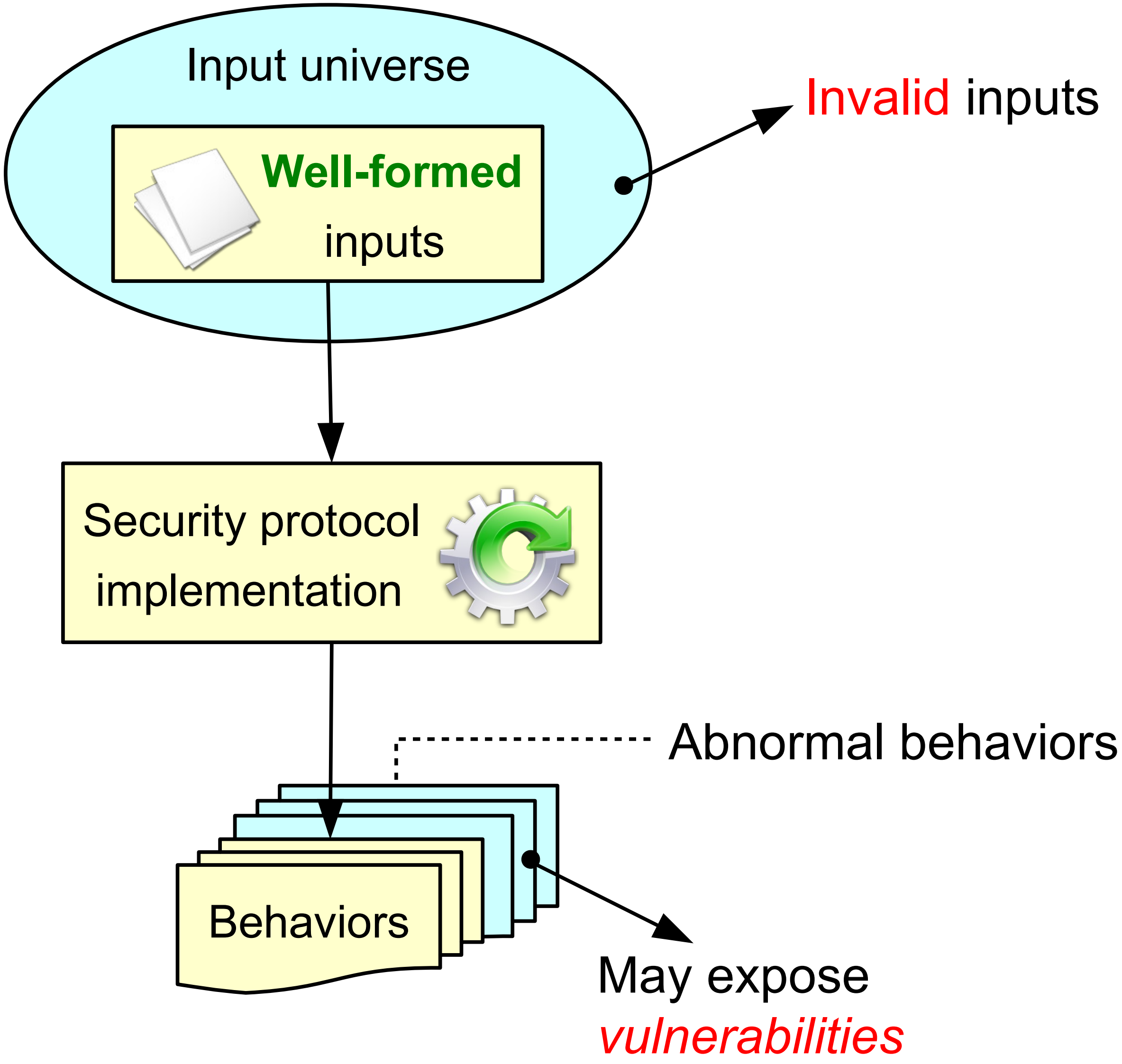
SECFUZZ: Fuzz-testing Security Protocols

