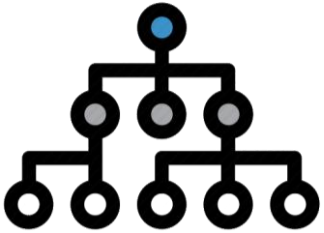
Symbolic Execution of Security Protocol Impl.: Handling Cryptographic Primitives

Mathy Vanhoef — @vanhoefm

USENIX WOOT, Baltimore, US, 14 August 2018

KU LEUVEN DistriNet

Overview



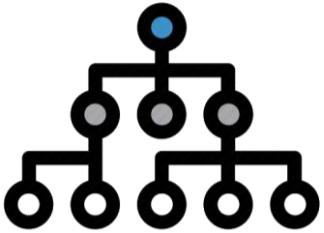
Symbolic Execution

4-way handshake

Handling Crypto

Results

Overview



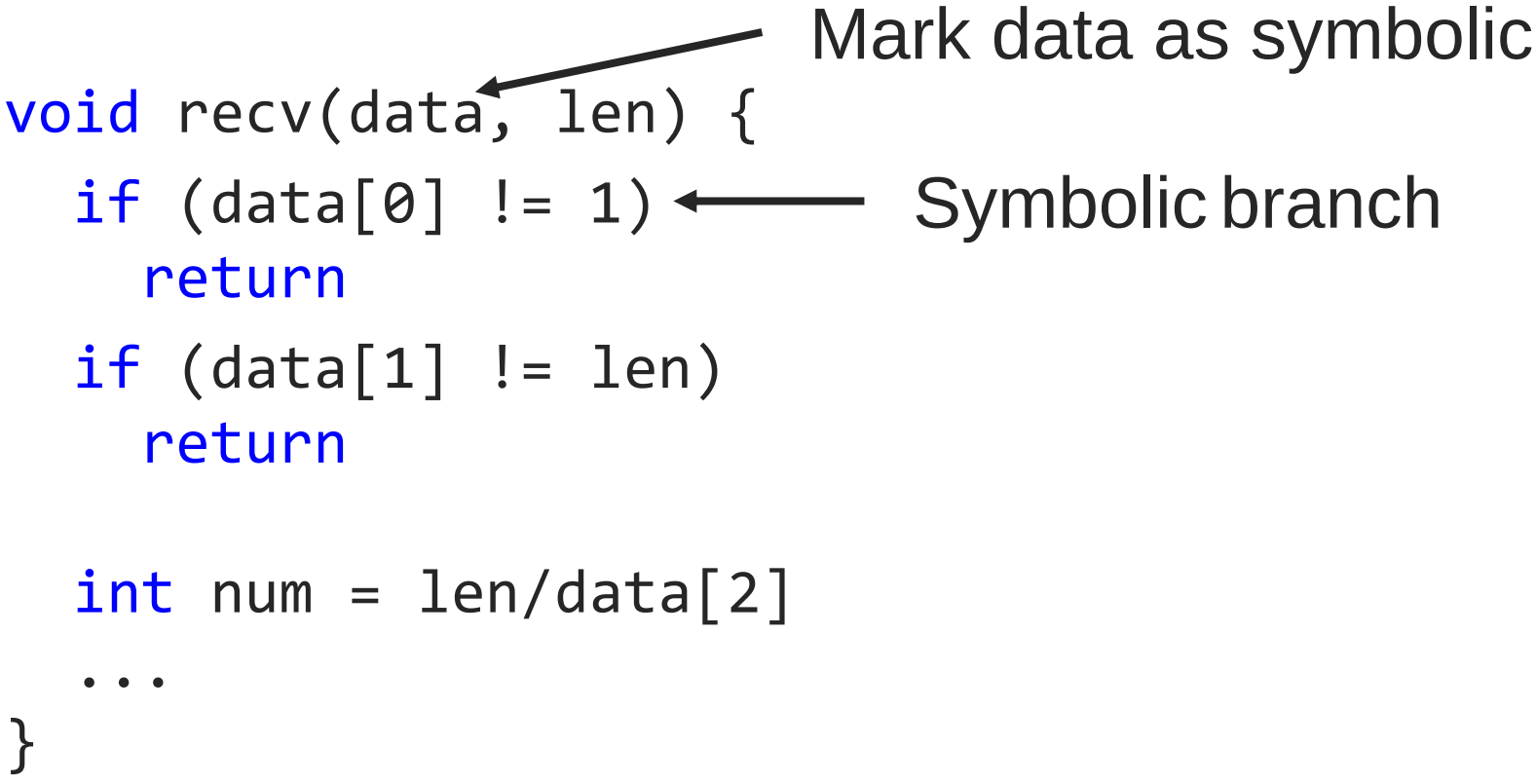
Symbolic Execution

4-way handshake

Handling Crypto

Results

Symbolic Execution



Symbolic Execution

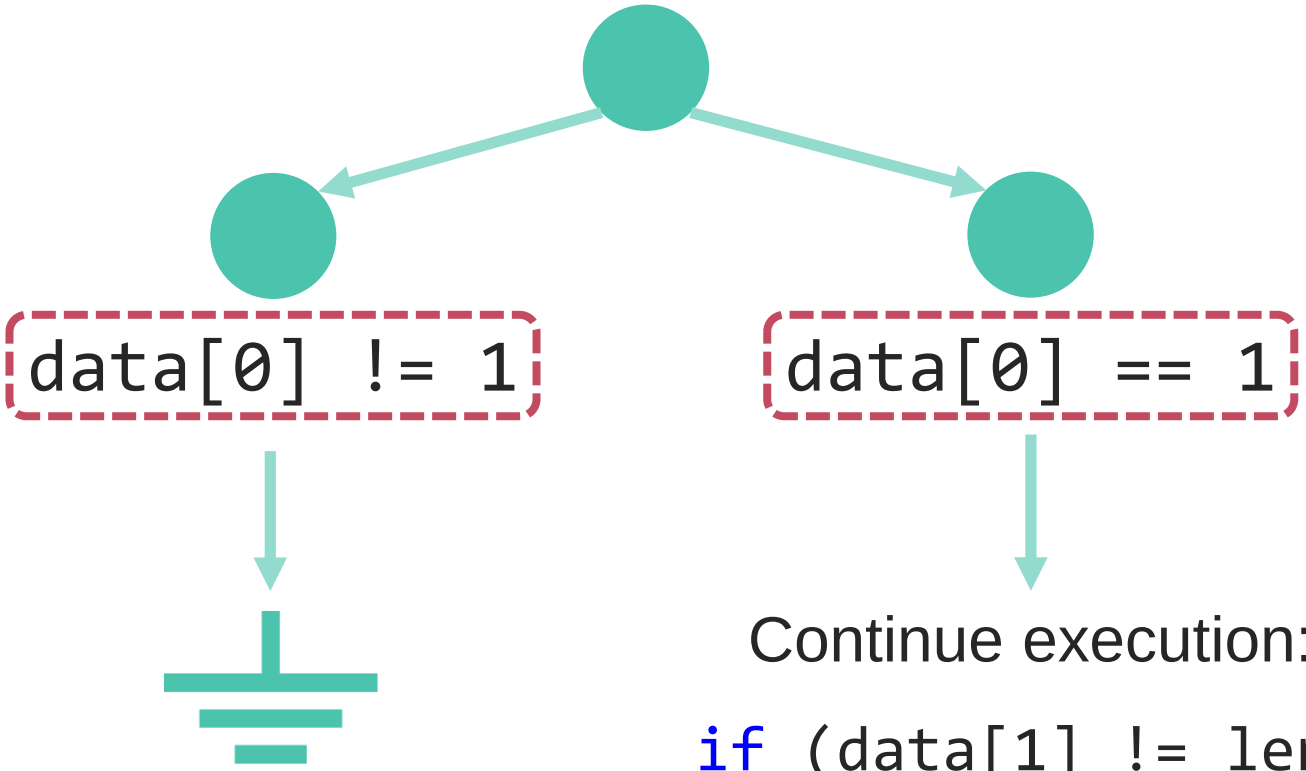
data[0] != 1

```
void recv(data, len) {
  if (data[0] != 1)
    return
  if (data[1] != len)
