WiFuzz: Detecting and Exploiting Logical Flaws in the Wi-Fi Cryptographic Handshake

Mathy Vanhoef - @vanhoefm

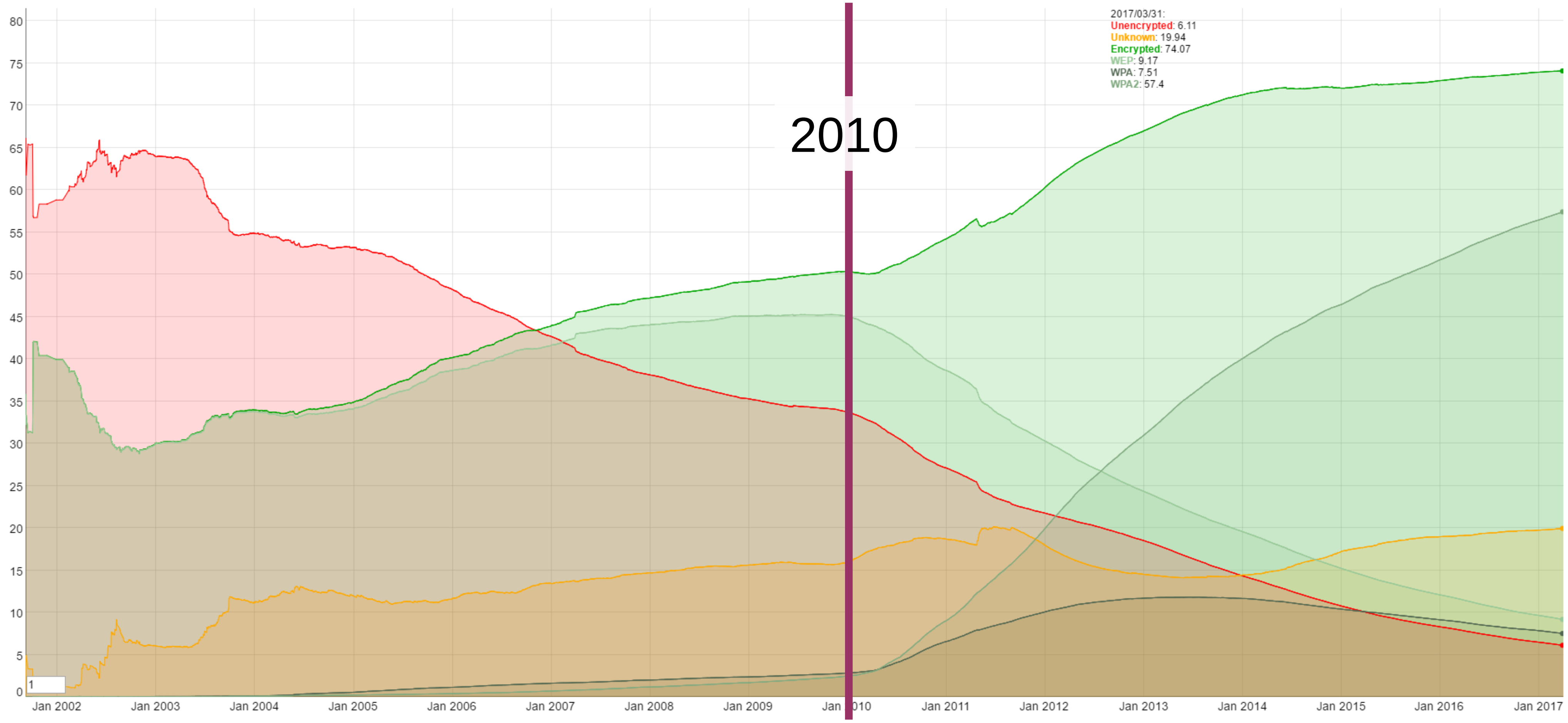
imec-DistriNet, KU Leuven

Black Hat, 27 July 2017

DistriNet

Introduction

More and more Wi-Fi network use encryption:



Most rely on the Wi-Fi handshake to generate session keys

How secure is the Wi-Fi handshake?

Design: formally analyzed and proven correct (CCS 2005)

Security of implementations?

- Some works fuzz network discovery stage
- Many stages are not tested, e.g. 4-way handshake.
- But do not tests for logical implementation bugs

→ Objective: test implementations of the full Wi-Fi handshake for logical vulnerabilities

[1] C. He, M. Sundararajan, A. Datta, A Derek, and J. Mitchell. A modular correctness proof of IEEE 802.11i and TLS.
[2] M. Vanhoef, D. Schepers, and F. Piessens. Discovering Logical Vulnerabilities in theWi-Fi Handshake Using Model-Based Testing.

Background: the Wi-Fi handshake

Main purposes:

- Network discovery
- Mutual authentication & negotiation of pairwise session keys
- Securely select cipher to encrypt data frames

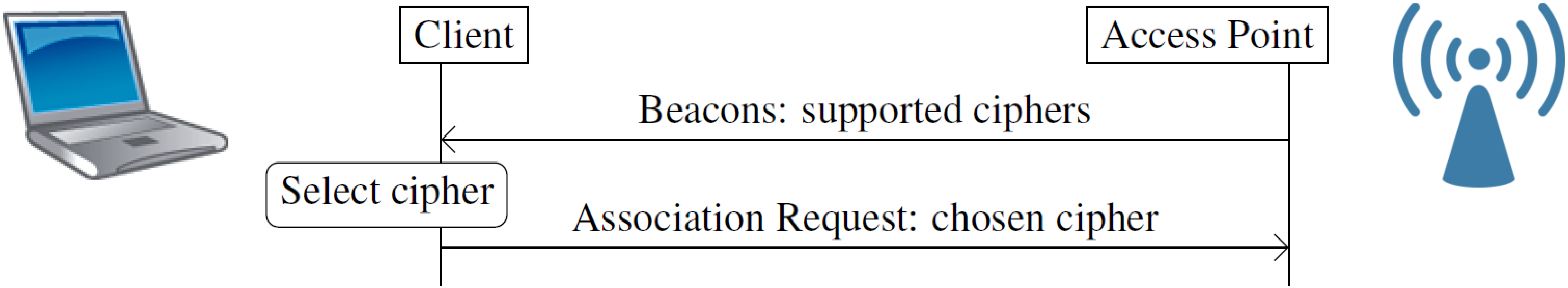
WPA-TKIP

Short-term solution that sacrificed some security, so it could run on old WEP-compatible hardware