Petar Tsankov, Mohammad Torabi Dashti, David Basin
ETH Zurich

Motivation



Input universe

Well-formed inputs

Invalid inputs

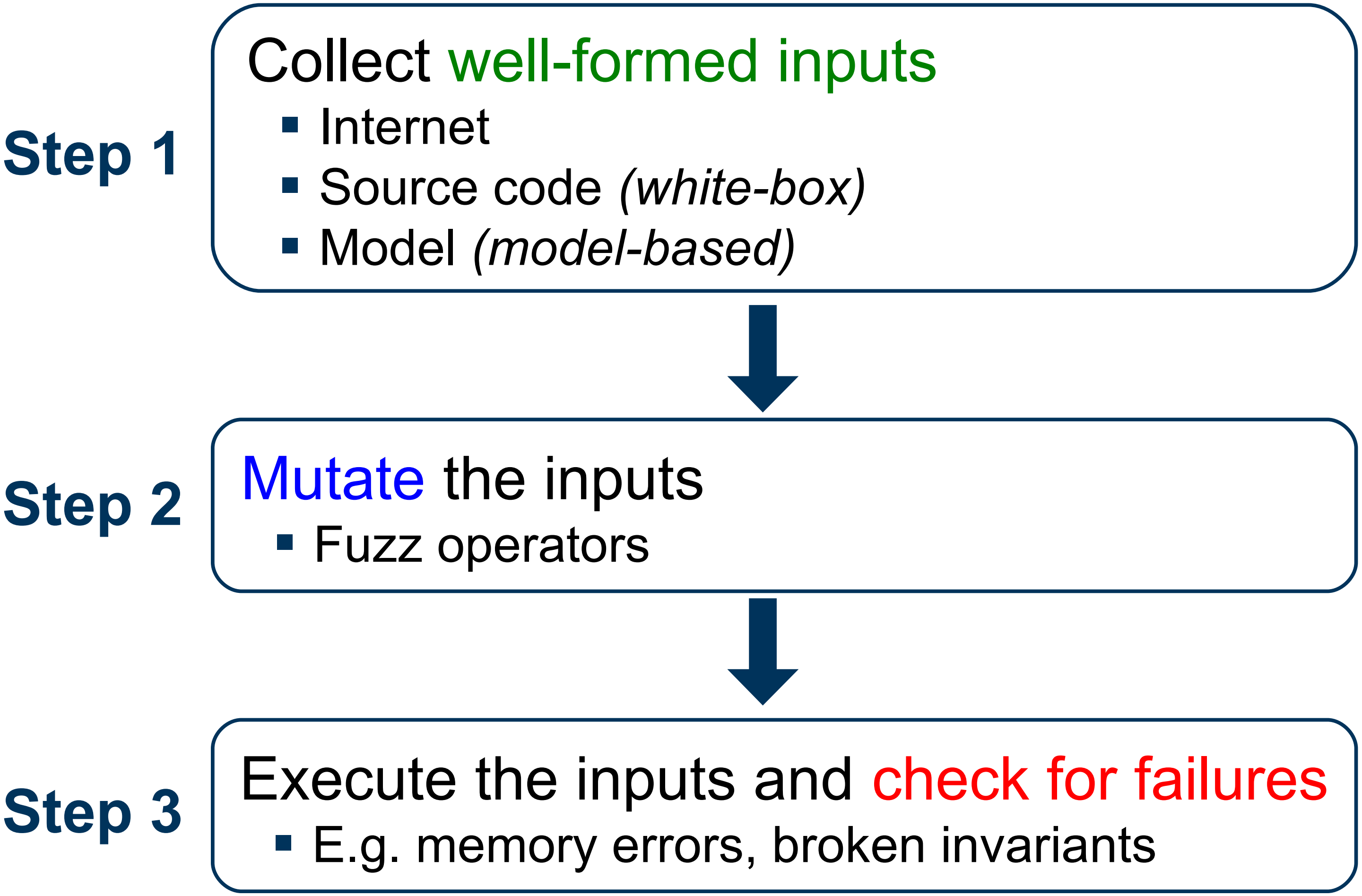
Security protocol implementation

Abnormal behaviors

Behaviors

May expose *vulnerabilities*

Fuzz-testing Security Protocols



Step 1

Collect well-formed inputs

- Internet
- Source code *(white-box)*
- Model *(model-based)*

Step 2

Mutate the inputs

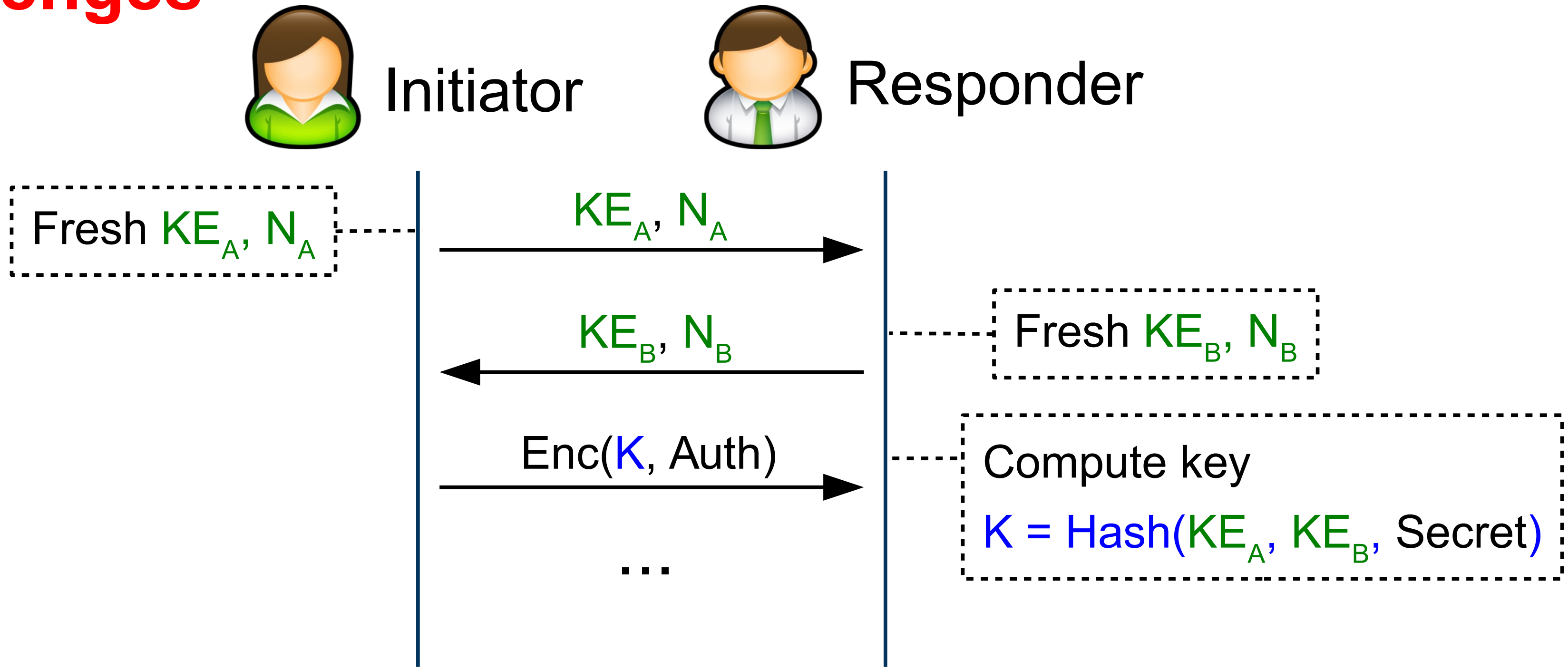
- Fuzz operators

Step 3

Execute the inputs and check for failures

- E.g. memory errors, broken invariants