    return

  int num = len/data[2]
  ...
}
```

data[0] == 1

```
void recv(data, len) {
  if (data[0] != 1)
    return
  if (data[1] != len)
    return

  int num = len/data[2]
  ...
}
```

Symbolic Execution

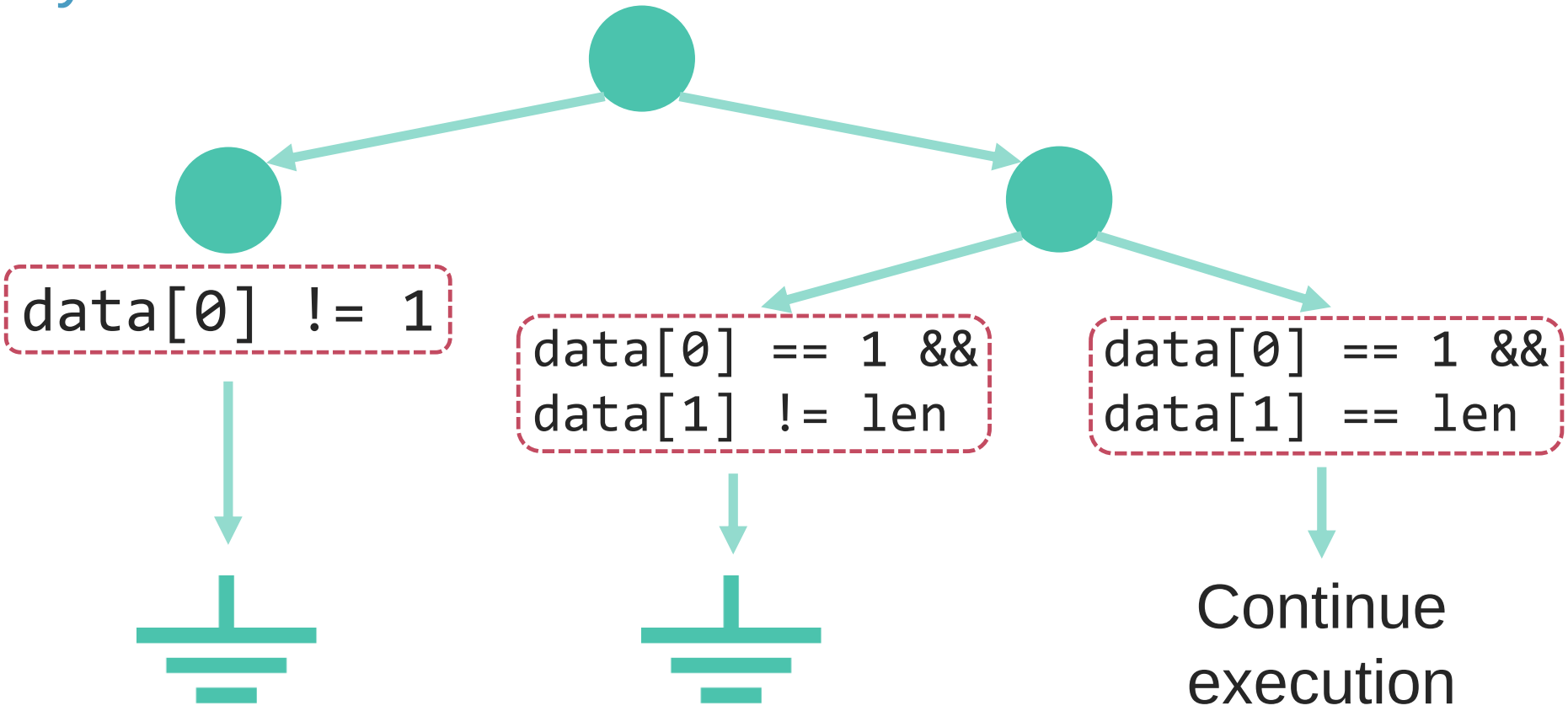


PC = Path Constraint

Continue execution:

```
if (data[1] != len)
```

Symbolic Execution



Symbolic Execution

```
data[0] == 1 &&
data[1] == len
-----------------------------------------
void recv(data, len) {
  if (data[0] != 1)
    return
  if (data[1] != len)
    return

  int num = len/data[2]
  ...
```

Can data[2] equal zero under the current PC?

