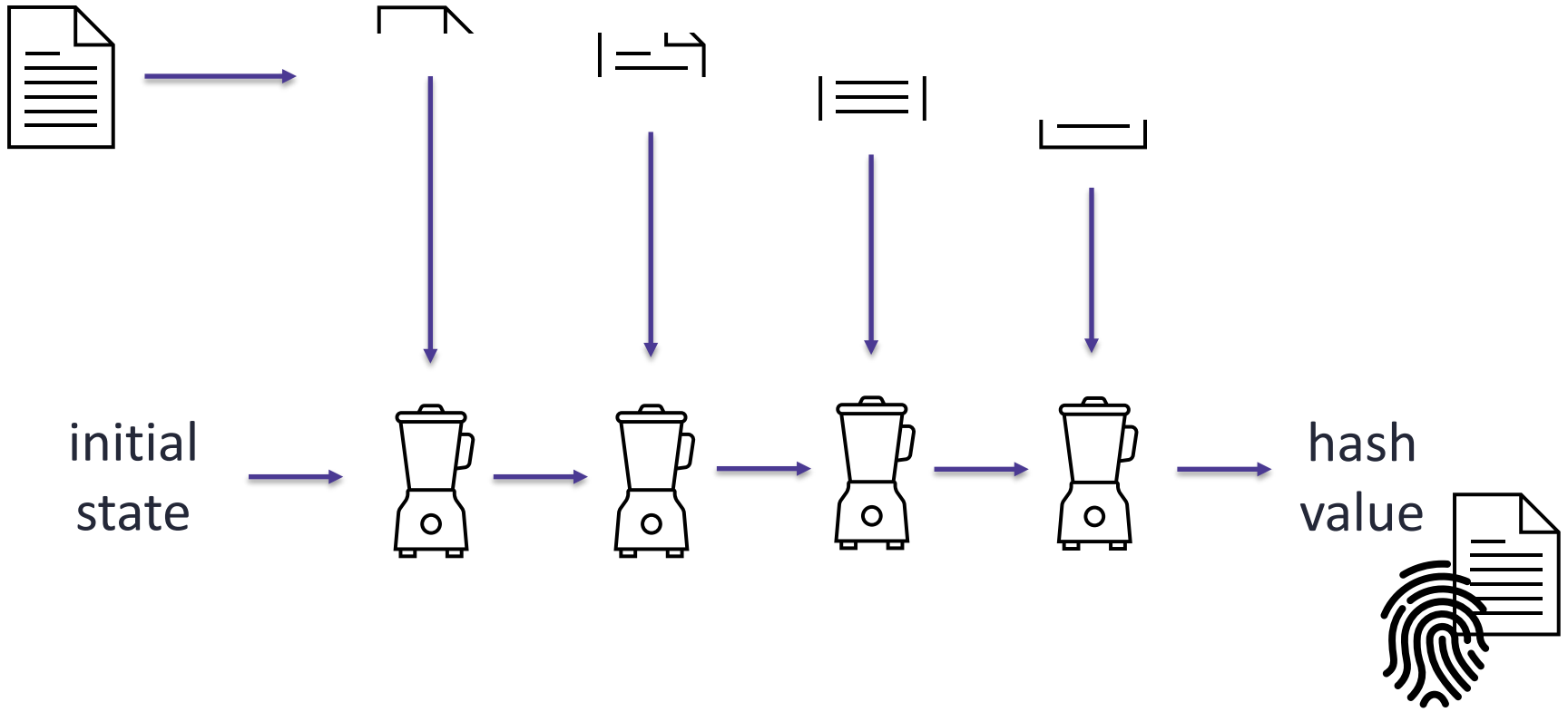
# How the SHA-3 Buffer Overflow Was Found using KLEE

**Nicky Mouha**

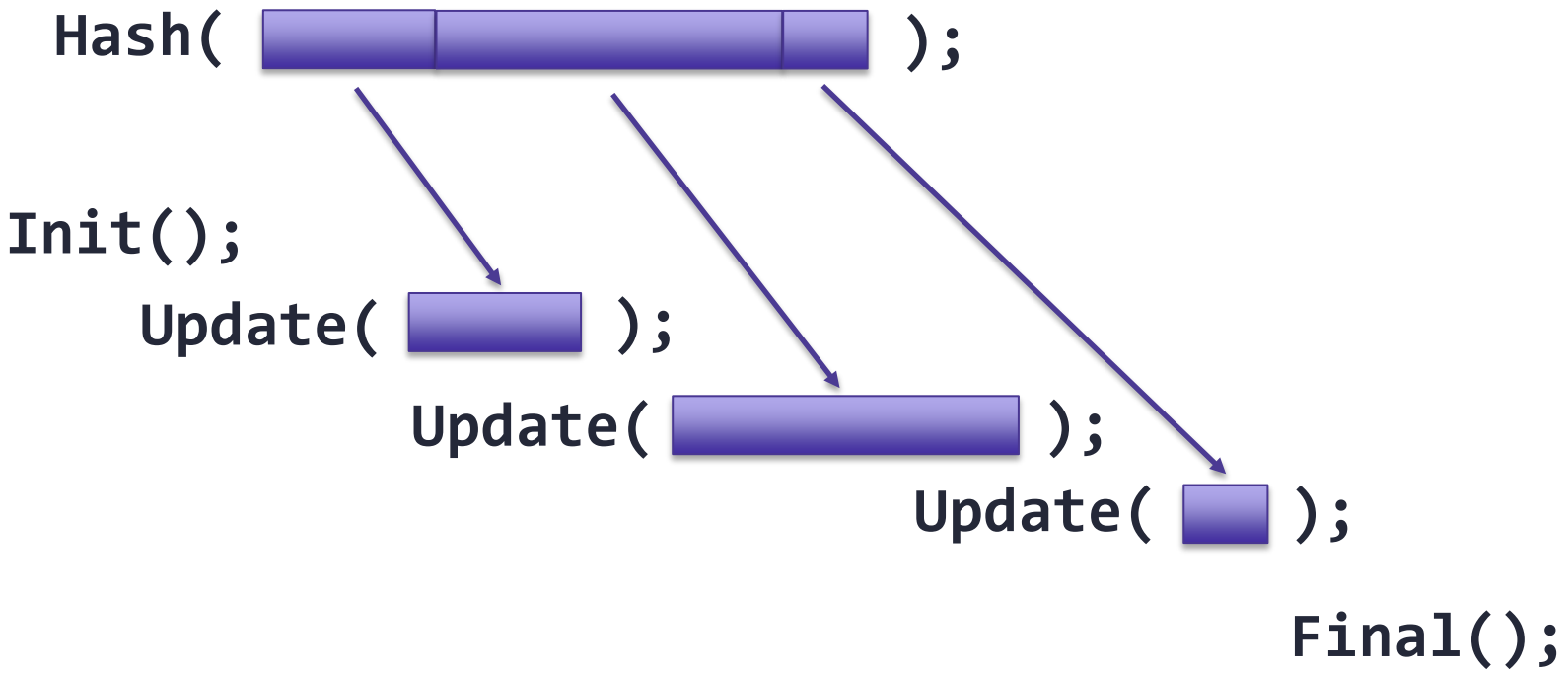
KLEE Workshop 2024

April 16, 2024

# Iterated Hash Functions



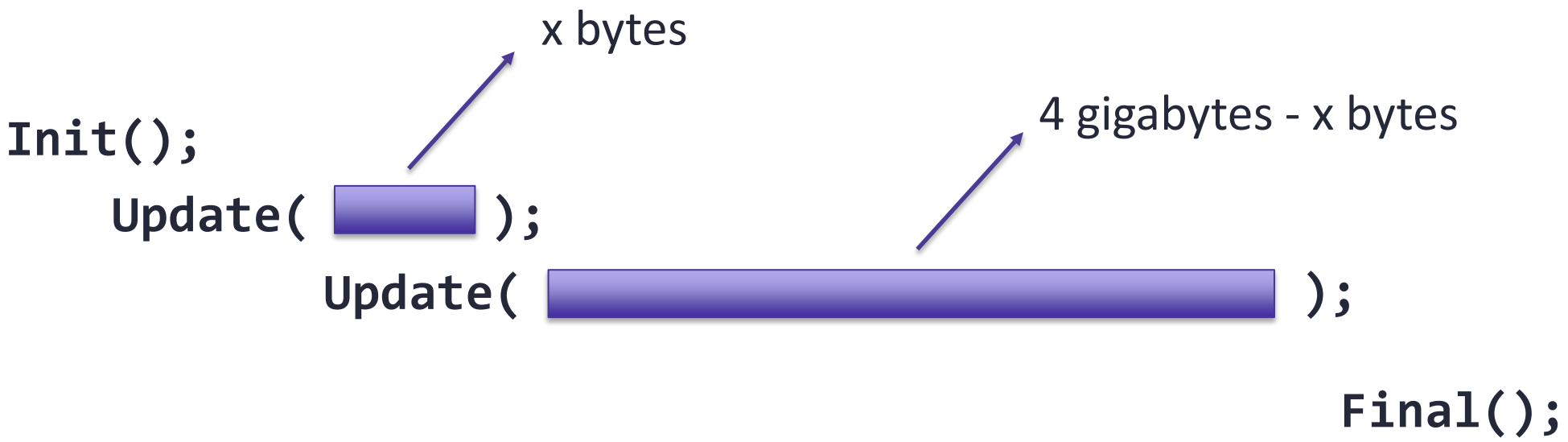
# Two Common Hash Function Interfaces




- Q: Where/when are they used?

