FragAttacks

Breaking Wi-Fi Through Frame Aggregation and Fragmentation

4-5 August, Black Hat USA (Virtual)

Mathy Vanhoef

NEW YORK UNIVERSITY

Advancements in Wi-Fi security

1999

Wired Equivalent Privacy (WEP)

- Horribly **broken** [FMS01]

Early 2000

Wi-Fi Protected Access (WPA and WPA2)

- Offline **dictionary attacks**
- KRACK and Kraken attack [VP17,VP18]
- KRACK defenses now proven secure [CKM20]

Advancements in Wi-Fi security

2018 Wi-Fi Protected Access 3 (WPA3)

Uses a **new handshake** to prevent dictionary attacks

- Vulnerable to Dragonblood: side-channel leaks [VR20]
- WPA3 certification updated to require defenses [WFA20]

Once connected, the **encryption of WPA2 & WPA3 is similar**

- The attacks in this presentation work against both

Advancements in Wi-Fi security

Late 2020 Two extra defenses standardized

- Operating channel validation [VBDOP18]
- Beacon protection [VAP20]

Would make presented **attacks harder but still possible**

- Still undergoing adoption → currently no practical impact

Advancements in Wi-Fi security

Despite these major advacements,

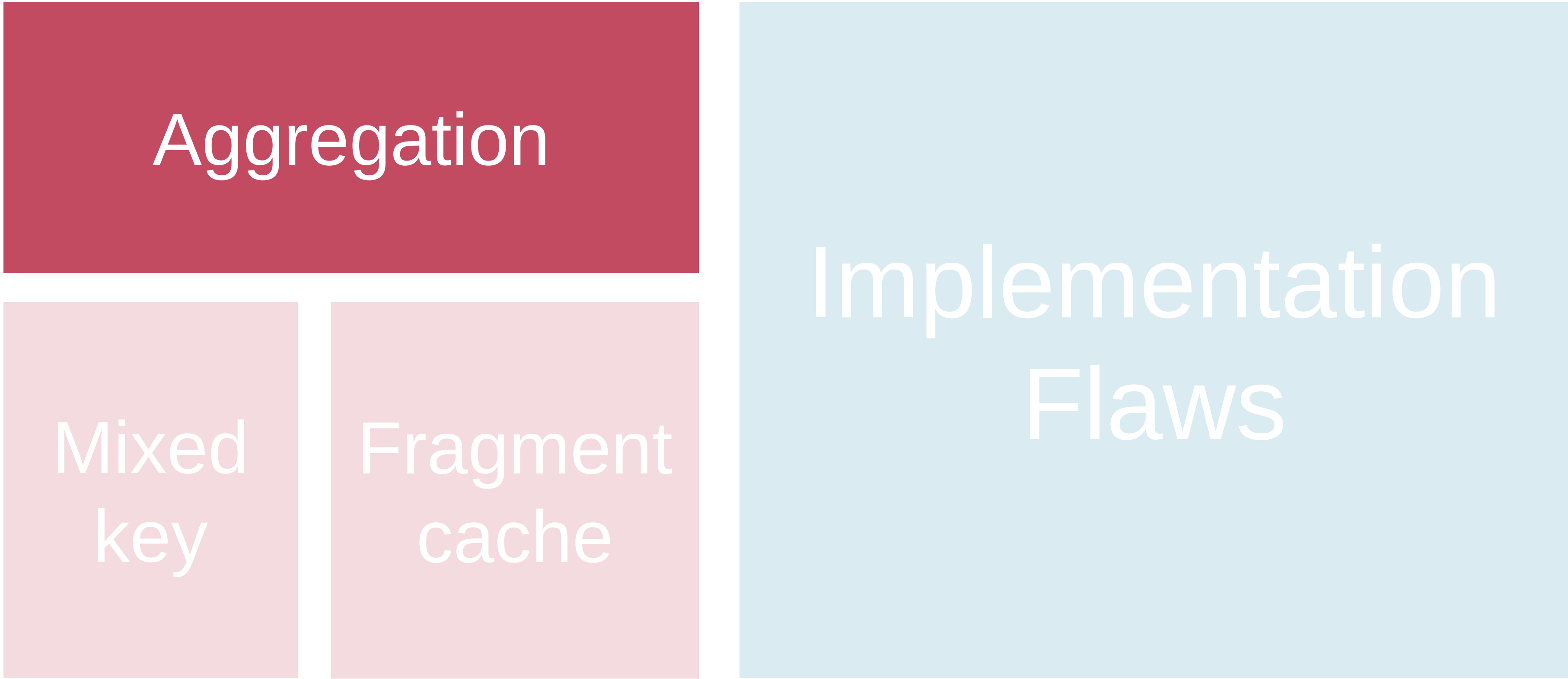
found new **flaws in all networks**

Design flaws

Implementation Flaws

Design flaws

Implementation Flaws



Aggregation

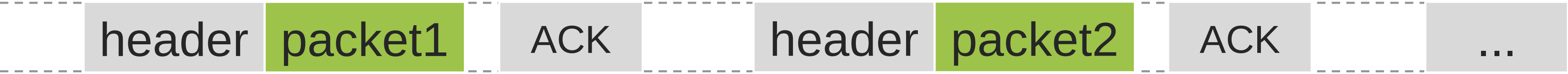
Mixed key

Fragment cache

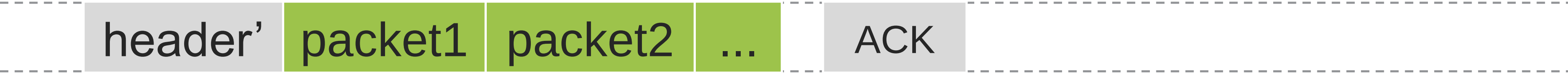
Implementation Flaws

Background

Sending small frames causes high overhead:

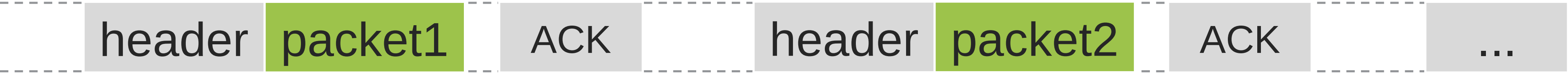


This can be avoided by **aggregating frames**:



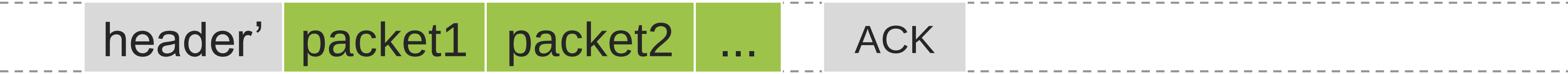
Background

Sending small frames causes high overhead:



header packet1 ACK header packet2 ACK ...

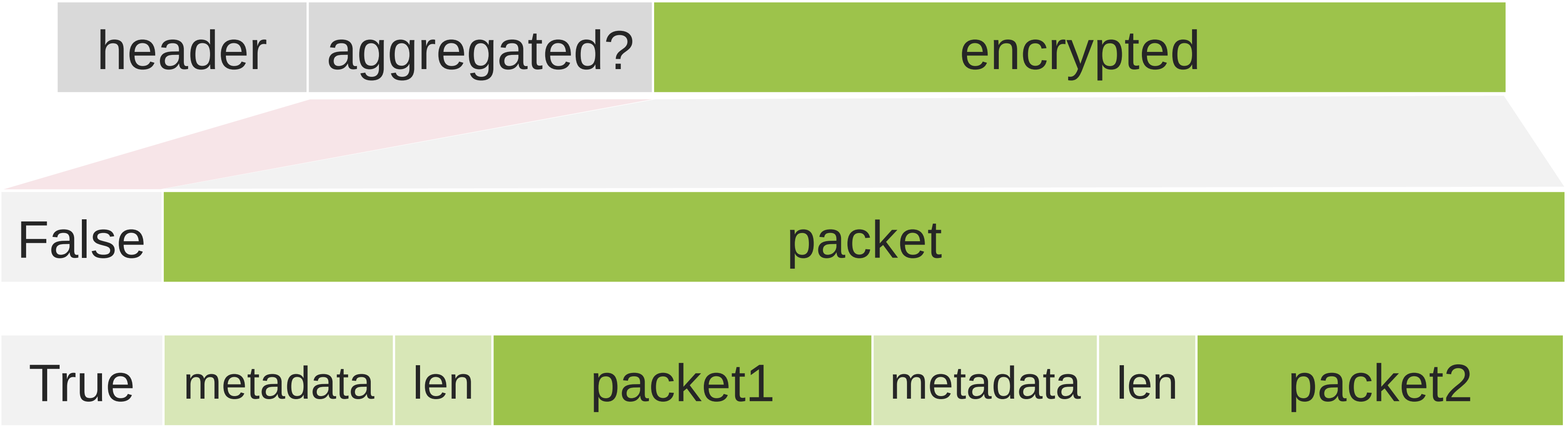
This can be avoided by **aggregating frames**:



header’ packet1 packet2 ... ACK

Problem: how to recognize aggregated frames?

