, (ii) lying about length in cert. but NOT in cert. wrapper, **and** (iii) DY Objective Oracle capabilities (checking agreement properties) 

Coverage

- DY-only (baseline) < bit-only on terms (+10%) < DY+bit (+20-22%).

WIP Evaluation and Results

New bug class DY cannot express

- HAVOC+DY rediscovered **Heartbleed** in minutes. Pure DY never reaches it. Lying length → root
- **New security bug** 🤖: WolfSSL suffers from a certificate malleability / agreement violation (under responsible disclosure). Found in ~15 min. Impacts consumer like ircd-hybrid
Requires: (i) HAVOC under encryption, (ii) lying about length in cert. but NOT in cert. wrapper, **and** (iii) DY Objective Oracle capabilities (checking agreement properties) 

Coverage

- DY-only (baseline) < bit-only on terms (+10%) < DY+bit (+20-22%).
- Gains land in byte-sensitive code: cipher/sig-alg value-space, certificate parsers, MAC/pad-failure, extensions, etc.

WIP Evaluation and Results

New bug class DY cannot express

- HAVOC+DY rediscovered **Heartbleed** in minutes. Pure DY never reaches it. Lying length → root
- **New security bug** 🐛: WolfSSL suffers from a certificate malleability / agreement violation (under responsible disclosure). Found in ~15 min. Impacts consumer like ircd-hybrid
Requires: (i) HAVOC under encryption, (ii) lying about length in cert. but NOT in cert. wrapper, **and** (iii) DY Objective Oracle capabilities (checking agreement properties)



Coverage

- DY-only (baseline) < bit-only on terms (+10%) < DY+bit (+20-22%).
- Gains land in byte-sensitive code: cipher/sig-alg value-space, certificate parsers, MAC/pad-failure, extensions, etc.
- HAVOC only right under opaque (which would be much easier): −15.5%

WIP Evaluation and Results

New bug class DY cannot express

- HAVOC+DY rediscovered **Heartbleed** in minutes. Pure DY never reaches it. Lying length → root
- **New security bug** 🐛: WolfSSL suffers from a certificate malleability / agreement violation (under responsible disclosure). Found in ~15 min. Impacts consumer like ircd-hybrid
Requires: (i) HAVOC under encryption, (ii) lying about length in cert. but NOT in cert. wrapper, **and** (iii) DY Objective Oracle capabilities (checking agreement properties) 

Coverage

- DY-only (baseline) < bit-only on terms (+10%) < DY+bit (+20-22%).
- Gains land in byte-sensitive code: cipher/sig-alg value-space, certificate parsers, MAC/pad-failure, extensions, etc.
- HAVOC only right under opaque (which would be much easier): −15.5%

Need further evaluations

- Time-to-find comparison with a larger benchmark of CVEs
- Compare all sort of combinations: presence DYonly/Bitonly/DY+Bit), schedule (DYthenBit/Interleave), locus (BitRoot/BitLeaf/BitOpaque/BitAny), etc.
- Evidence-based fine-tuning all parameters (mutations weight/proba, etc.)

Conclusion

Conclusion

- Reunifying bit-level and structured fuzzing is challenging
- An interesting instantiation of this is HAVOC+DY fuzzing
- We propose an approach that seems to work nicely but needs further evaluation

Conclusion

- Reunifying bit-level and structured fuzzing is challenging
- An interesting instantiation of this is HAVOC+DY fuzzing
- We propose an approach that seems to work nicely but needs further evaluation

Other directions we are pursuing on puffin

- Dolev-Yao / Structured feedback to fight code base coverage plateau (bad metric)
- Robust test-bed and evaluation framework for puffin: puffin-bench (eval. and CI)
- (Semi-)automated mapper to ease integration of new protocols
- More protocols (more YYYYpuffin) and PUTs (soon: SSHpuffin, OPCUApuffin, SPpuffin)
- Connect further with DY verifiers (ProVerif, Tamarin) 