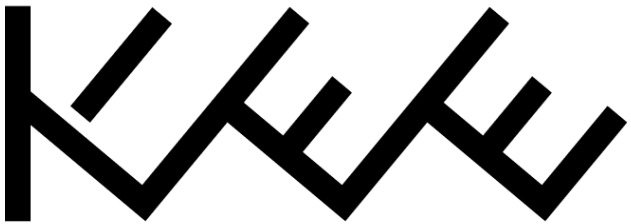
Symbolic Execution

```
data[0] == 1 &&
data[1] == len
-----------------------------------------
void recv(data, len) {
  if (data[0] != 1)
    return
  if (data[1] != len)
    return

  int num = len/data[2]
  ...
```

Can data[2] equal zero under the current PC?

Yes! Bug detected!

Implementations



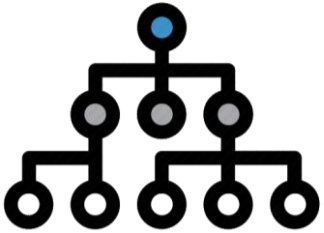
We build upon KLEE

- Works on LLVM bytecode
- Actively maintained

Practical limitations:

- $|paths| = 2^{|if-statements|}$
- Infinite-length paths
- SMT query complexity

Overview



Symbolic Execution

4-way handshake

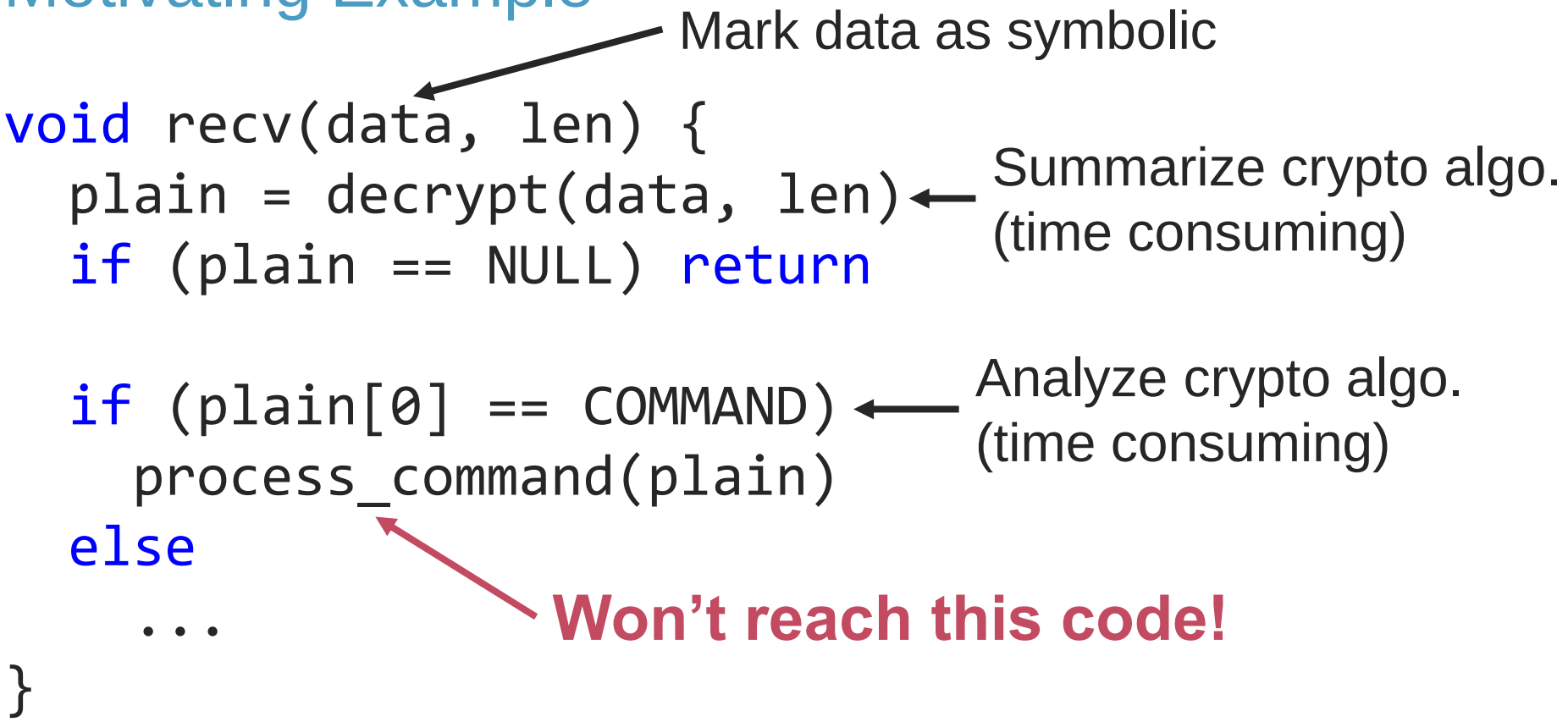
Handling Crypto

Results

Motivating Example


```
void recv(data, len) {
  plain = decrypt(data, len)
  if (plain == NULL) return

  if (plain[0] == COMMAND)
    process_command(plain)
  else
    ...
}
```

Motivating Example



Efficiently handling decryption?

Decrypted output

=

fresh symbolic variable

Example

Mark data as symbolic

```
void recv(data, len) {
  plain = decrypt(data, len)
  if (plain == NULL) return

  if (plain[0] == COMMAND)
    process_command(plain)
  else
    ...
}
```

create fresh symbolic variable

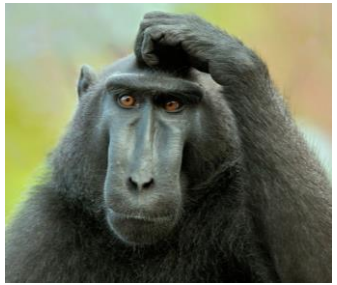
Normal analysis

→ Can now **analyze code that parses decrypted data**

Other Applications

Handling hash functions

- Output = fresh symbolic variable
- Also works for HMACs (Message Authentication Codes)



Tracking use of crypto primitives?