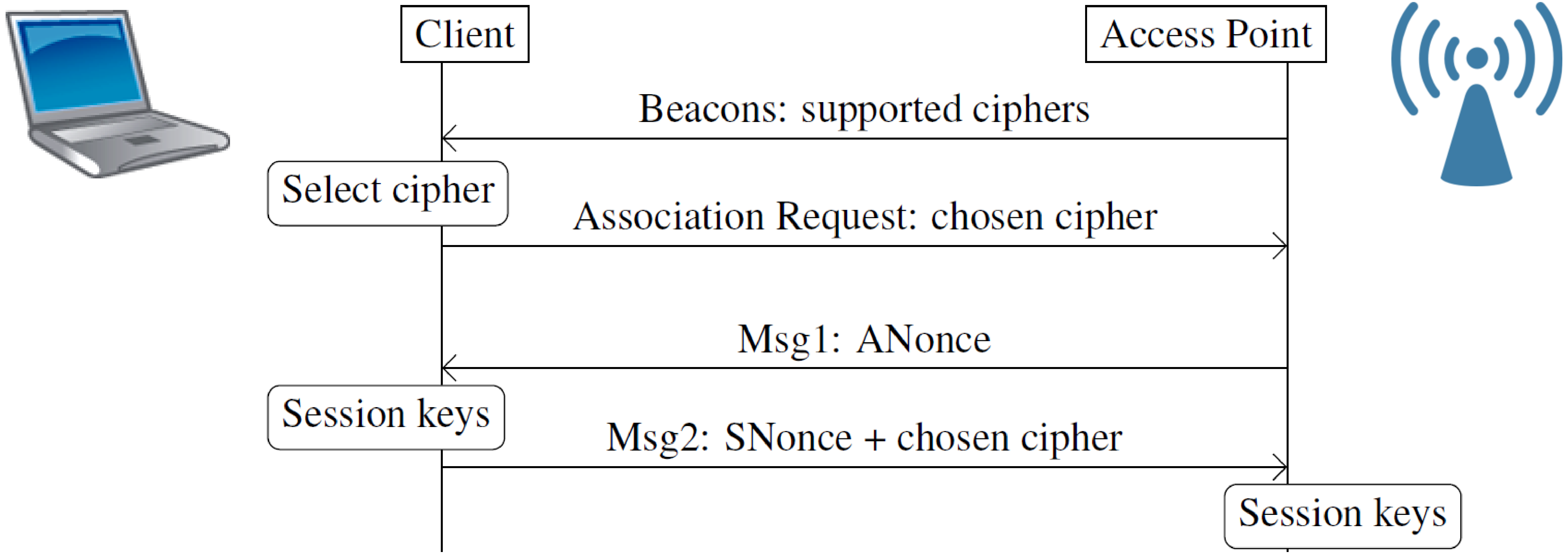
AES-CCMP

Long-term solution based on modern cryptographic primitives

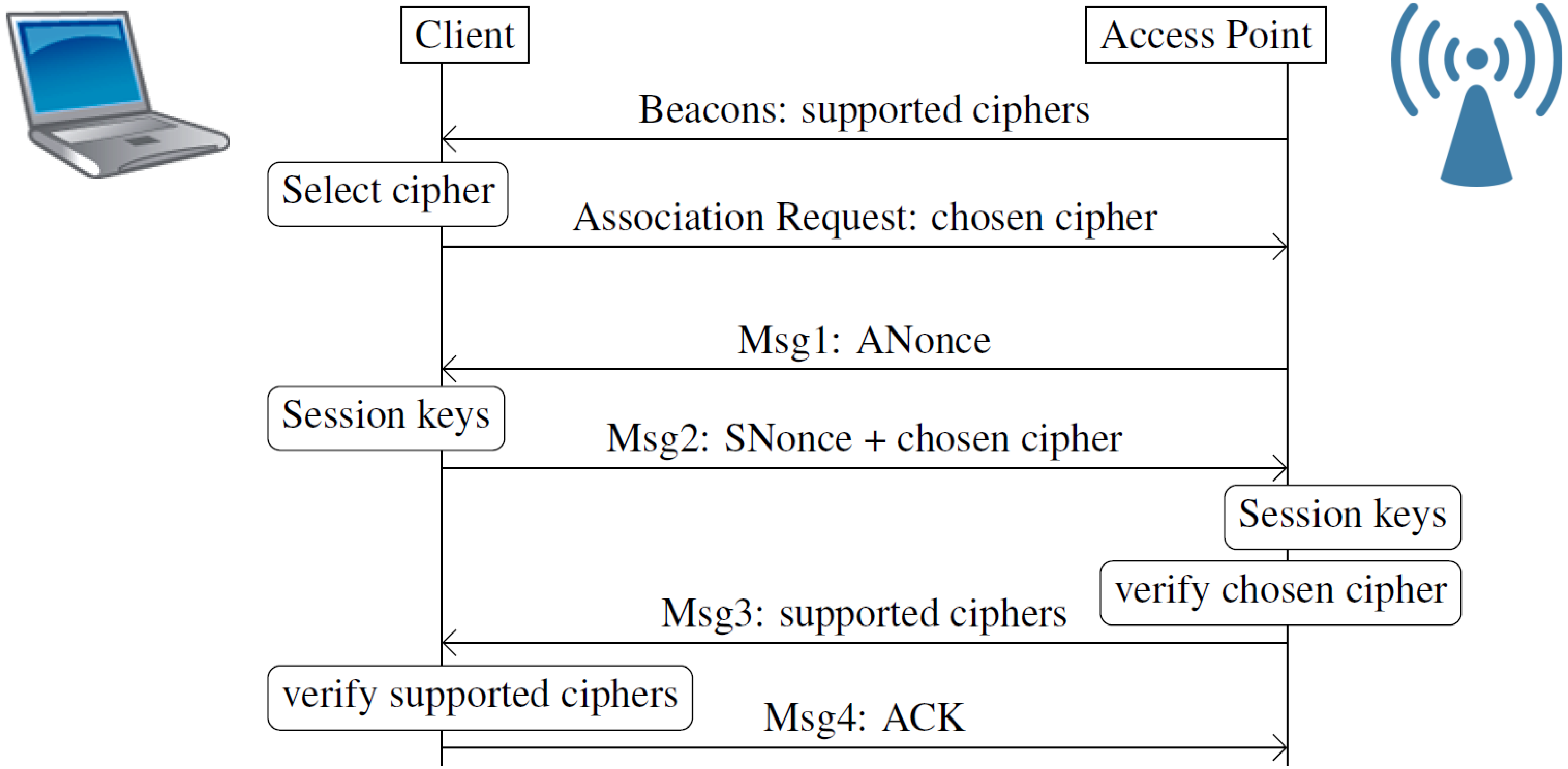
Wi-Fi handshake (simplified)