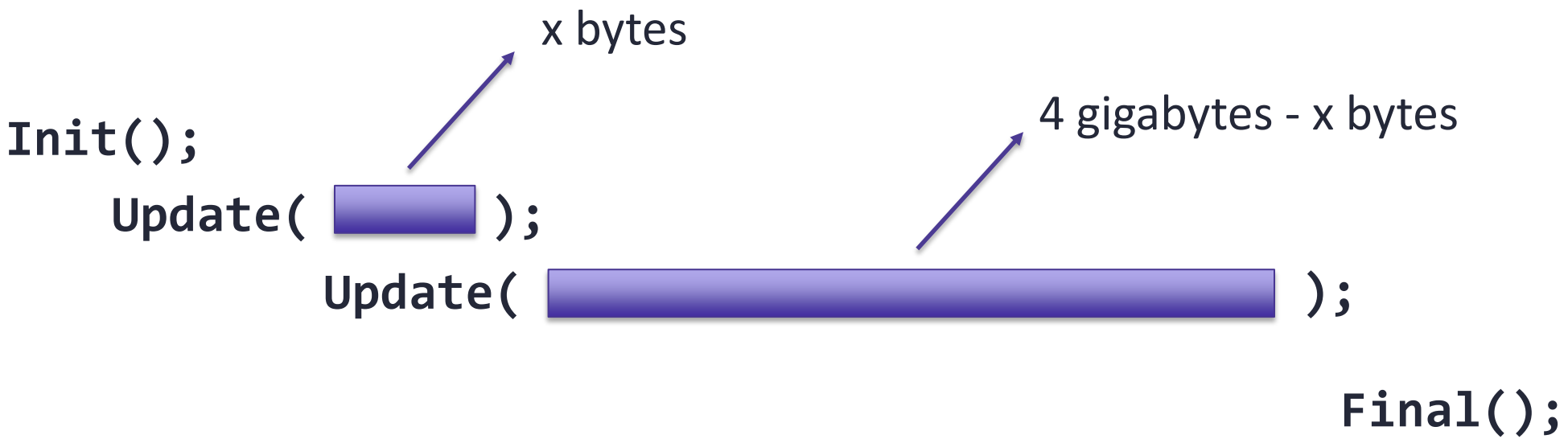
# SHA-3 Bug (CT-RSA 2023)

- Appeared in 2011 (final-round Keccak submission)
- CVE-2022-37454, NVD: 9.8 CRITICAL



# SHA-3 Bug (CT-RSA 2023)

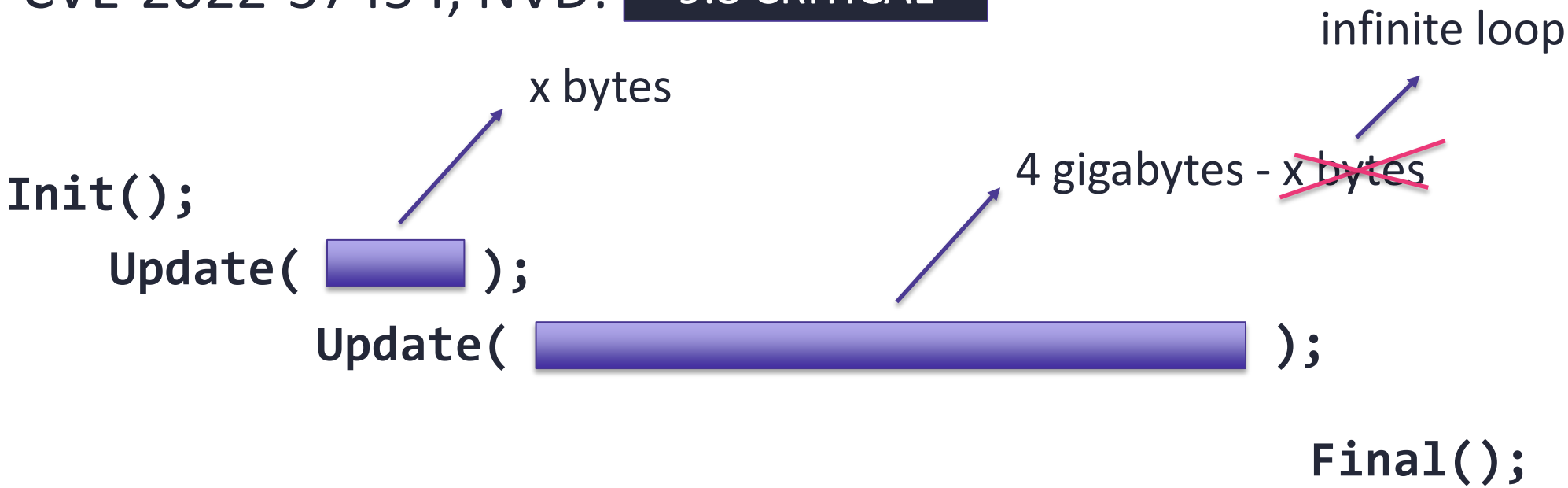
- Appeared in 2011 (final-round Keccak submission)
- CVE-2022-37454, NVD: 9.8 CRITICAL




buffer overflow!
(arbitrary code execution)

# SHA-3 Bug (CT-RSA 2023)

- Appeared in 2011 (final-round Keccak submission)
- CVE-2022-37454, NVD: 9.8 CRITICAL



# Proof of Concept

```python
#!/usr/bin/python
import hashlib
h = hashlib.sha3_224()
h.update(b"\x00" * 1)
h.update(b"\x00" * 4294967295)
print(h.hexdigest())
```

```php
<?php
$ctx = hash_init("sha3-224");
hash_update($ctx, str_repeat("\x00", 1));
hash_update($ctx, str_repeat("\x00", 4294967295));
echo hash_final($ctx);
?>
```

# KeccakSponge.inc

```
partialBlock = (unsigned int) (dataByteLen - i);
if (partialBlock + instance->byteIOIndex > rateInBytes) {
    partialBlock = rateInBytes - instance->byteIOIndex;
}
```

# KeccakSponge.inc

```
partialBlock = (unsigned int) (dataByteLen - i);
if (partialBlock > rateInBytes - instance->byteIOIndex) {
    partialBlock = rateInBytes - instance->byteIOIndex;
}
```

# KeccakSponge.inc

```
partialBlock = (unsigned int) (dataByteLen - i);
if (dataByteLen - i > rateInBytes - instance->byteIOIndex) {
    partialBlock = rateInBytes - instance->byteIOIndex;
}
```

# KeccakSponge.inc

```
if (dataByteLen - i > rateInBytes - instance->byteIOIndex) {
    partialBlock = rateInBytes - instance->byteIOIndex;
} else {
    partialBlock = (unsigned int) (dataByteLen - i);
}
```

# KLEE: Approach (ACISP 2023)

```
Update(&state,  NULL, databitlen);
Update(&state2, NULL, databitlen1);
Update(&state2, NULL, databitlen2);

if (state.bitsInQueue != state2.bitsInQueue)
  klee_assert(0);
```

# KLEE: Approach (ACISP 2023)

```
Update(&state,  NULL, databitlen);
Update(&state2, NULL, databitlen1);
Update(&state2, NULL, databitlen2);

if (state.bitsInQueue != state2.bitsInQueue)
  klee_assert(0);
```

- All variables are symbolic
- Message input is not used (NULL)
- `Update()` contains large loop ($\approx 2^{64}$/block_size iterations)
- Infinite loop in implementation → infinite loop in KLEE

# KLEE: Runtimes

| Implementation | Buggy | Correct |
|---|---|---|
| BLAKE | 5 s | 11 m 59 s |
| Grøstl | 6 s | 10 s |
| JH | — | 50 s |
| Keccak | 1 s | 39 s (*) |
| Skein | — | 34 s |

| | | |
|---|---|---|
| XKCP (SHA-3) | 1 s | 20 s (*) |
| CoreCrypto | 1 s | 2 m 41 s |

# KLEE: Asterisk on Previous Slide

| Algorithm | Block Size |
|---|---|
| SHA3-224 | 144 |
| SHA3-256 | 136 |
| SHA3-384 | 104 |
| SHA3-512 | 72 |
| SHAKE128 | 168 |
| SHAKE256 | 136 |

| | |
|---|---|
| Asterisk On Previous Slide | 128 |

# Conclusion

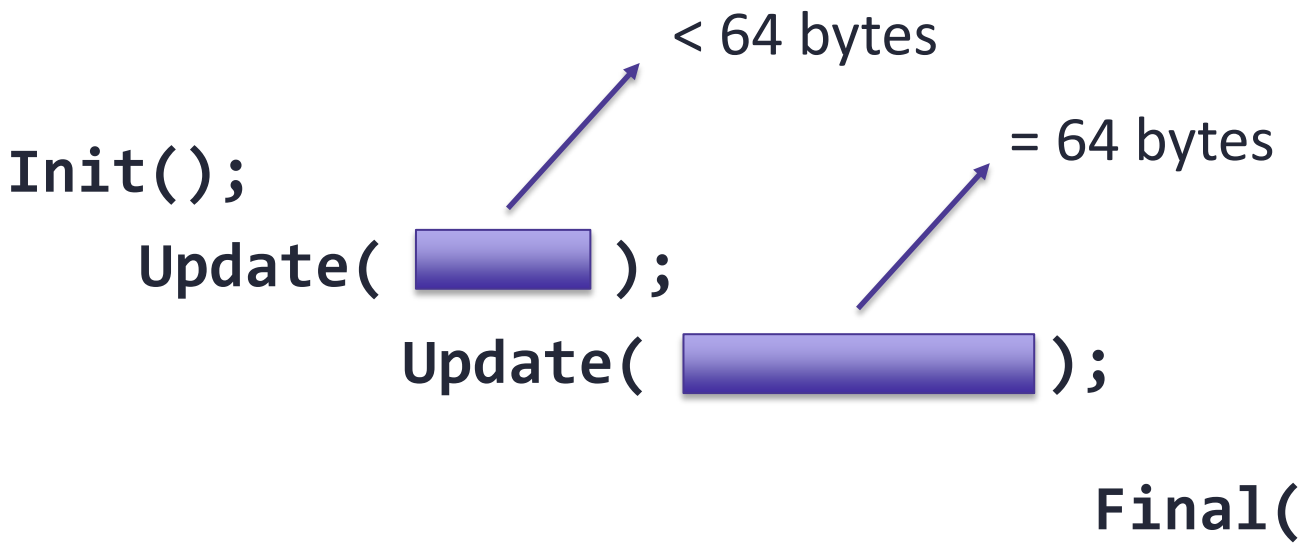
- KLEE approach:
  - Focus only on counter variables
  - All variables symbolic
  - Update($\ell_1$); Update($\ell_2$) vs Update($\ell_1 + \ell_2$)
- KLEE runtimes:
  - Few seconds to find bug
  - 1-10 minutes to prove correctness (block size power of two)

- Questions?