Aggregation design flaw

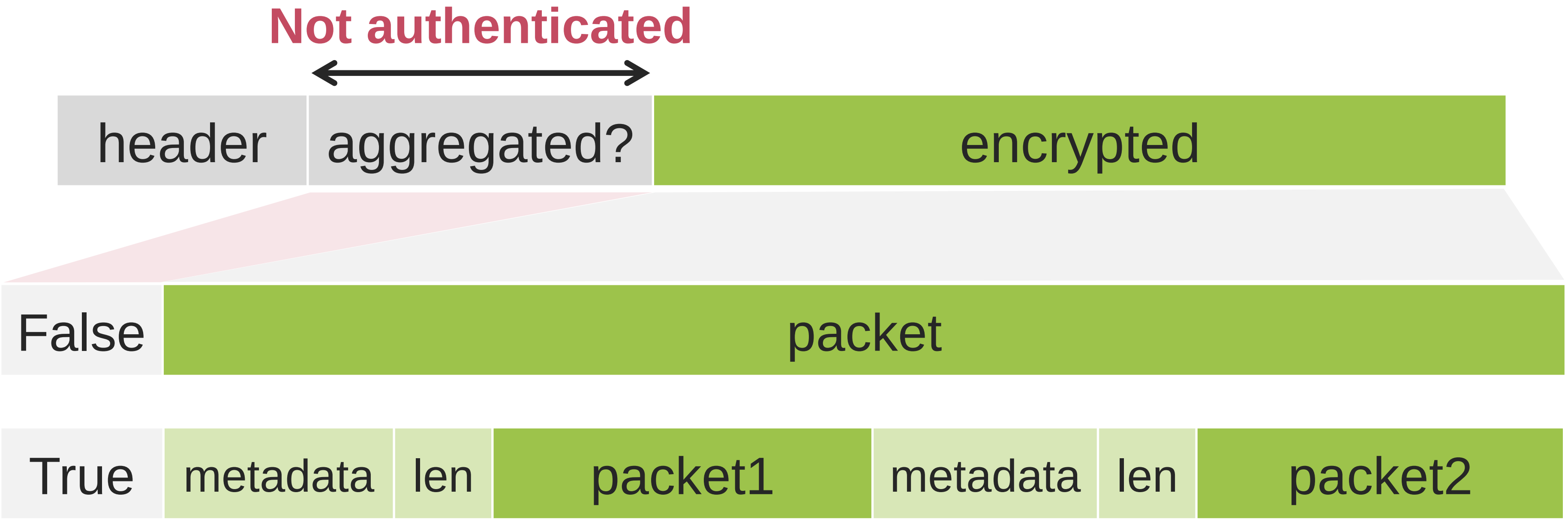


header | aggregated? | encrypted

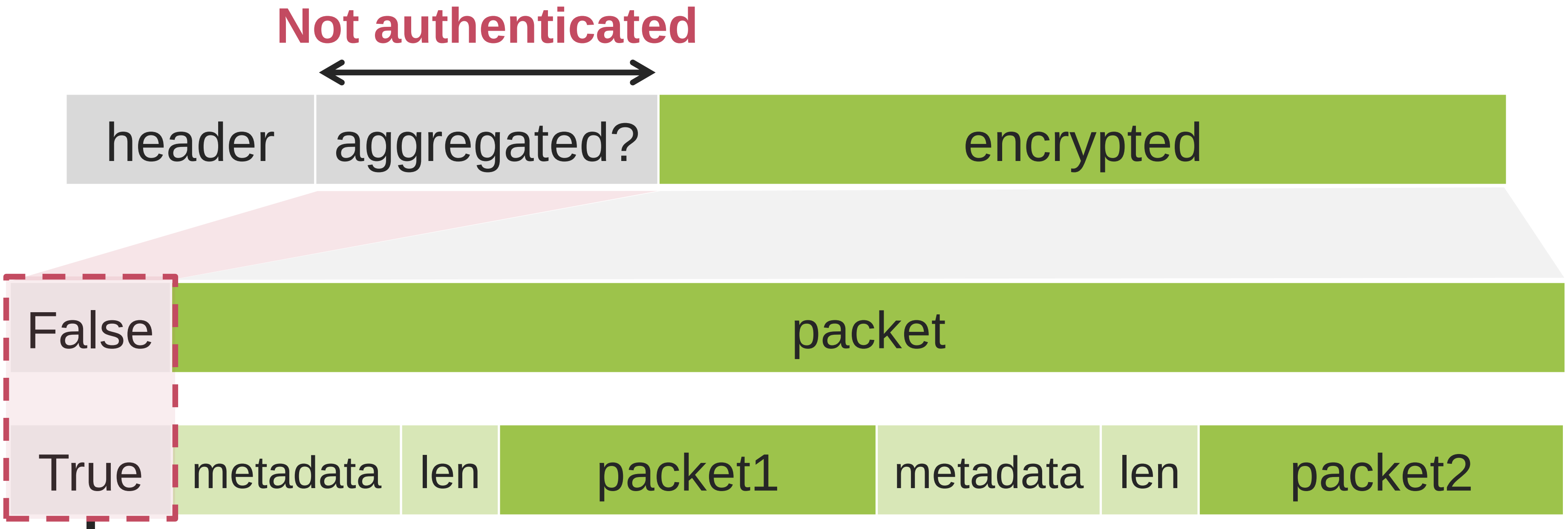
False | packet

True | metadata | len | packet1 | metadata | len | packet2

Aggregation design flaw



Aggregation design flaw



Not authenticated

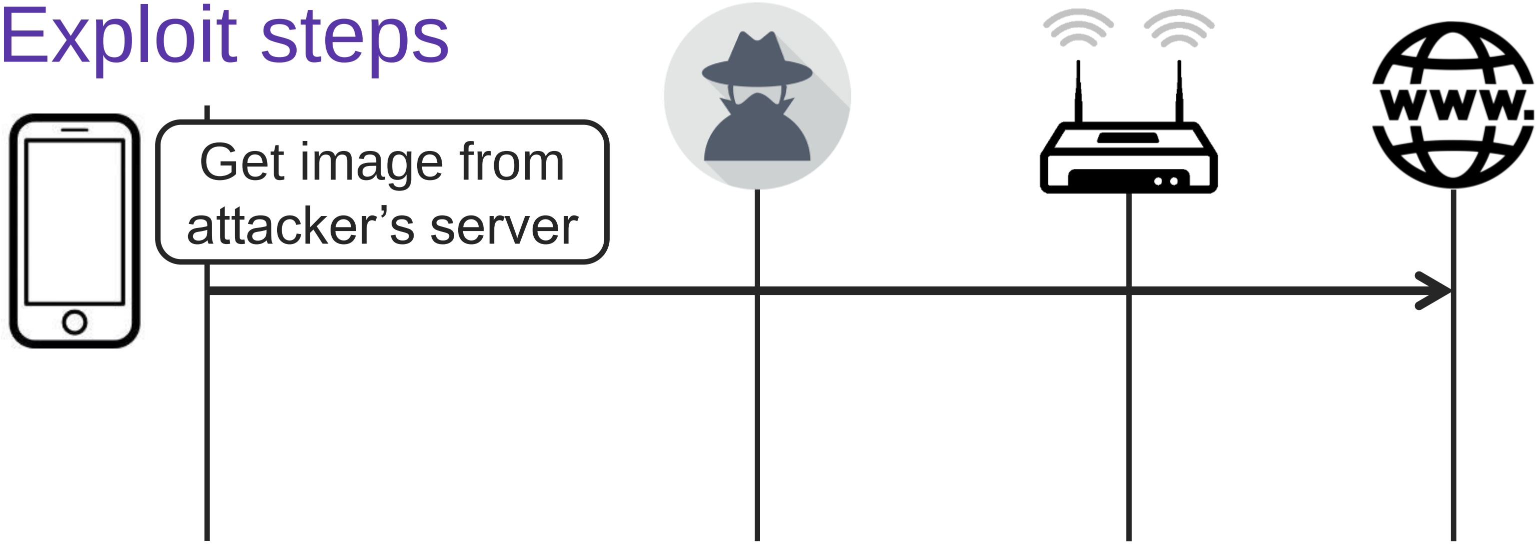
header | aggregated? | encrypted

False | packet

True | metadata | len | packet1 | metadata | len | packet2

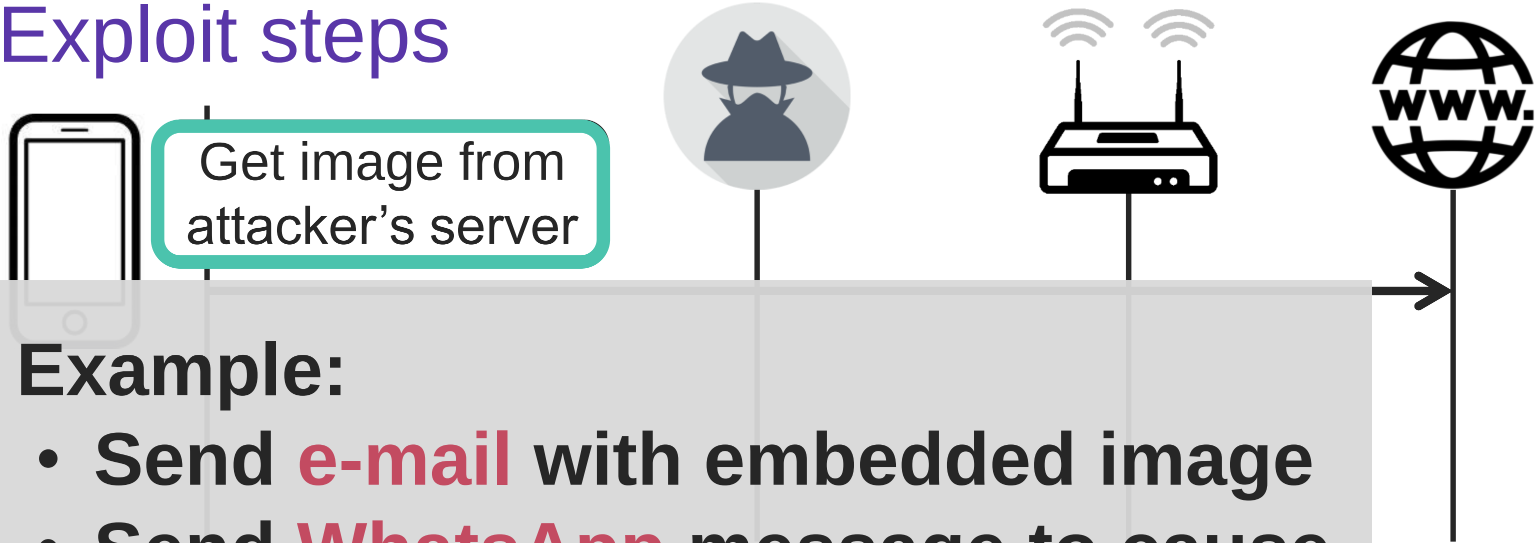
Flip flag → decrypted payload is parsed in wrong manner

Exploit steps



Get image from attacker's server

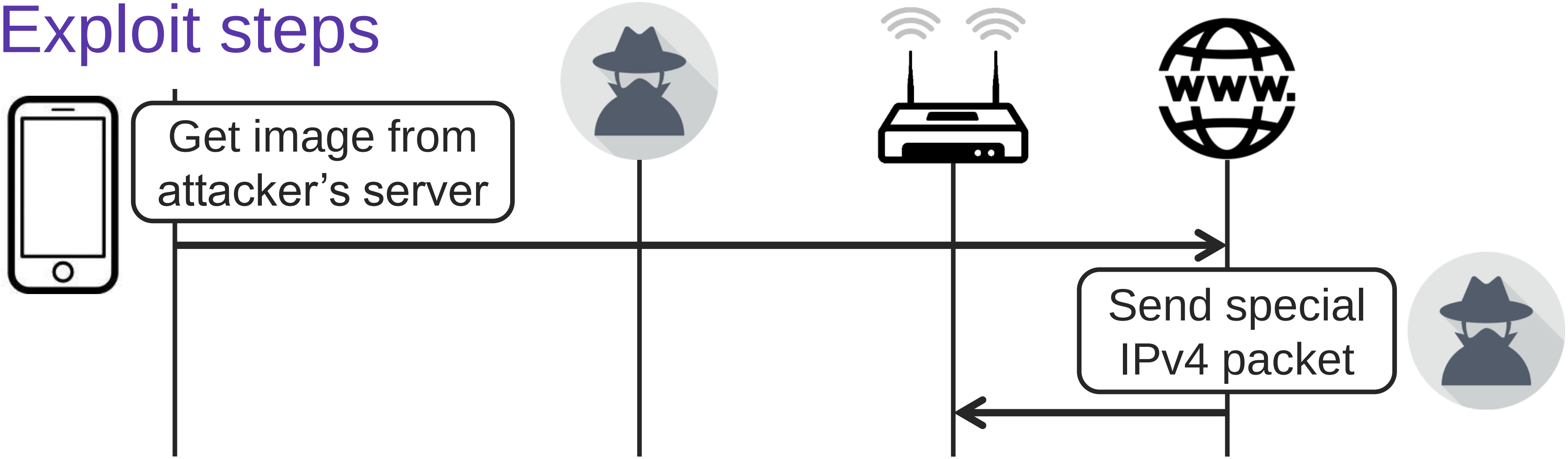
Exploit steps



Example:

- **Send e-mail with embedded image**
- **Send WhatsApp message to cause link/image preview**