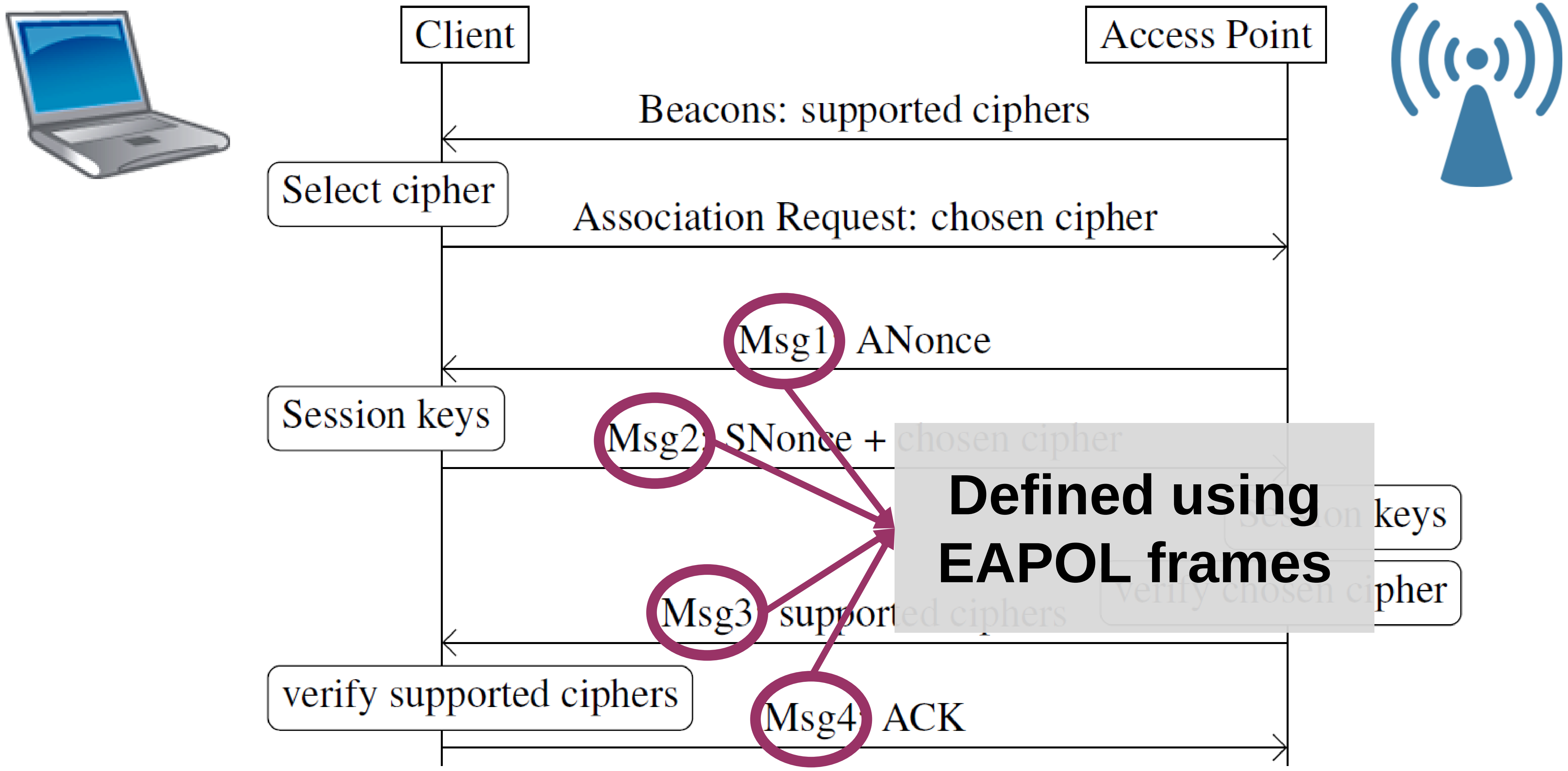
Wi-Fi handshake (simplified)



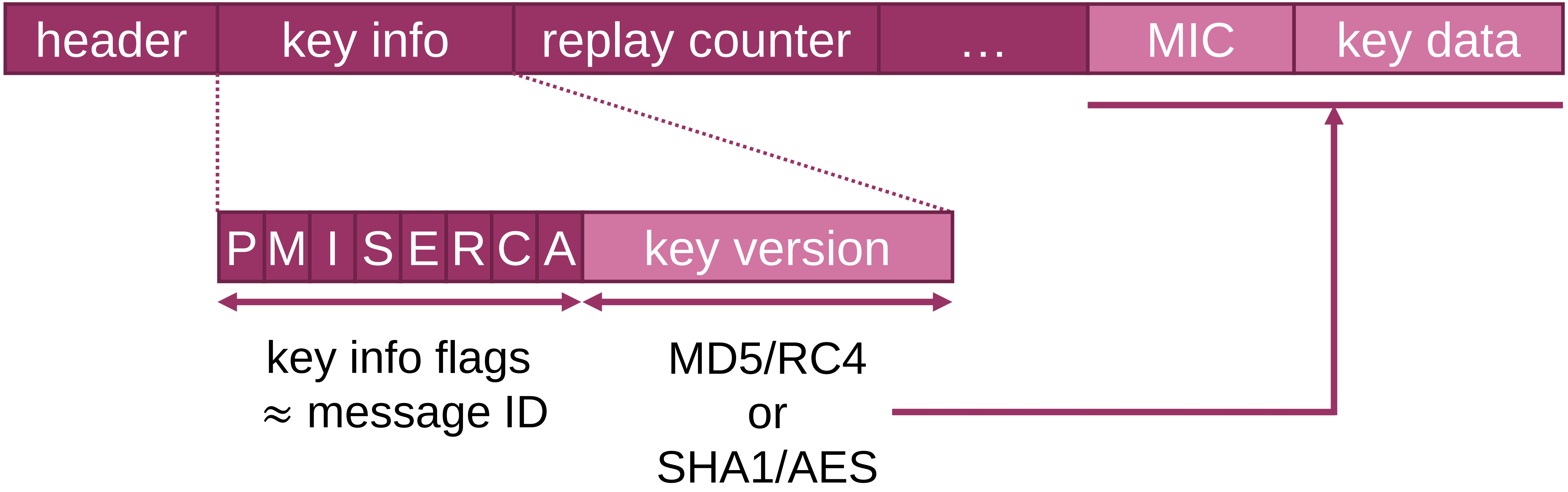
Wi-Fi handshake (simplified)



Wi-Fi handshake (simplified)



EAPOL frame layout (simplified)



How to test implementations?

Model-based testing!