# Backup Slides

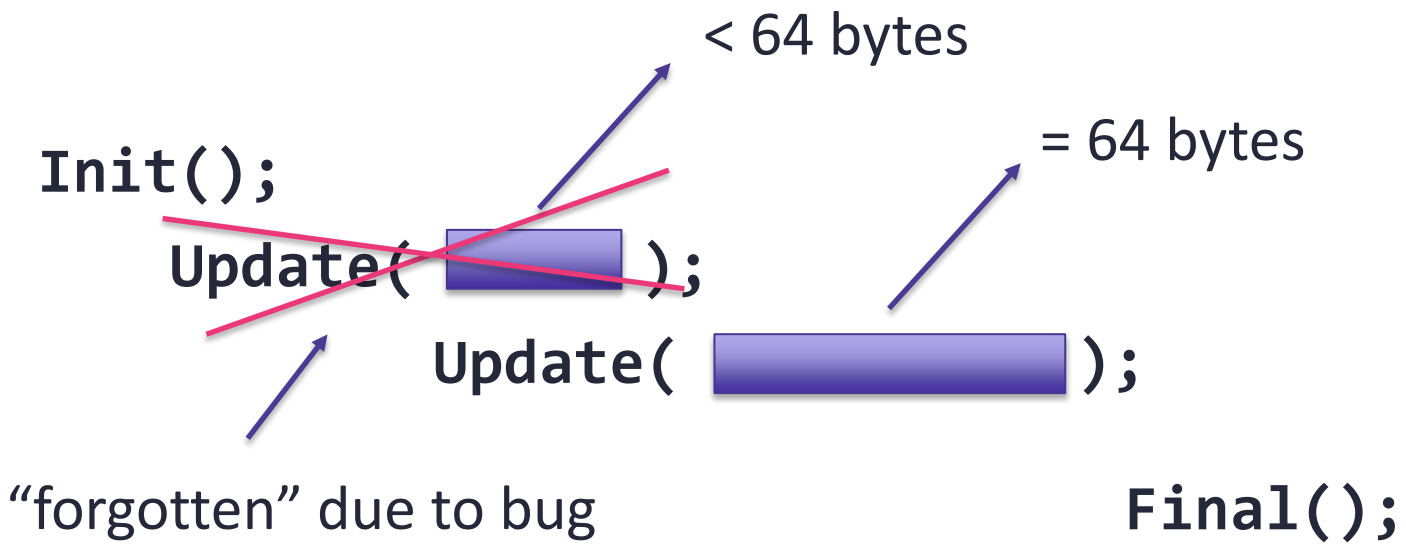
# BLAKE Bug

- Appeared in 2008, disclosed in 2015
- BLAKE-256: process message in blocks of 64 bytes



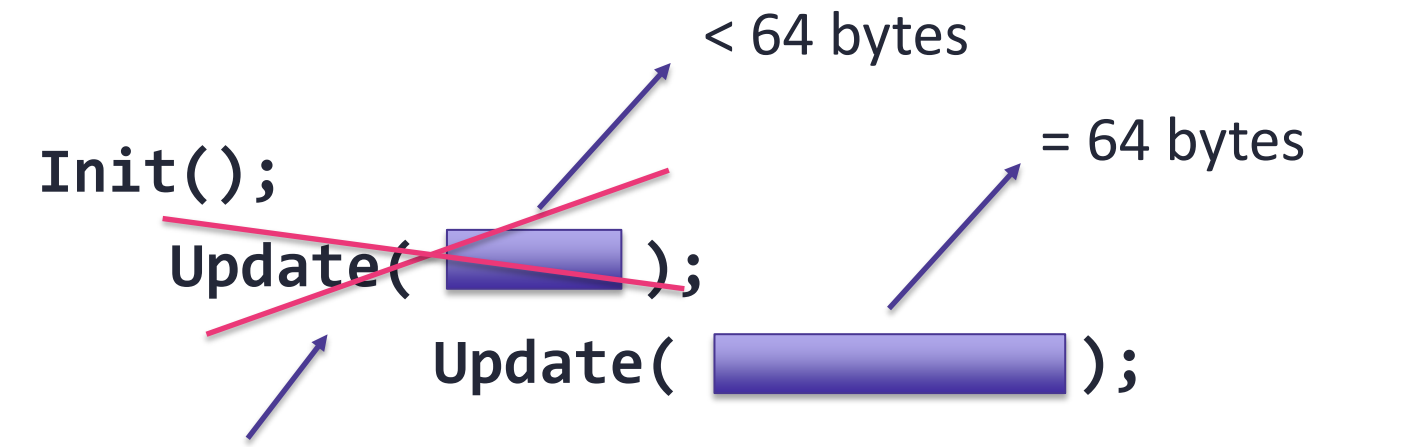
# BLAKE Bug

- Appeared in 2008, disclosed in 2015
- BLAKE-256: process message in blocks of 64 bytes



# BLAKE Bug

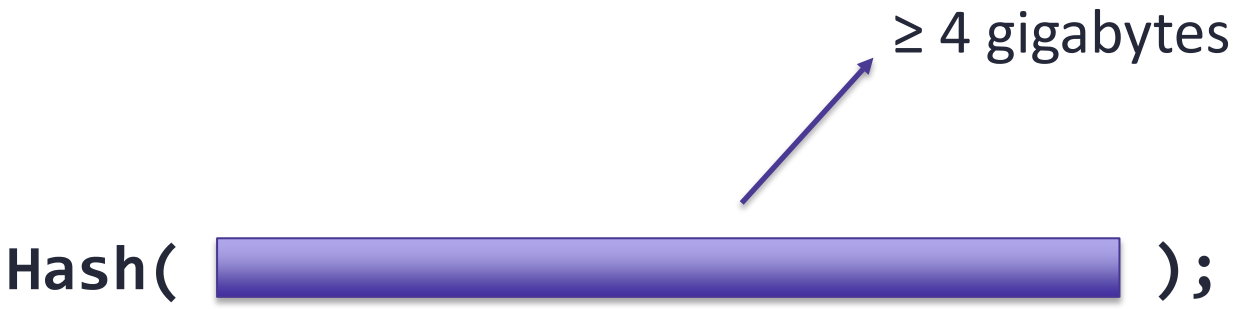
- Appeared in 2008, disclosed in 2015
- BLAKE-256: process message in blocks of 64 bytes




“second preimage”
(modified message, same hash)

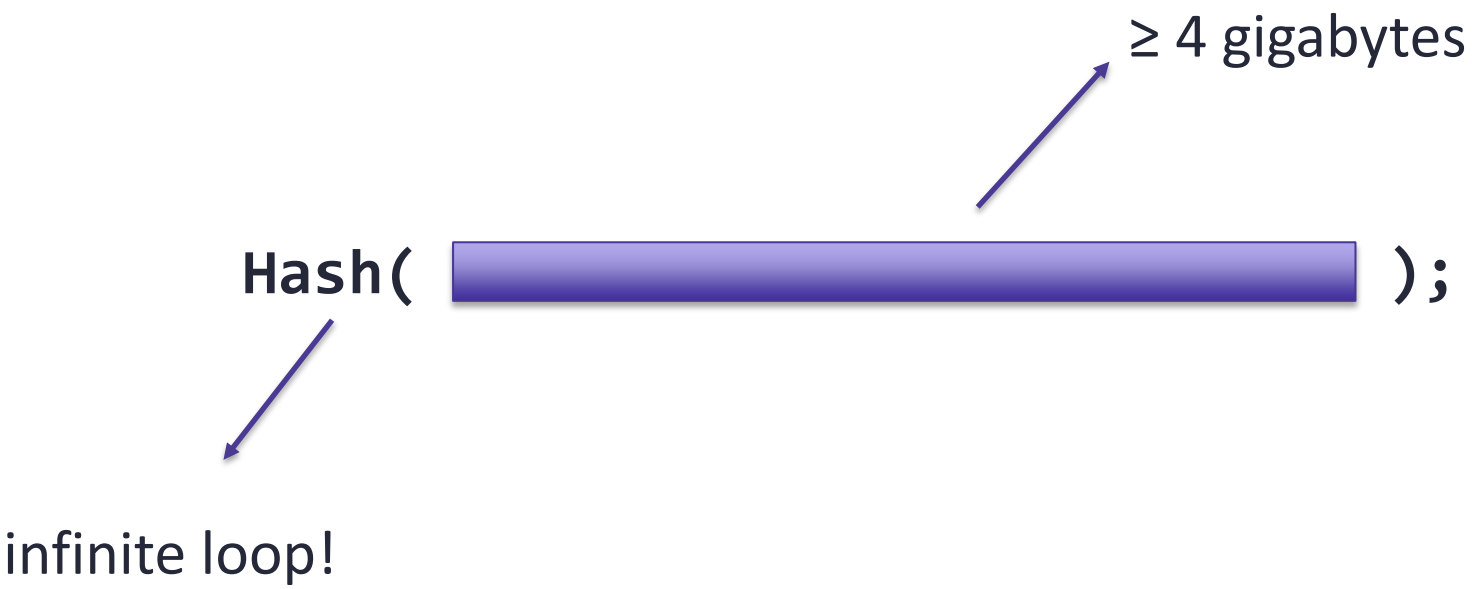
# Apple CoreCrypto Bug (CT-RSA 2020)