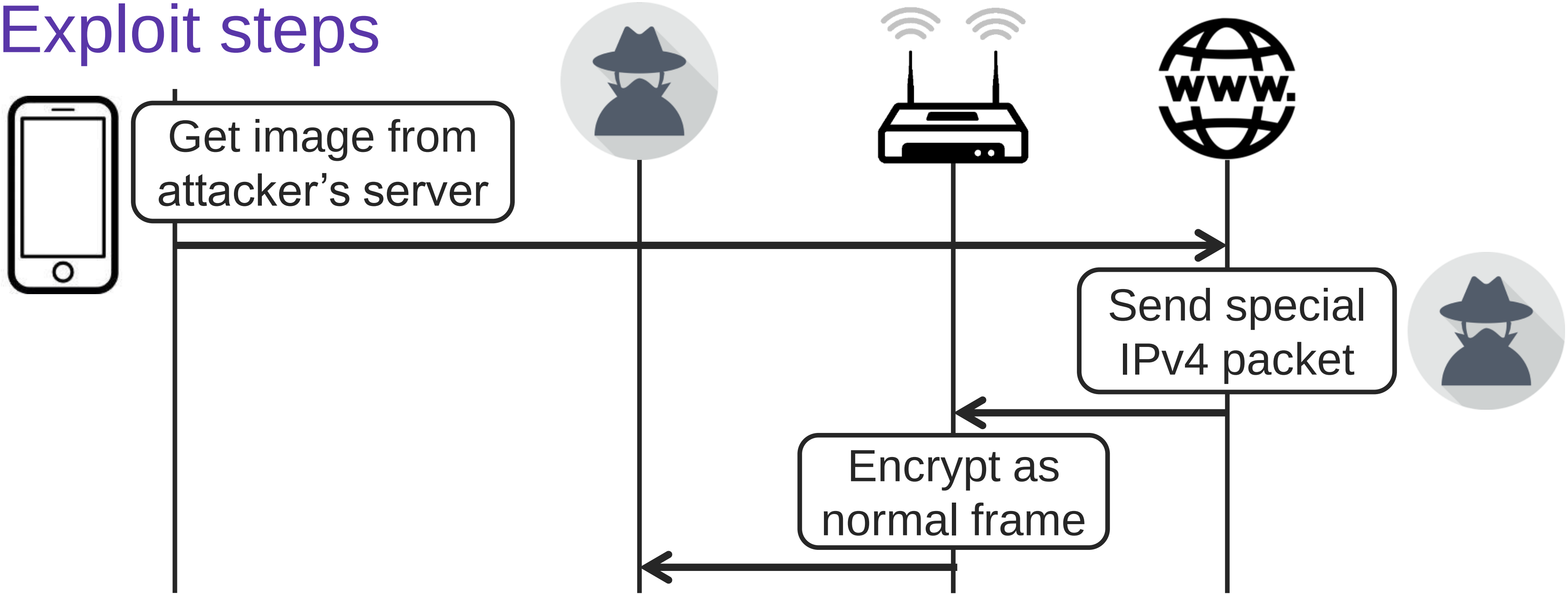
Exploit steps



Get image from attacker’s server

Send special IPv4 packet

Exploit steps

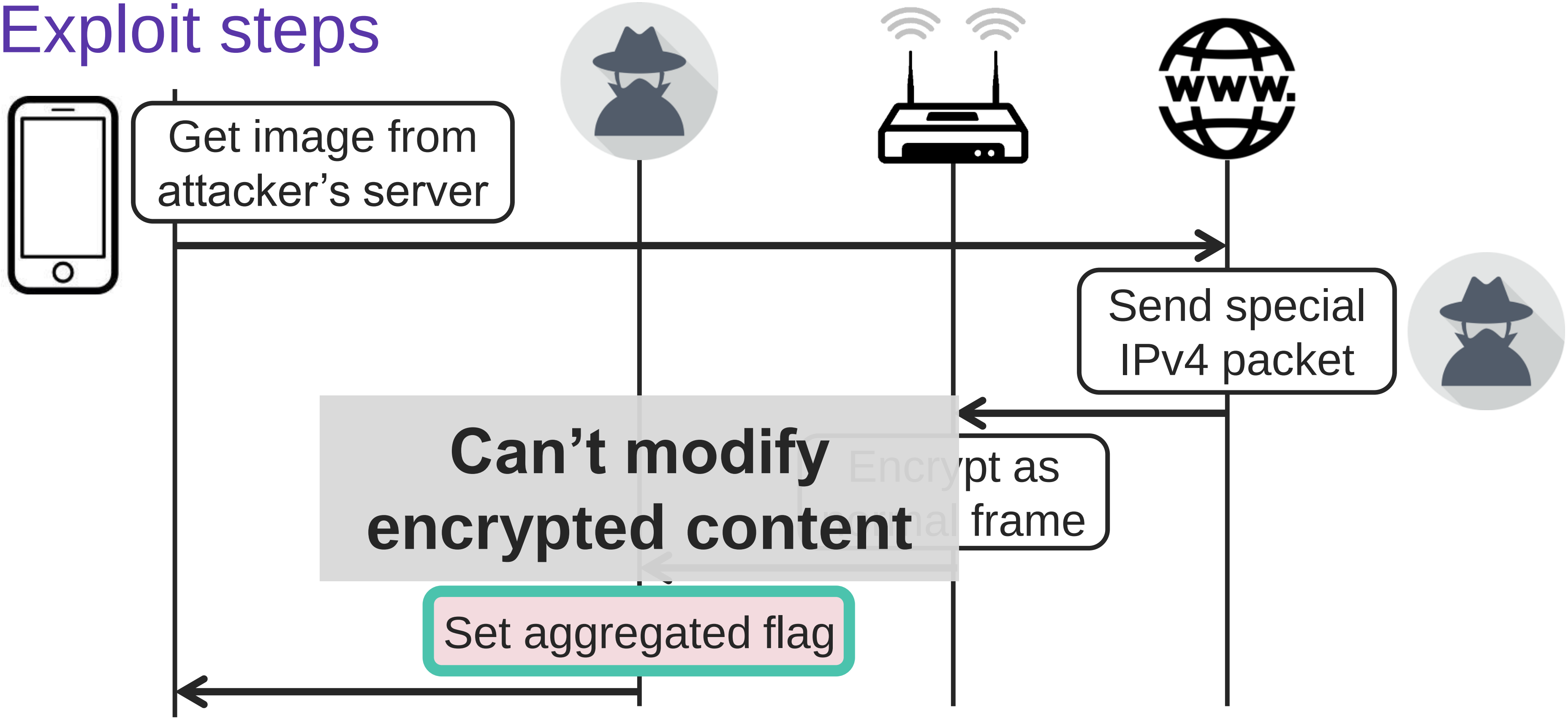


Get image from attacker’s server

Send special IPv4 packet

Encrypt as normal frame

Exploit steps



Get image from attacker's server

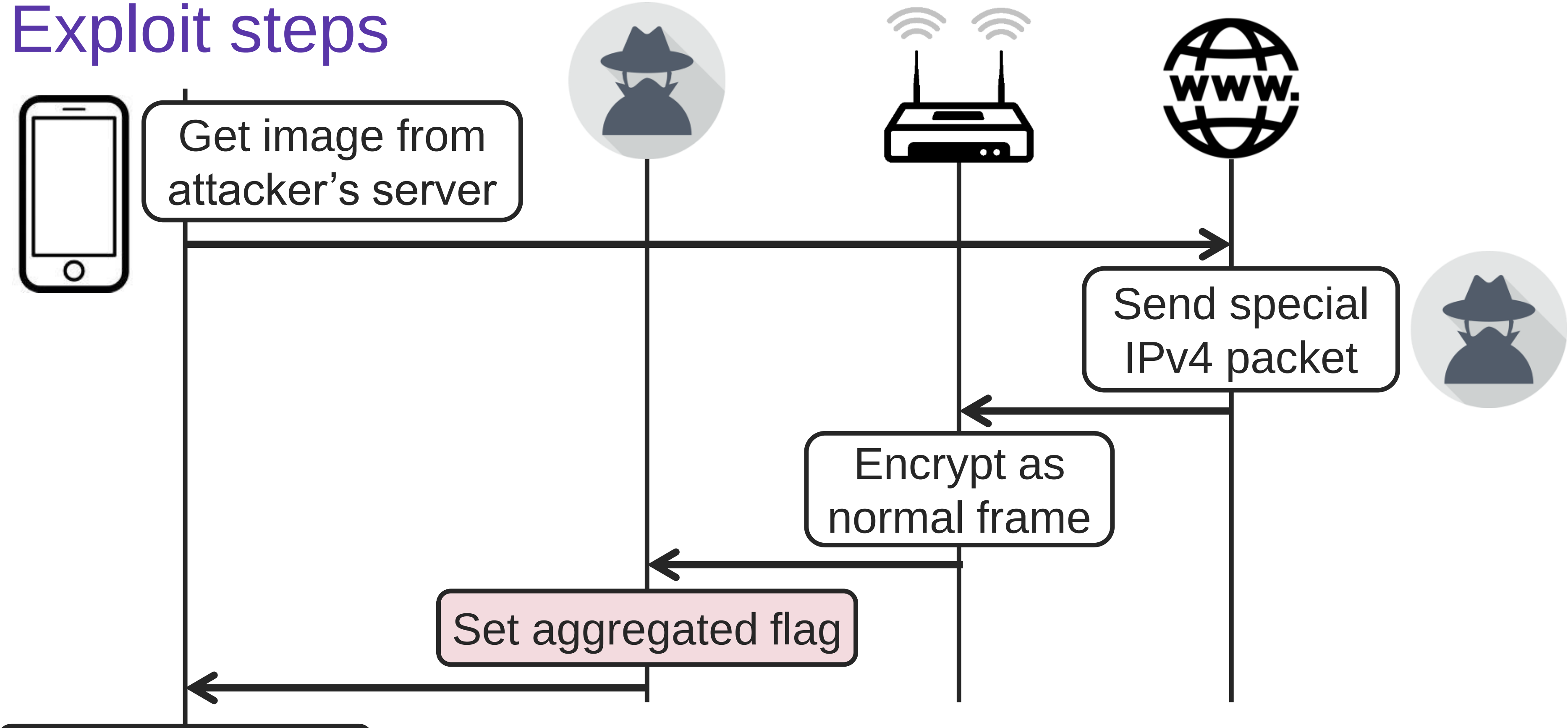
Send special IPv4 packet

Encrypt as normal frame

Can't modify encrypted content

Set aggregated flag

Exploit steps



Get image from attacker's server

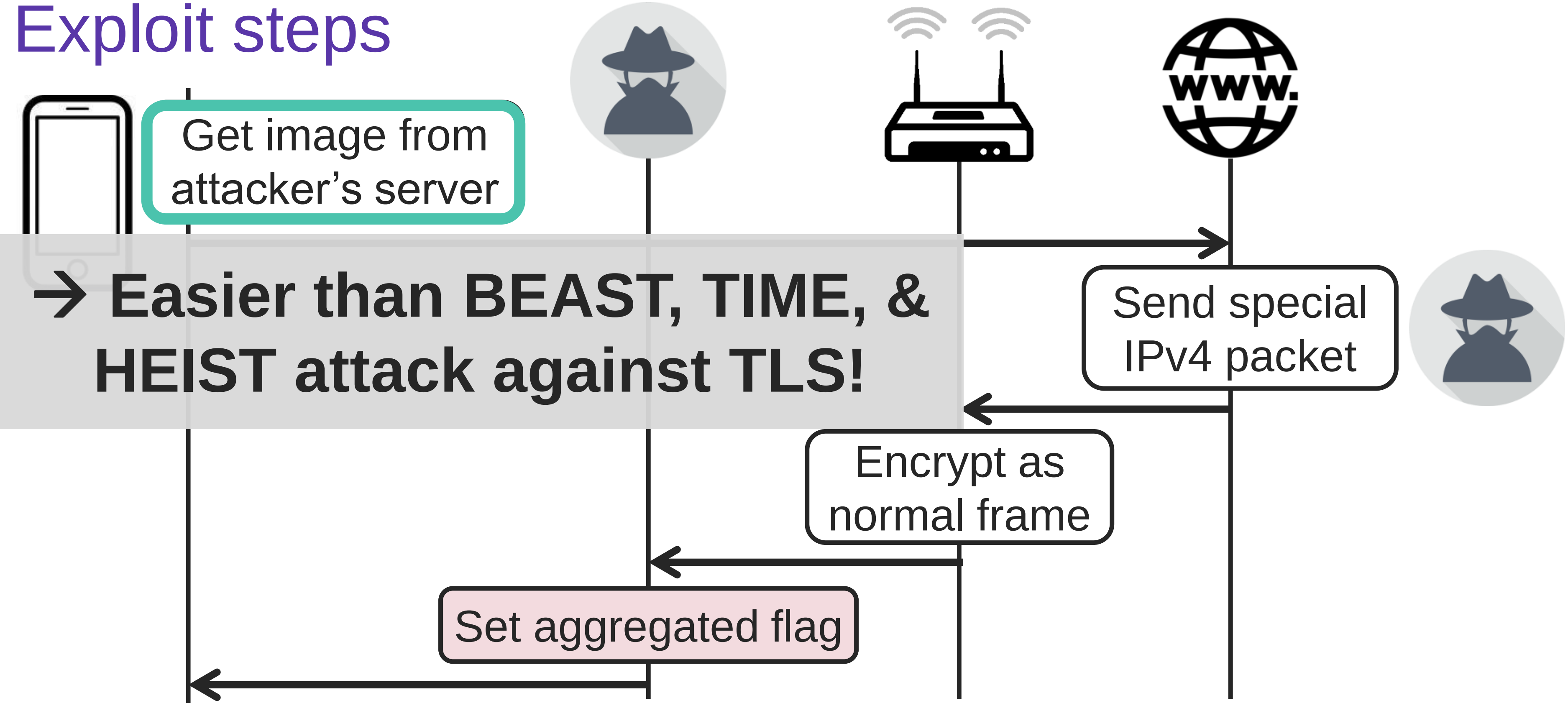
Send special IPv4 packet

Encrypt as normal frame

Set aggregated flag

Inject any packet → **Inject** ICMPv6 RA with **malicious DNS server**

Exploit steps



Get image from attacker's server

Send special IPv4 packet

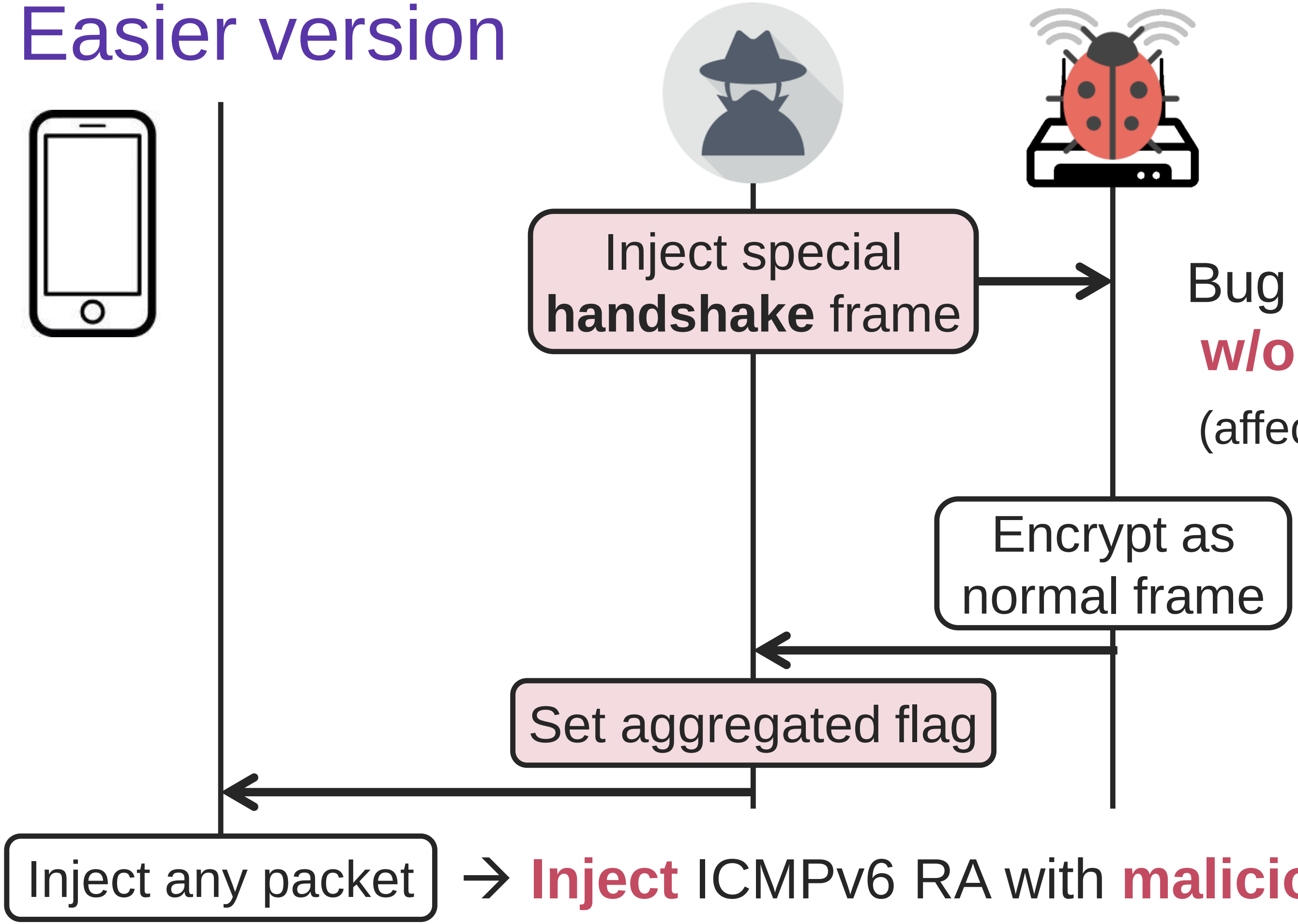
Encrypt as normal frame

Set aggregated flag

→ Easier than BEAST, TIME, & HEIST attack against TLS!