- Test if program behaves according to some abstract model
- Proved successful against TLS

  ➢ Apply model-based approach on the Wi-Fi handshake

Model-based testing: our approach

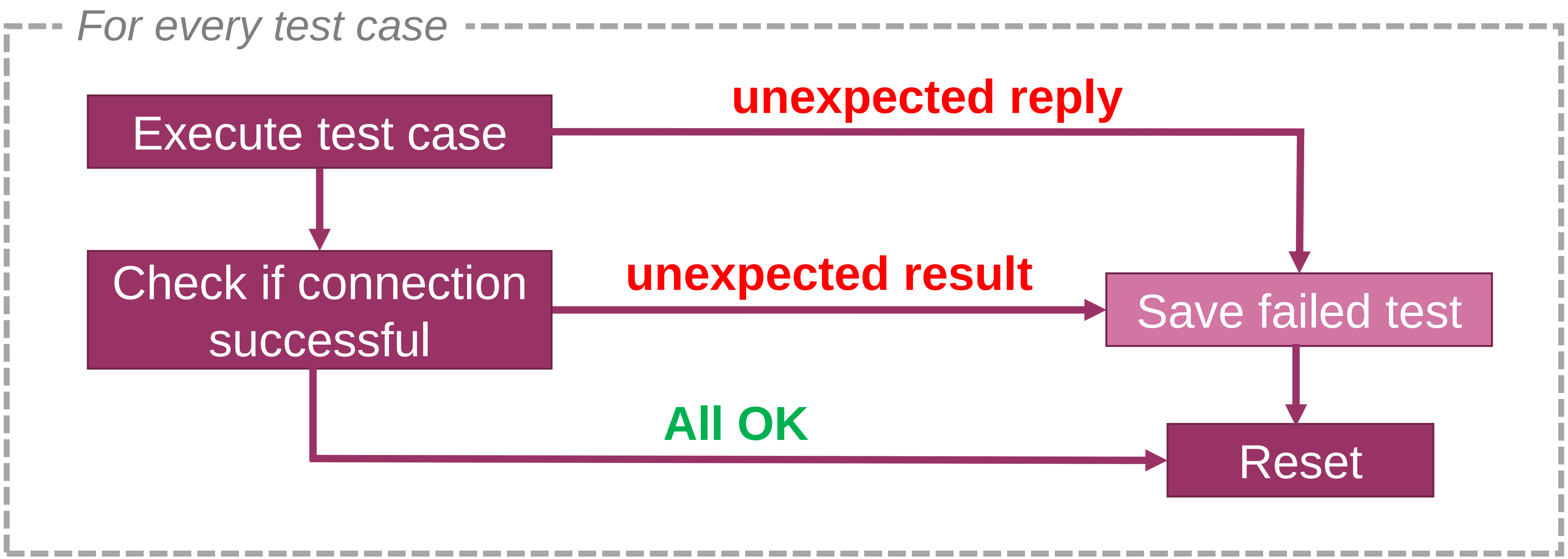
A test case defines:

1. Messages to send & expected replies
2. Results in successful connection?

Generation rules:

- Can test various edge cases, allows some creativity
- Are assumed to be independent (avoid state explosion)

Executing test cases



Afterwards Inspect <u>failed</u> test cases

- Experts determines impact and exploitability

Test generation rules

Test generation rules manipulating messages as a whole:

1. Drop a message
2. Inject/repeat a message

Test generation rules that modify fields in messages:

1. Wrong selected cipher suite in message 2
2. Bad EAPOL replay counter
3. Bad EAPOL key info flags (used to identify message)
4. Bad EAPOL key version (switch SHA1/AES with MD5/RC4)
5. Bad EAPOL Message Integrity Check (MIC)
6. …

Evaluation

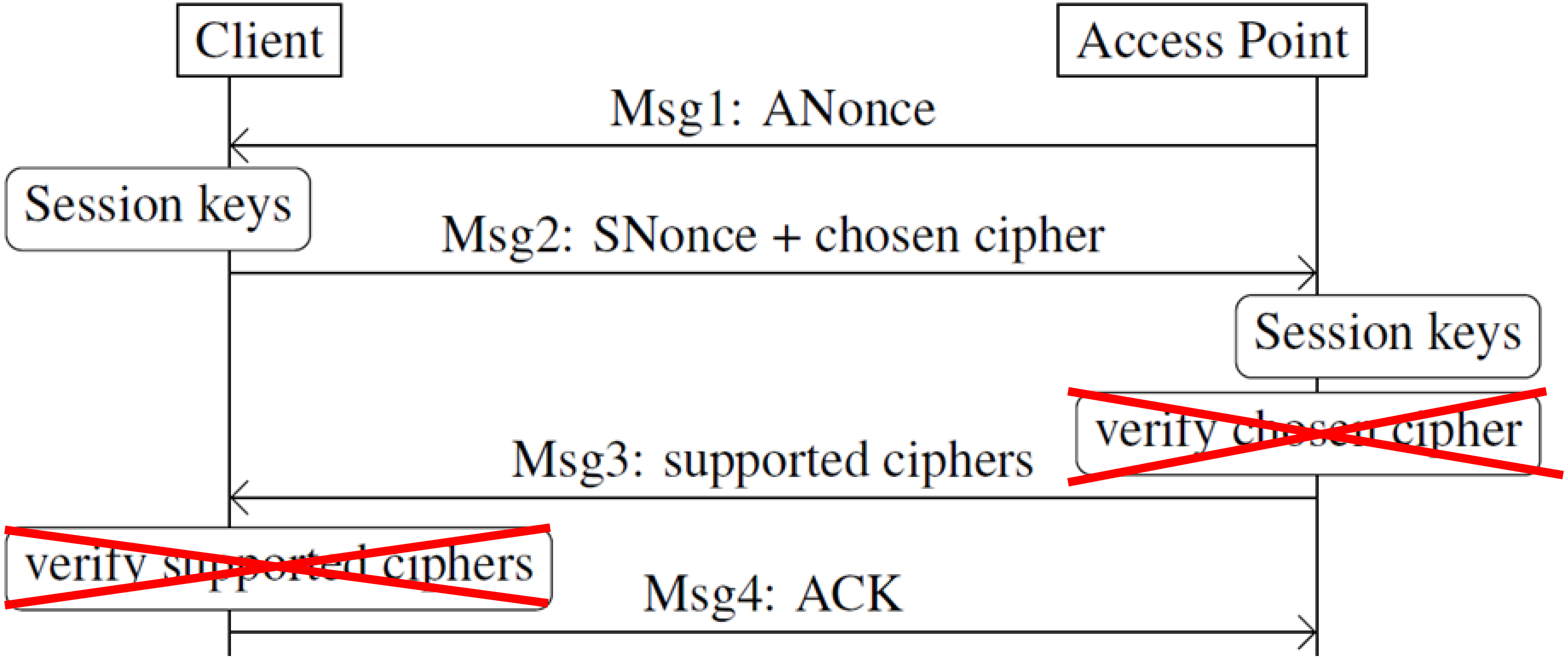
We tested 12 access points:

- Open source: OpenBSD, Linux's Hostapd
- Leaked source: Broadcom, MediaTek (home routers)
- Closed source: Windows, Apple, Telenet
- Professional equipment: Aerohive, Aironet

Discovered several issues!