- Affected 11/12 hash functions
- CVE-2019-8741, NVD: 7.5 HIGH



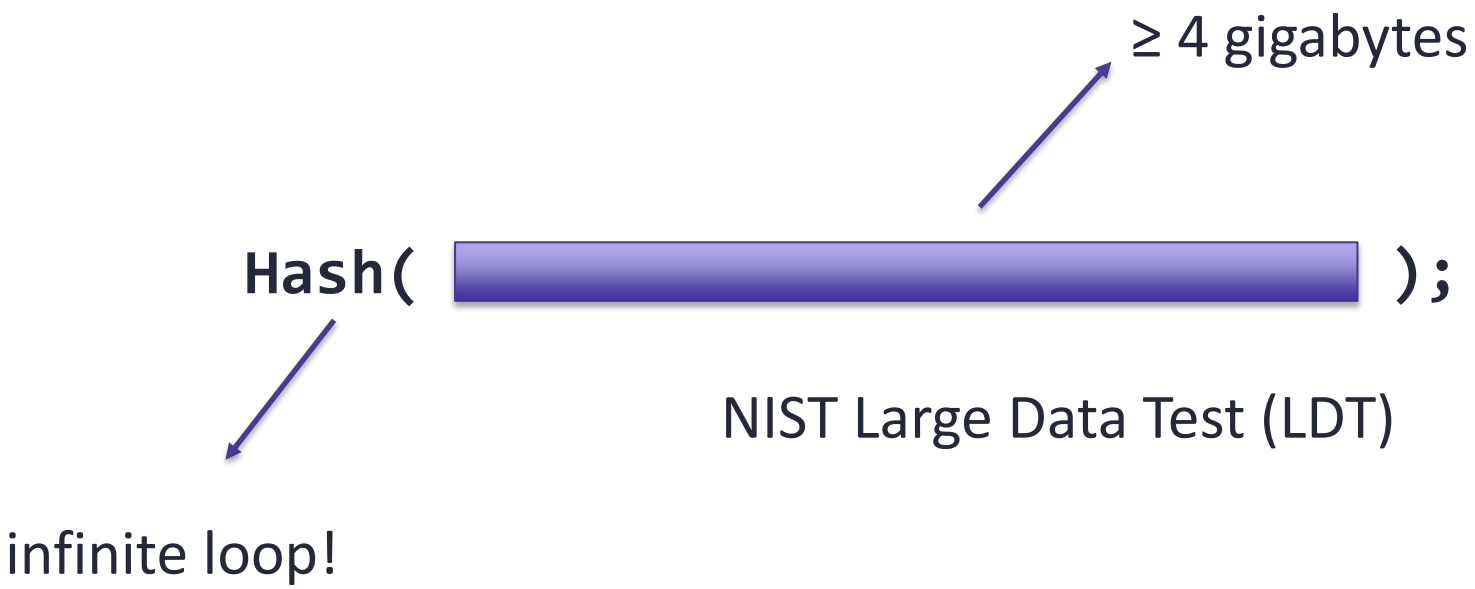
# Apple CoreCrypto Bug (CT-RSA 2020)

- Affected 11/12 hash functions
- CVE-2019-8741, NVD: 7.5 HIGH



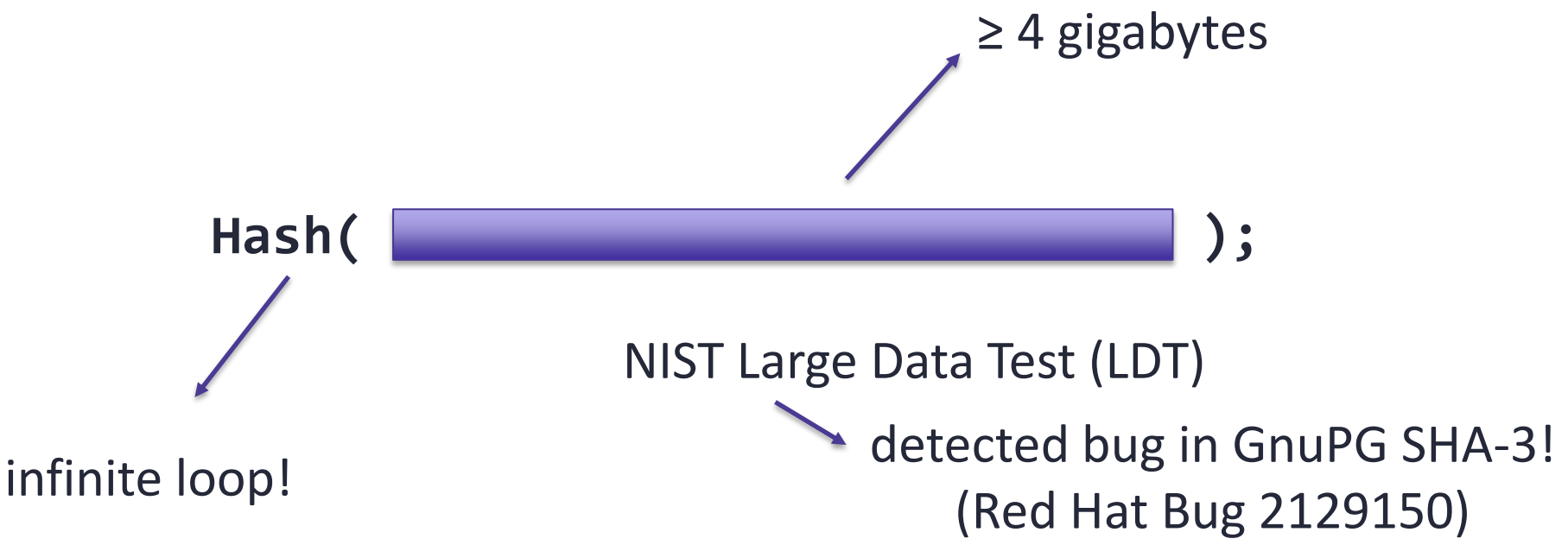
# Apple CoreCrypto Bug (CT-RSA 2020)

- Affected 11/12 hash functions
- CVE-2019-8741, NVD: 7.5 HIGH



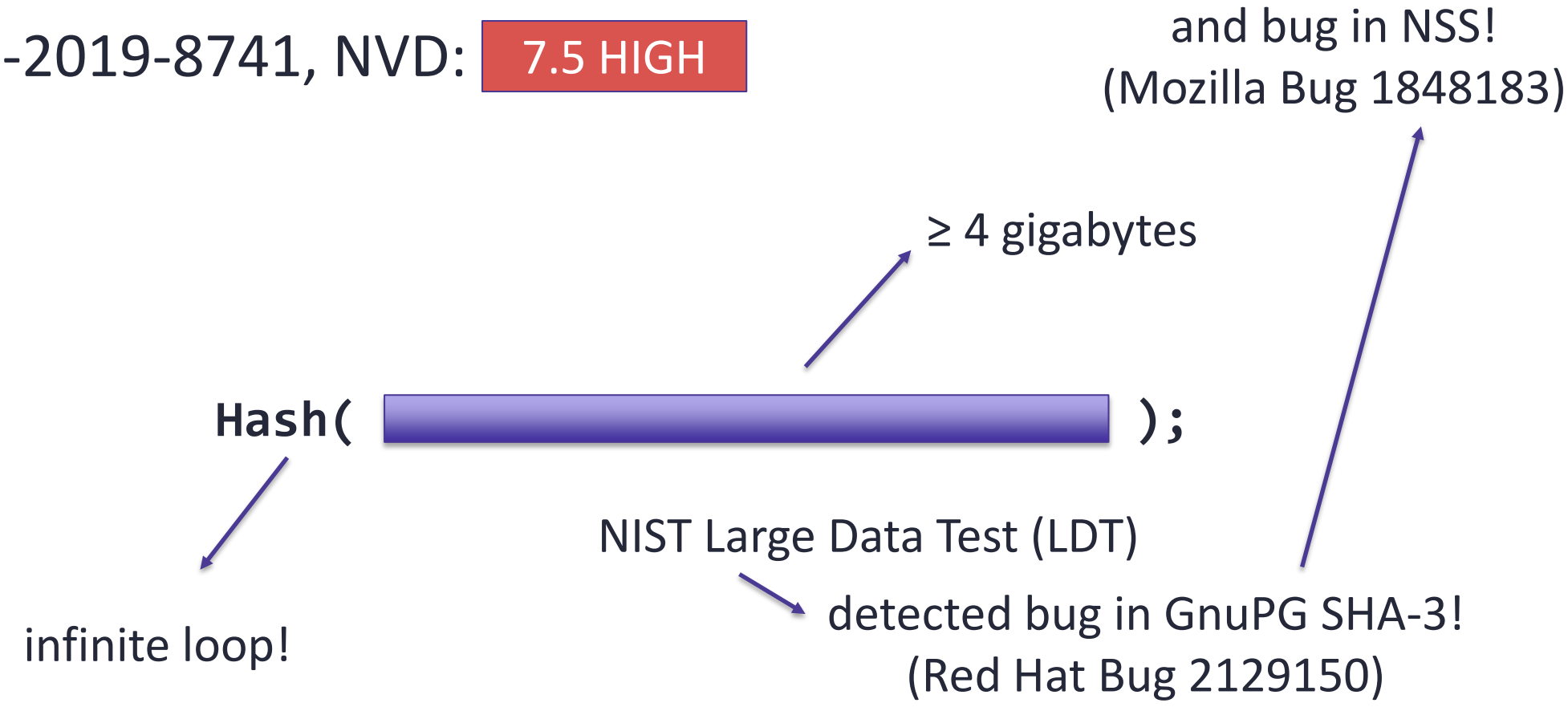
# Apple CoreCrypto Bug (CT-RSA 2020)

