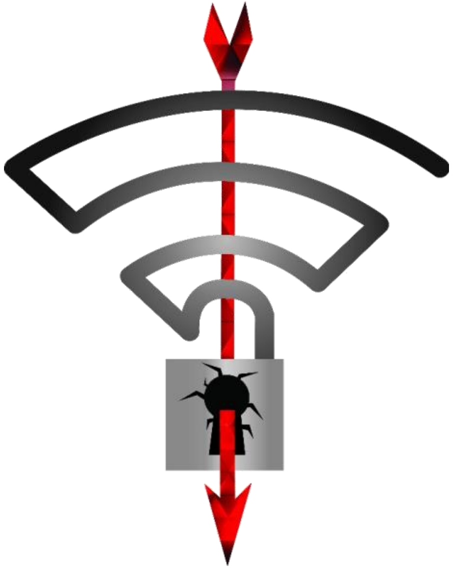
Release the Kraken:

New KRACKs in the 802.11 Standard

Mathy Vanhoef — @vanhoefm

Toronto, Canada, 16 October 2018

KU LEUVEN DistriNet



Key reinstallations in the 4-way handshake

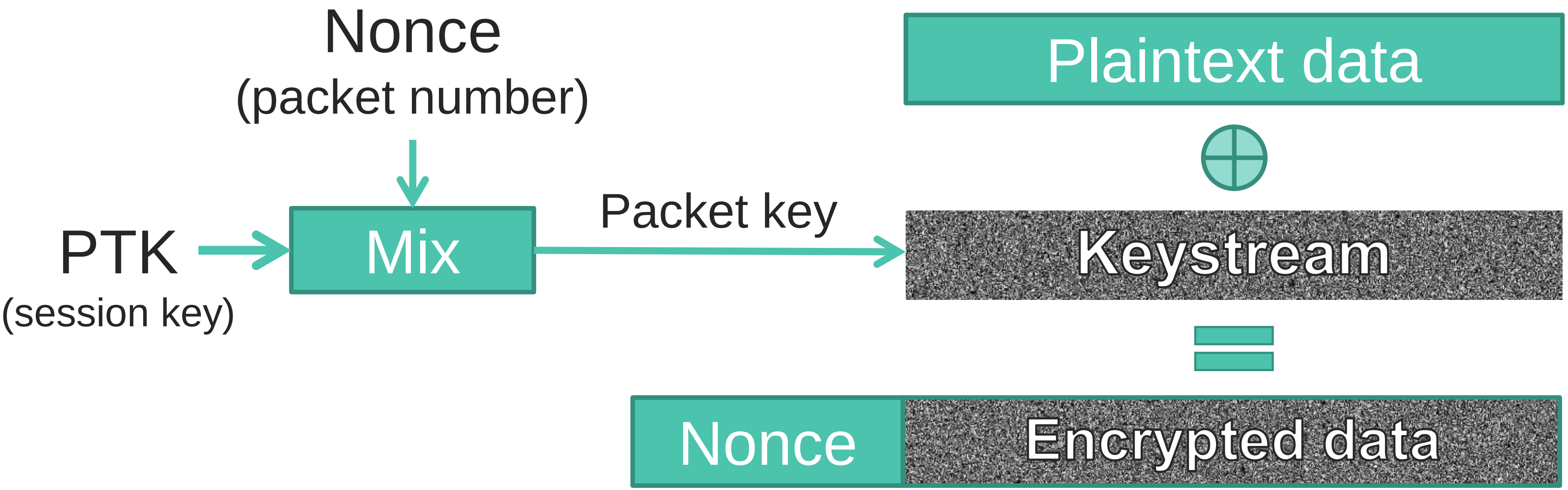
WPA2: 4-way handshake

Used to connect to any protected Wi-Fi network

Mutual authentication

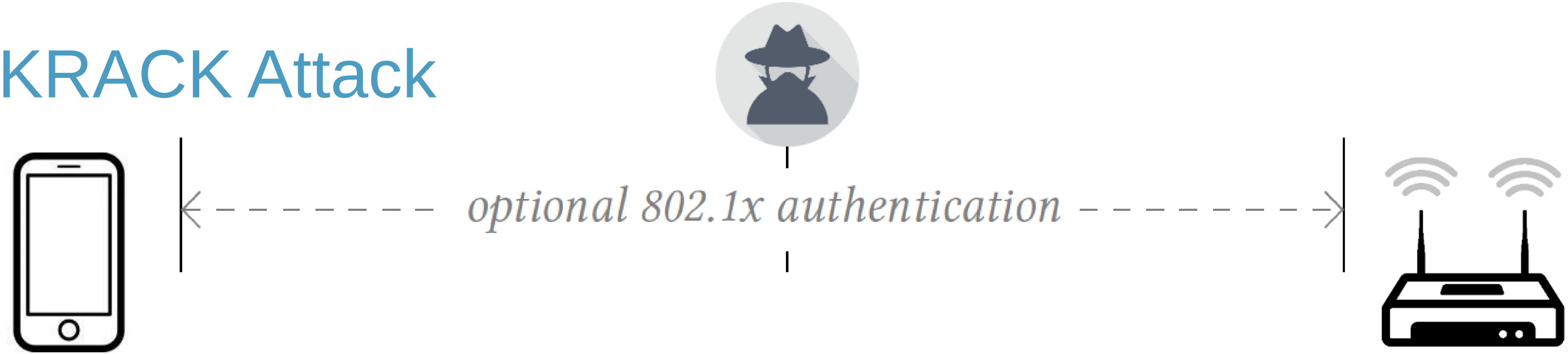
Negotiates fresh PTK: pairwise transient key

WPA2: Encryption algorithm



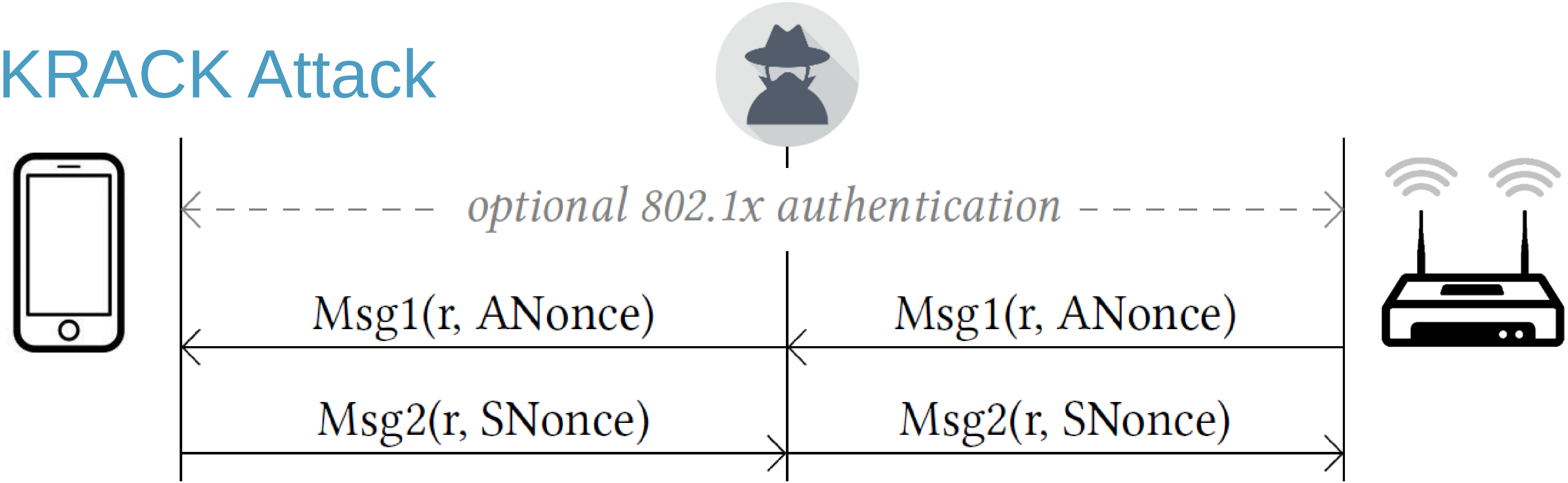
→ Nonce reuse implies keystream reuse (in all WPA2 ciphers)

KRACK Attack

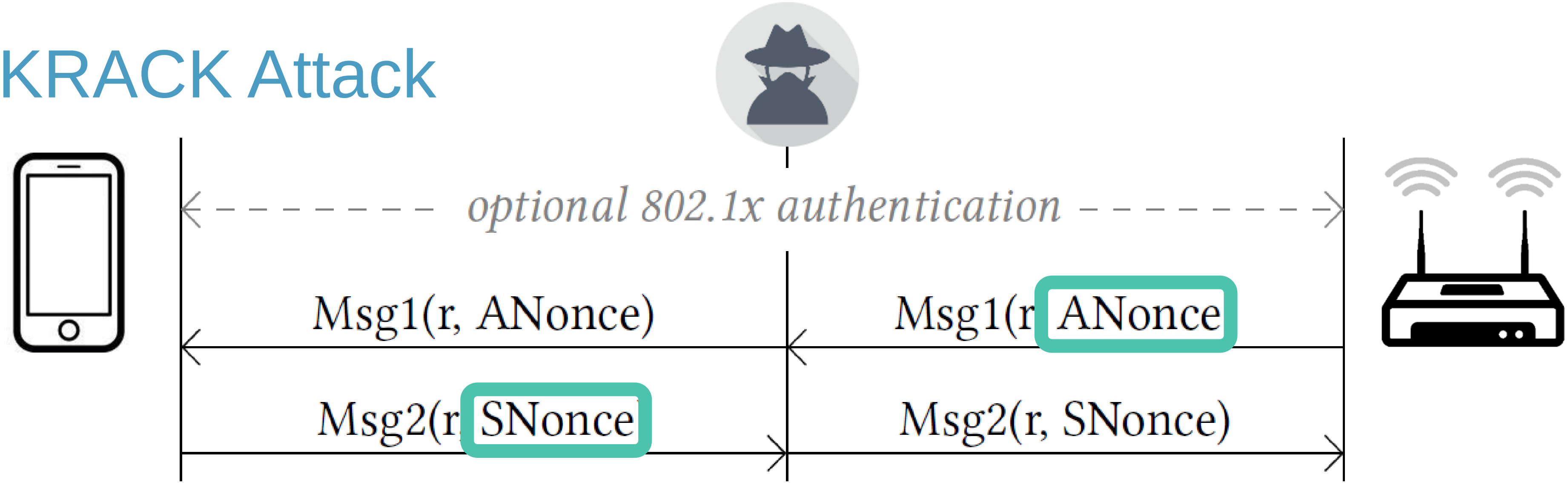


optional 802.1x authentication

KRACK Attack



KRACK Attack



optional 802.1x authentication

Msg1(r, ANonce)