- Recording relationship between input & output
- Treating fresh variable as information flow taint

Detecting Crypto Misuse

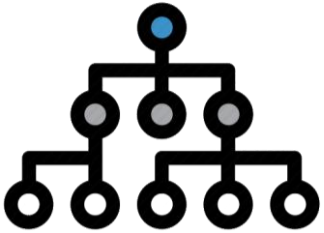
Timing side-channels

- $\forall(paths)$: all bytes of MAC in path constraint?
- If not: comparison exits on first difference

Decryption oracles

- Behavior depends on unauth. decrypted data
- Decrypt data is in path constraint, but not in MAC

Overview



Symbolic Execution

4-way handshake

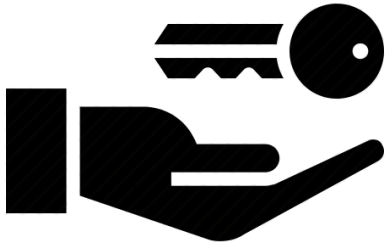
Handling Crypto

Results

The 4-way handshake

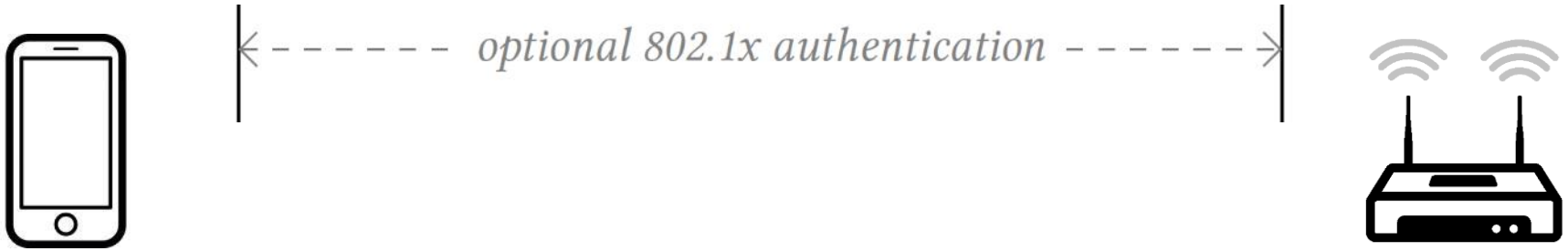
Used to connect to any protected Wi-Fi network

Mutual authentication

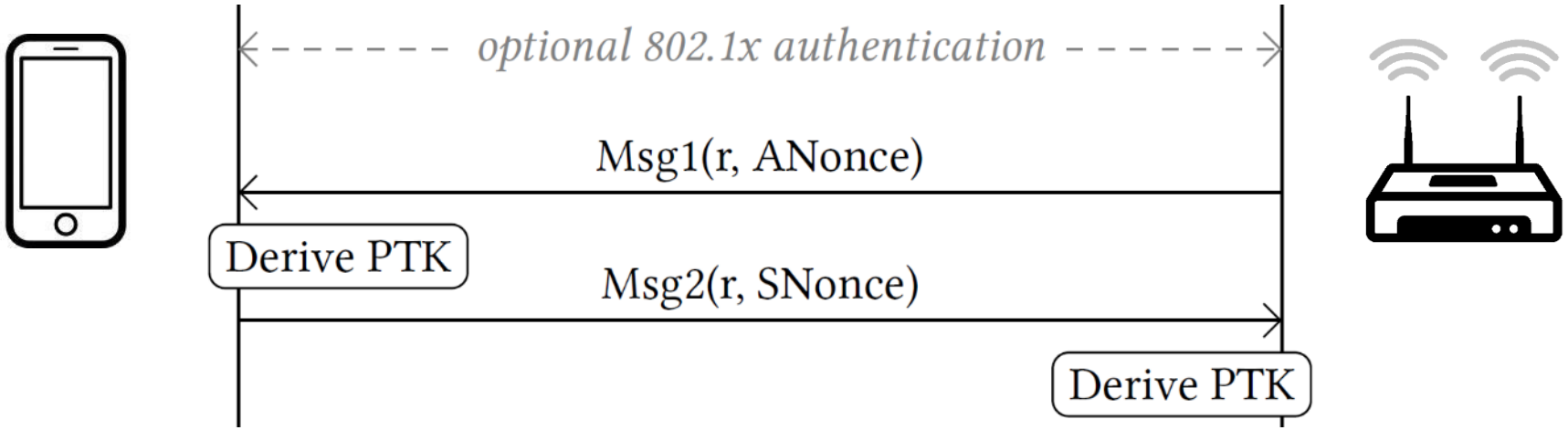


Negotiates fresh PTK:
pairwise transient key

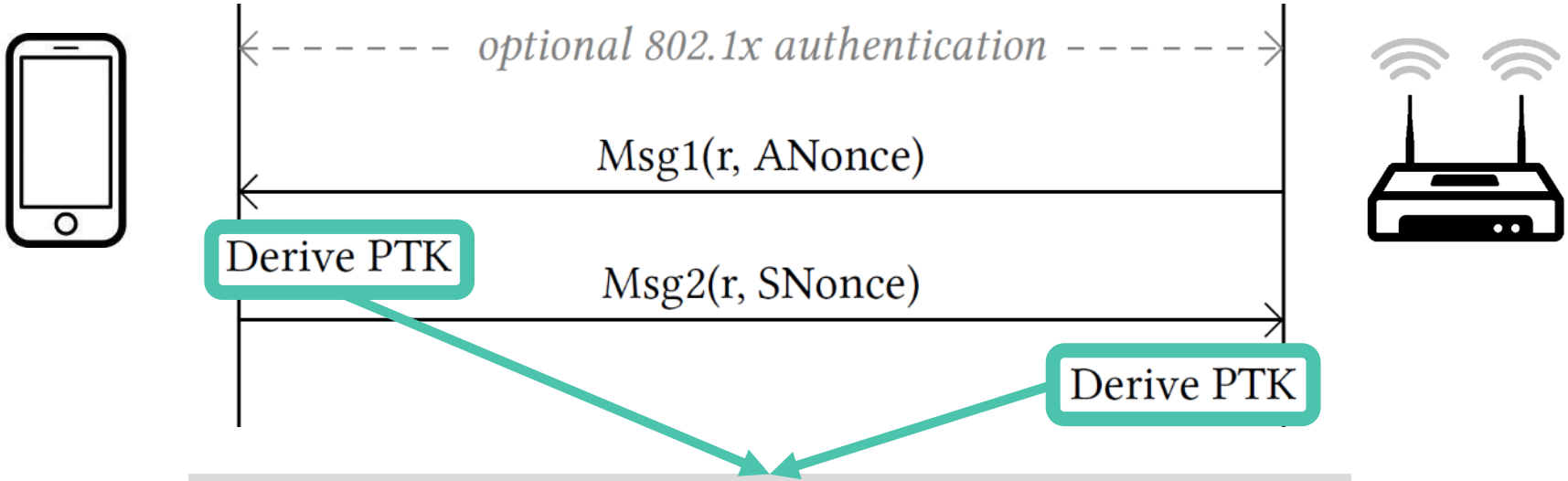
4-way handshake (simplified)