Challenges



Initiator

Responder

Fresh KE_A, N_A

KE_A, N_A

KE_B, N_B

Fresh KE_B, N_B

Enc(K, Auth)

Compute key

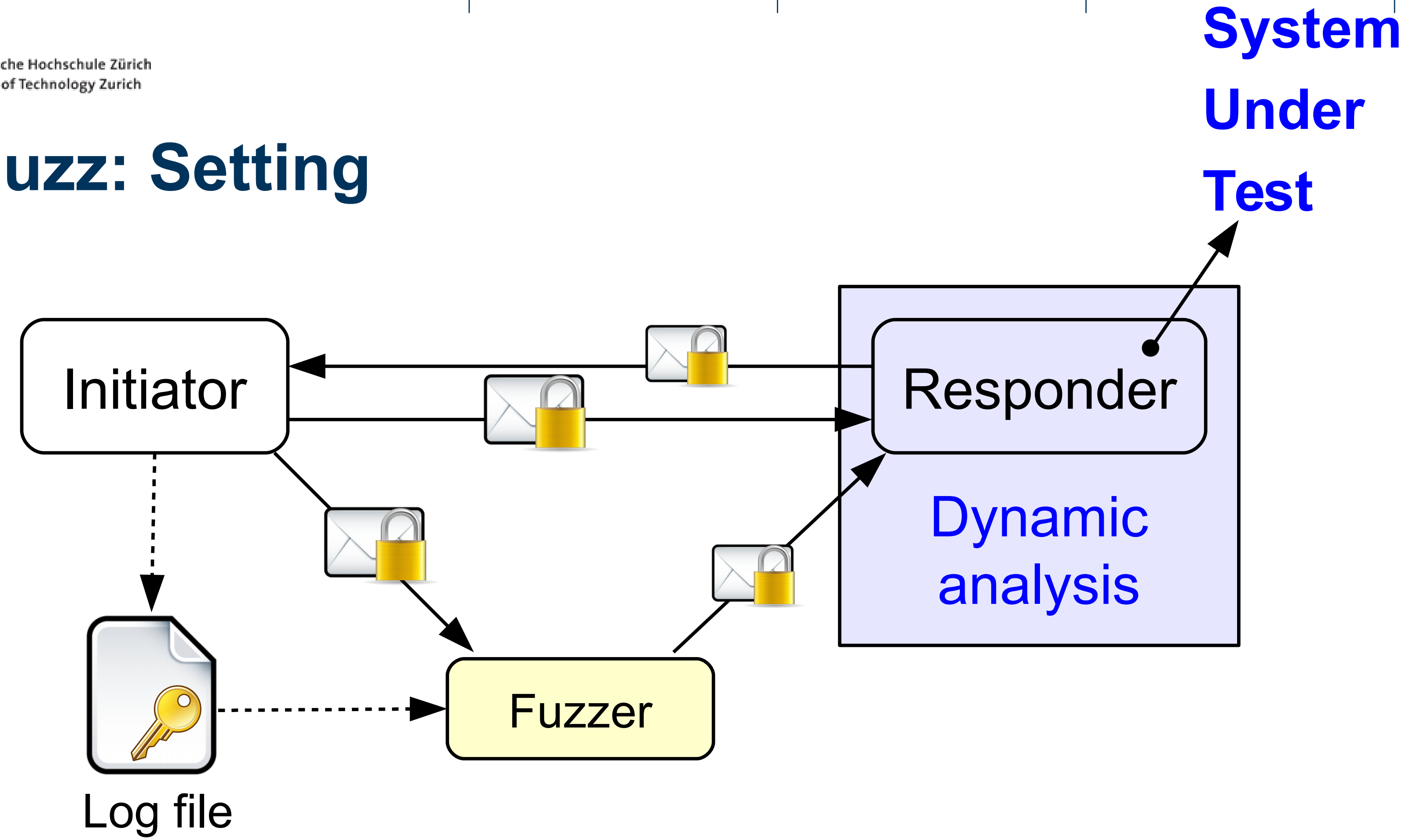
$K = Hash(KE_A, KE_B, Secret)$

…

Challenges:

- *Encrypted messages*
- *Security protocols are stateful*
- *Messages are non-replayable*

SecFuzz: Setting



Key advantages:

- *Light-weight and modular approach*
- *Fresh messages*
- *Fuzzer can decrypt messages*

Input Mutation

A fuzz operator:

- Mutates a well-formed input.
- The mutated input is *likely* to expose vulnerabilities.