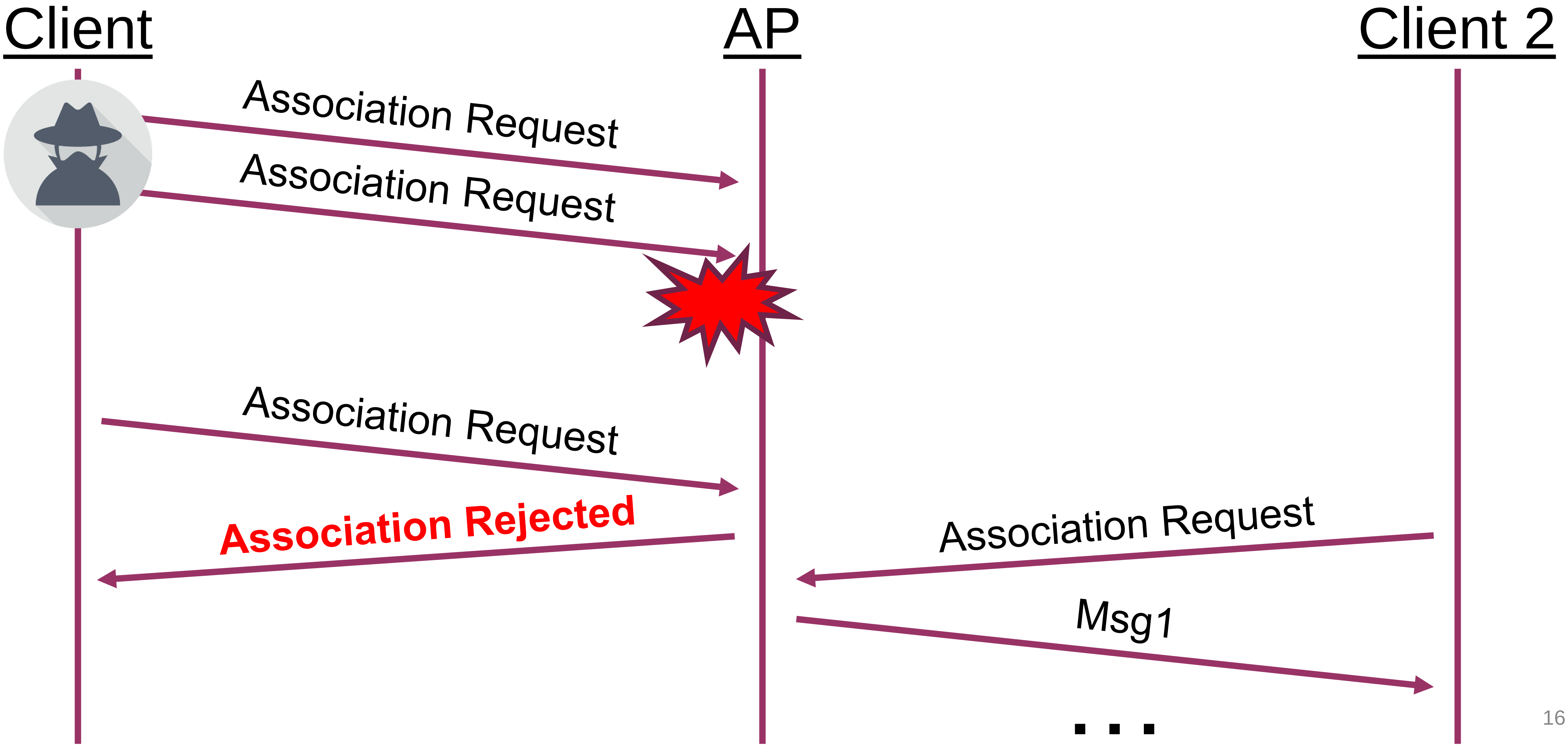
Missing downgrade checks

1. MediaTek & Telenet don't verify selected cipher in message 2
2. MediaTek also ignores supported ciphers in message 3

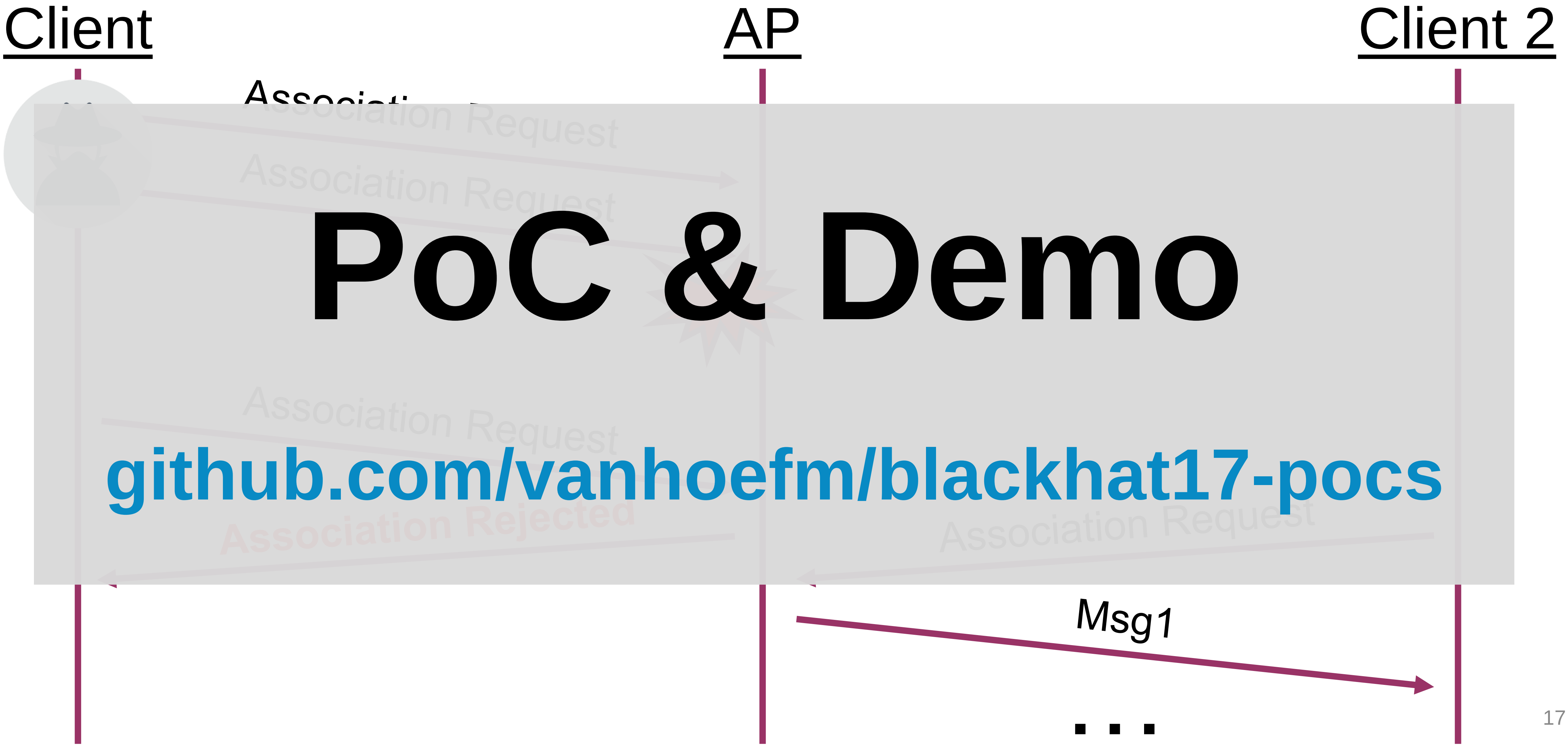


→ MediaTek clients can be trivially downgraded

Windows 7 targeted DoS



Windows 7 targeted DoS



PoC & Demo

github.com/vanhoefm/blackhat17-pocs

Broadcom downgrade

Broadcom cannot distinguish message 2 and 4

- Can be abused to downgrade the AP to TKIP

Hence message 4 is essential in preventing downgrade attacks