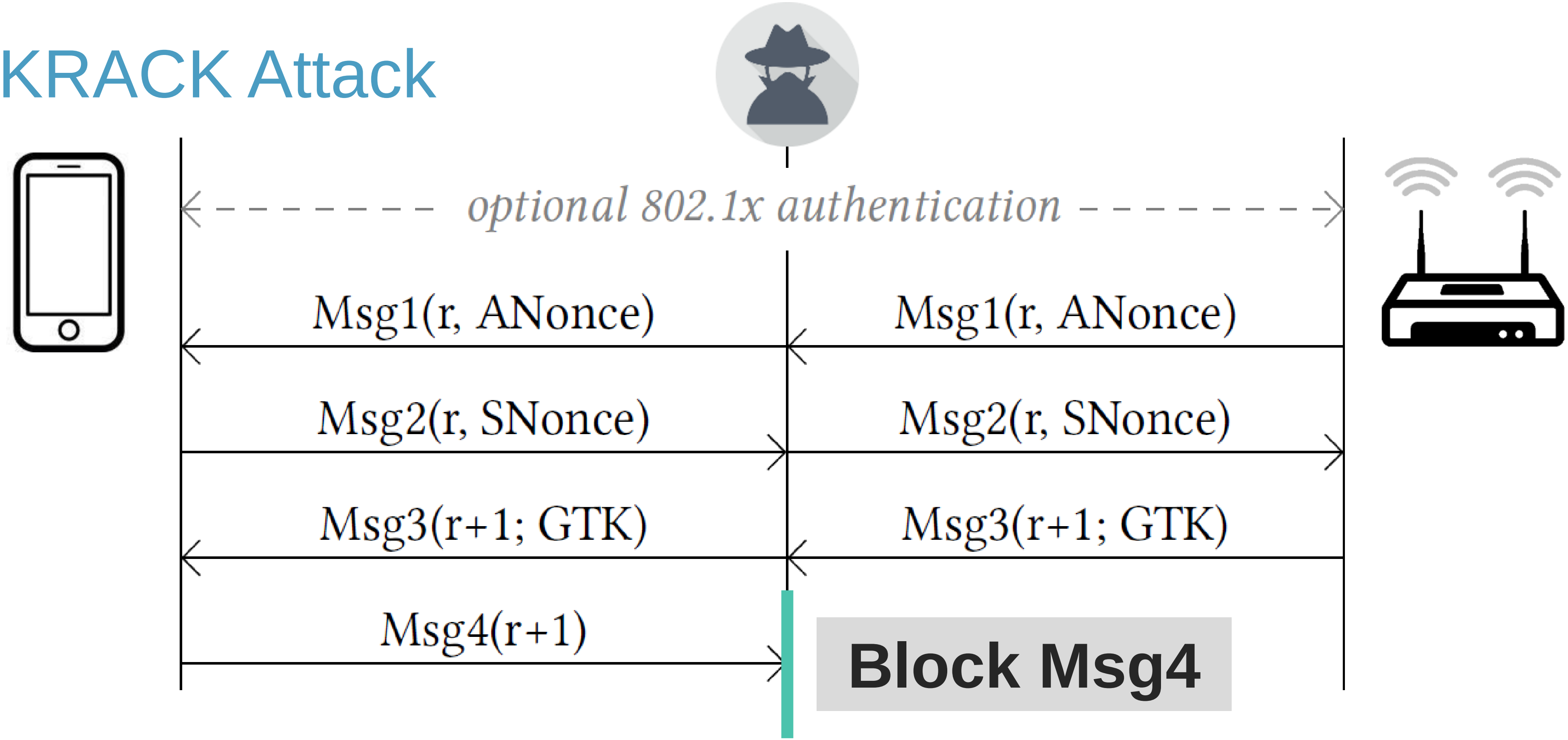
Msg1(r, ANonce)

Msg2(r, SNonce)

Msg2(r, SNonce)

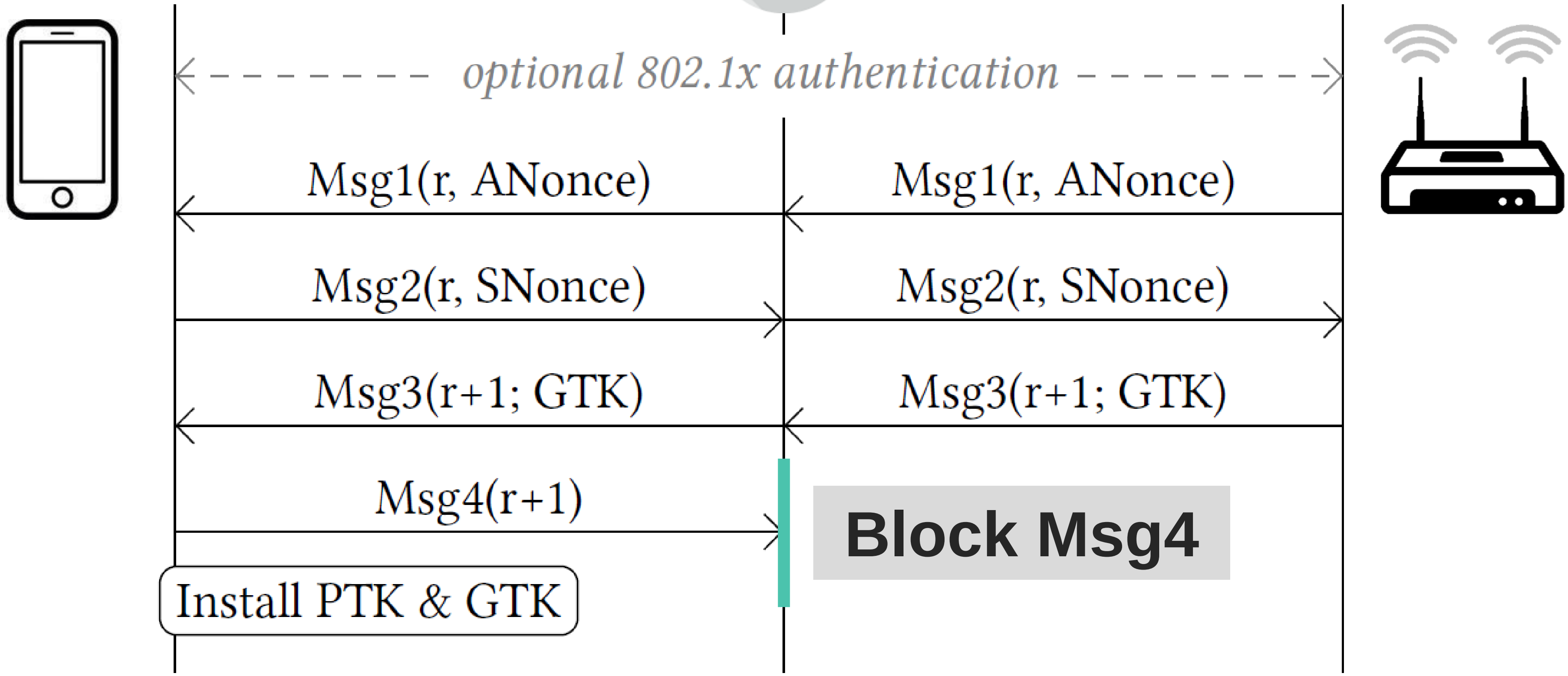
PTK = Combine(shared secret, ANonce, SNonce)

KRACK Attack



Block Msg4

KRACK Attack



optional 802.1x authentication

Msg1(r, ANonce)

Msg1(r, ANonce)

Msg2(r, SNonce)

Msg2(r, SNonce)

Msg3(r+1; GTK)

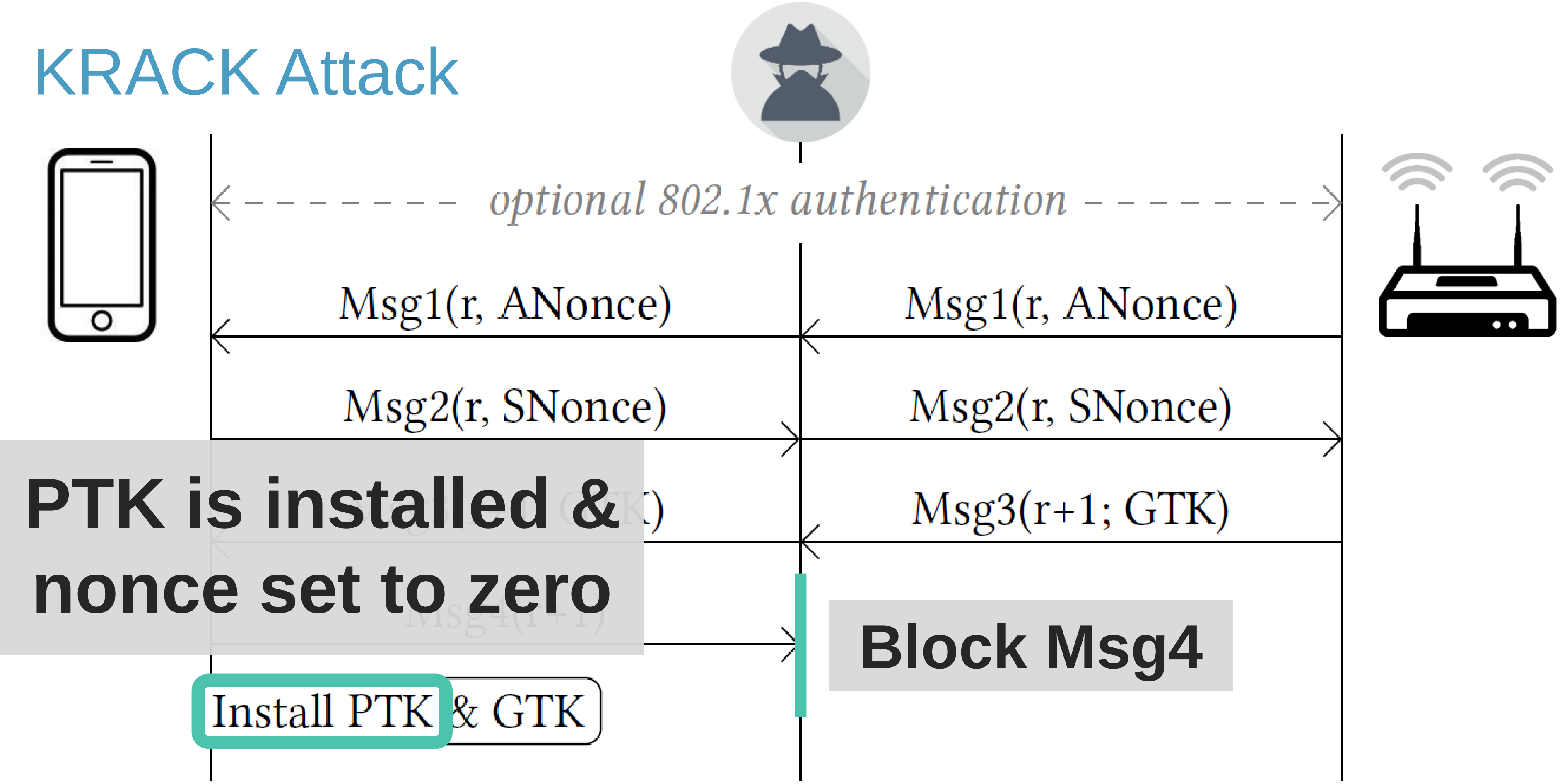
Msg3(r+1; GTK)

Msg4(r+1)