The fuzz operators should produce mutated inputs that expose common programming mistakes.

Input Structure

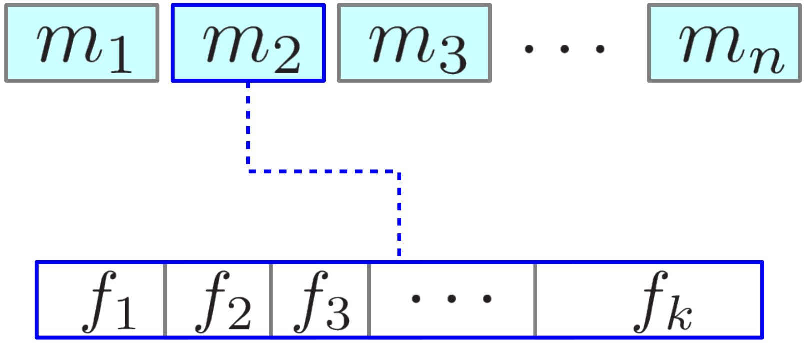
An input i consists of:

- a sequence of messages

$$i = m_1 \cdot m_2 \cdots m_n$$

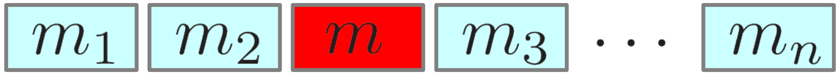
- a message consists of fields

$$m_k = f_1 \cdot f_2 \cdots f_k$$

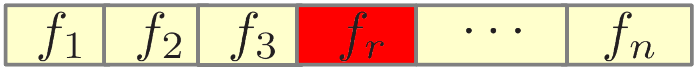


Fuzz operators

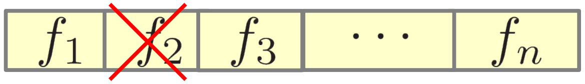
- Message fuzz operators
  - Insert random (well-formed) message



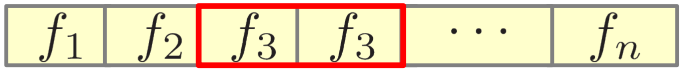
- Field fuzz operator
  - Insert random field