4-way handshake (simplified)

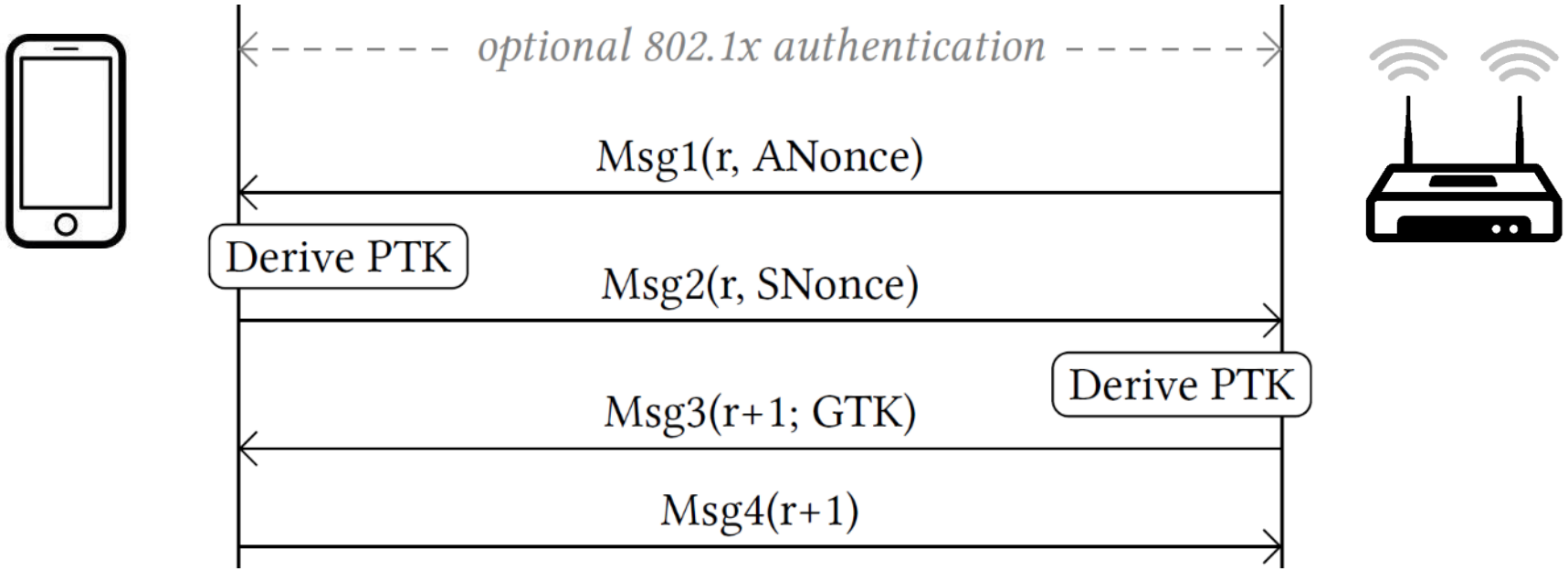


4-way handshake (simplified)

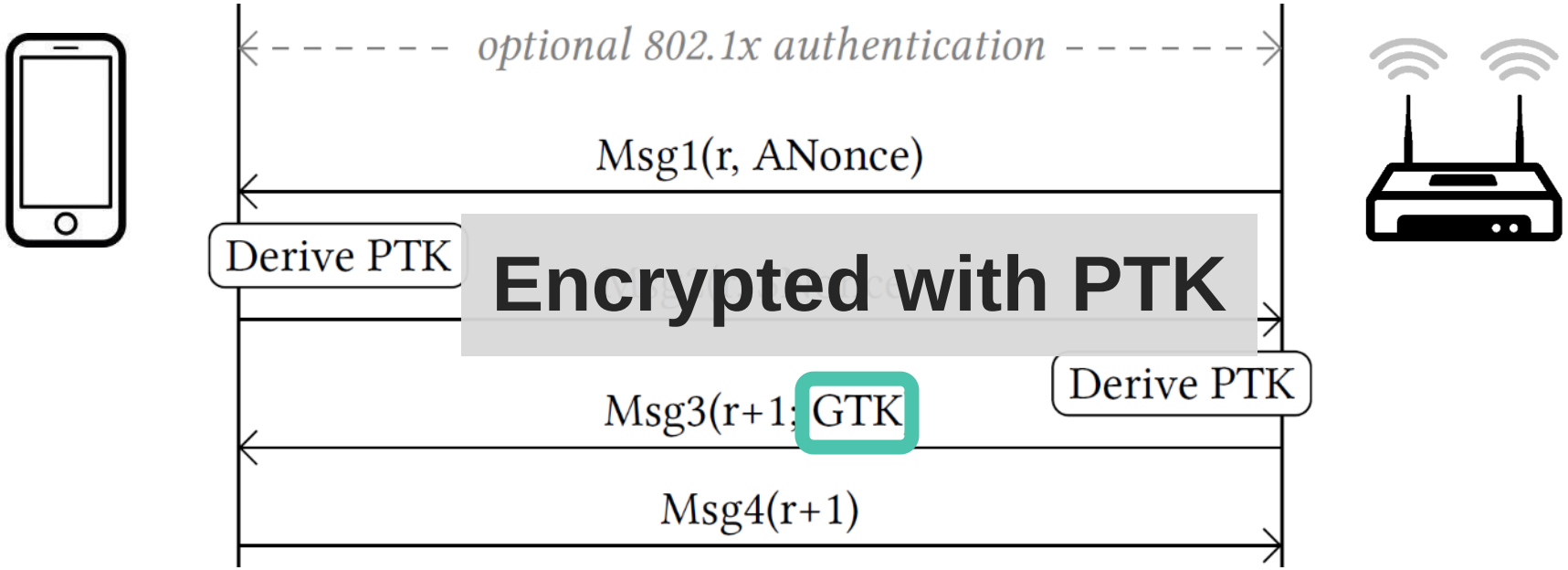


PTK = Combine(shared secret, ANonce, SNonce)