Block Msg4

Install PTK & GTK

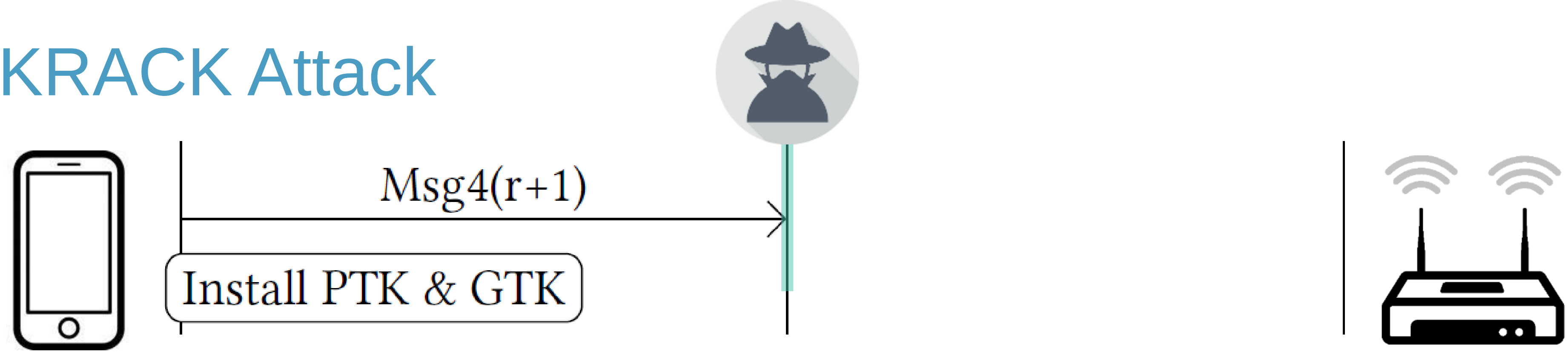
KRACK Attack



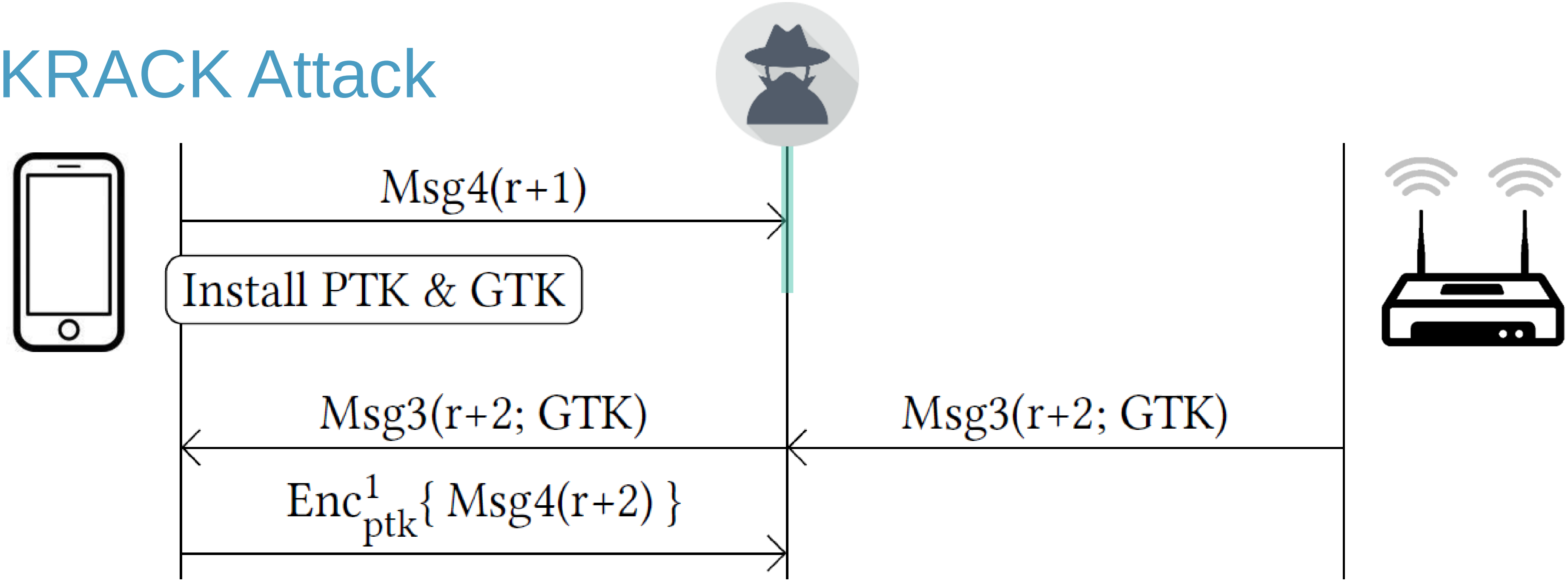
PTK is installed & nonce set to zero

Block Msg4

KRACK Attack



KRACK Attack



Msg4(r+1)

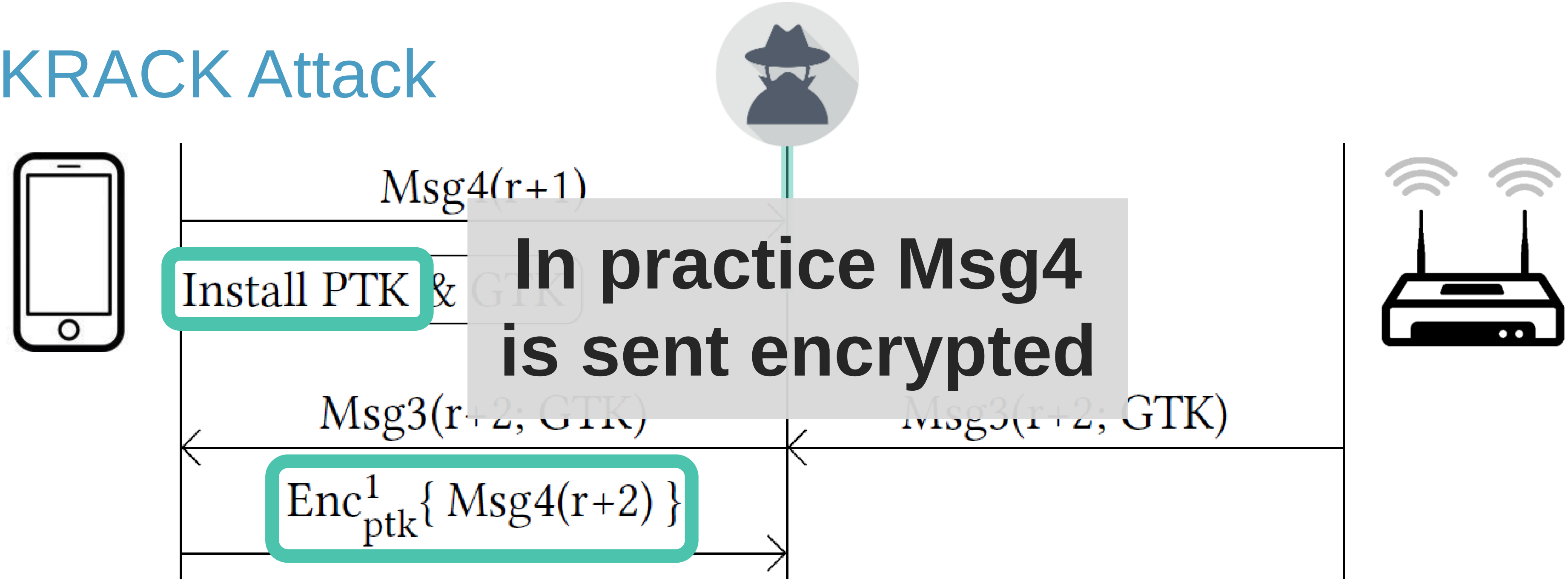
Install PTK & GTK

Msg3(r+2; GTK)

Msg3(r+2; GTK)

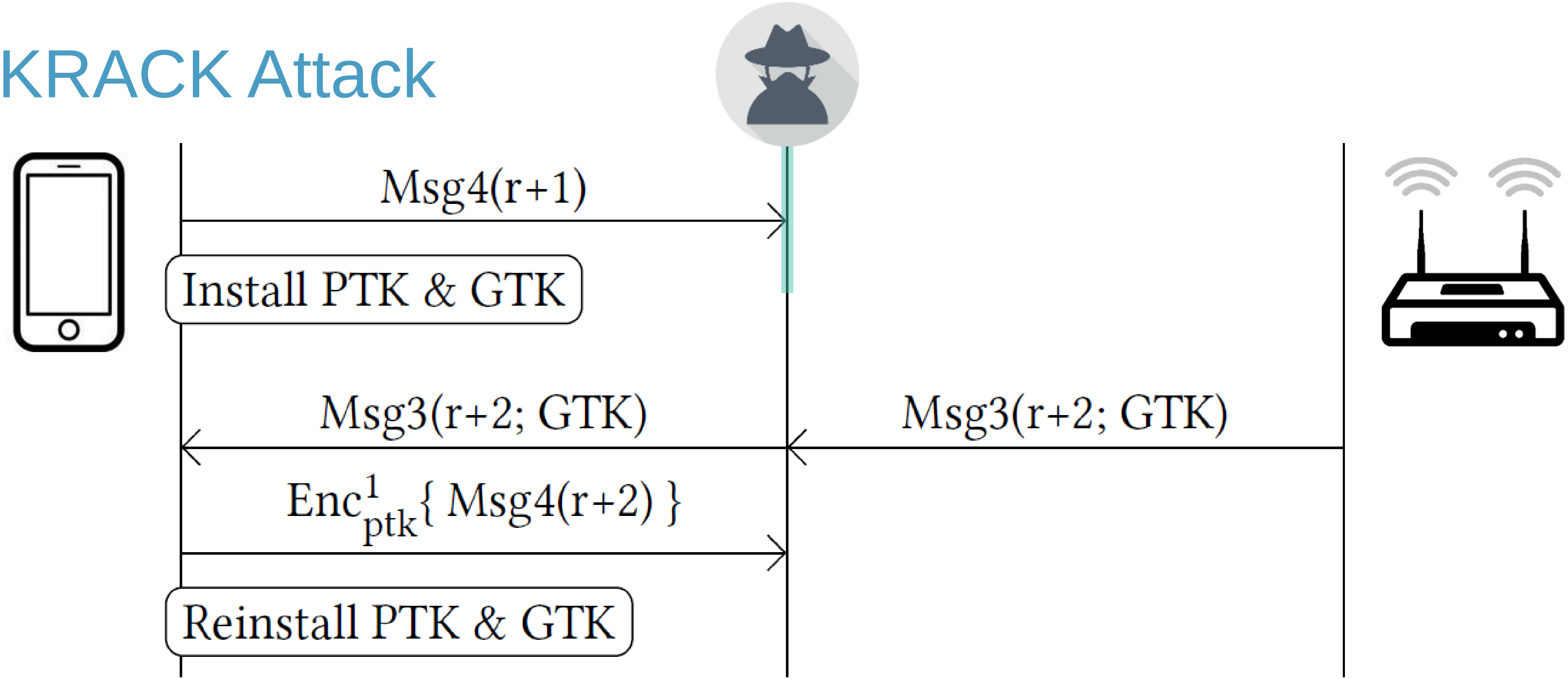
$\text{Enc}^{1}_{\text{ptk}}\{\text{ Msg4(r+2) }\}$

KRACK Attack



In practice Msg4 is sent encrypted

KRACK Attack



Msg4(r+1)

Install PTK & GTK

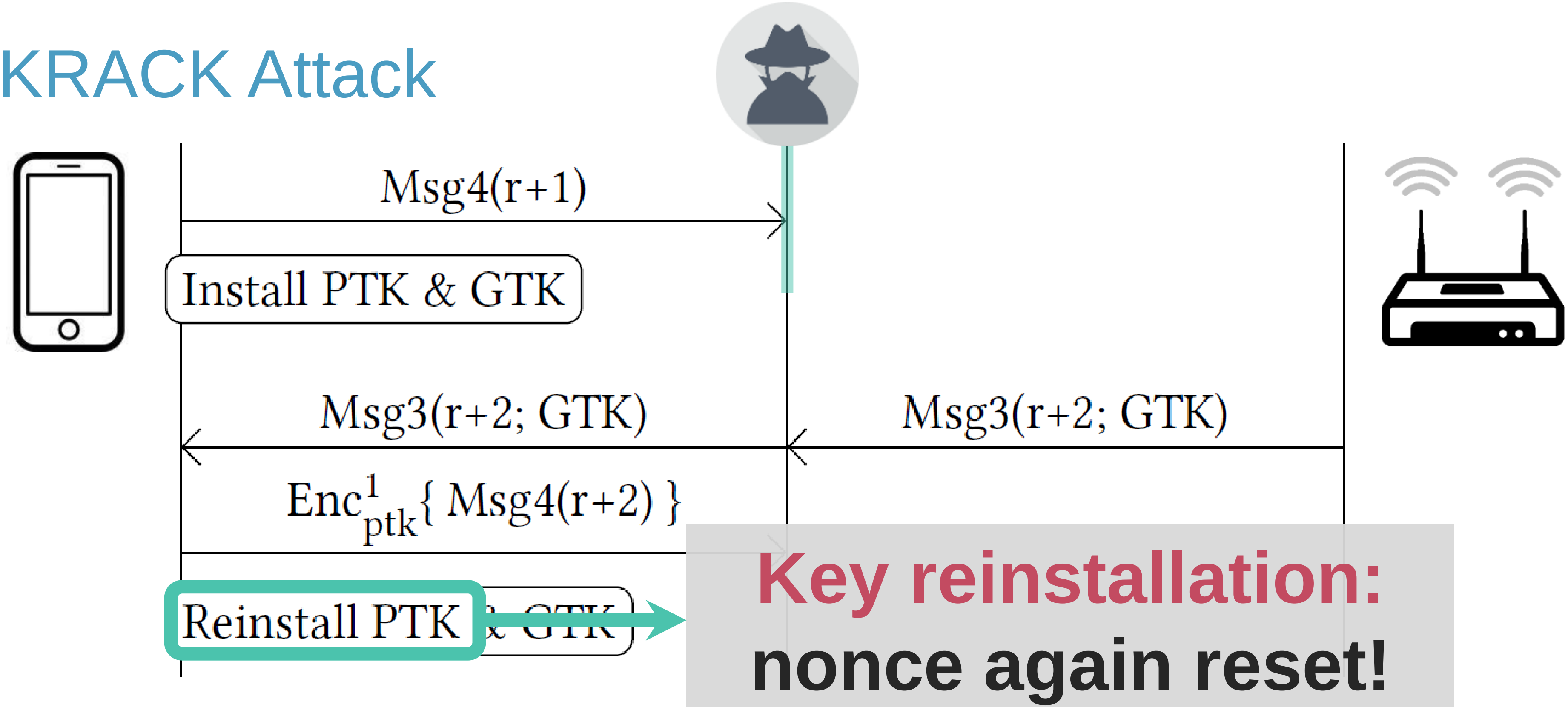
Msg3(r+2; GTK)