Inject any packet → **Inject** ICMPv6 RA with **malicious DNS server**

Easier version



Inject special **handshake** frame

Encrypt as normal frame

Set aggregated flag

Inject any packet → **Inject** ICMPv6 RA with **malicious DNS server**

Bug in AP → do attack **w/o user interaction**

(affected $^2/_4$ of home APs)

DEMO! ☺

Impact

All major operating systems affected

Only NetBSD & some IoT devices unaffected

How to construct the special IPv4/TCP packet?

hdr	len	ID		TCP	data
45 00	01 0C	00 22	...	...	Frame to inject

How to construct the special IPv4/TCP packet?

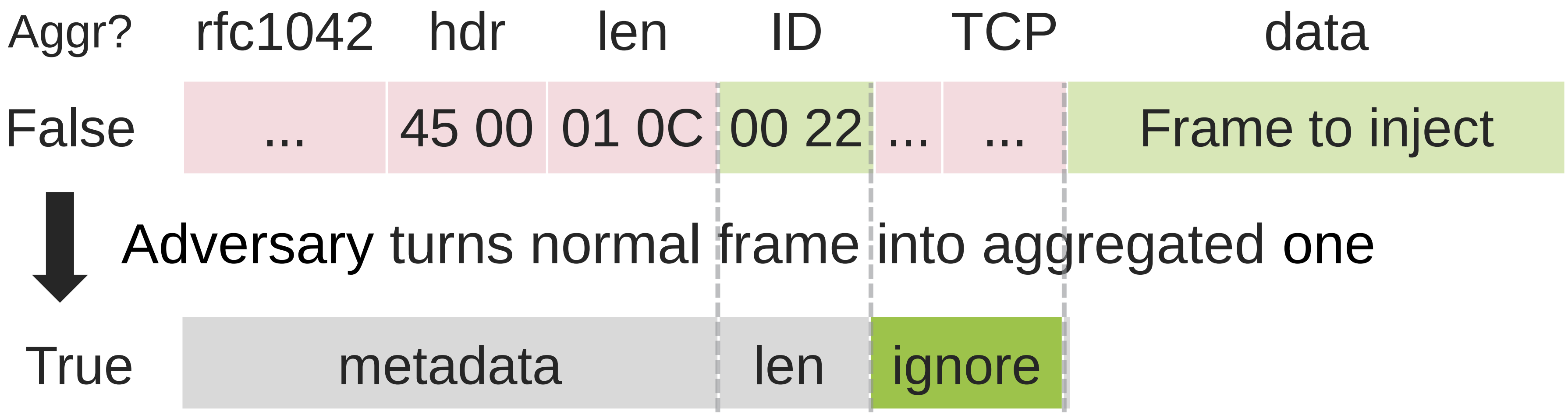
Aggr?	rfc1042	hdr	len	ID		TCP	data
False	...	45 00	01 0C	00 22	...	...	Frame to inject

How to construct the special IPv4/TCP packet?

Aggr?	rfc1042	hdr	len	ID		TCP	data
False	...	45 00	01 0C	00 22	...	...	Frame to inject

Adversary turns normal frame into aggregated one

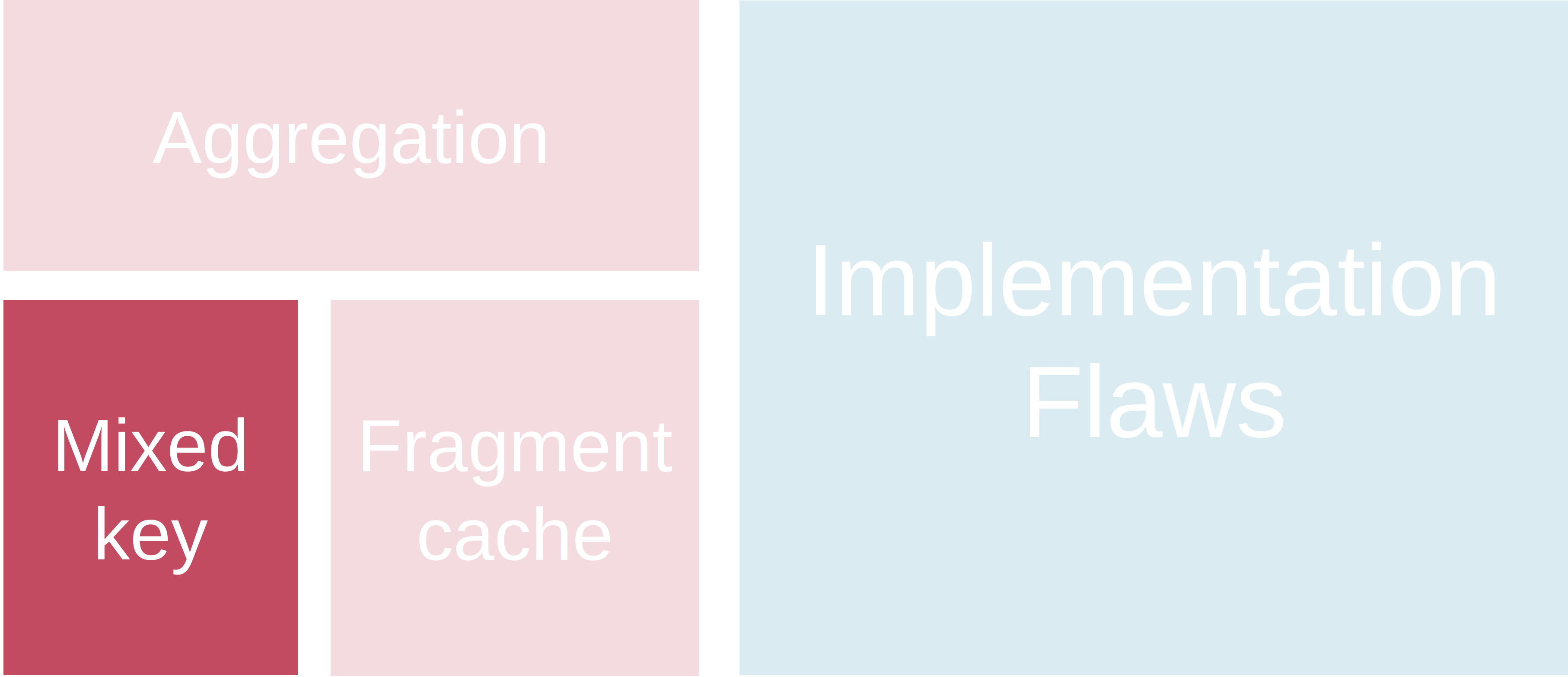
How to construct the special IPv4/TCP packet?