- This highlights incorrect claims in the 802.11 standard:

"**While Message 4 serves no cryptographic purpose**, it serves as an acknowledgment to Message 3. **It is required to ensure reliability** and to inform the Authenticator that the Supplicant has installed the PTK and GTK and hence can receive encrypted frames."

OpenBSD: DoS against AP

Two bugs in OpenBSD:

1. TKIP countermeasures are never stopped
   - TKIP is weak: detects frame forging attempts
   - Possible forge attempt → send MIC failure report to AP

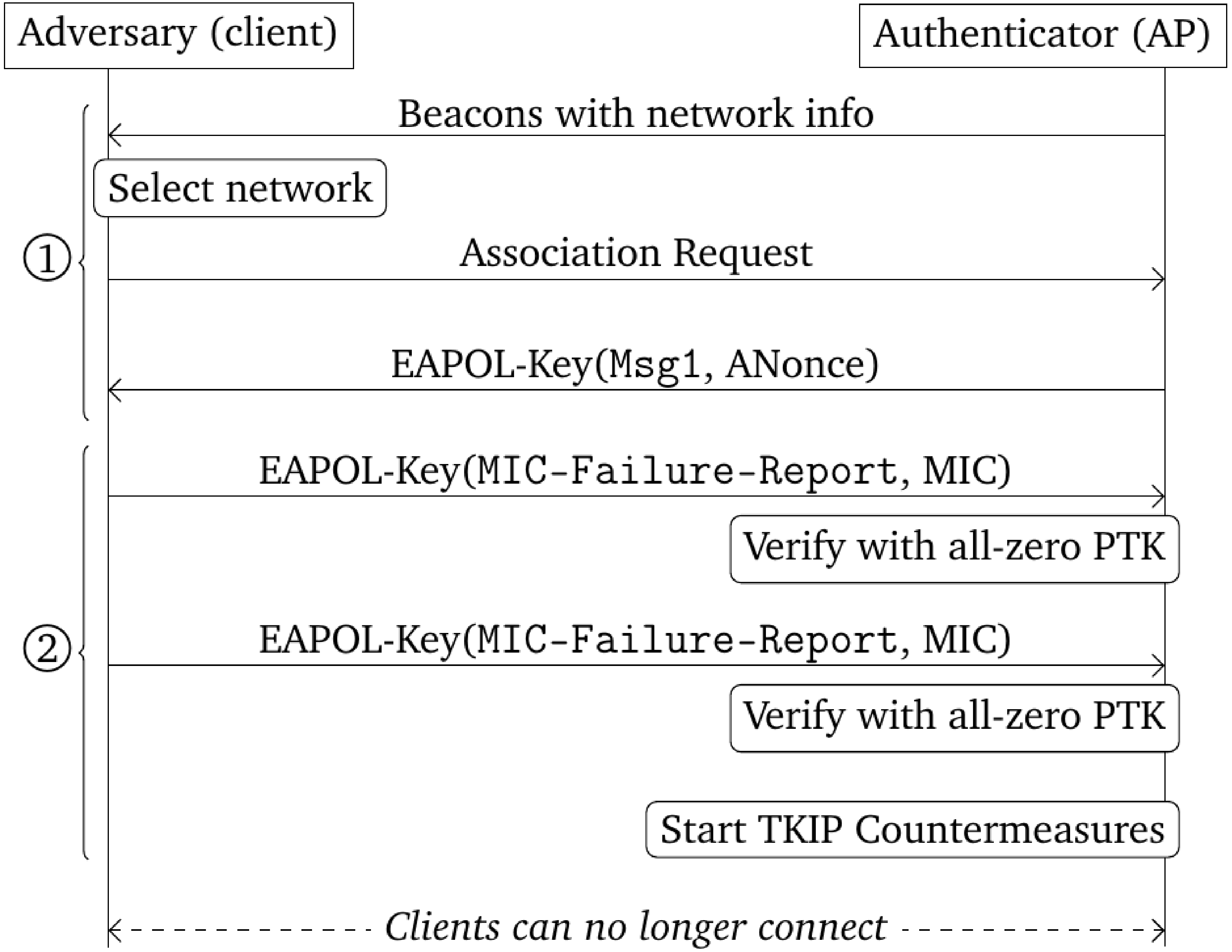
If (two MIC failure reports within a minute)