Msg3(r+2; GTK)

$\text{Enc}^{1}_{\text{ptk}}\{ \text{Msg4}(r+2) \}$

Reinstall PTK & GTK

KRACK Attack



Msg4(r+1)

Install PTK & GTK

Msg3(r+2; GTK)

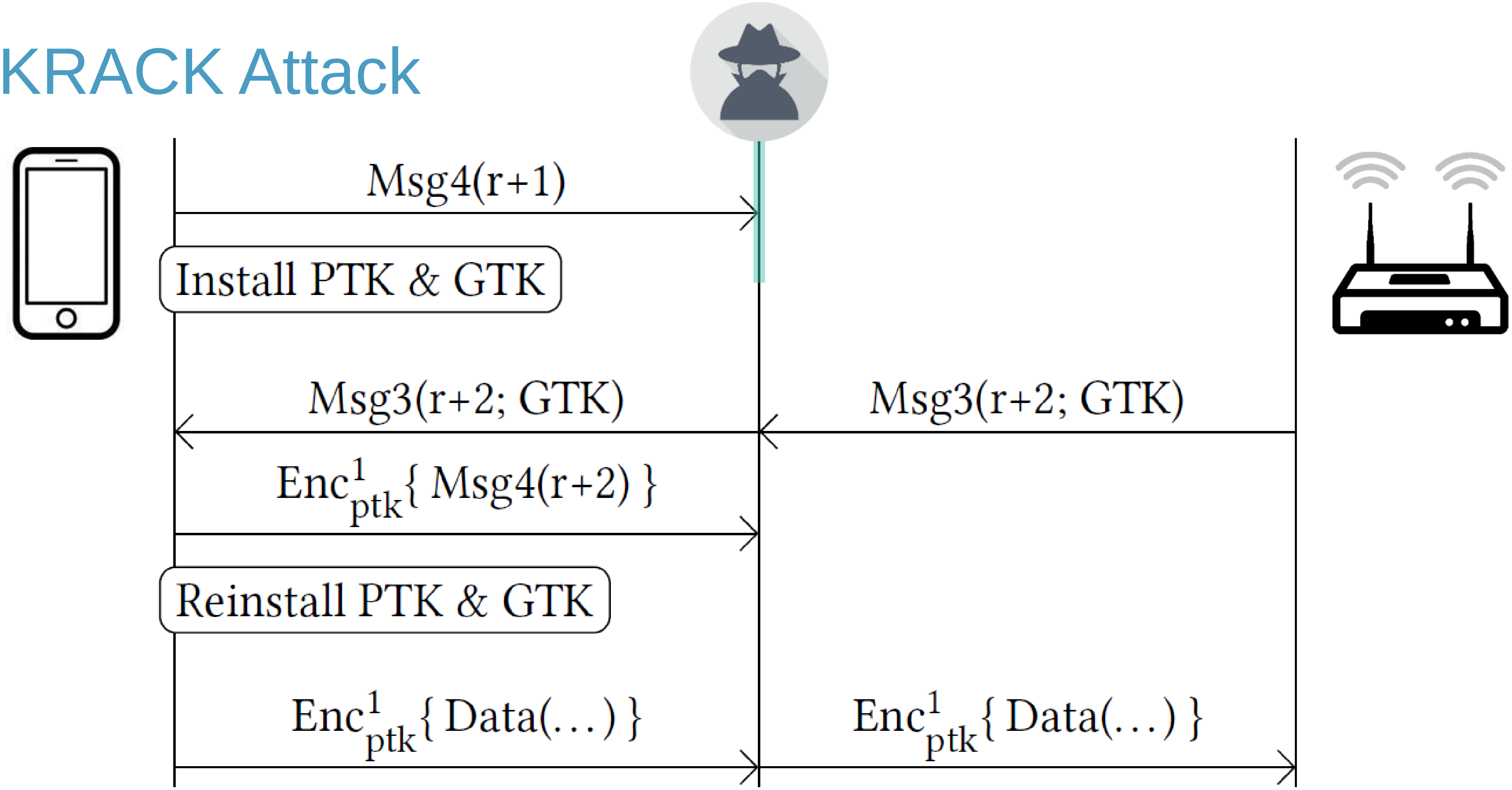
Msg3(r+2; GTK)

$\text{Enc}^{1}_{\text{ptk}}\{\text{ Msg4(r+2) }\}$

Reinstall PTK & GTK

Key reinstallation:
nonce again reset!

KRACK Attack



Msg4(r+1)

Install PTK & GTK

Msg3(r+2; GTK)

Msg3(r+2; GTK)

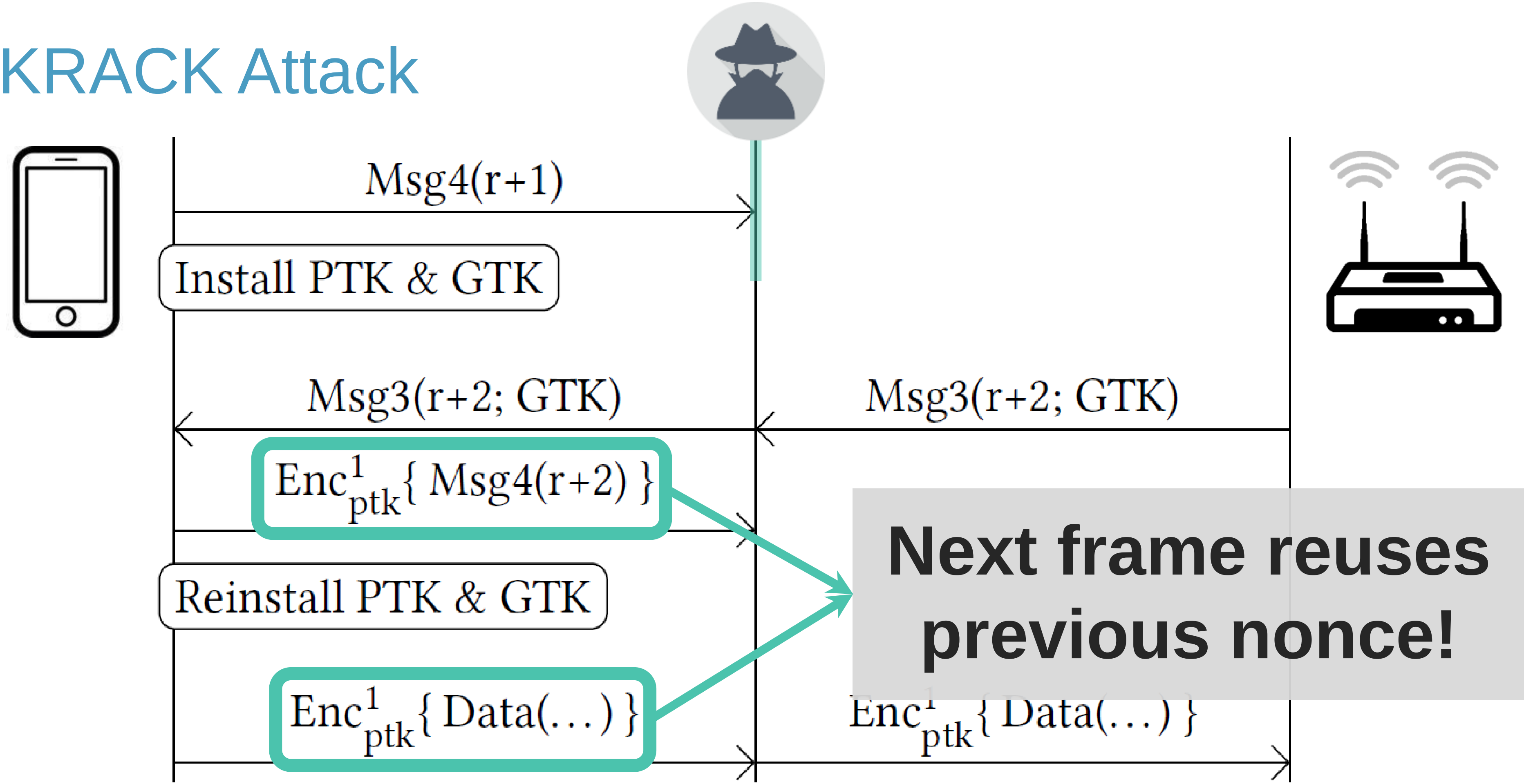
$\text{Enc}^{1}_{\text{ptk}}\{\ \text{Msg4(r+2)}\ \}$

Reinstall PTK & GTK

$\text{Enc}^{1}_{\text{ptk}}\{\ \text{Data(...)}\ \}$

$\text{Enc}^{1}_{\text{ptk}}\{\ \text{Data(...)}\ \}$

KRACK Attack



Msg4(r+1)

Install PTK & GTK

Msg3(r+2; GTK)

Msg3(r+2; GTK)

$\text{Enc}^{1}_{\text{ptk}}\{\ \text{Msg4}(r+2)\ \}$

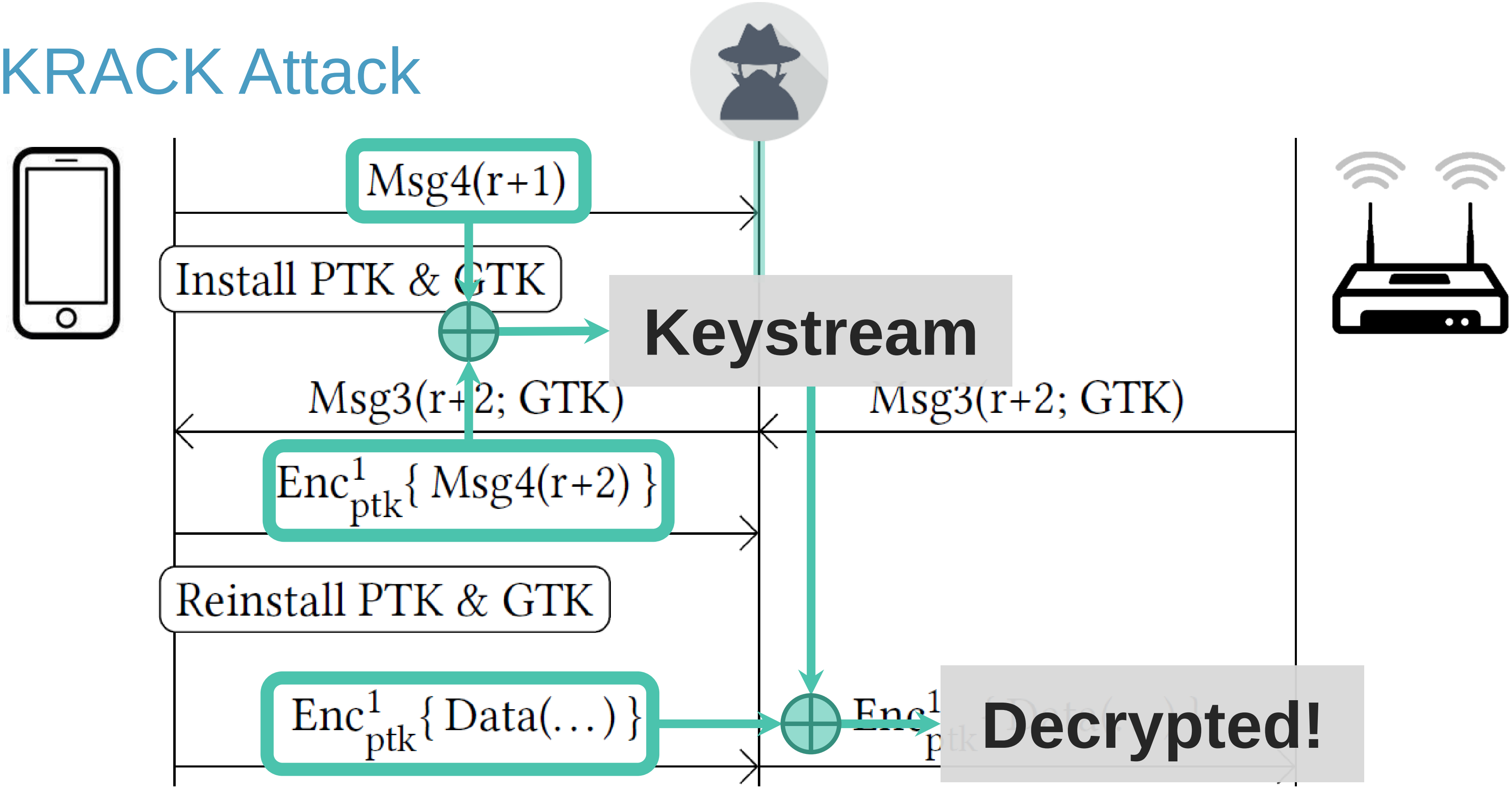
Reinstall PTK & GTK

$\text{Enc}^{1}_{\text{ptk}}\{\ \text{Data}(\ldots)\ \}$

$\text{Enc}^{1}_{\text{ptk}}\{\ \text{Data}(\ldots)\ \}$

Next frame reuses previous nonce!