- Affected 11/12 hash functions
- CVE-2019-8741, NVD: 7.5 HIGH



# Apple CoreCrypto Bug (CT-RSA 2020)

- Affected 11/12 hash functions
- CVE-2019-8741, NVD: 7.5 HIGH



The LANE hash function

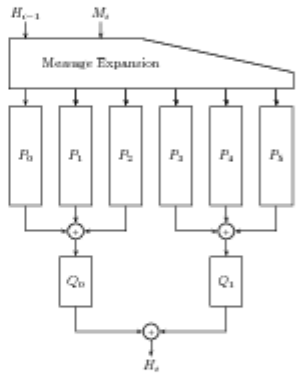
cosic.esat.kuleuven.be/lane/

# The LANE hash function



## What is LANE?

LANE is a cryptographic hash function that has been entered as a candidate in the NIST SHA-3 competition by the COSIC research group of the Katholieke Universiteit Leuven, Belgium. The aims of LANE are to be secure, easy to understand, elegant and flexible in implementation. It reuses components from the AES block cipher. LANE can take advantage of the parallelism offered by modern high-performance CPUs, but also scales down to embedded systems. Another advantages of LANE is the fact that its design is supported by a clear design rationale and a comprehensive security analysis.




*The LANE compression function*

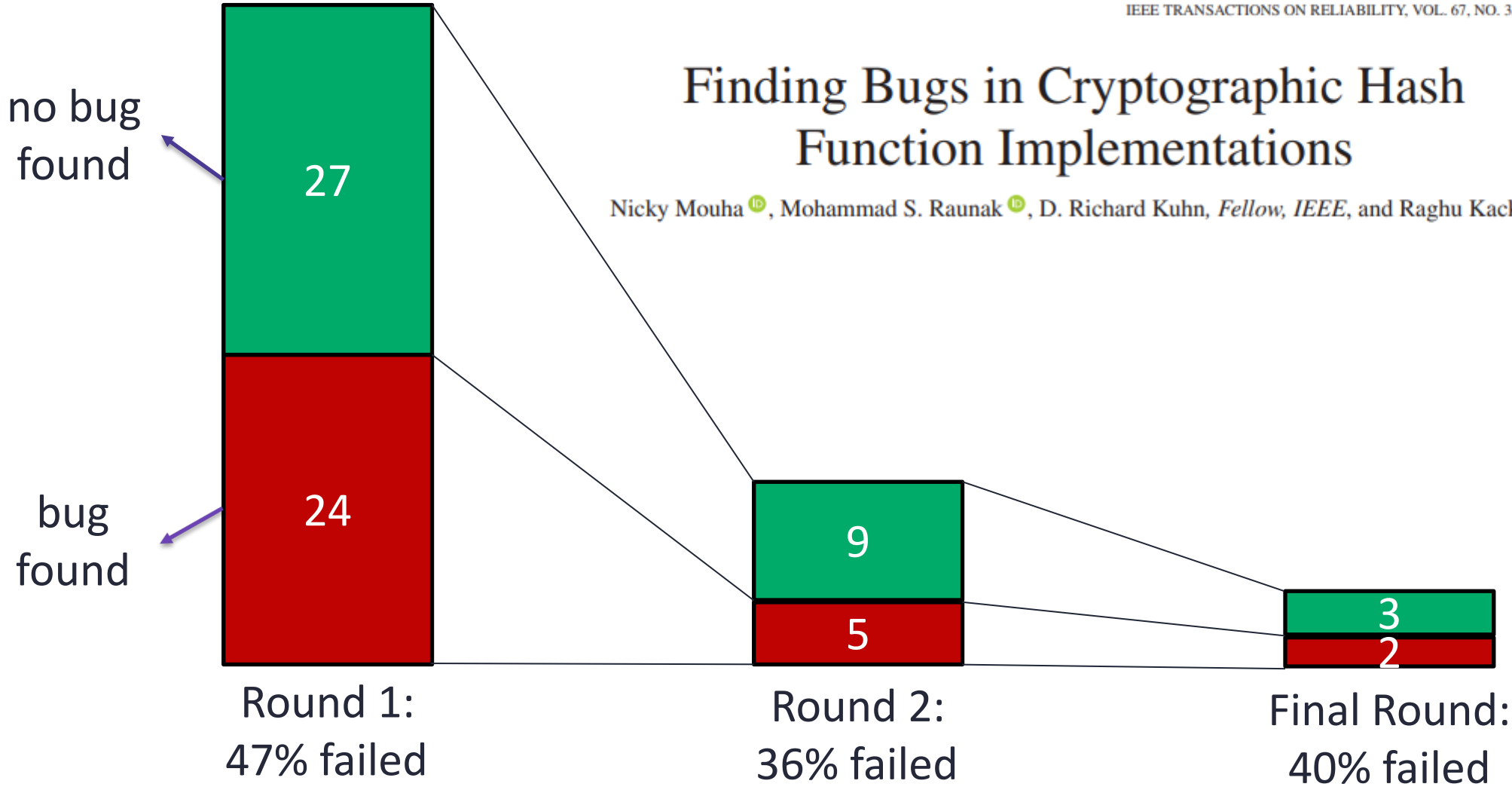
LANE was designed by Sebastiaan Indesteege. Important contributions to the design and the security analysis were made by Elena Andreeva, Christophe De Cannière, Orr Dunkelman, Emilia Käsper, Svetla Nikova, Bart Preneel and Elmar Tischhauser, all from the COSIC research group of the Katholieke Universiteit Leuven, Belgium.

More information on LANE is available here.

## The SHA-3 competition

The American National Institute of Standards and Technology (NIST) has initiated a public competition, the SHA-3 competition, to develop a new cryptographic hash algorithm. This is an algorithm that maps a message of variable length into a short, fixed-length digest, such that certain security properties, like collision resistance and preimage resistance, are achieved.