Aggr?	rfc1042	hdr	len	ID	TCP		data
False	...	45 00	01 0C	00 22	...	...	Frame to inject
True	metadata			len	ignore		

Adversary turns normal frame into aggregated one

- At Wi-Fi layer 1st sub-packet is ignored
- **Control IP ID** & part of **TCP data** → **inject arbitrary packets**



Aggregation

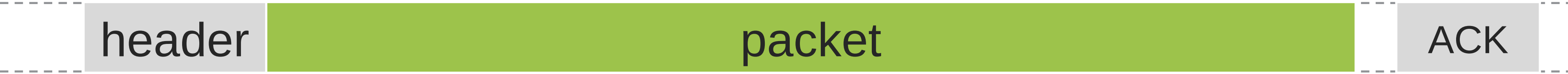
Mixed key

Fragment cache

Implementation Flaws

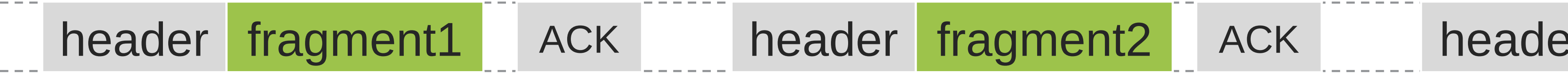
Background

Large frames have a high chance of being corrupted:



header packet ACK

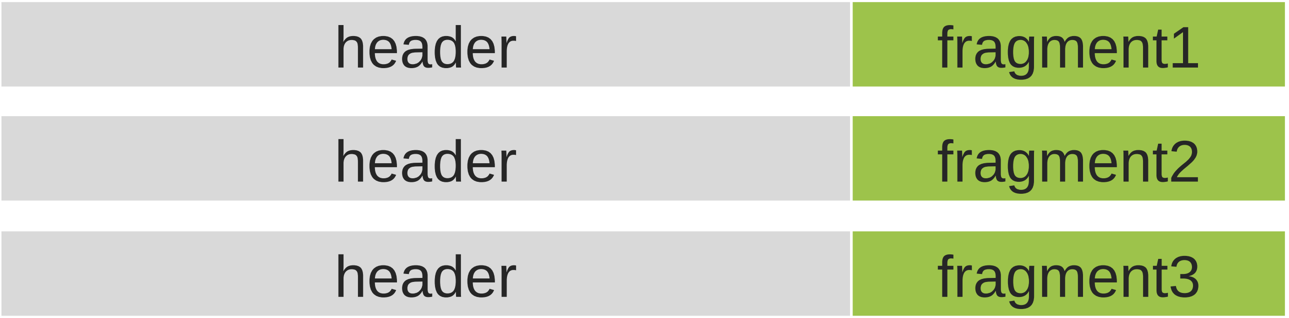
Avoid by **fragmenting** & only retransmitting lost fragments:



header fragment1 ACK header fragment2 ACK heade

Problem: **how to (securely) reassemble** the fragments?

Reassembling **plaintext** fragments



header	fragment1
header	fragment2
header	fragment3

Reassembling plaintext fragments

header	s	fragment1
header	s	fragment2
header	s	fragment3

- Fragments have the **same sequence number s**

Reassembling plaintext fragments

header	s	0	fragment1
header	s	1	fragment2
header	s	2	fragment3

- Fragments have the **same sequence number s**
- All fragments also have a **fragment number** ...

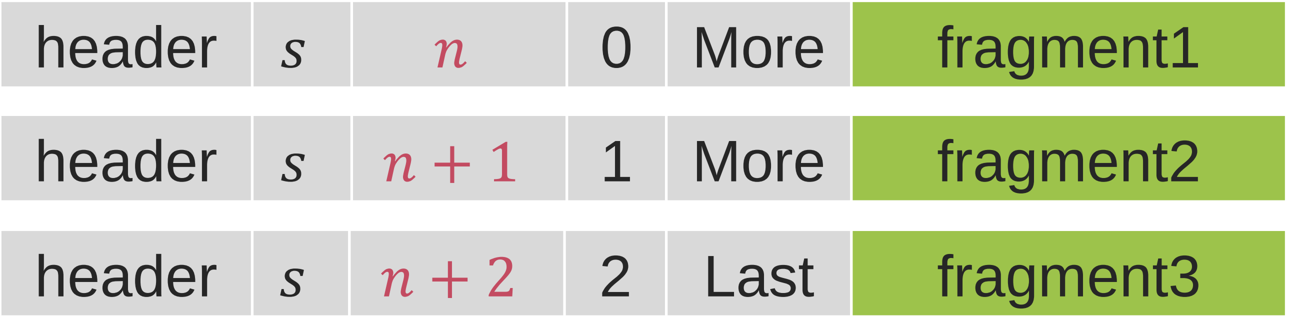
Reassembling plaintext fragments

header	*s*	0	More	fragment1
header	*s*	1	More	fragment2
header	*s*	2	Last	fragment3

- Fragments have the **same sequence number *s***
- All fragments also have a **fragment number** ...

  ... and a flag to **identify the last** fragment

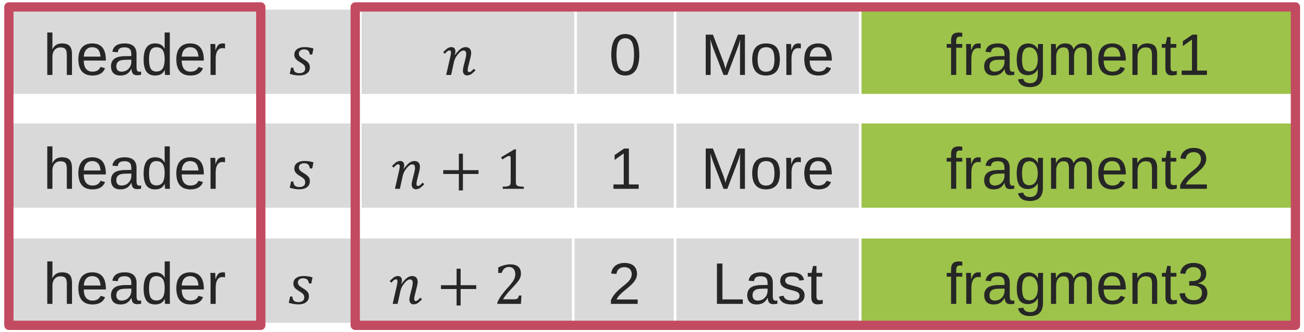
Reassembling **encrypted** fragments



header	s	n	0	More	fragment1
header	s	$n+1$	1	More	fragment2
header	s	$n+2$	2	Last	fragment3

- Encrypted frames have a **packet number** to detect replays

Reassembling encrypted fragments



header	s	n	0	More	fragment1
header	s	$n+1$	1	More	fragment2
header	s	$n+2$	2	Last	fragment3

Authenticated

Authenticated

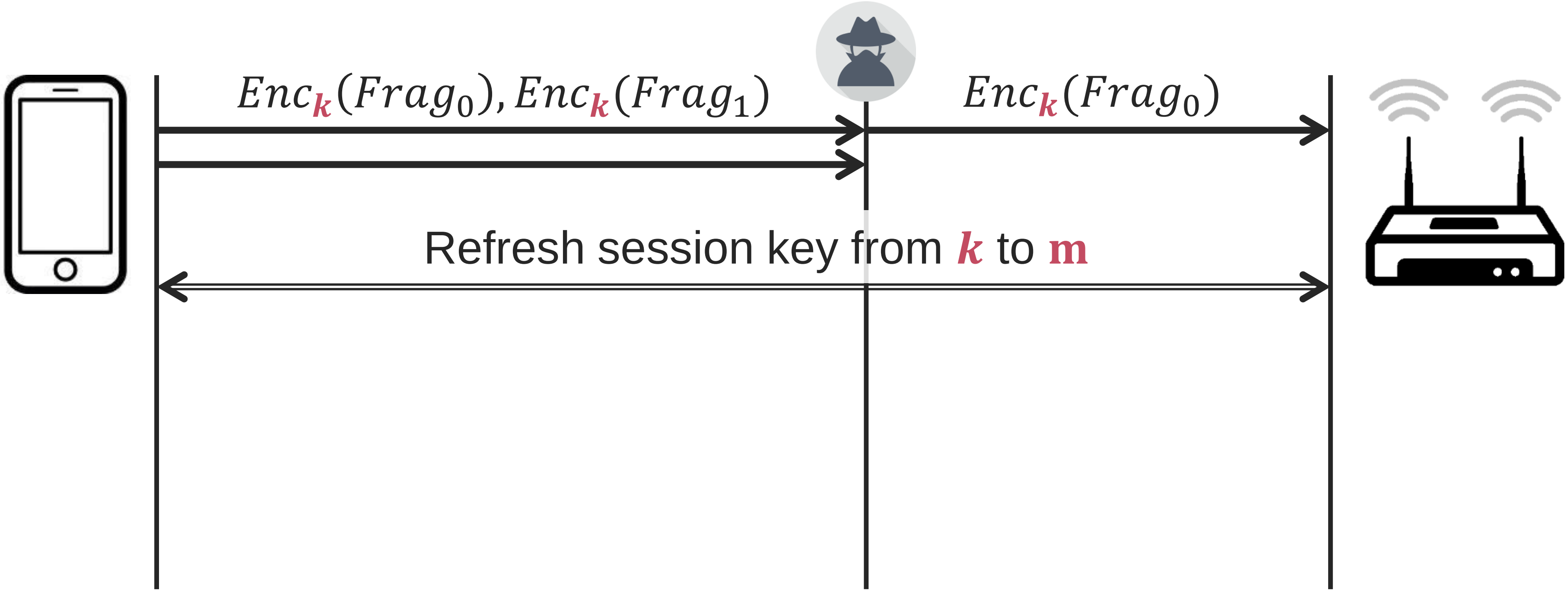
- Encrypted frames have a **packet number** to detect replays
- If packet & fragment numbers are **not consecutive, drop it**