KRACK Attack



Msg4(r+1)

Install PTK & GTK

Keystream

Msg3(r+2; GTK)

Msg3(r+2; GTK)

$\text{Enc}^1_{\text{ptk}}$\{ Msg4(r+2) \}

Reinstall PTK & GTK

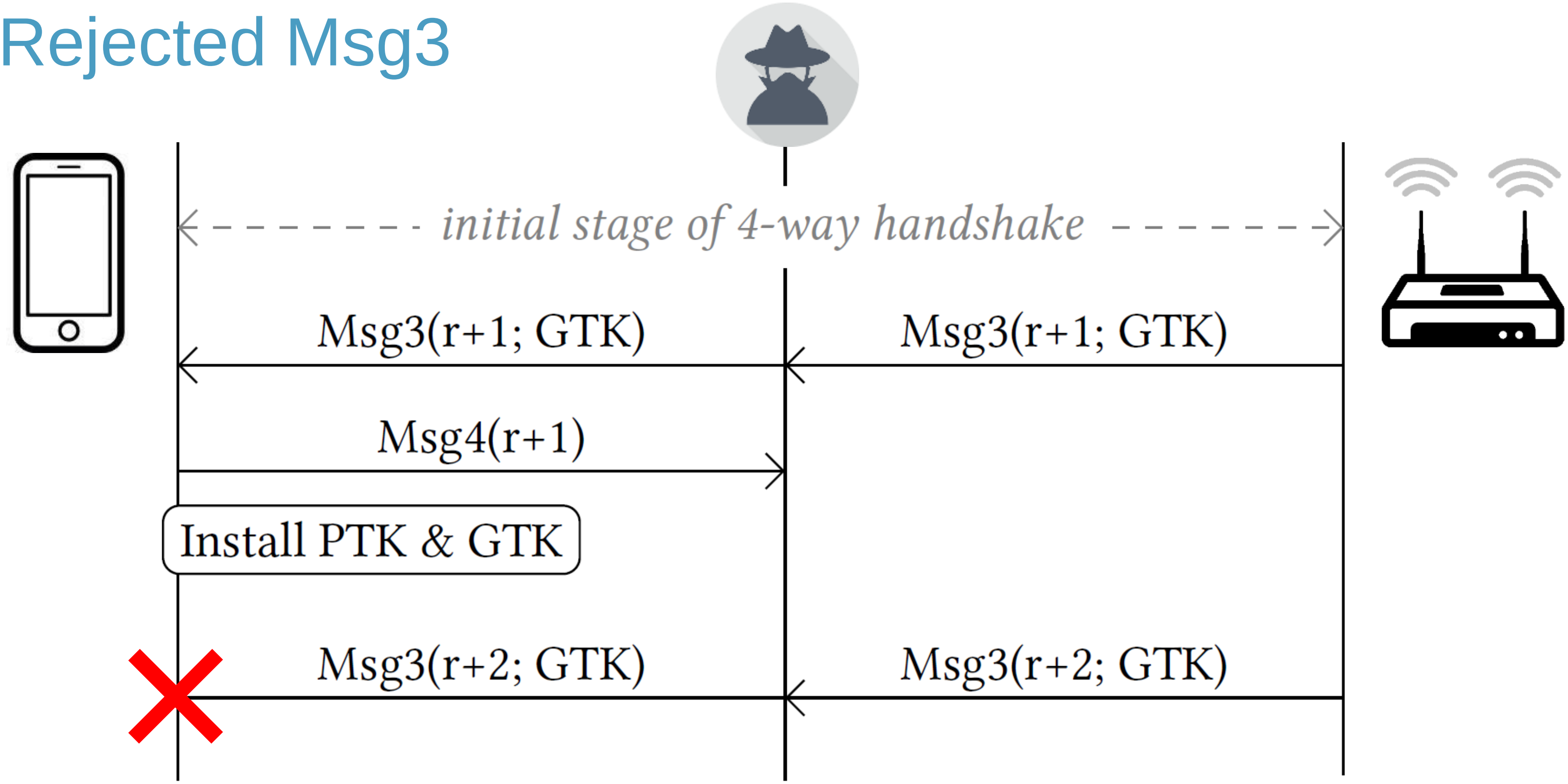
$\text{Enc}^1_{\text{ptk}}$\{ Data(…) \}

$\text{Enc}^1_{\text{ptk}}$

Decrypted!

Practical Obstacles

Rejected Msg3



initial stage of 4-way handshake

Msg3(r+1; GTK)

Msg3(r+1; GTK)

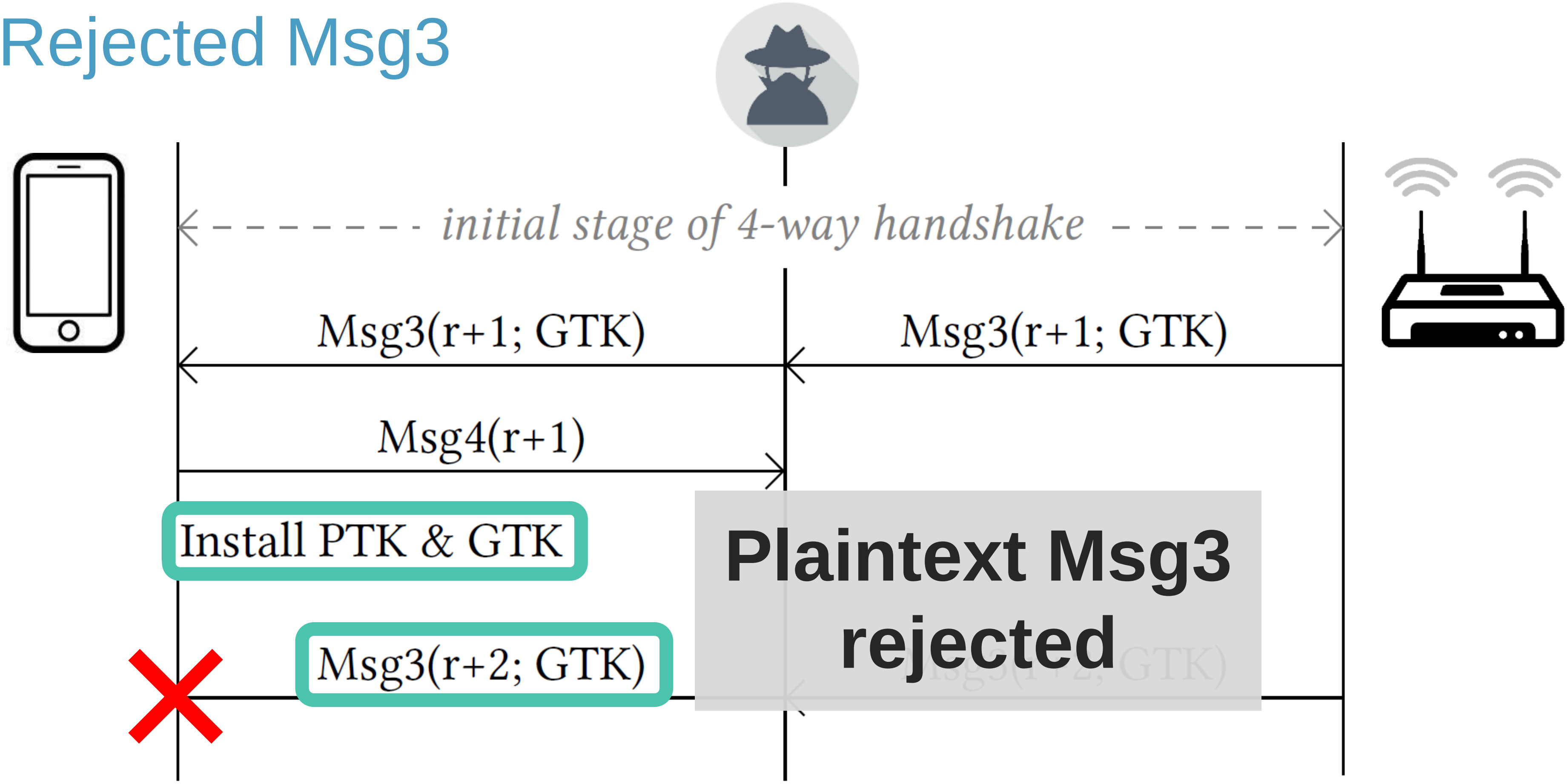
Msg4(r+1)

Install PTK & GTK

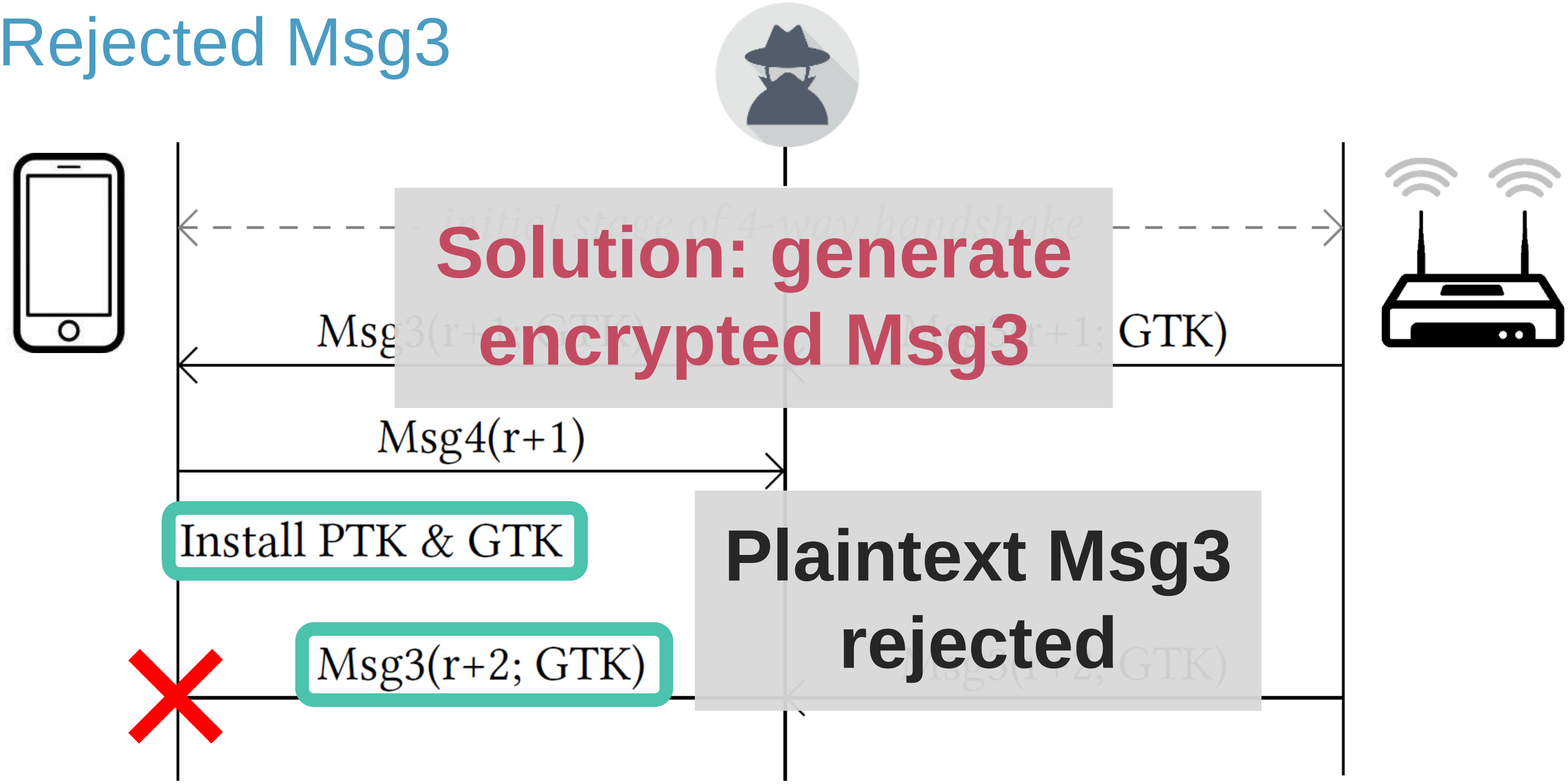
Msg3(r+2; GTK)