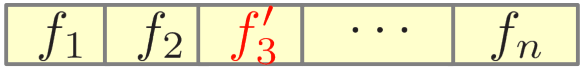
  - Remove field



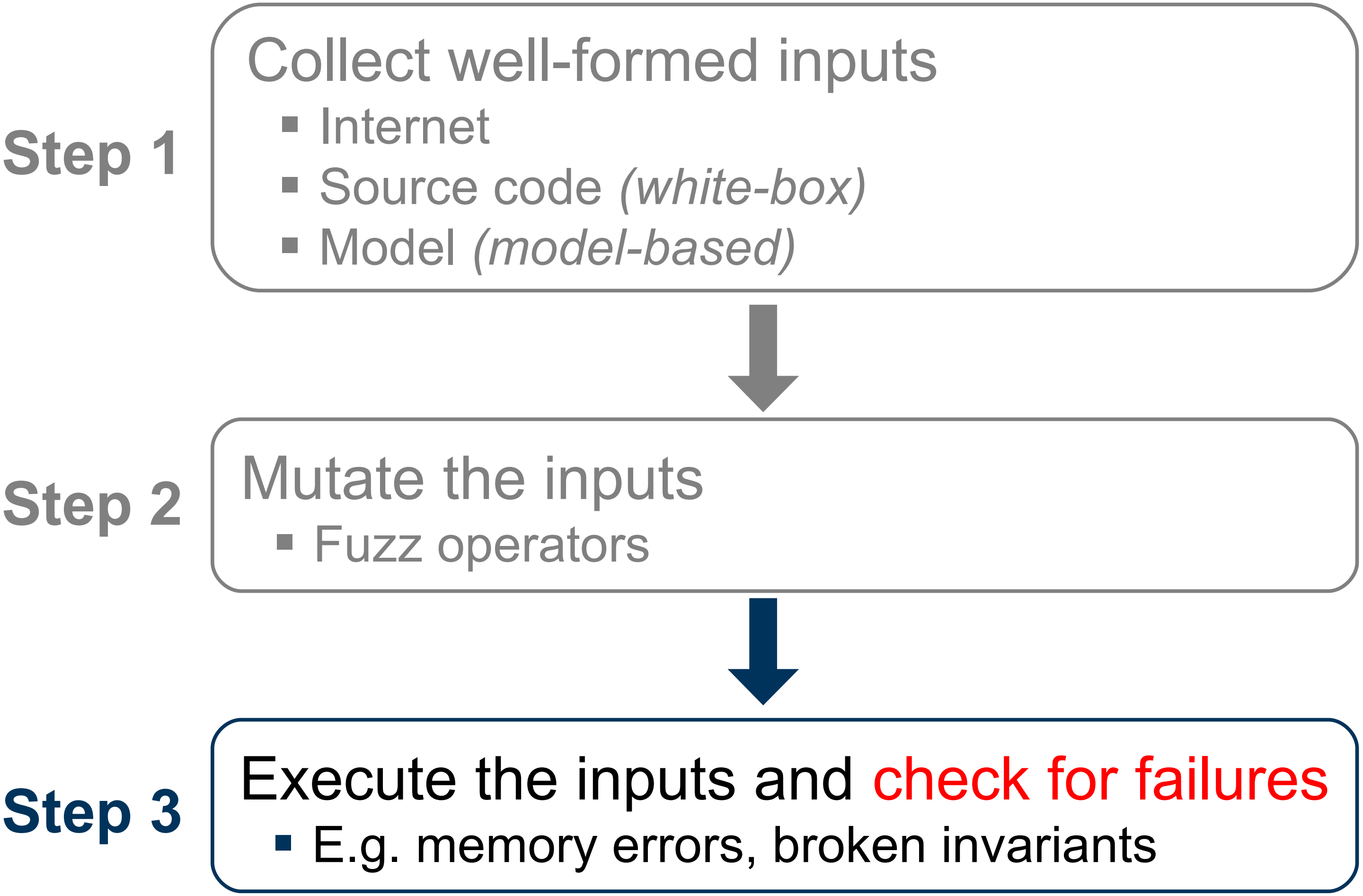
  - Duplicate field



  - Modify field



Fuzz-testing Security Protocols



Step 1

Collect well-formed inputs

- Internet
- Source code *(white-box)*
- Model *(model-based)*

Step 2

Mutate the inputs

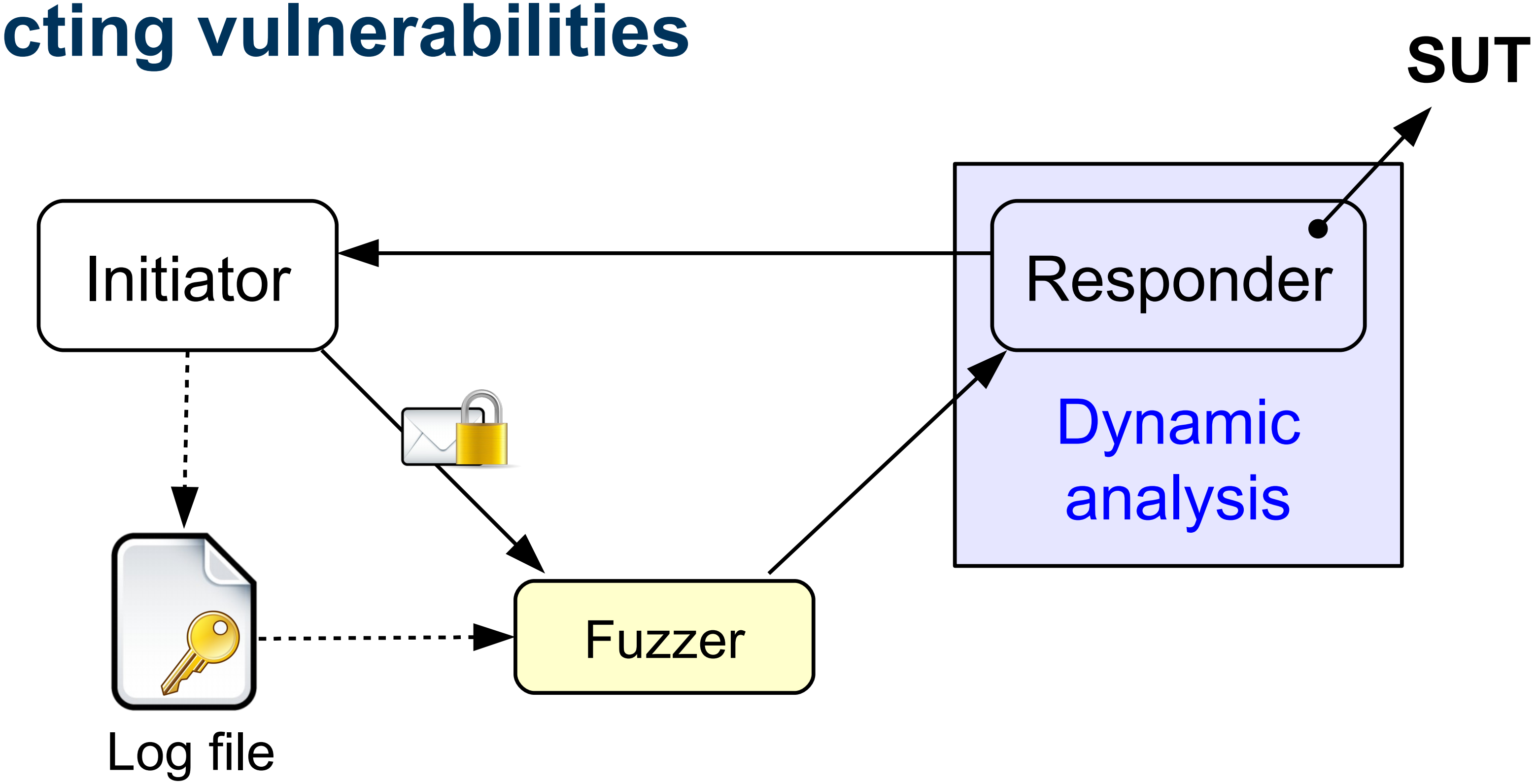
- Fuzz operators

Step 3

Execute the inputs and check for failures

- E.g. memory errors, broken invariants

Detecting vulnerabilities



- The dynamic analysis monitors the SUT and reports failures.
- Memory errors are a common source of vulnerabilities:
  - Tools: Valgrind's Memcheck, IBM's Purify