# SHA-3 Competition Bugs

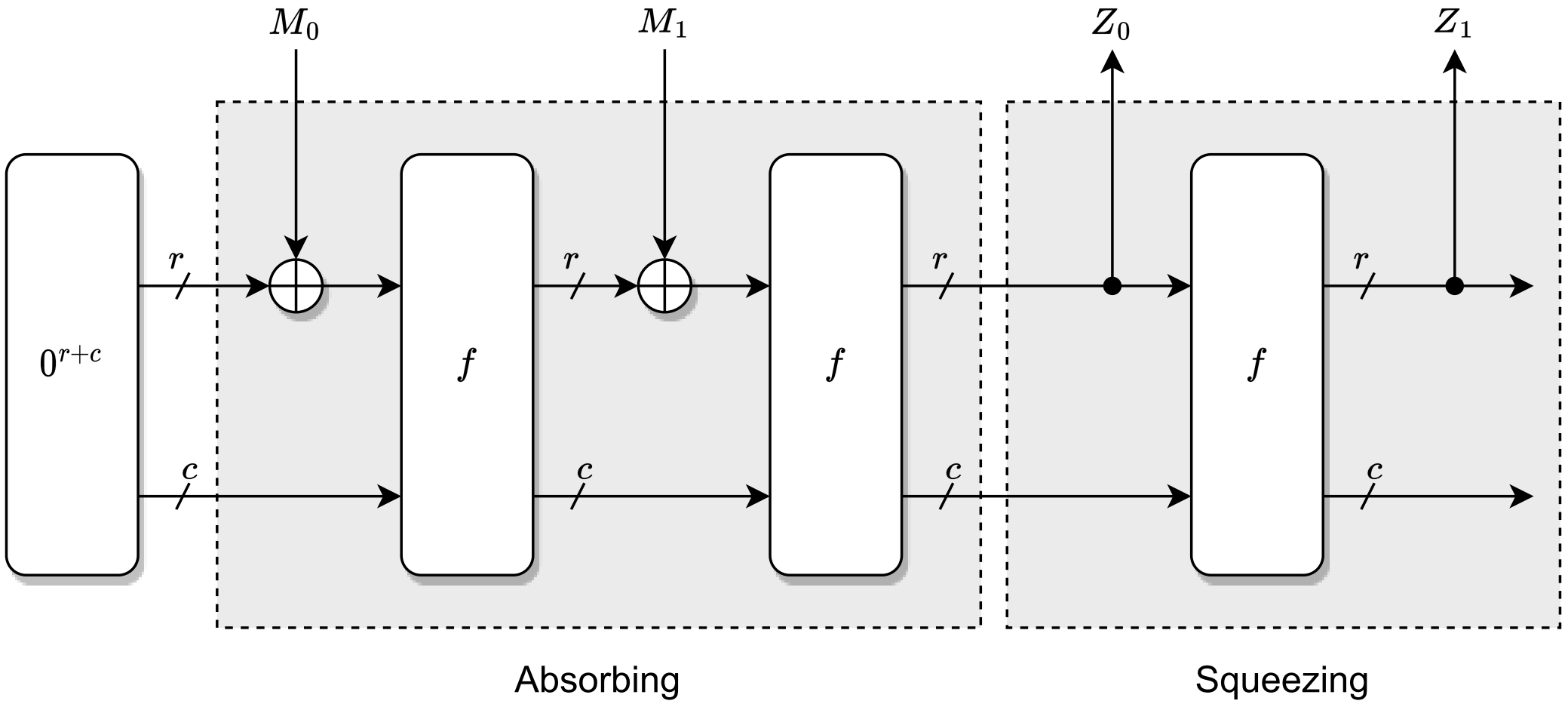



IEEE TRANSACTIONS ON RELIABILITY, VOL. 67, NO. 3, SEPTEMBER 2018

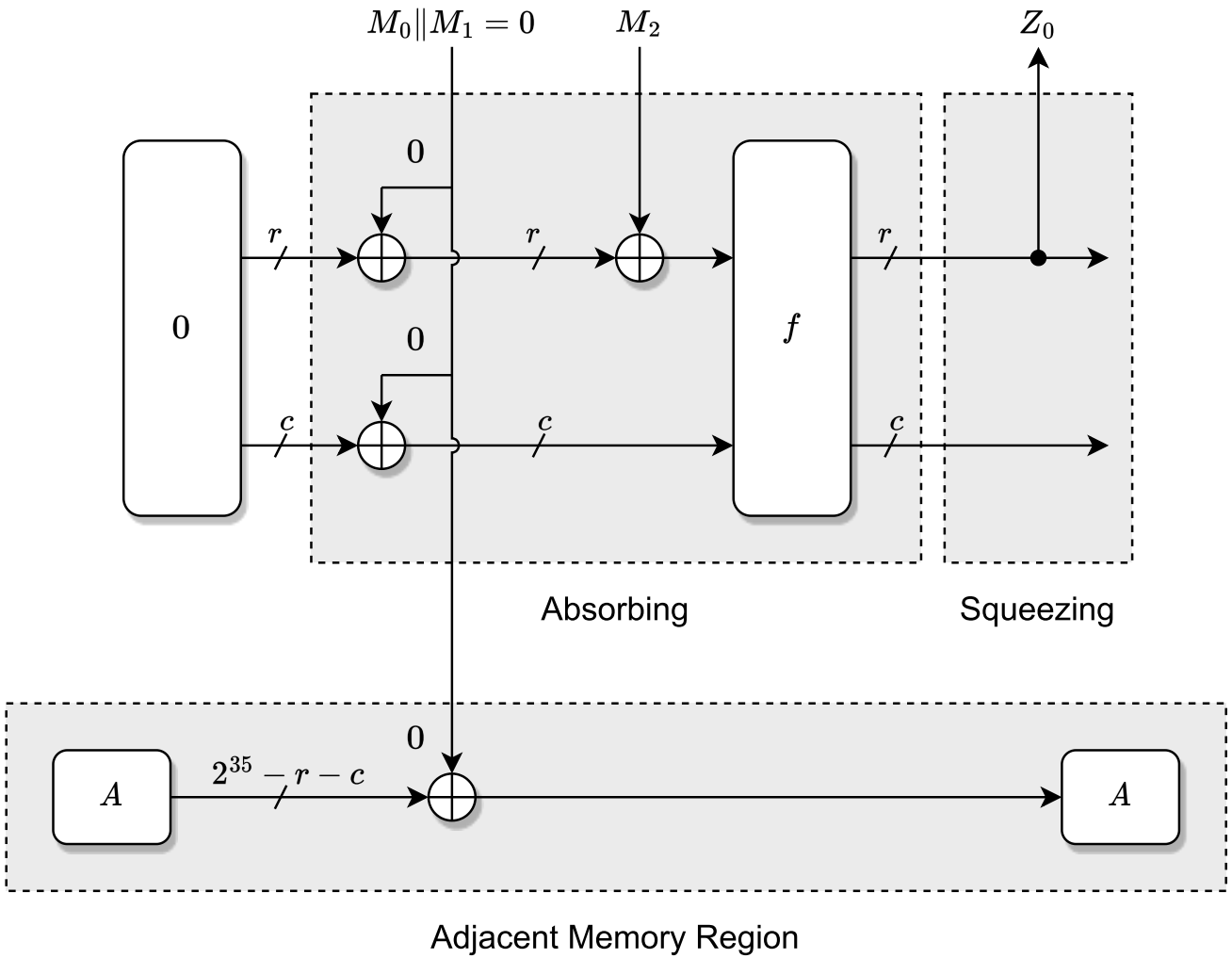
Finding Bugs in Cryptographic Hash Function Implementations

Nicky Mouha, Mohammad S. Raunak, D. Richard Kuhn, *Fellow, IEEE*, and Raghu Kacker