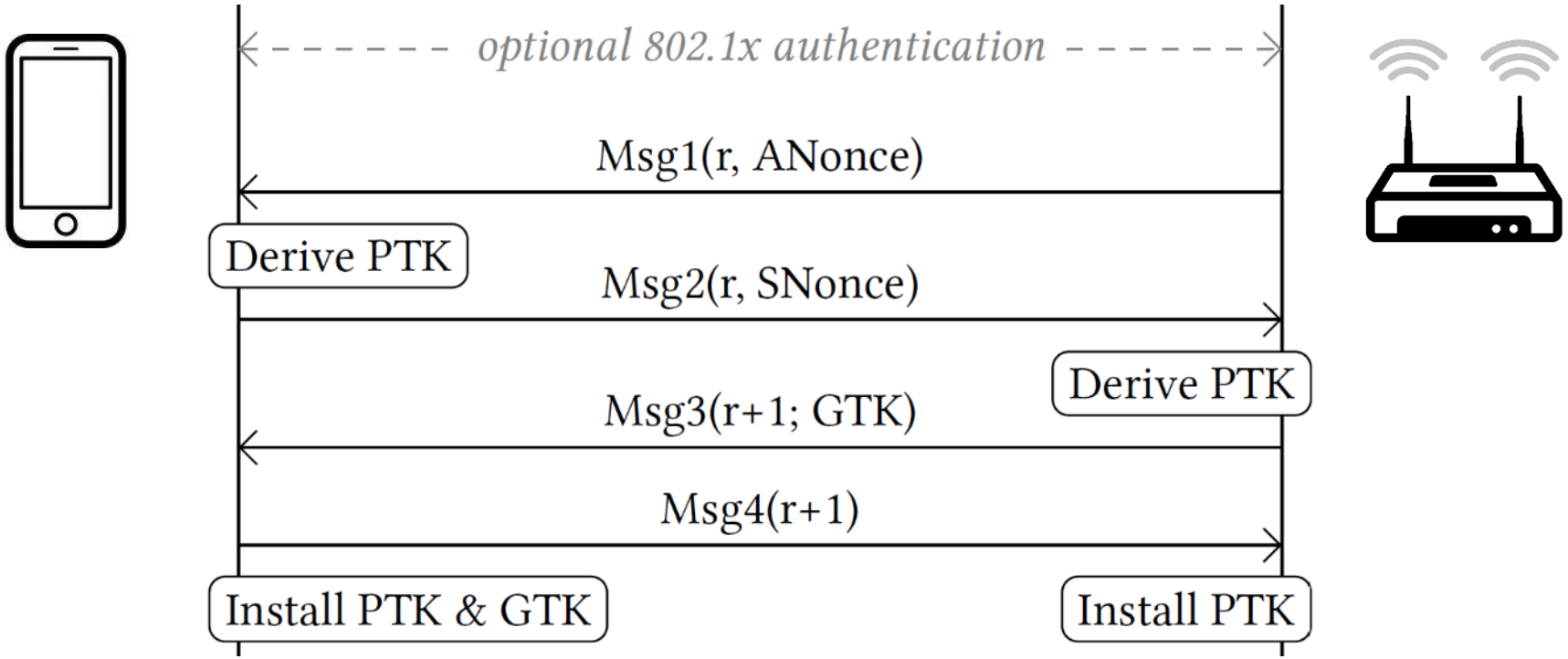
4-way handshake (simplified)