Internet Key Exchange Case Study

Experiment 1

Test subject: OpenSwan v2.6.35

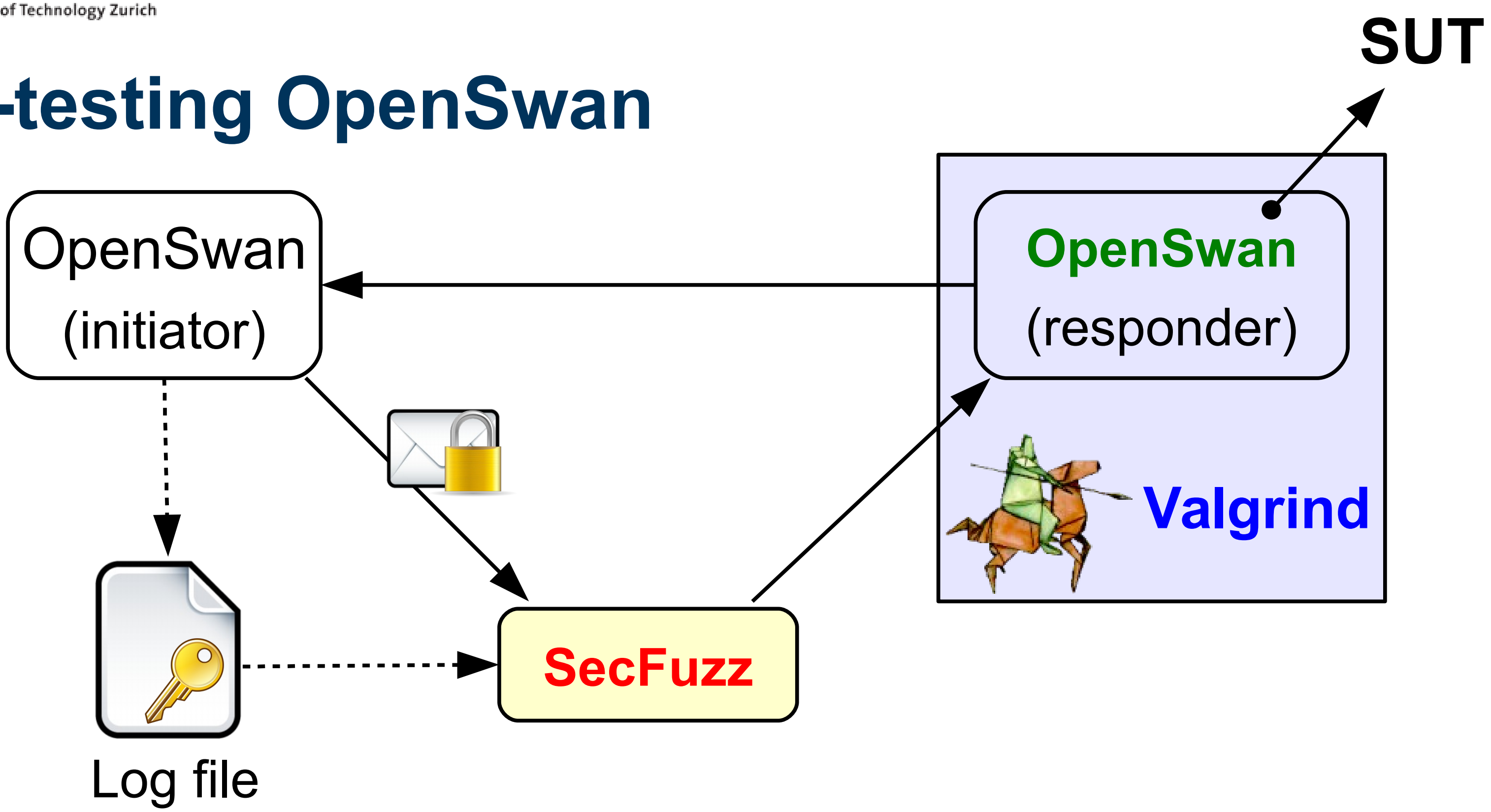
Results: Discovered a previously unknown use-after-free vulnerability.