halt all traffic ~~for 1 minute~~ forever

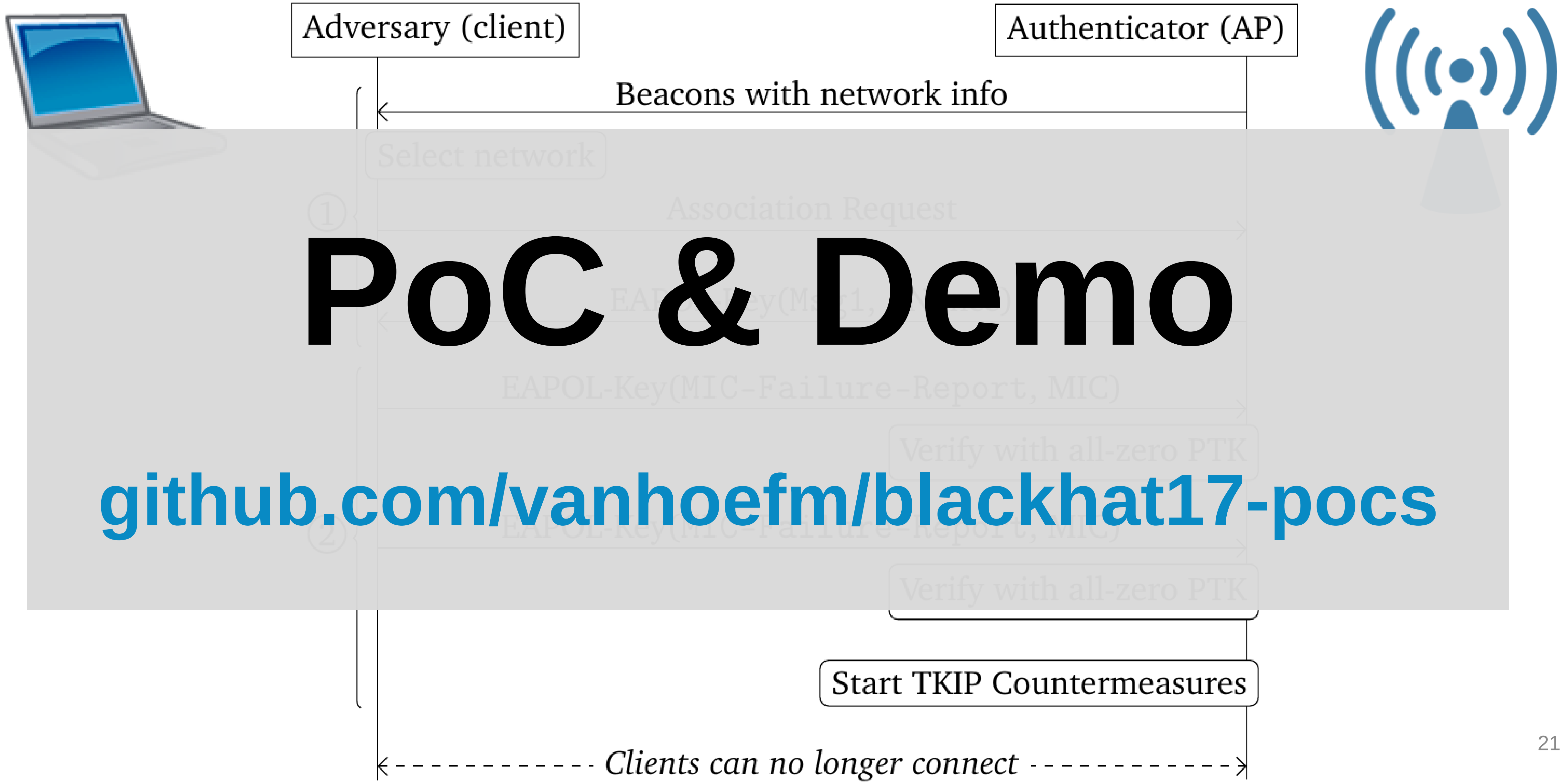
2. MIC failure report accepted before 4-way handshake

Combined: unauthenticated permanent DoS

OpenBSD: DoS against AP



OpenBSD: DoS against AP



PoC & Demo

github.com/vanhoefm/blackhat17-pocs

OpenBSD: client man-in-the-middle

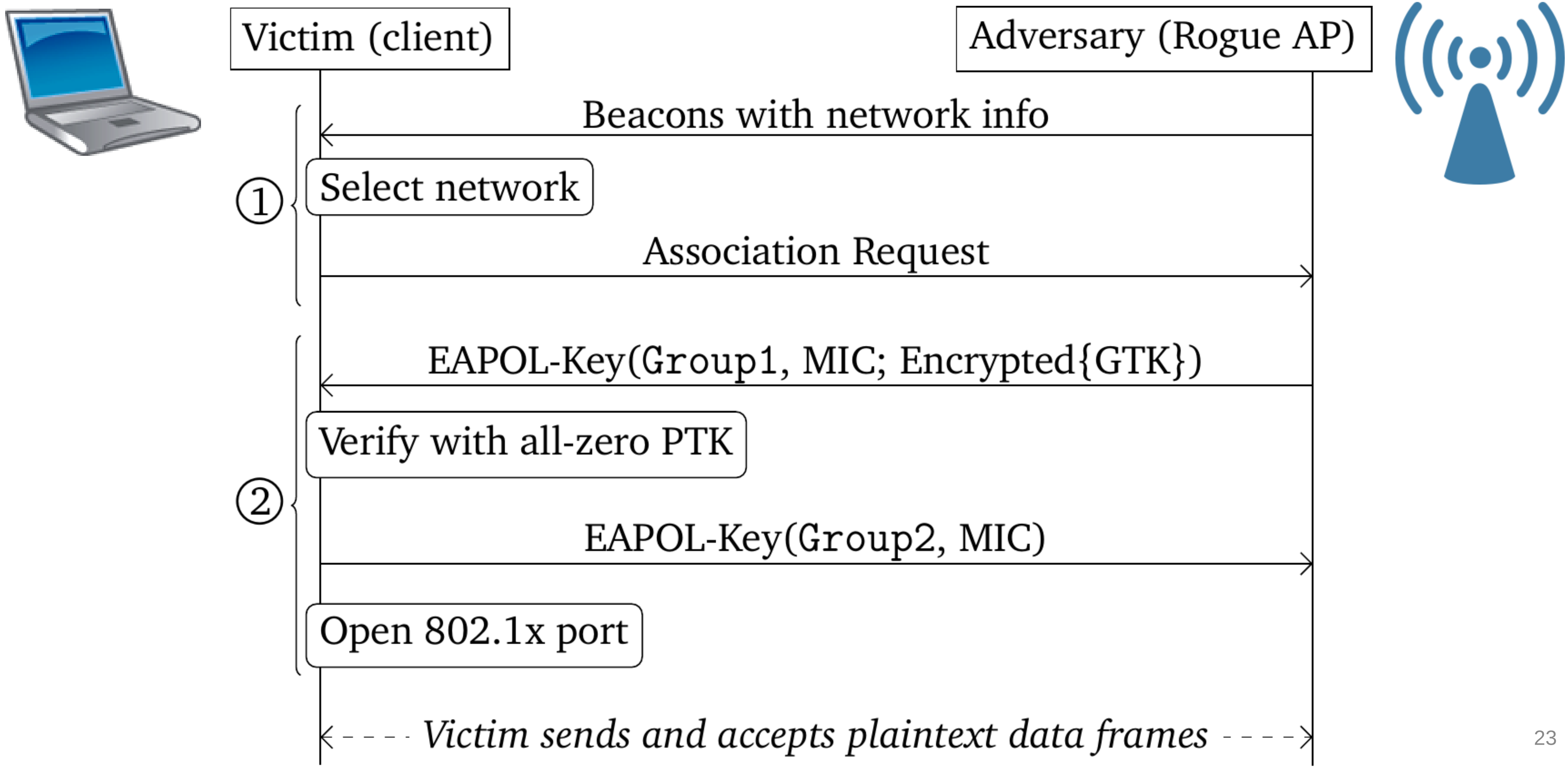
Manual inspection of OpenBSD client.