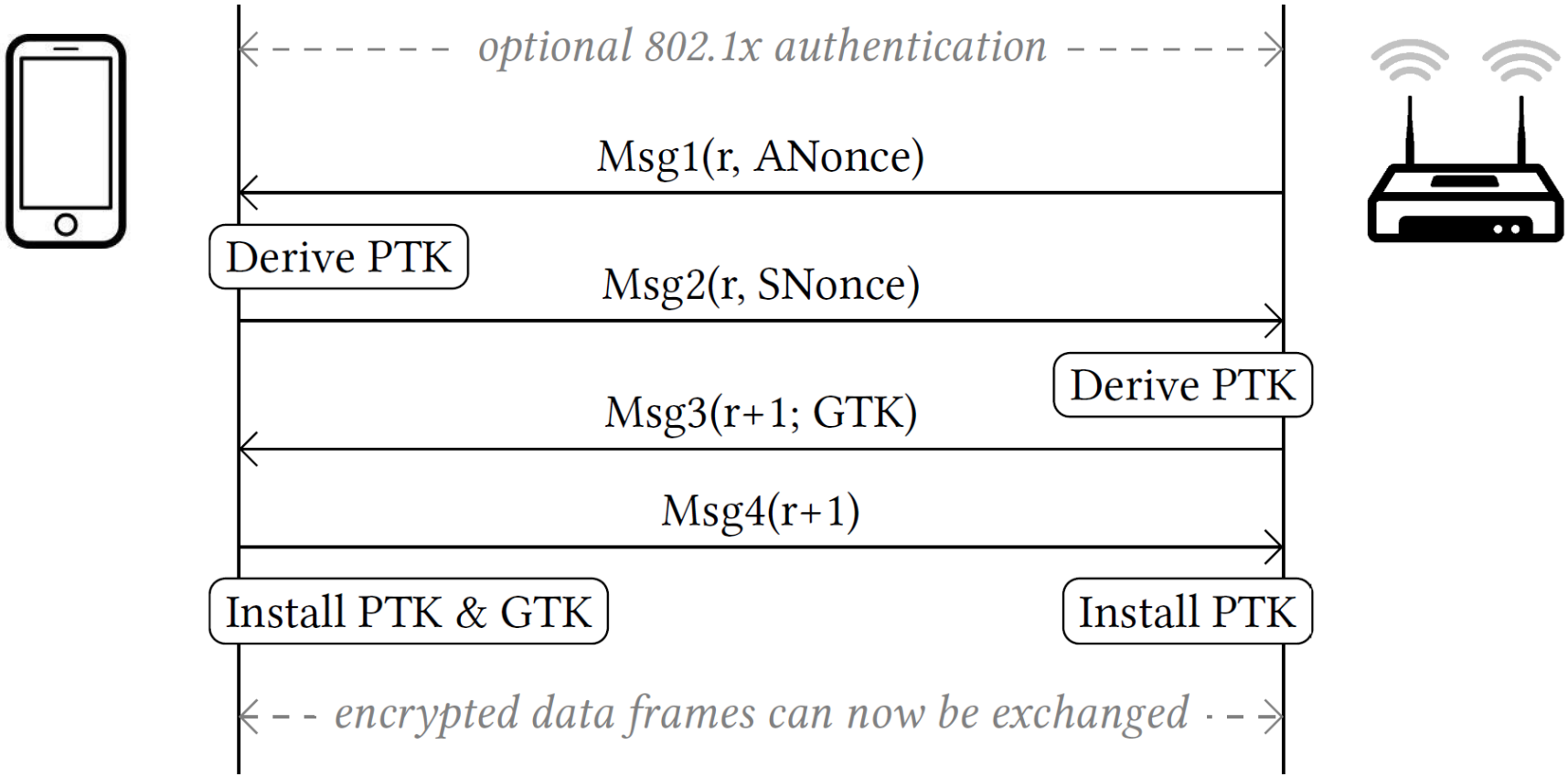
4-way handshake (simplified)



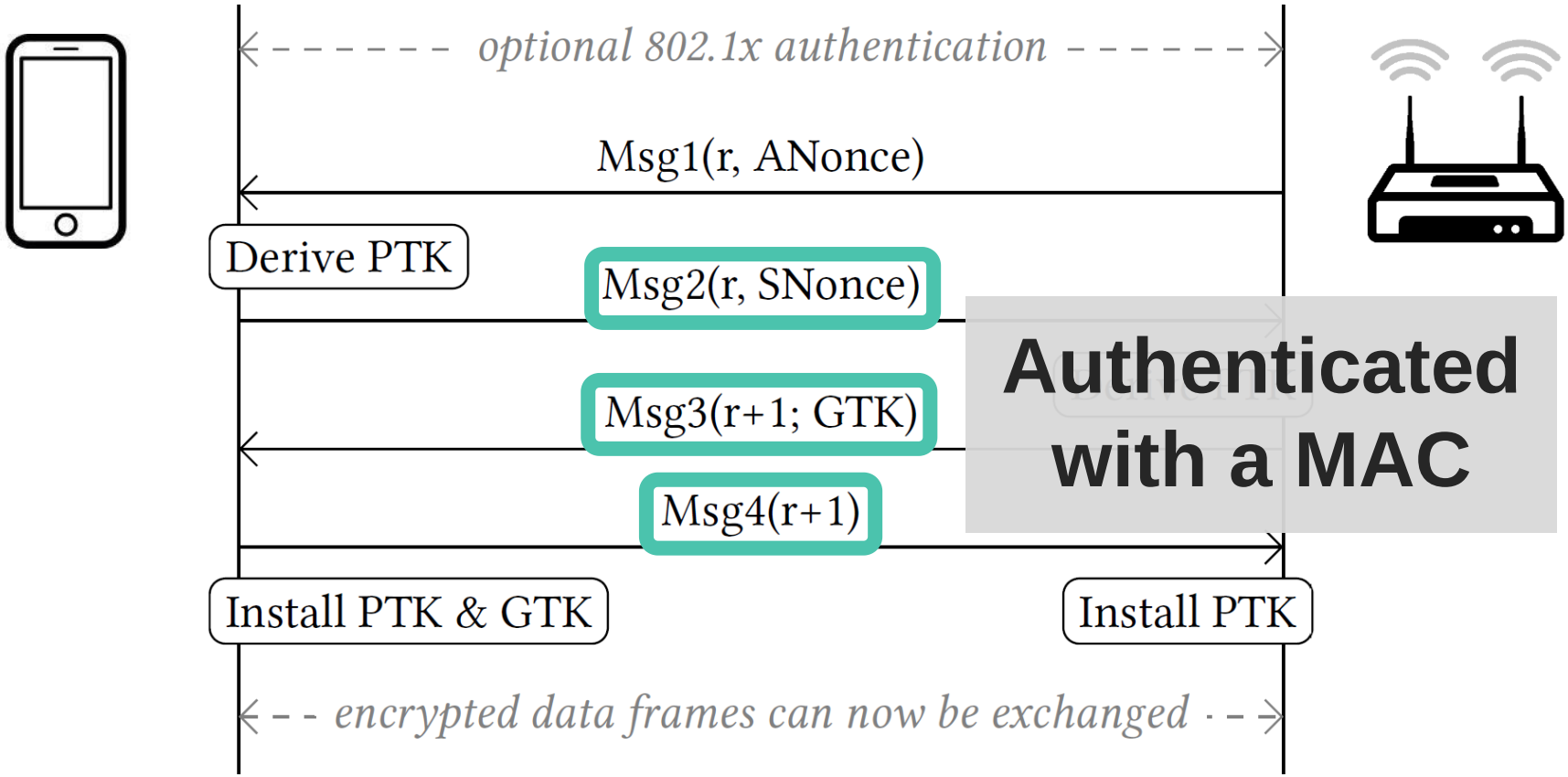
4-way handshake (simplified)



4-way handshake (simplified)

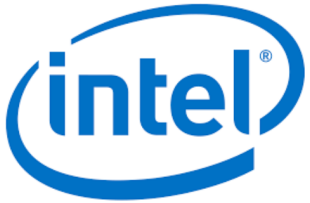


4-way handshake (simplified)



We focus on the client

Symbolic execution of



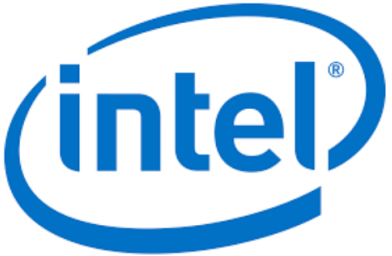
Intel’s iwd deamon

wpa_supplicant

kernel driver

How to get these working under KLEE?