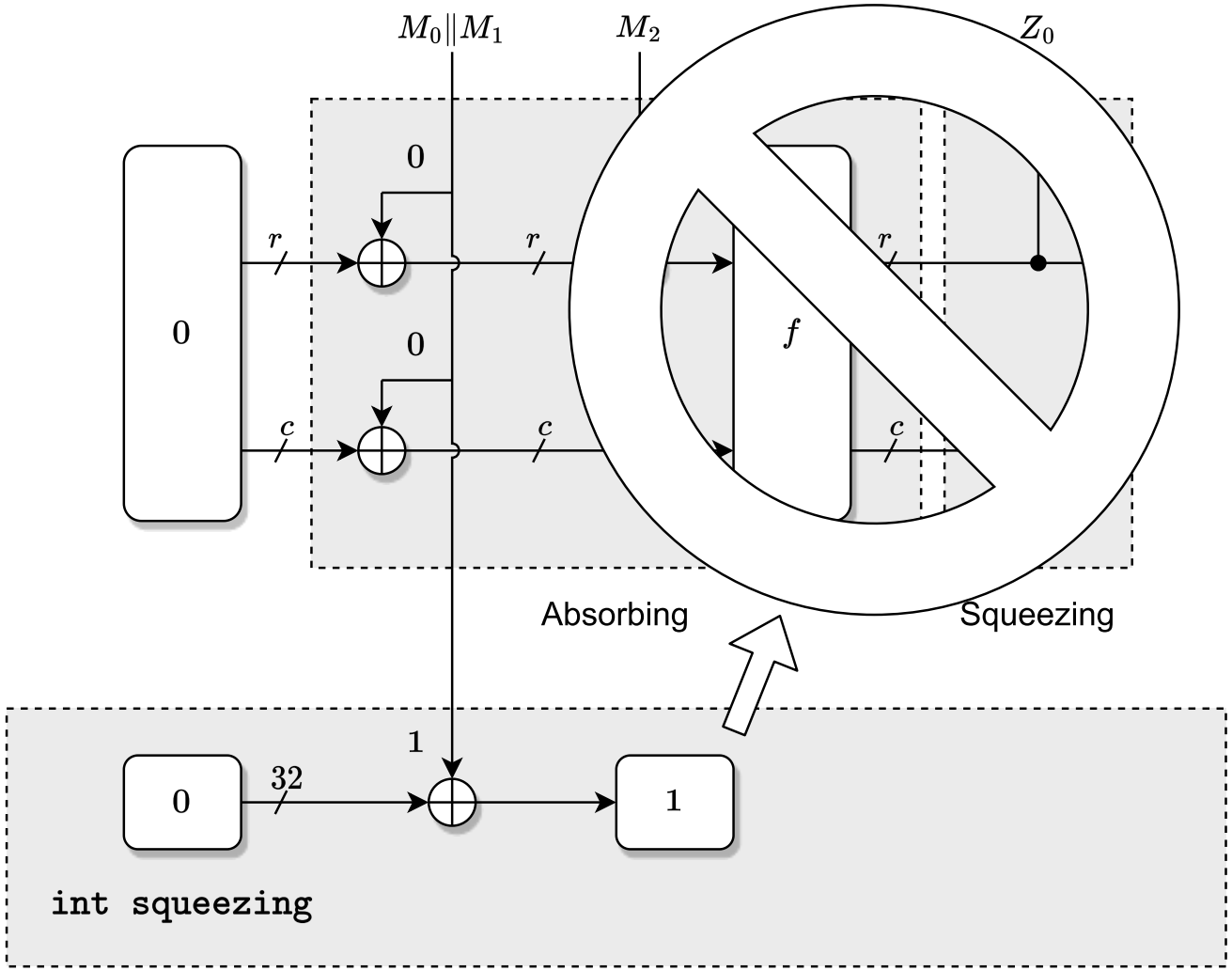
# SHA-3: Sponge Function



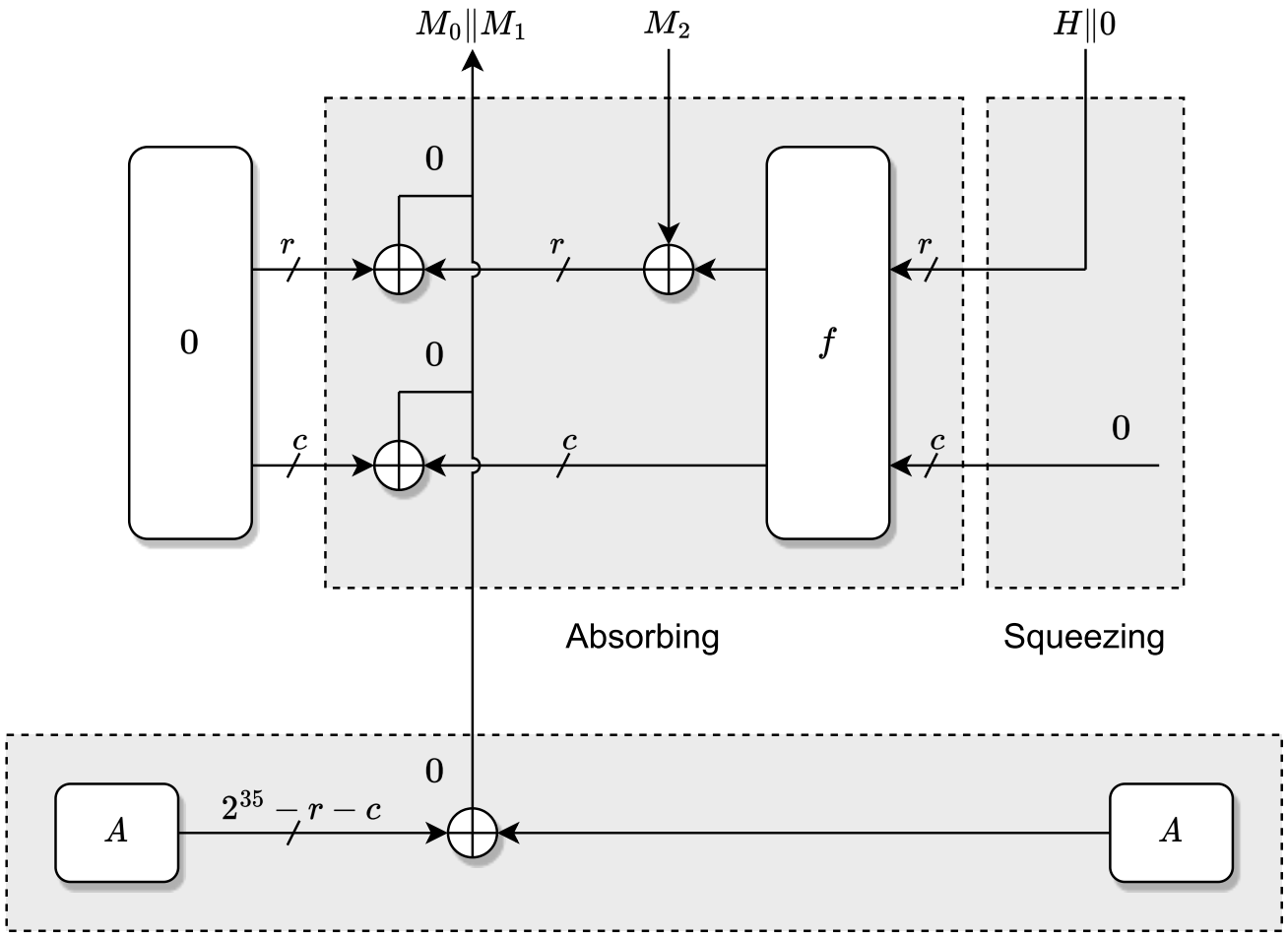
# SHA-3: Second Preimage



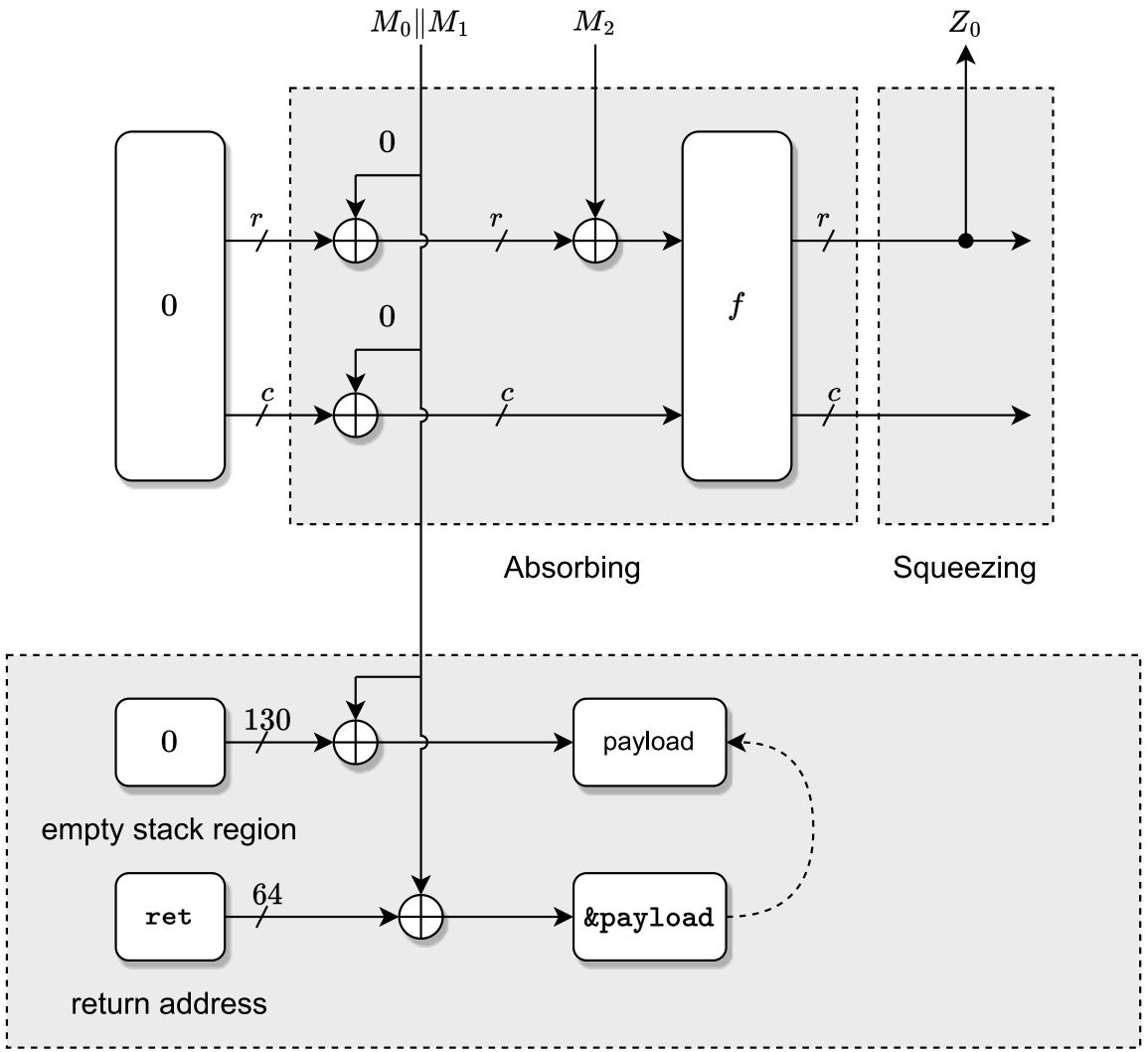
# SHA-3: Preimage of Zero



# SHA-3: Preimage of Any Hash Value



# SHA-3: Exploit



# Init-Update-Final Test (IUFT)

```
{
    "messages": [
        { "message": "00", "length": 8 },
        { "largeMessage": { "content": "00",
                            "contentLength": 8,
                            "fullLength": 34359738360,
                            "expansionTechnique": "repeating" }
                          } ]
}
```

# CodeQL

"Add query for CVE-2022-37454" (https://github.com/github/codeql/pull/12036)

```
todo = digest_len;
if (done + todo > out_len) {
  todo = out_len - done;
}
OPENSSL_memcpy(out_key + done, previous, todo);
done += todo;
```

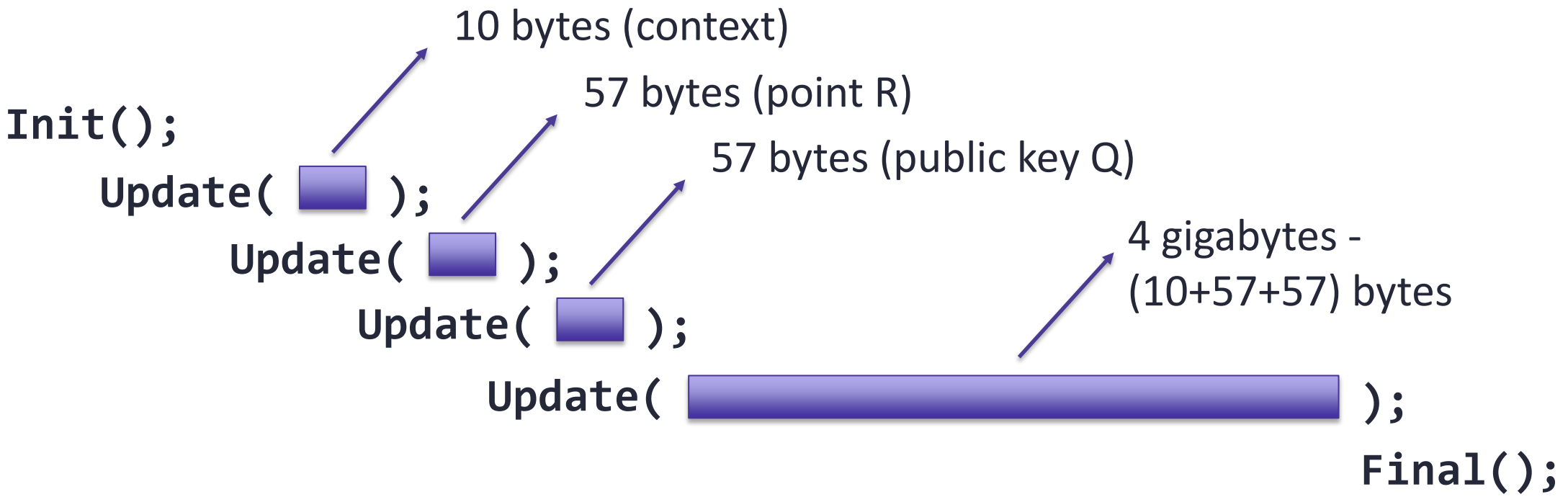
# CodeQL

"Add query for CVE-2022-37454" (https://github.com/github/codeql/pull/12036)