Problem: key renewal

- Session key can be periodically renewed ...
- ... or updated when roaming between APs

- During rekey packet numbers restart from zero
- Problem: receiver is allowed to **reassemble fragments encrypted under different keys** (i.e. mixed keys)

Mixed key design flaw

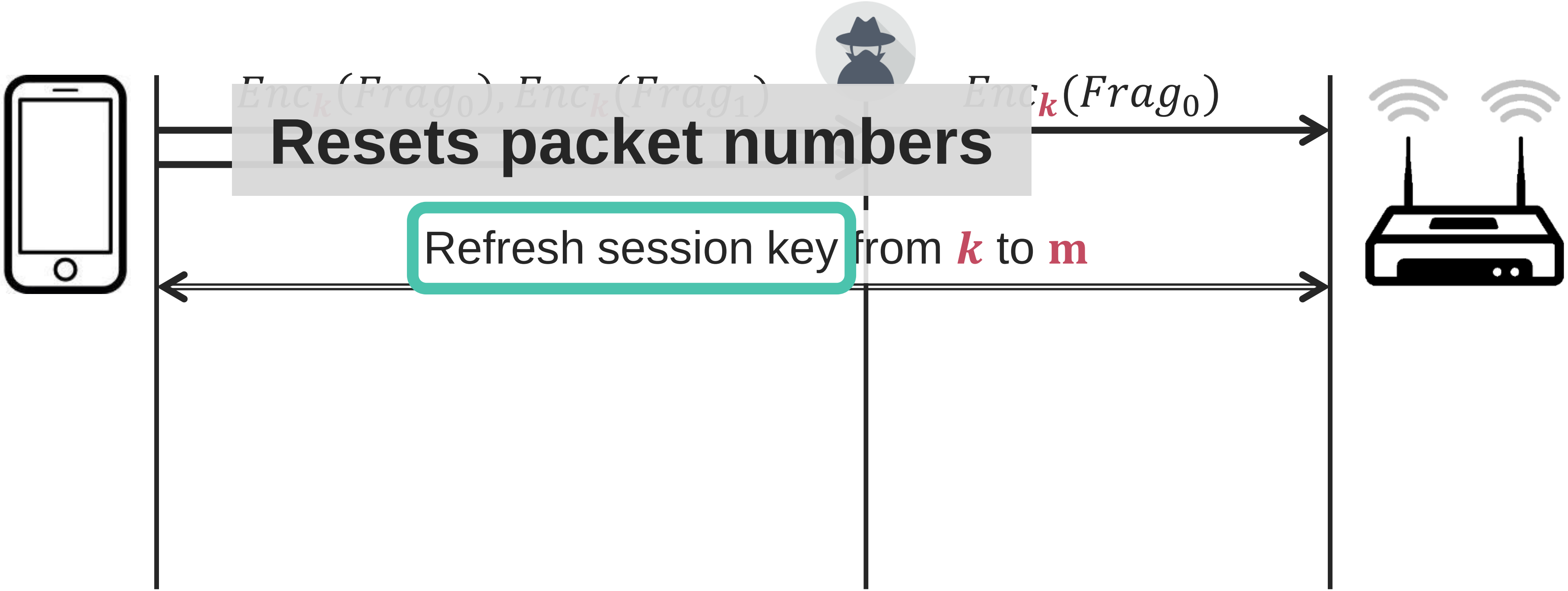


$Enc_{k}(Frag_0), Enc_{k}(Frag_1)$

$Enc_{k}(Frag_0)$

Refresh session key from $\boldsymbol{k}$ to $\mathbf{m}$

Mixed key design flaw



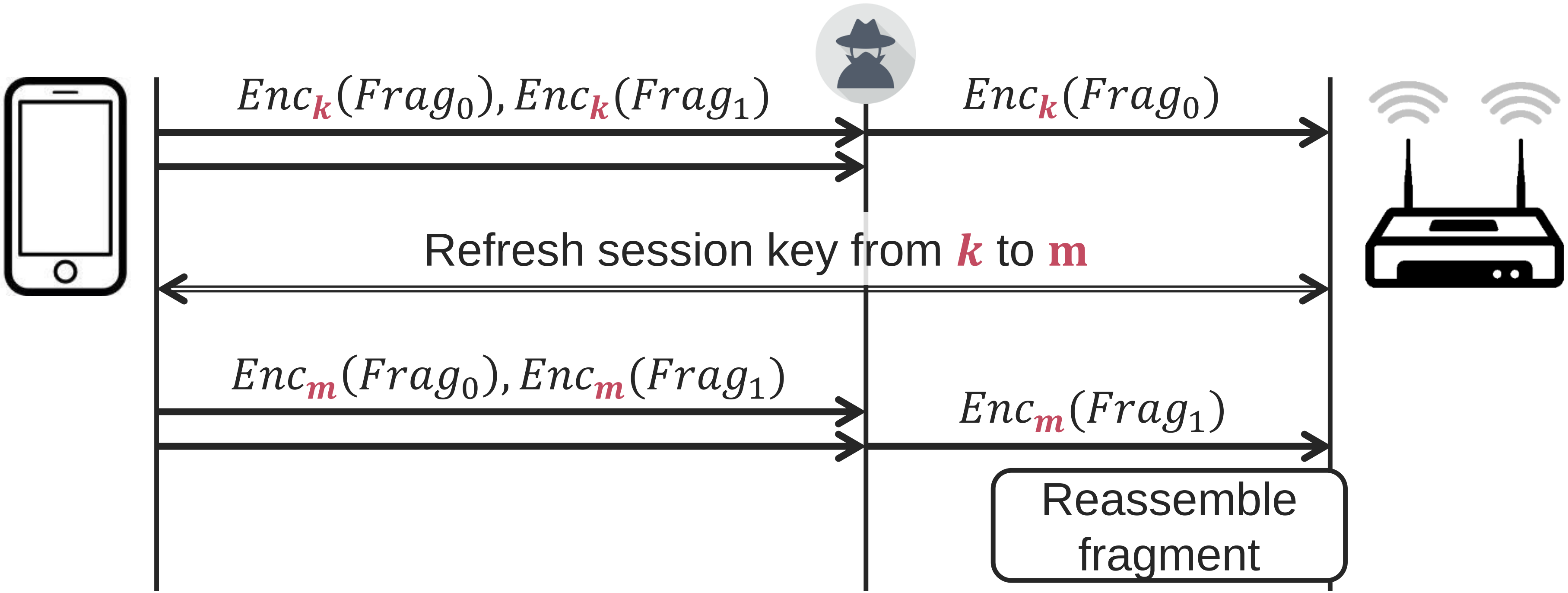
$Enc_k(Frag_0), Enc_k(Frag_1)$

$Enc_k(Frag_0)$

Resets packet numbers

Refresh session key from k to $\mathbf{m}$

Mixed key design flaw


$Enc_k(Frag_0), Enc_k(Frag_1)$
$Enc_k(Frag_0)$
Refresh session key from k to $\mathbf{m}$
$Enc_m(Frag_0), Enc_m(Frag_1)$
$Enc_m(Frag_1)$
Reassemble fragment

→ Can **mix fragments of different frames**

Summary of impact

Abuse to **exfiltrate data** assuming:

1. Someone sends fragmented frames (rare unless Wi-Fi 6)
2. Victim will connect to server of attacker
3. Network periodically refreshes the session key

Summary of impact

Abuse to **exfiltrate data** assuming:

1. Someone sends fragmented frames (rare unless Wi-Fi 6)
2. Victim will connect to server of attacker
3. ~~Network periodically refreshes the session key~~
   - **Combine with implementation flaw** to avoid this condition

How to exfiltrate data?

	$Frag_0$	$Frag_1$
Frame 1	`192.168.1.2 to 3.5.1.1`	`GET /image.png HTTP/1.1`

Frame 2	`192.168.1.2 to 8.8.8.8`	`POST /login.php HTTP/1.1`
		`user=admin&pass=SeCr3t`

How to exfiltrate data?

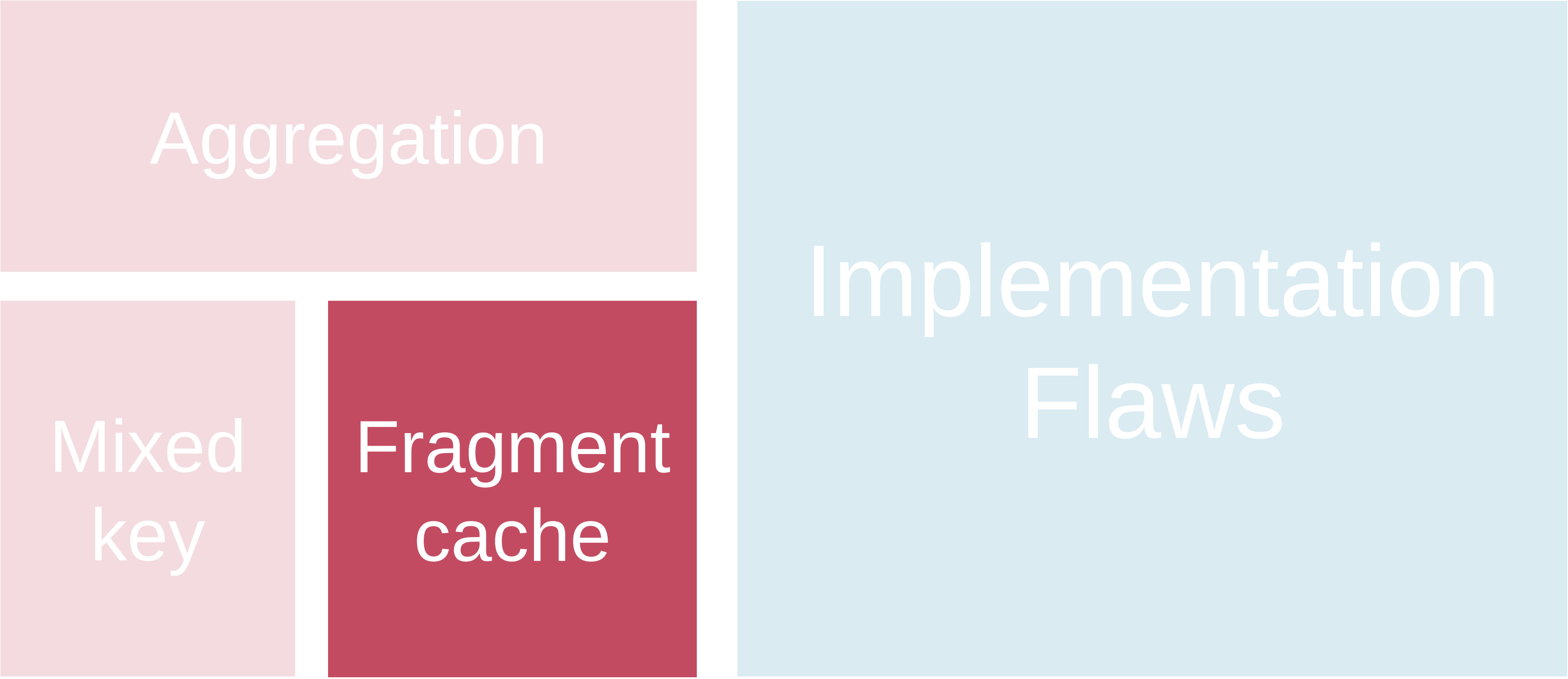
	$Frag_0$	$Frag_1$
Frame 1	`192.168.1.2 to 3.5.1.1`	`GET /image.png HTTP/1.1`