Intel’s iwd



Avoid running full program under KLEE

- Would need to model Wi-Fi stack symbolically

Our approach

- iwd contains unit test for the 4-way handshake
- Reuse initialization code of unit test!
- Symbolically execute only receive function

wpa_supplicant

Unit test uses virtual hardware and runs full AP

- Still need to simulate Wi-Fi stack…

Alternative approach:

- Write unit test that isolates 4-way handshake like iwd
- Then symbolically execute receive function!
- Need to modify code of wpa_supplicant (non-trivial)

MediaTek's Driver

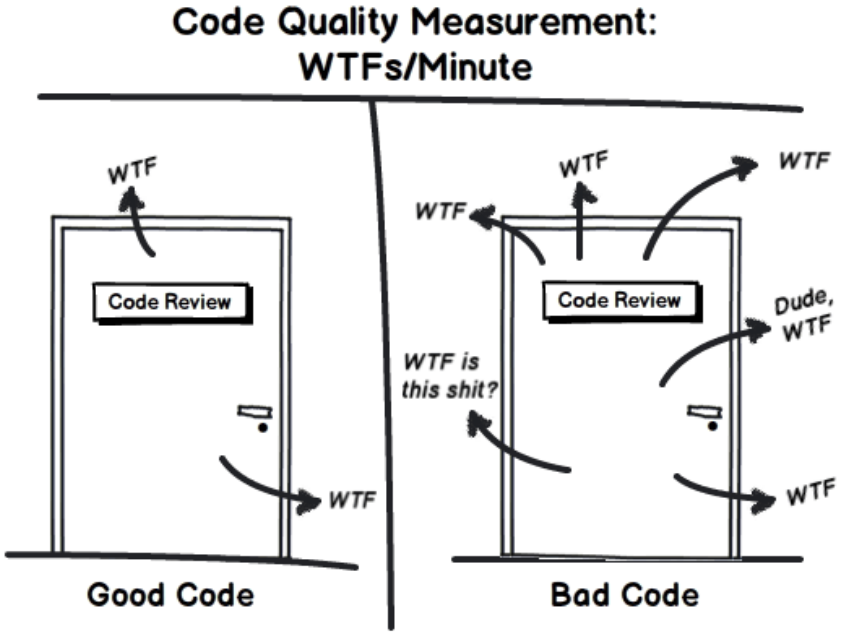
No unit tests & it's a Linux driver

- Symbolically executing the Linux kernel?!

Inspired by previous cases

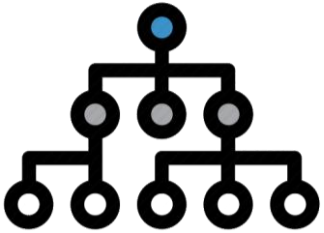
- Write unit test & simulate used kernel functions in userspace
- Verify extracted code is correctly simulated in userspace!

Not all our unit tests are created equally



https://github.com/vanhoefm/woot2018

Overview



Symbolic Execution

4-way handshake

Handling Crypto

Results

Discovered Bugs I

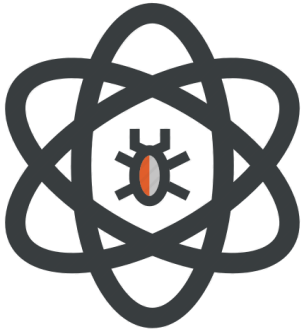
Timing side-channels

- Authentication tag not checked in constant time
- MediaTek and iwd are vulnerable

Denial-of-service in iwd

- Caused by integer underflow
- Leads to huge malloc that fails

Discovered Bugs II



Buffer overflow in MediaTek kernel driver

- Occurs when copying the group key
- May lead to **remote code execution**

Flawed AES unwrap crypto primitive

- Also in MediaTek's kernel driver
- **Manually discovered**

Decryption oracle in wpa_supplicant

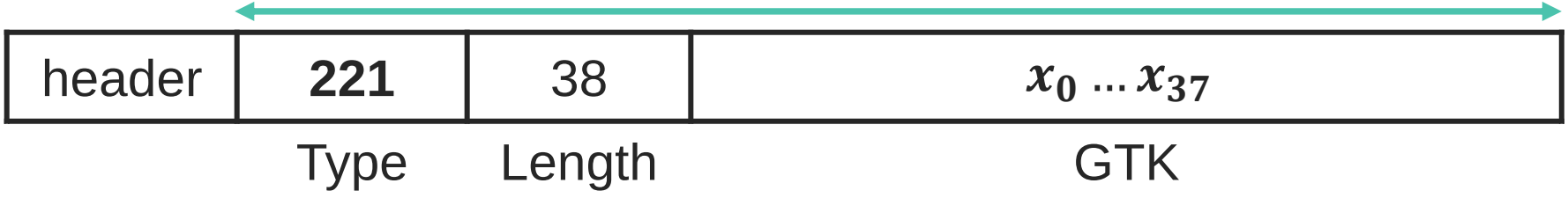
Decryption oracle:

- Doesn’t check authenticity of malformed handshake message
- But does decrypt and process data

→ **Decrypt group key (GTK) in Message 3 (Msg3)**

Decryption oracle in wpa_supplicant II

Msg3': decrypted using RC4, but not authenticated



→ Parsing only succeeds if x_{37} is zero

Future work

Short-term

- Efficiently simulate reception of multiple packets
- If 1st packet doesn't affect state, stop exploring this path

Long-term

- Extract packet formats and state machine
- Verify basic properties of protocol

Conclusion

- Symbolic execution of protocols
- Simple simulation of crypto
- Interesting future work

As a final note…

Thank you!

Questions?