Experiment 2

Test subject: ShrewSoft's VPN Client for Windows v2.1.7

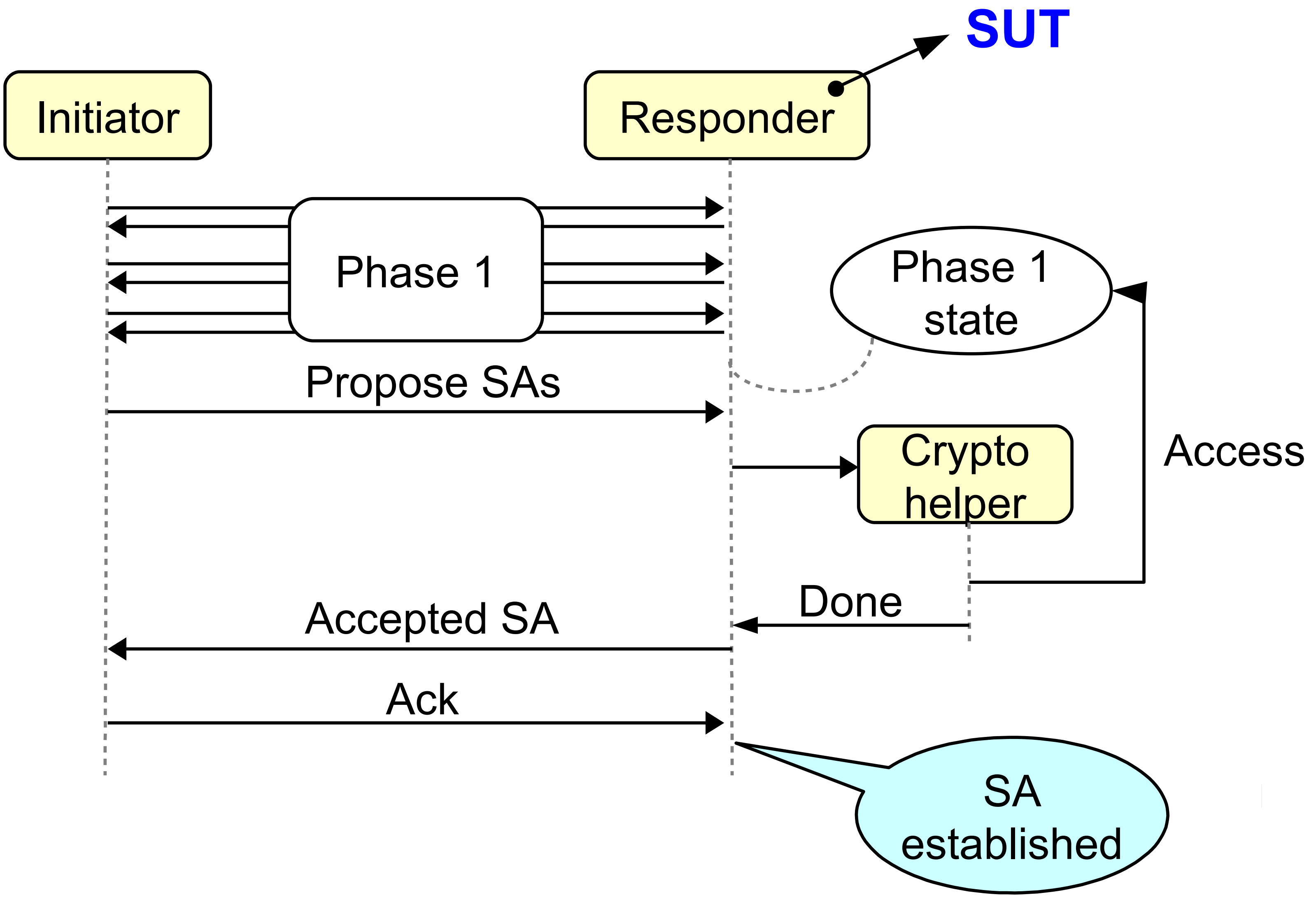
Results: Discovered a previously unknown unhandled exception vulnerability.