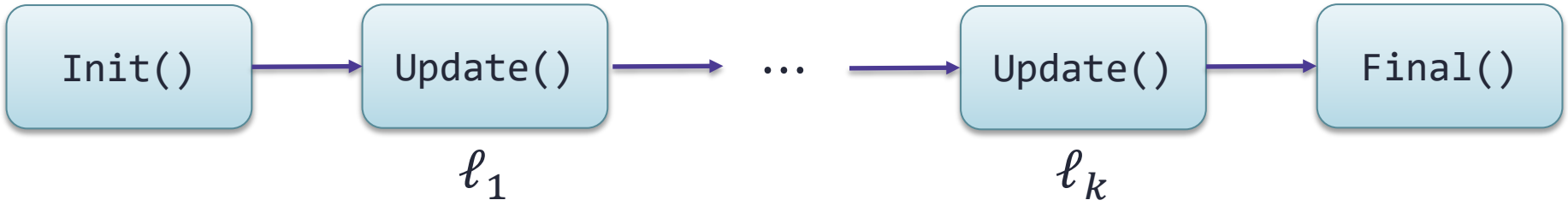
```
todo = digest_len;
if (todo > out_len - done) {
  todo = out_len - done;
}
OPENSSL_memcpy(out_key + done, previous, todo);
done += todo;
```

# EdDSA (FIPS 186-5)

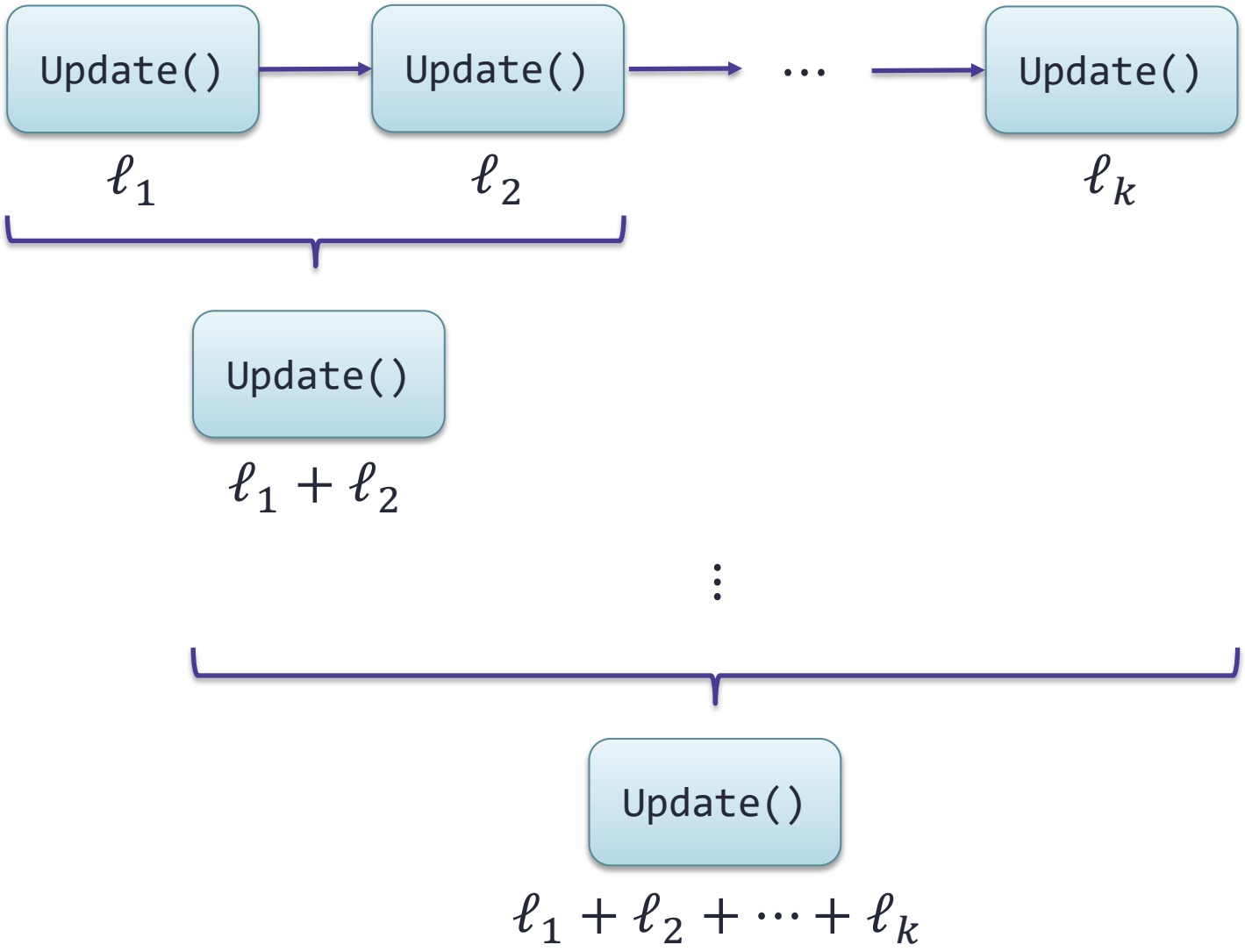
- Ed448: requires SHAKE256 (block size: 136 bytes)
- Attack **verification**!



# Init-Update-Final

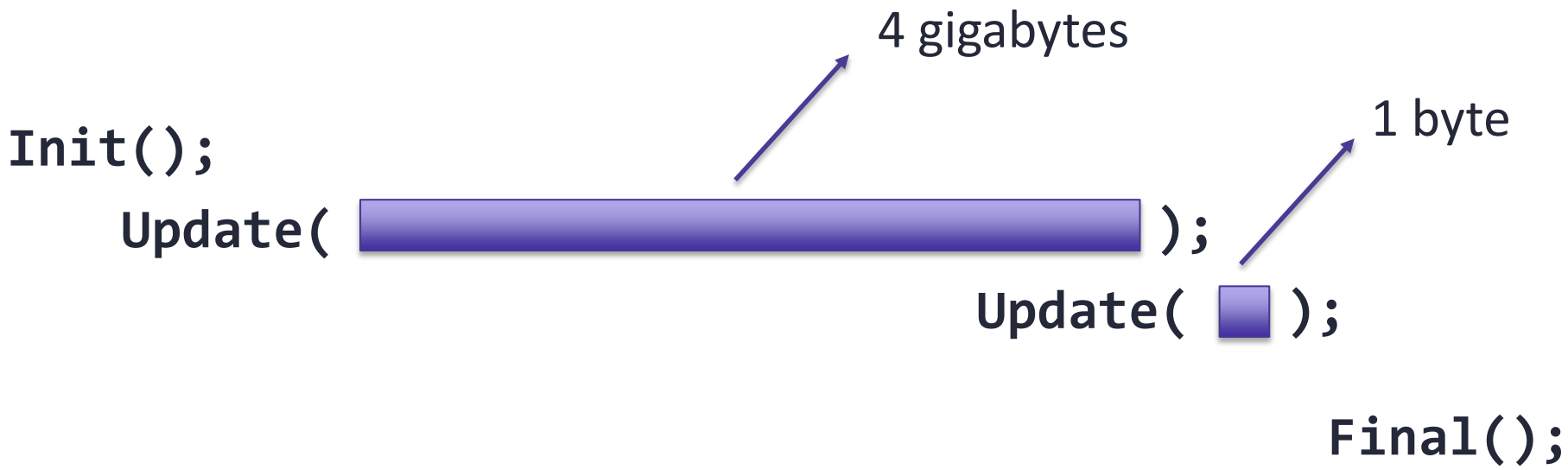


# Consistency of Update()



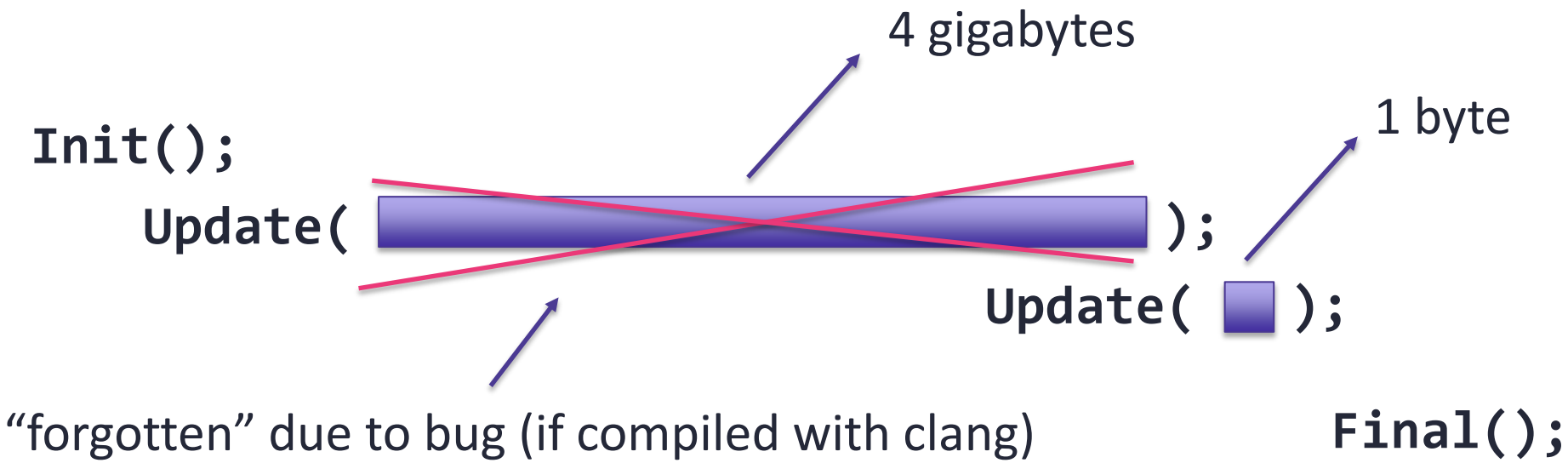
# Grøstl Bug (ACISP 2023)

- Appeared in 2008
- First disclosed at ACISP 2023!



# Grøstl Bug (ACISP 2023)

- Appeared in 2008
- First disclosed at ACISP 2023!



# Grøstl Bug (ACISP 2023)

- Appeared in 2008
- First disclosed at ACISP 2023!