Msg3(r+2; GTK)

Rejected Msg3



Rejected Msg3



initial stage of 4-way handshake

Msg3(r+1; GTK)

Msg3(r+1; GTK)

Msg4(r+1)

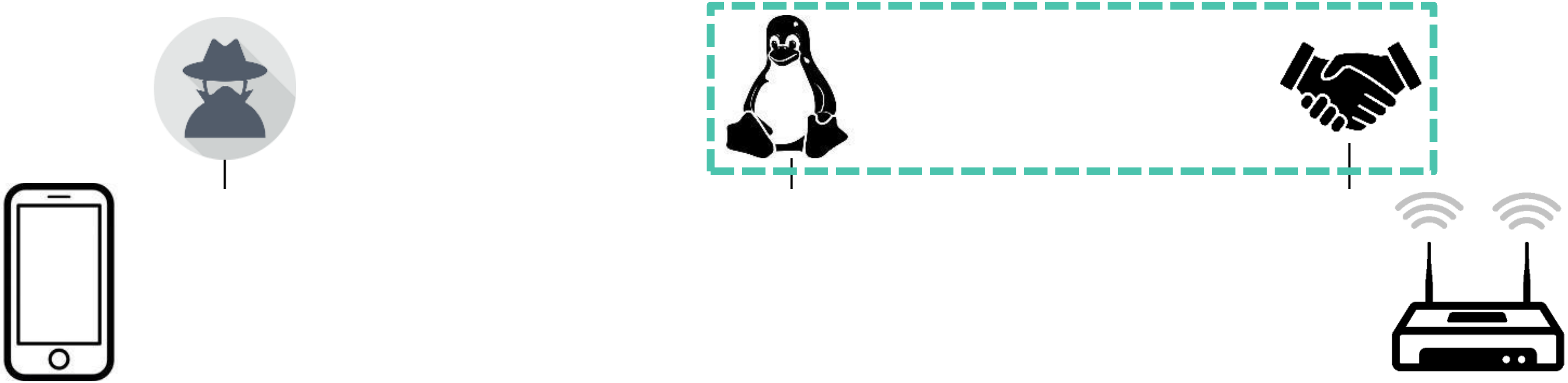
Install PTK & GTK

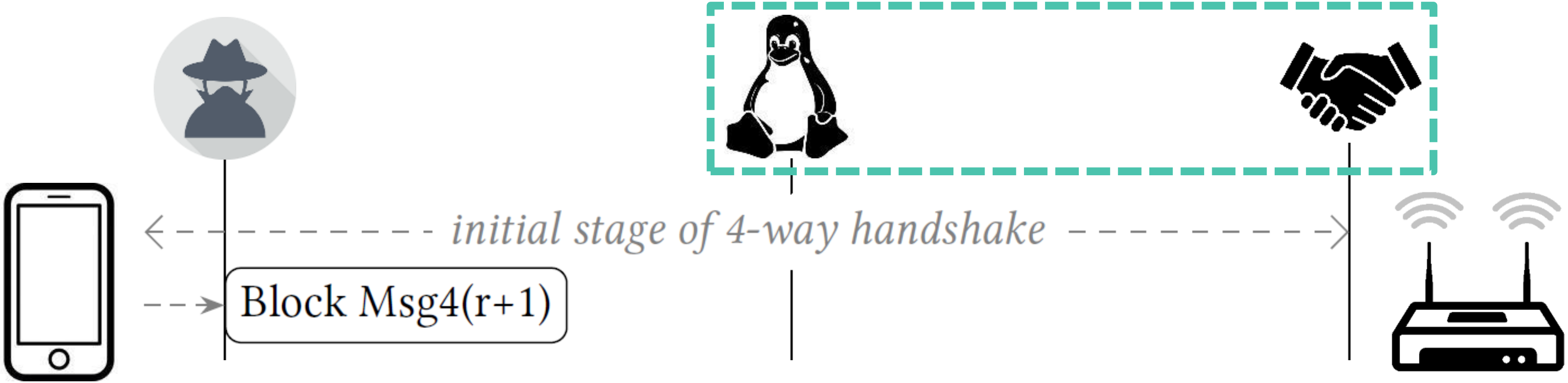
Msg3(r+2; GTK)

Msg3(r+2; GTK)

Solution: generate encrypted Msg3

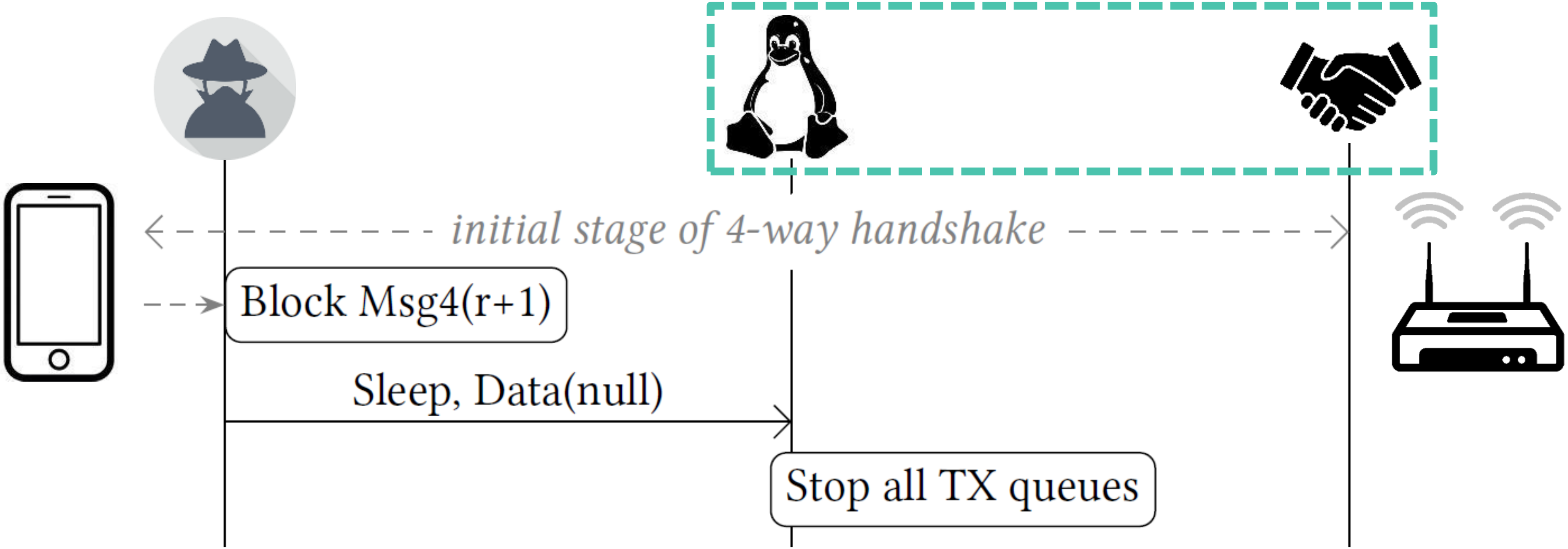
Plaintext Msg3 rejected

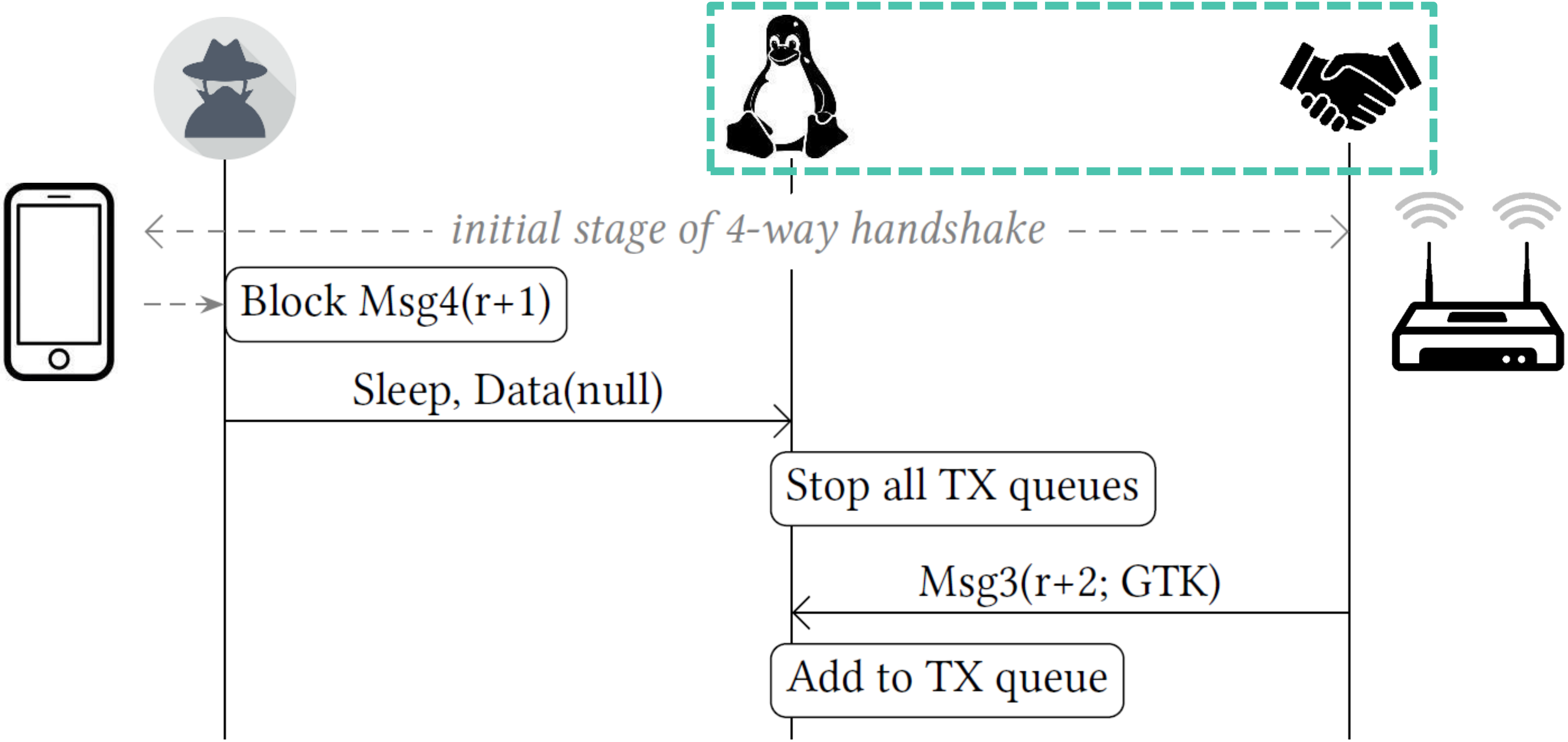




initial stage of 4-way handshake

Block Msg4(r+1)


initial stage of 4-way handshake
Block Msg4(r+1)
Sleep, Data(null)
Stop all TX queues