Frame 2	`192.168.1.2 to 8.8.8.8`	`POST /login.php HTTP/1.1`
		`user=admin&pass=SeCr3t`

⬇ Adversary **mixes different fragments**

`192.168.1.2 to 3.5.1.1`	`POST /login.php HTTP/1.1`
	`user=admin&pass=SeCr3t`

→ Login **info is sent to attacker’s server**



Aggregation

Mixed key

Fragment cache

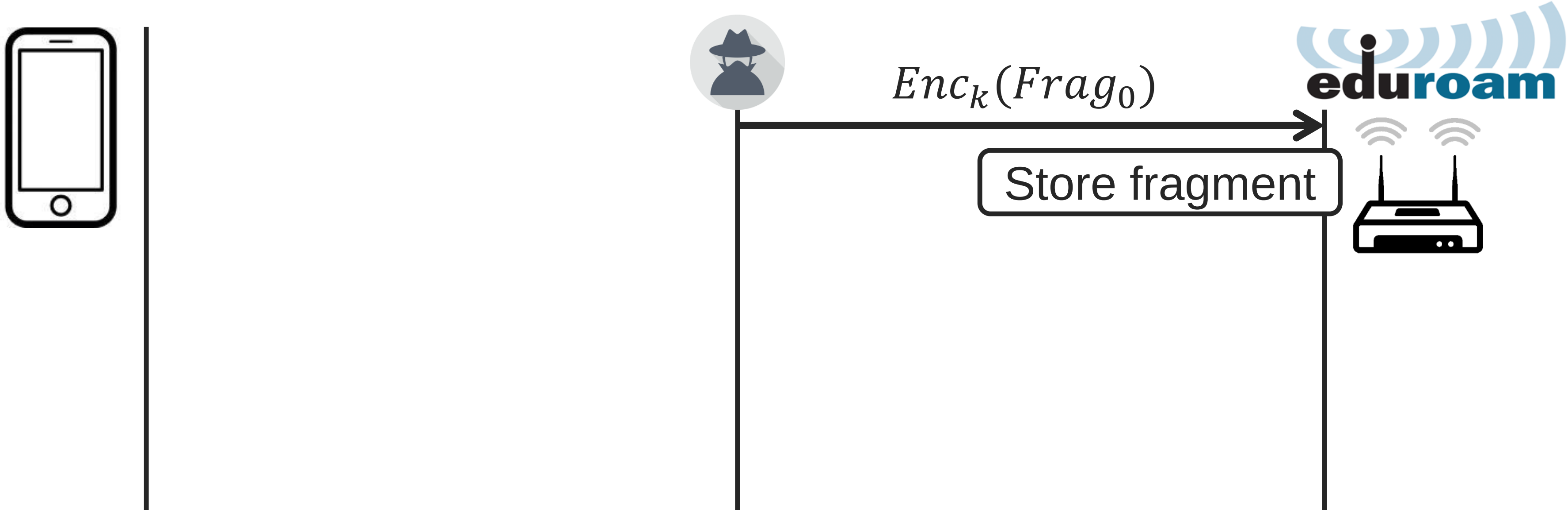
Implementation Flaws

Fragment cache design flaw

Fragments aren’t removed after disconnecting:

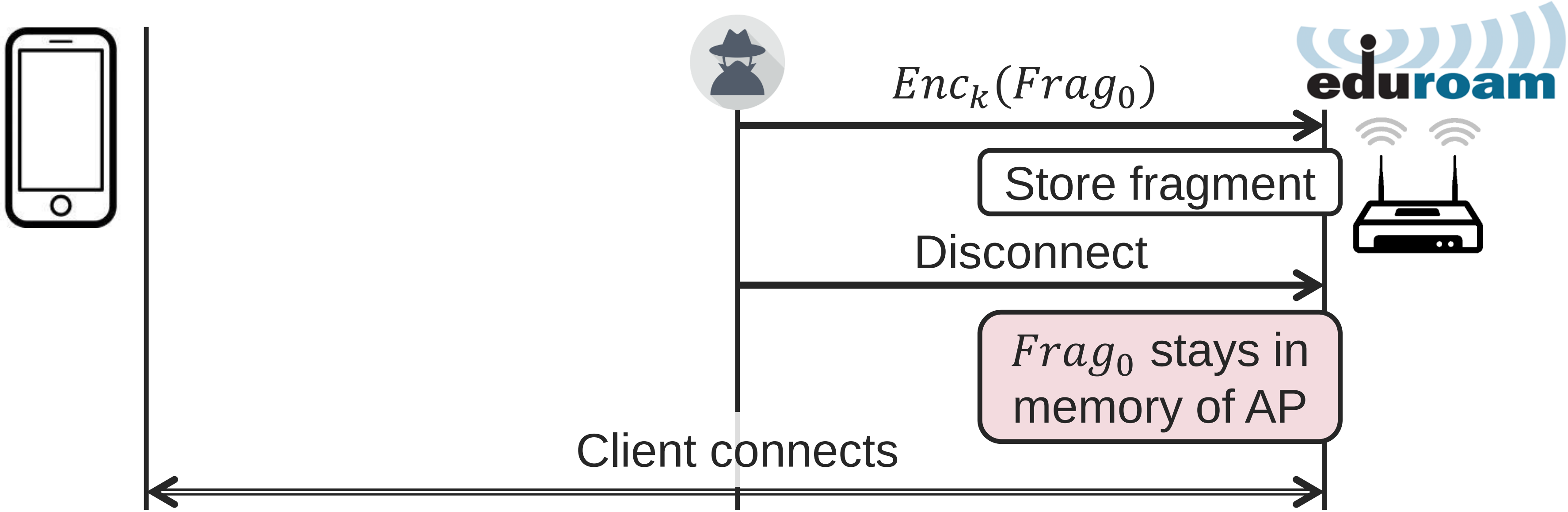
Fragment cache design flaw

Fragments aren’t removed after disconnecting:



Fragment cache design flaw

Fragments aren’t removed after disconnecting:



$Enc_k(Frag_0)$

Store fragment

Disconnect

$Frag_0$ stays in memory of AP

Client connects

Summary of impact

Abuse to **exfiltrate or inject packets** assuming:

1. Hotspot-like network where users distrust each other
2. Client sends fragmented frames (rare unless Wi-Fi 6)

Summary of impact

Abuse to **exfiltrate or inject packets** assuming:

1. Hotspot-like network where users distrust each other
2. Client sends fragmented frames (rare unless Wi-Fi 6)

Even the ancient **WEP protocol is affected**!

› WEP is also affected by the mixed key design flaw

→ Design flaws have been **part of Wi-Fi since 1997**

Defenses

Preventing aggregation-based attacks

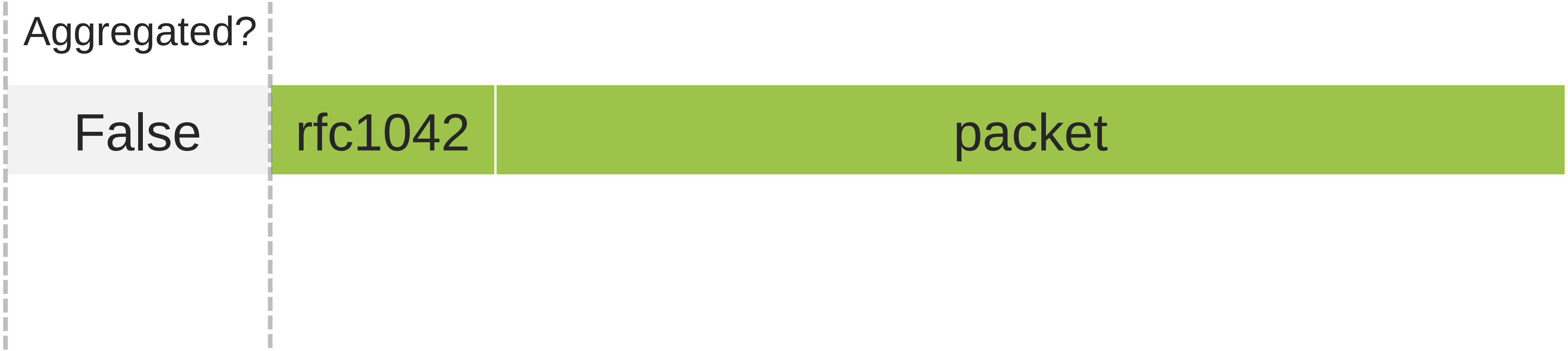
Aggregation design flaw

- Protect the “is aggregated” flag. Not backwards-compatible.
- Current fix: **prevent known attacks** by dropping aggregated frames whose first 6 bytes equal an rfc1042 header

Preventing aggregation-based attacks

Aggregation design flaw

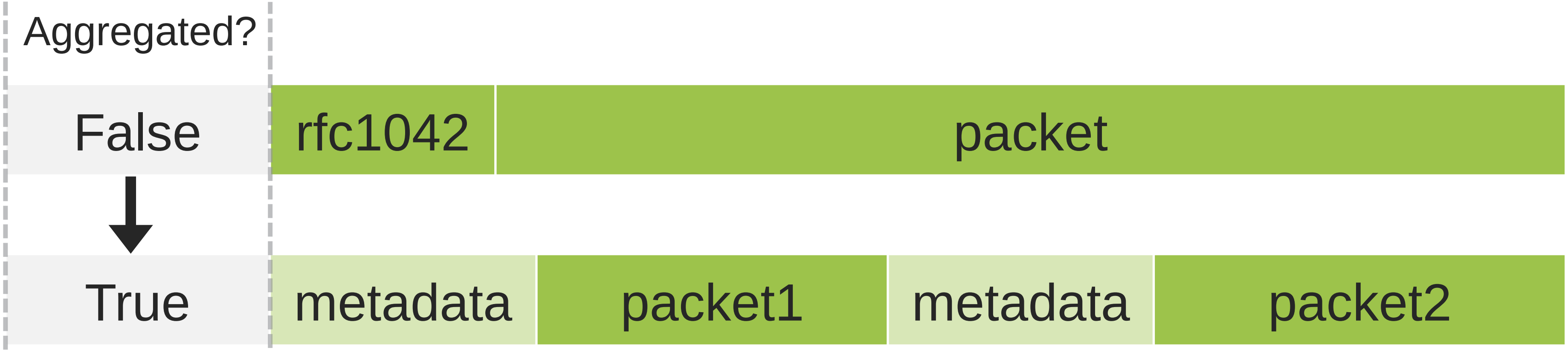
- Protect the “is aggregated” flag. Not backwards-compatible.
- Current fix: **prevent known attacks** by dropping aggregated frames whose first 6 bytes equal an rfc1042 header



Preventing aggregation-based attacks

Aggregation design flaw

- Protect the “is aggregated” flag. Not backwards-compatible.
- Current fix: **prevent known attacks** by dropping aggregated frames whose first 6 bytes equal an rfc1042 header



Aggregated?