Fuzz-testing OpenSwan

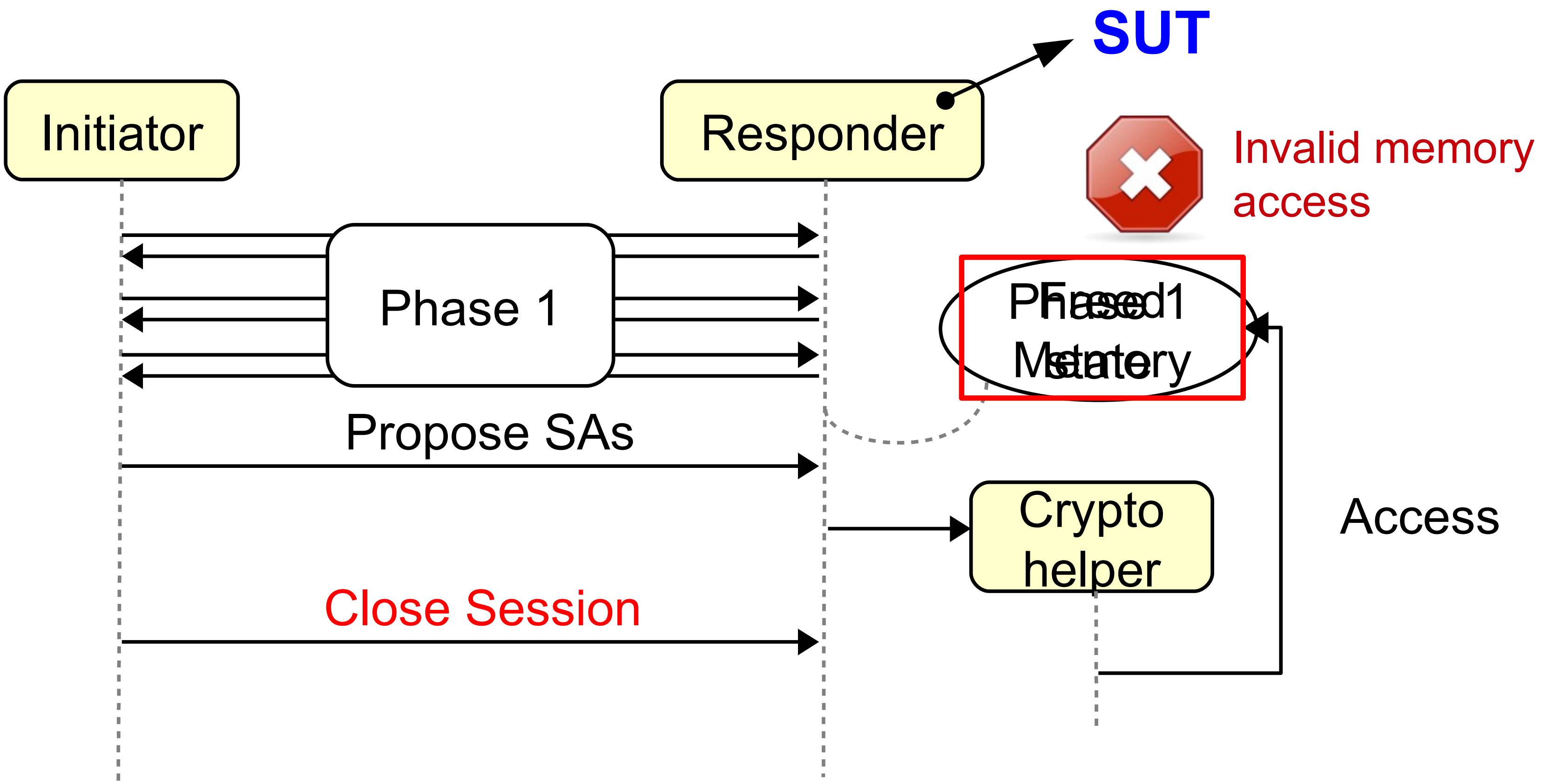


- SUT: **OpenSwan** v2.6.35
  - A popular IPSec implementation for Linux.
- Dynamic analysis: **Valgrind's Memcheck**
  - Detects different types of memory access errors.
- Fuzzer: **SecFuzz**, implemented using Python / Scapy.

OpenSwan: IKE Implementation details

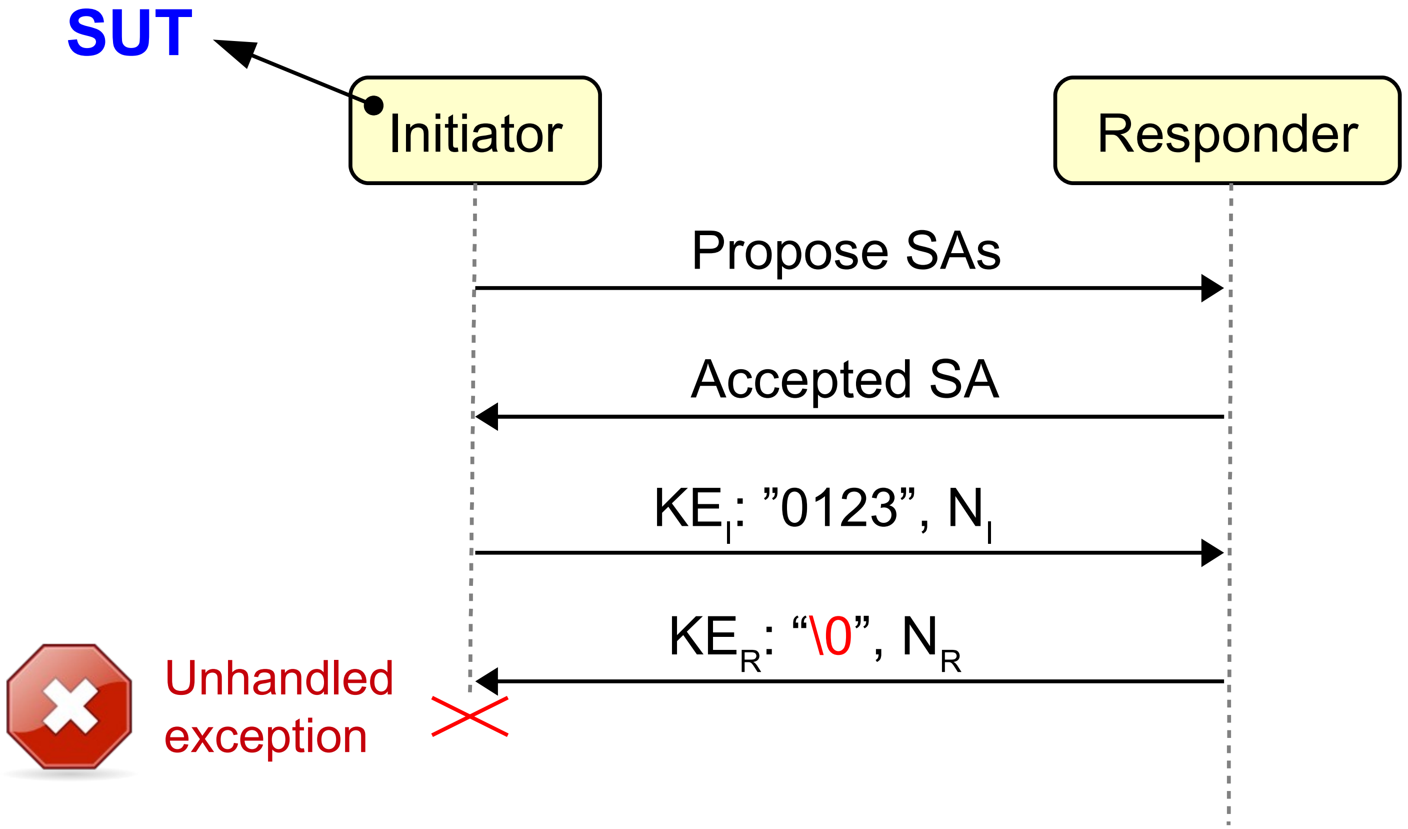


OpenSwan: Use-after-free Vulnerability



The vulnerability was reported and a security patch was released in CVE-2011-4073.

ShrewSoft's VPN Client: Unhandled Exception



The vulnerability details will appear in CVE-2012-0784.