initial stage of 4-way handshake

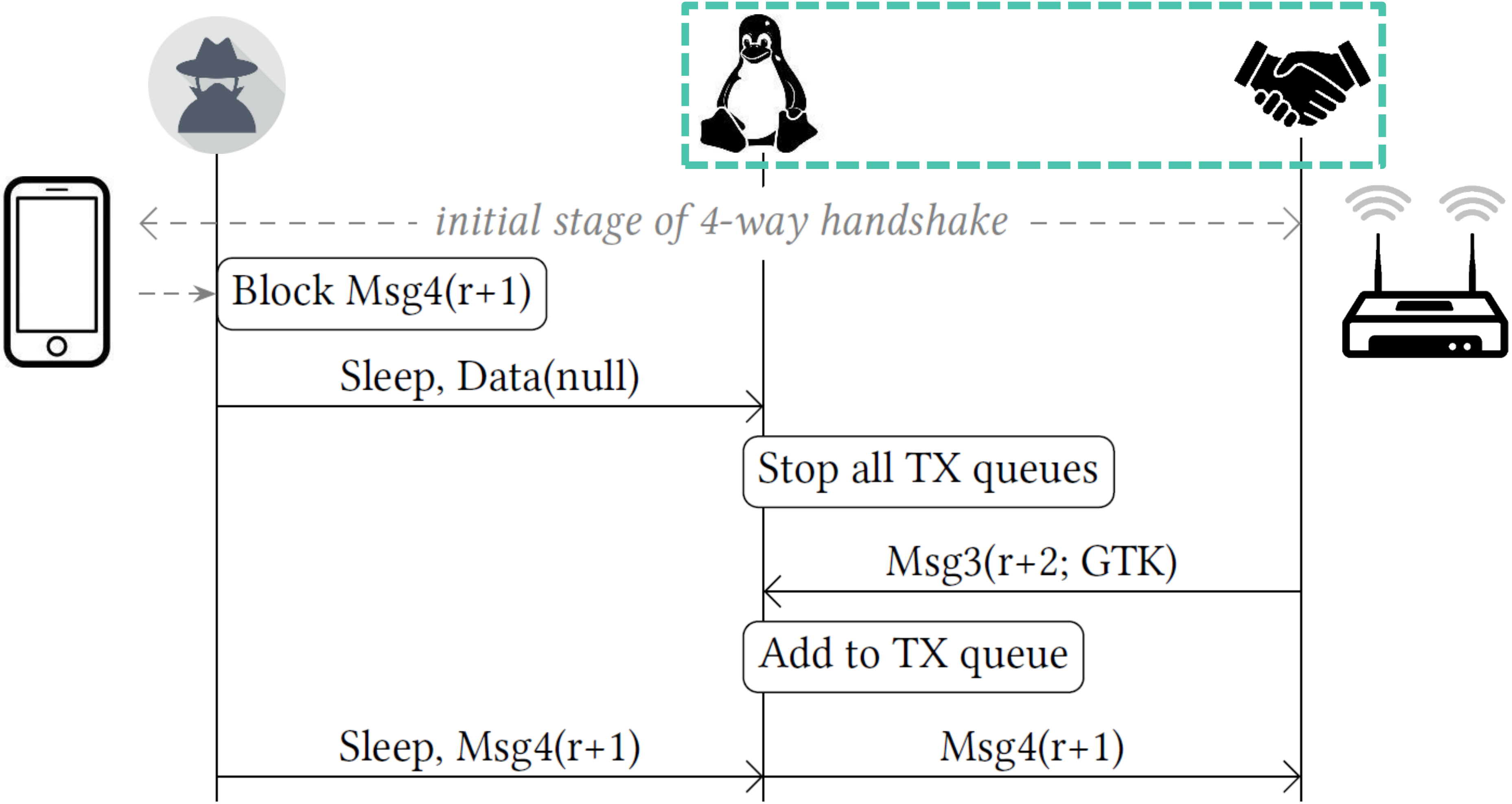
Block Msg4(r+1)

Sleep, Data(null)

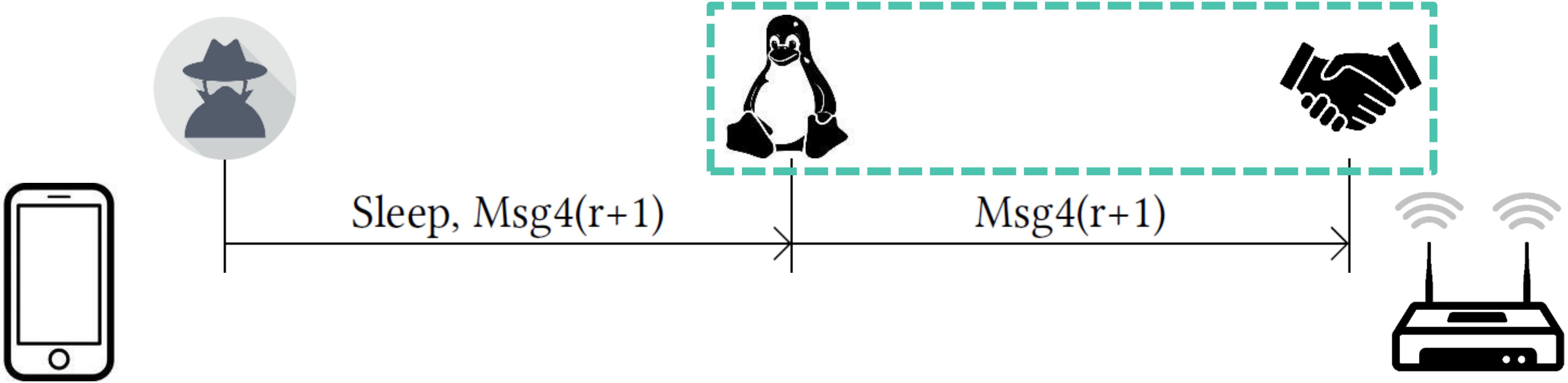
Stop all TX queues

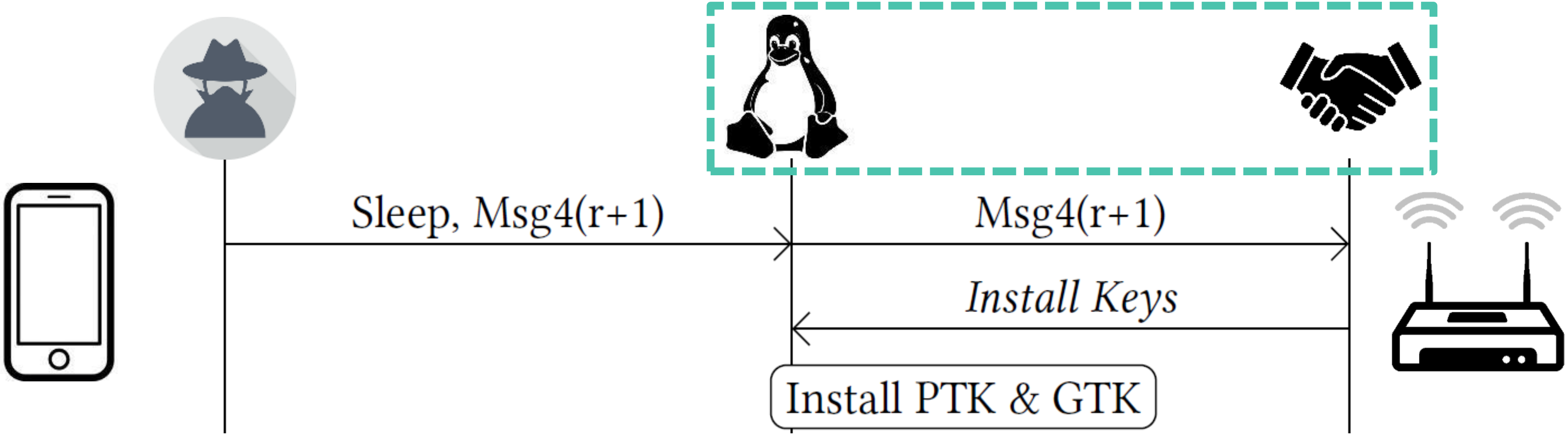
Msg3(r+2; GTK)