State machine missing! Attack possible:

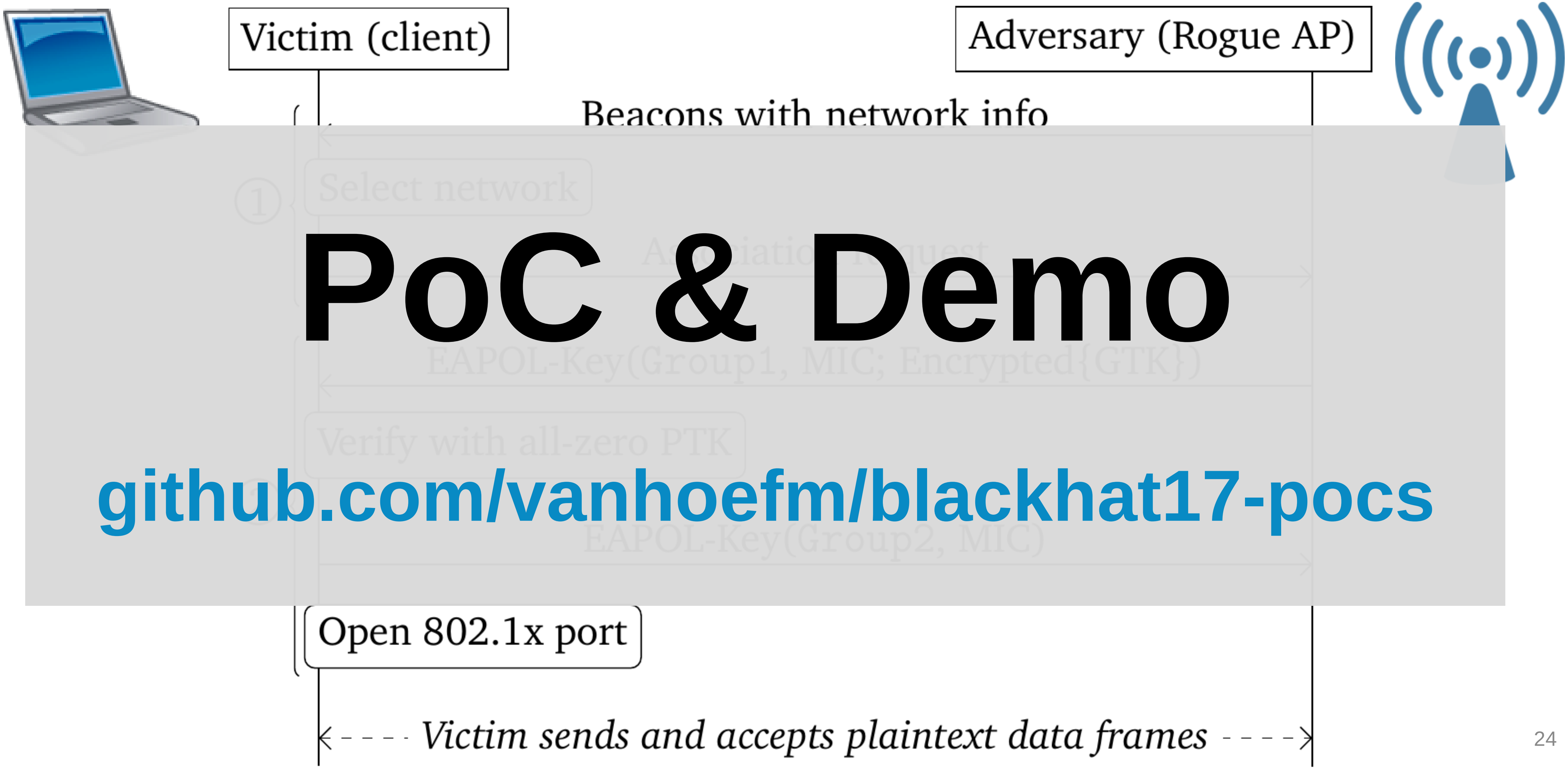
1. Rouge AP: skip 4-way handshake, send Group Message 1
2. Client verifies authenticity of message using all-zero key
3. Message accepted, client now allows normal data traffic

Proof of concept and demo!

OpenBSD: client man-in-the-middle



OpenBSD: client man-in-the-middle



PoC & Demo

github.com/vanhoefm/blackhat17-pocs

Other results: see white paper!

- Fingerprinting techniques!
- Permanent DoS attack against Broadcom
- DoS attack against Windows 10, Broadcom, Aerohive
- Inconsistent parsing of selected and supported cipher suite(s)
- …

Technique (Dis)advantages & Limitations

General remaks:

- ✓ Black-box testing mechanism: no source code needed
- ✓ Fairly simple handshake, but still several logical bugs!
- ○ But time consuming to implement & requires an expert

Limitations:

- Amount of code coverage is unknown
- Only used well-formed (albeit invalid) packets
- Test generation rules applied independently
- Only tested Access Points (not clients)

Conclusion

Wi-Fi implementations are less secure than expected

- New attacks (will) keep popping up

Need more advanced tools to detect logical flaws

- Current testing framework is quite basic
- Complex bugs currently remain undetected

WiFuzz: Detecting and Exploiting Logical Flaws in the Wi-Fi Cryptographic Handshake

Mathy Vanhoef - @vanhoefm
imec-DistriNet, KU Leuven

Questions?