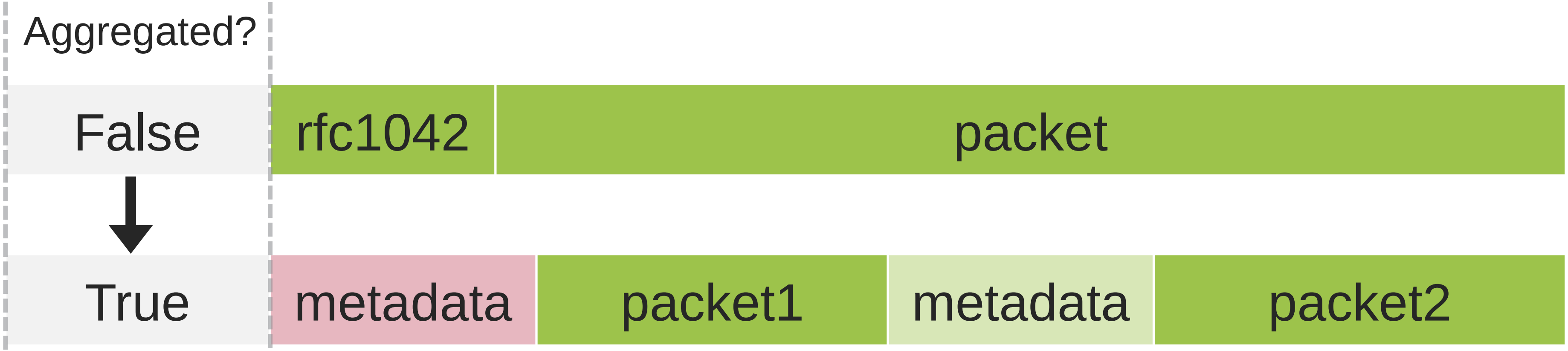
False | rfc1042 | packet

True | metadata | packet1 | metadata | packet2

Preventing aggregation-based attacks

Aggregation design flaw

- Protect the “is aggregated” flag. Not backwards-compatible.
- Current fix: **prevent known attacks** by dropping aggregated frames whose first 6 bytes equal an rfc1042 header



Preventing fragmentation-based attacks

Mixed key attack:

- Only reassemble fragments decrypted under the same key

Fragment cache attack:

- Clear unused fragments when the corresponding key is removed

Design flaws

Implementation Flaws

Design flaws

Plaintext frames

Mixed fragments

Out of order frag

Broadcast fragments

EAPOL forwarding

Cloacked A-MSDUs

Out of order fragments

Trivial frame injection

Plaintext frames wrongly accepted:

- Depending if **fragmented**, **broadcasted,** or while **connecting**

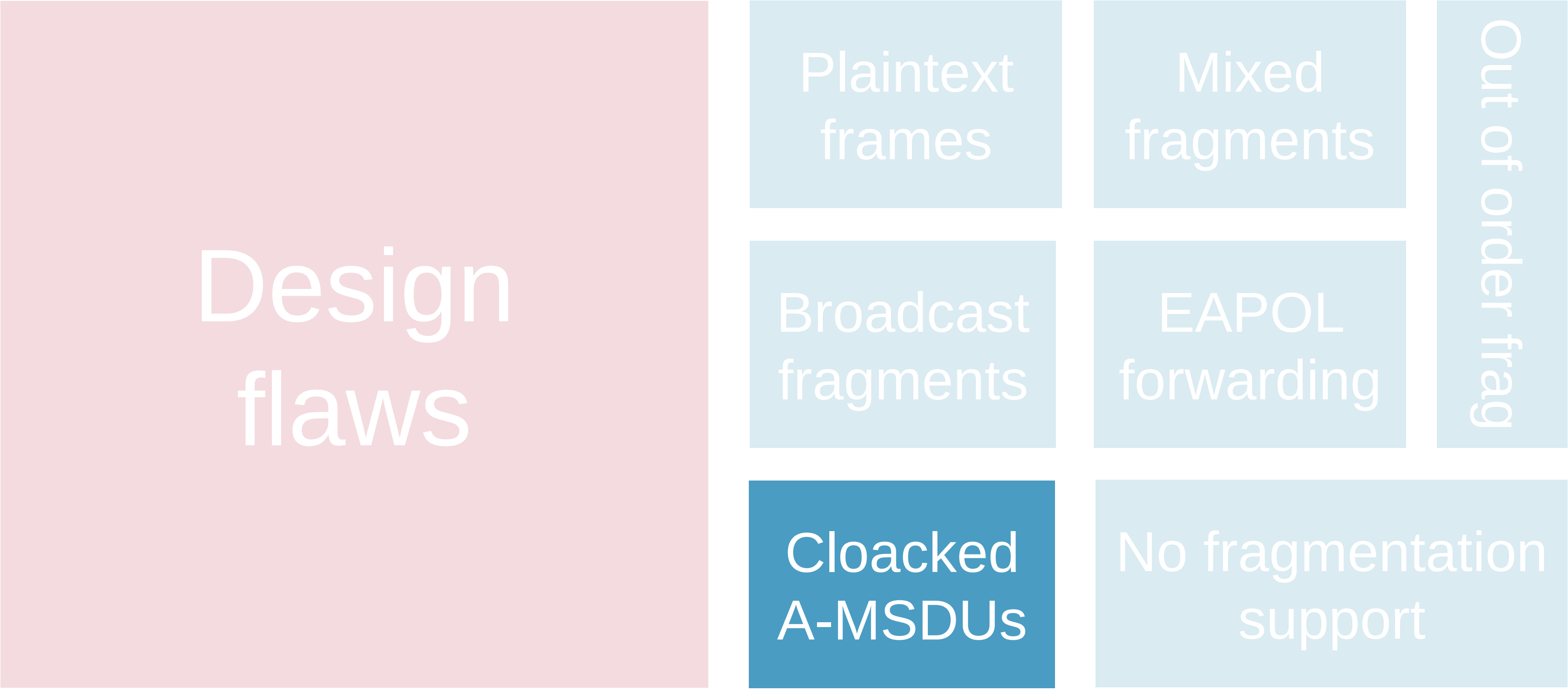
Trivial frame injection

Plaintext frames wrongly accepted:

- Depending if **fragmented**, **broadcasted,** or while **connecting**
- Examples: Apple and some Android devices, some Windows dongles, home and professional APs, and many others!

→ Can trivially **inject frames**

DEMO! ☺



Design flaws

Plaintext frames

Mixed fragments

Out of order frag

Broadcast fragments

EAPOL forwarding

Cloacked A-MSDUs

No fragmentation support

Cloacked aggregated (A-MSDU)frames

Set “is aggregated” flag and send as plaintext:

True	AA AA 03 00 00 00 00 00 88 8E	...	2nd subpacket

Normally: first deaggregate & then check if handshake frame

Cloacked aggregated (A-MSDU)frames

Set “is aggregated” flag and send as plaintext:

True	AA AA 03 00 00 00 00 00 88 8E	...	2nd subpacket

Some switch the order!

Normally: first deaggregate & then check if handshake frame

...	2nd subpacket
1st subpacket is ignored because it has invalid metadata	Plaintext data packet is rejected

Cloacked aggregated (A-MSDU)frames

Set “is aggregated” flag and send as plaintext:

True	AA AA 03 00 00 00 00 00 88 8E	...	2nd subpacket

Handshake header → accept full frame

Vulnerable order: check if handshake & then deaggregate

Cloacked aggregated (A-MSDU)frames

Set “is aggregated” flag and send as plaintext:

True	AA AA 03 00 00 00 00 00 88 8E	...	2nd subpacket

Handshake header → accept full frame

Vulnerable order: check if handshake & then deaggregate

...	2nd subpacket
1st subpacket is ignored because it has invalid metadata	**Plaintext data is also accepted!**

Cloacked aggregated (A-MSDU)frames

Affects FreeBSD, some Windows dongles, several Androids, **3 out of 4 home routers**, 1 out of 3 professional APs, etc.

DEMO! ☺

Design flaws

Plaintext frames

Mixed fragments

Out of order frag

Broadcast fragments

EAPOL forwarding

Cloacked A-MSDUs

No fragmentation support

Flaw: mixed plaintext/encrypted fragments

Only require that the **first fragment is encrypted**

- Affects nearly all network cards on Windows & Linux
- Simplifies aggregation & cache attack

Only require the **last fragment to be encrypted**

- Affects nearly all network cards on Free/NetBSD
- Trivial to **inject & exfiltrate** data

Design flaws

Plaintext frames

Mixed fragments

Out of order frag

Broadcast fragments

EAPOL forwarding

Cloacked A-MSDUs

No fragmentation support

Flaw: non-consective packet numbers

header	s	n	0	More	fragment1
header	s	$n+1$	1	More	fragment2
header	s	$n+2$	2	Last	fragment3

- **Nobody** but Linux **checks** if packet numbers are consecutive
- Can do mixed key attack without periodic rekeys

Design flaws

Plaintext frames

Mixed fragments

Out of order frag

Broadcast fragments

EAPOL forwarding

Cloacked A-MSDUs

No fragmentation support

No fragmentation support

Some devices don’t support fragmentation

- But they **treat fragmented frames as full frames**
- Examples: OpenBSD and Espressif chips

→ Abuse to **inject frames** under right conditions

→ **All devices are vulnerable** to one or more flaws!

Created tool to test devices

Has **45+ test cases** for both **clients and APs**

- Can detect all vulnerabilities
- Needs network password (not an attack tool)
- Can also be used as basis for other Wi-Fi research [SVR21]

https://github.com/vanhoefm/fragattacks

Discussion

Design flaws took **two decades** to discover

- Without modified drivers some attacks will fail

Discussion

Design flaws took **two decades** to discover

- Without modified drivers some attacks will fail
- Fragmentation & aggregation wasn’t considered important

Discussion

Design flaws took **two decades** to discover

- Without modified drivers some attacks will fail
- Fragmentation & aggregation wasn’t considered important

Long-term lessons:

- **Adopt defences early** even if concerns are theoretic
- Isolate **security contexts** (data decrypted with different keys)
- **Keep fuzzing** devices. Wi-Fi Alliance can help here!

Coordinated disclosure

Wi-Fi Alliance & ICASI contacted vendors

- Embargo of roughly 9 months
- Test tool (= PoC) received several updates during embargo!

Currently doing following-up work

- Updating the IEEE 802.11 standard to fix design flaws
- Maintaining test tool and checking some vendor patches

Looking back

Was it the long disclosure worth it?

- Some companies had patches for most devices but still weren’t happy... ¯_(ツ)_/¯
- Others appreciated this even if not all devices had patches!
- Props to: Cisco, LANCOM, Aruba, Huawei, Ubiquity, MediaTek, Samsung, NETGEAR, as well as others

Conclusion

- Discovered three **design flaws**
- Multiple **implementation flaws**
- Implementation flaws easy to abuse, but design flaws hard to abuse
- More info: www.fragattacks.com

References

- Presentation is based on: **Fragment and Forge: Breaking Wi-Fi Through Frame Aggregation and Fragmentation.** https://papers.mathyvanhoef.com/usenix2021.pdf
- [VP17] Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2
- [VP18] Release the Kraken: New KRACKs in the 802.11 Standard
- [CKM20] A Formal Analysis of IEEE 802.11's WPA2: Countering the Kracks Caused by Cracking the Counters
- [VR20] Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd
- [WFA20] Wi-Fi Alliance Wi-Fi Security Roadmap and WPA3 Updates. https://wi-fi.org/file/wi-fi-security-roadmap-and-wpa3-updates-december-2020
- [VBDOP18] Operating Channel Validation: Preventing Multi-Channel Man-in-the-Middle Attacks Against Protected Wi-Fi Networks
- [VAP20] Protecting Wi-Fi Beacons from Outsider Forgeries
- [SVR21] DEMO: A Framework to Test and Fuzz Wi-Fi Devices. https://papers.mathyvanhoef.com/wisec2021-demo.pdf