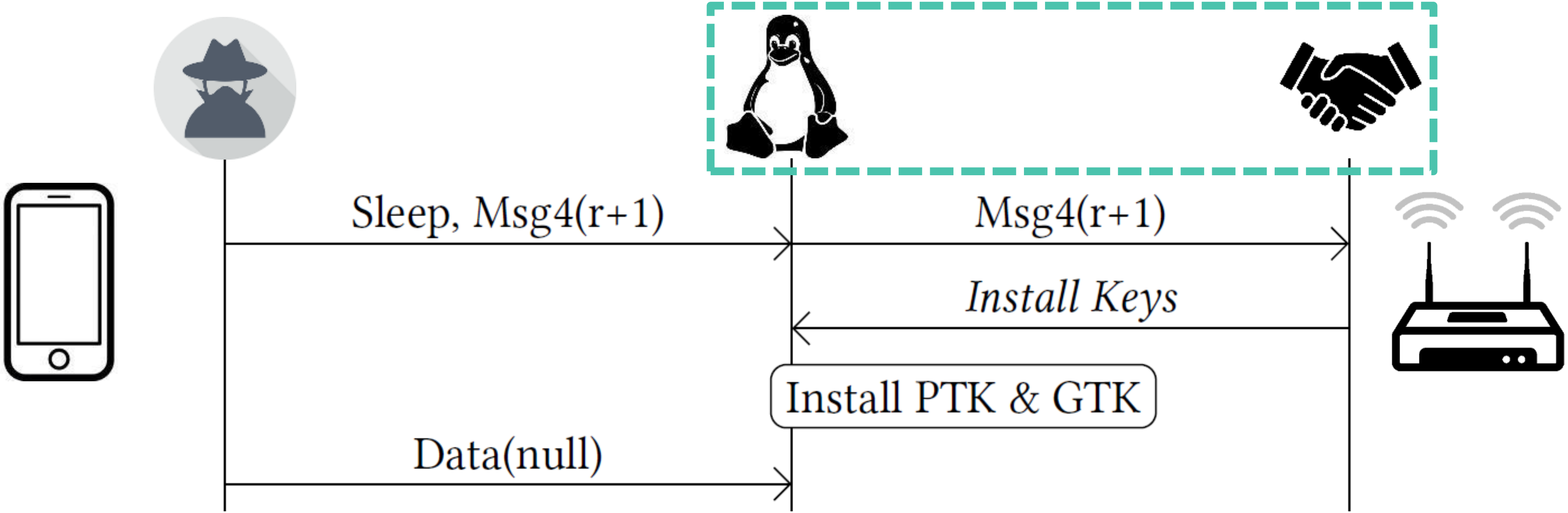
Add to TX queue

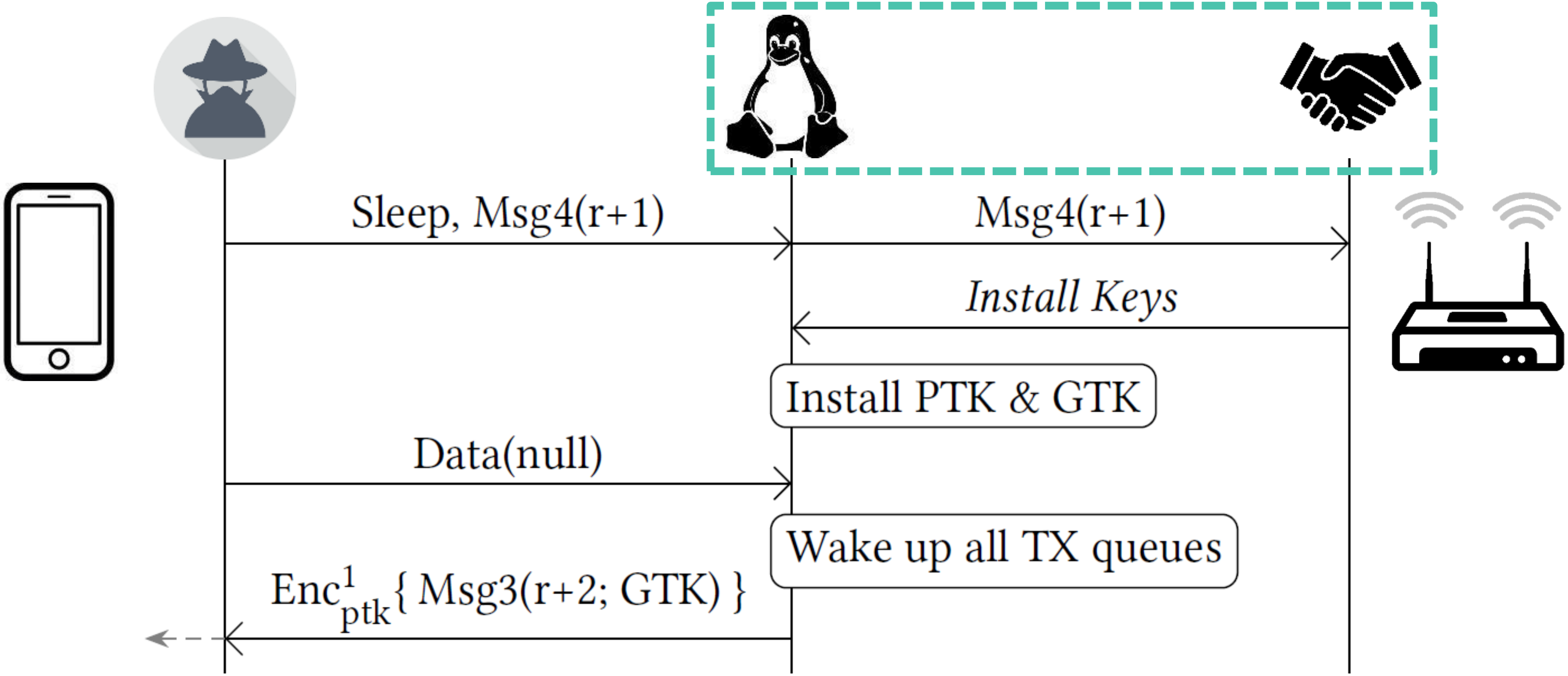


initial stage of 4-way handshake
Block Msg4(r+1)
Sleep, Data(null)
Stop all TX queues
Msg3(r+2; GTK)
Add to TX queue
Sleep, Msg4(r+1)
Msg4(r+1)


Sleep, Msg4(r+1)
Msg4(r+1)


Sleep, Msg4(r+1)
Msg4(r+1)
Install Keys
Install PTK & GTK


Sleep, Msg4(r+1)
Msg4(r+1)
Install Keys
Install PTK & GTK
Data(null)



Sleep, Msg4(r+1)

Msg4(r+1)

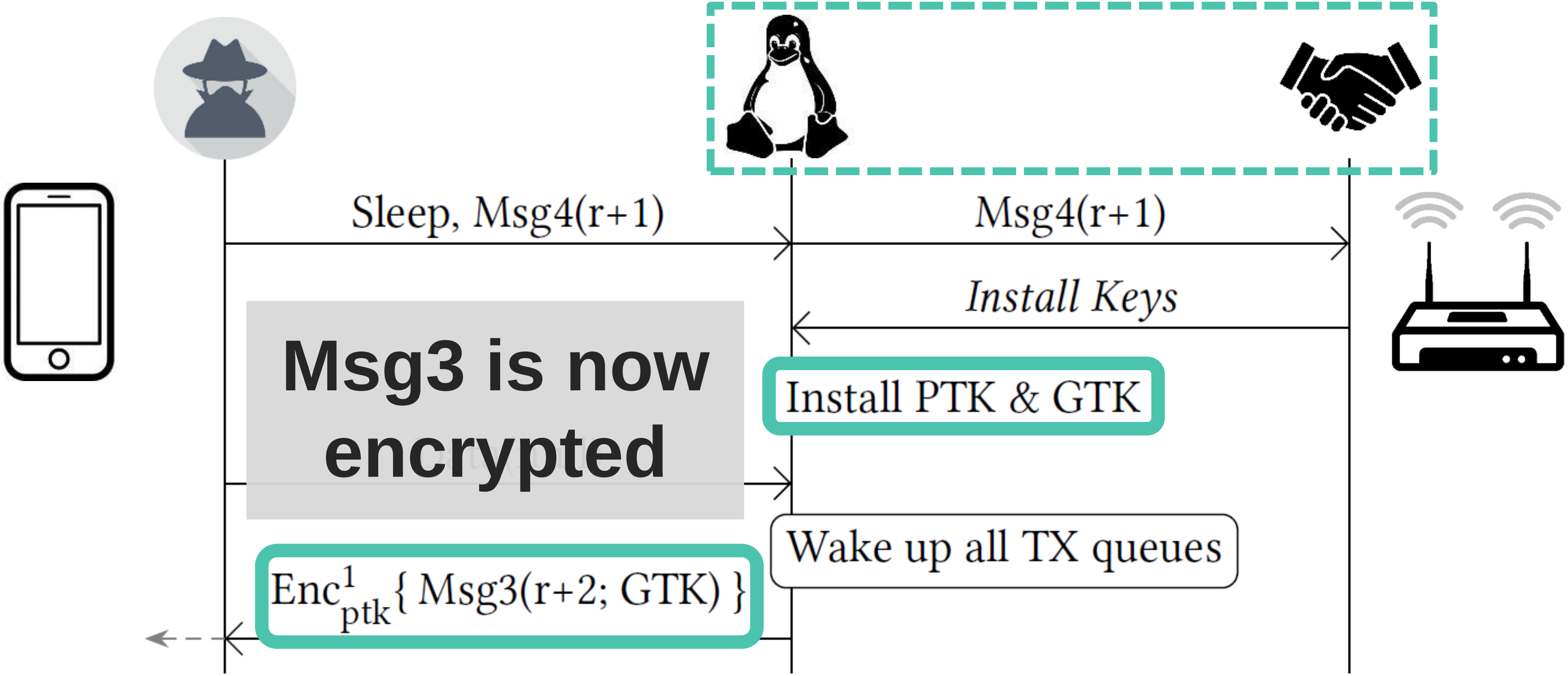
Install Keys

Install PTK & GTK

Data(null)

Wake up all TX queues

$\text{Enc}^{1}_{\text{ptk}}\{\ \text{Msg3}(r{+}2;\ \text{GTK})\ \}$



Sleep, Msg4(r+1)

Msg4(r+1)

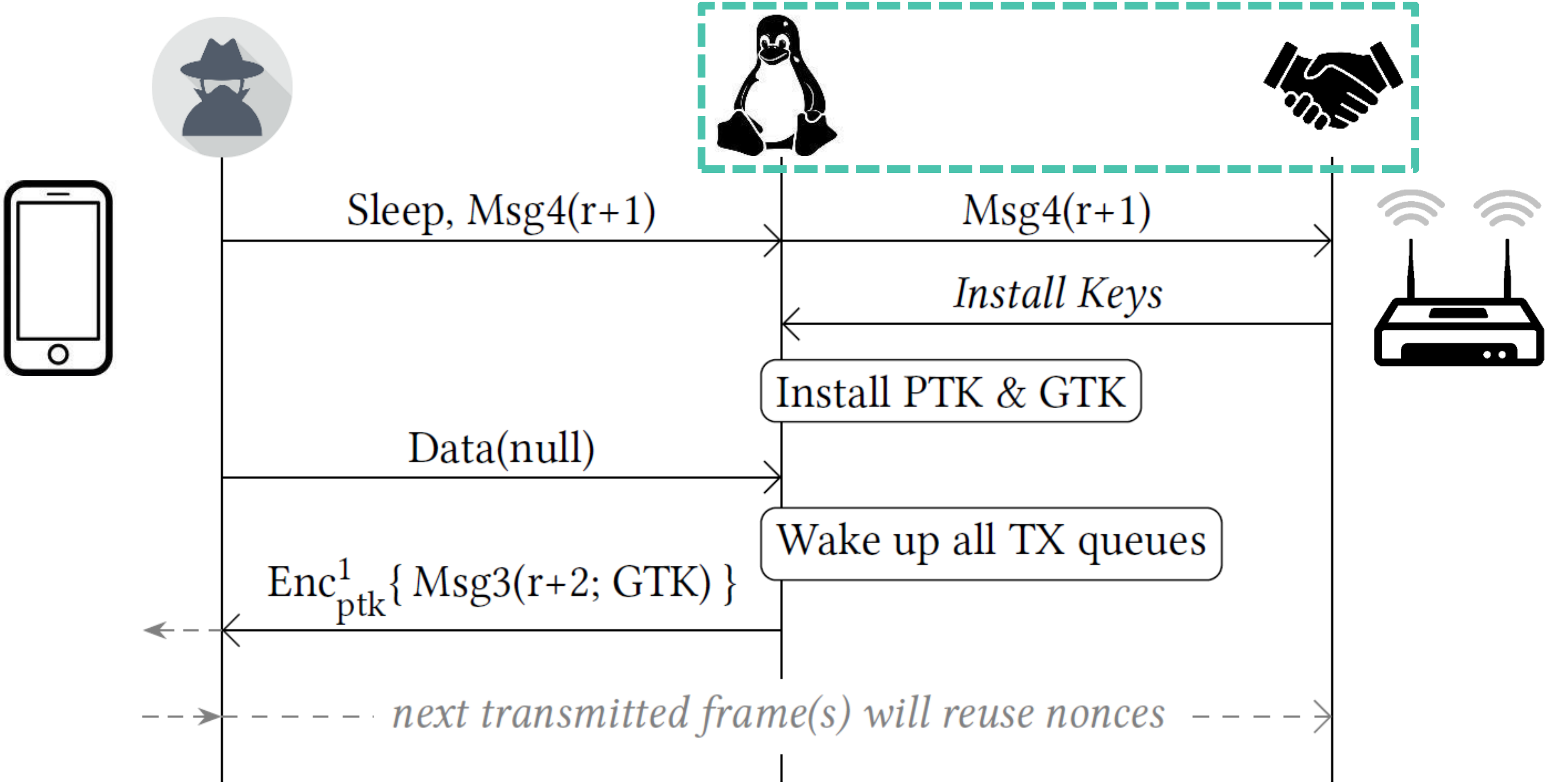
Install Keys

Install PTK & GTK

Msg3 is now encrypted

Wake up all TX queues

$\text{Enc}^{1}_{\text{ptk}}$\{ Msg3(r+2; GTK) \}


Sleep, Msg4(r+1)
Msg4(r+1)
Install Keys
Install PTK & GTK
Data(null)
Wake up all TX queues
$\text{Enc}^{1}_{\text{ptk}}\{\text{ Msg3(r+2; GTK) }\}$
next transmitted frame(s) will reuse nonces

Flawed countermeasure

802.11’s official countermeasure

“When the Key, Address, Key Type, and Key ID parameters identify an **existing key, the MAC shall not change the current transmitter TSC/PN/IPN counter** or the receiver replay counter values associated with that key.”

Bypassing 802.11’s countermeasure

Group key transported in two frames

- EAPOL-Key frames
- WNM-Sleep frames

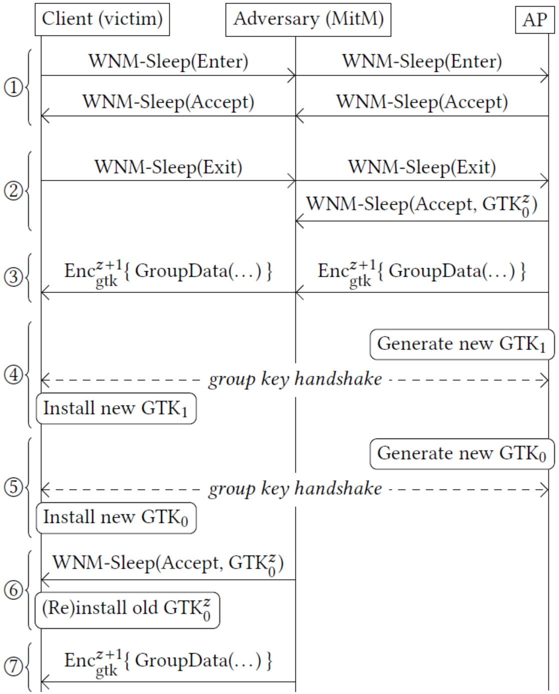
We can mix these frames

- WNM-Sleep installs new key
- Then EAPOL-Key reinstall old key

→ Can **reinstall the group key**

Details are non-trivial

WNM & Group HS



Client (victim)
Adversary (MitM)
AP

①
WNM-Sleep(Enter)
WNM-Sleep(Enter)
WNM-Sleep(Accept)
WNM-Sleep(Accept)

②
WNM-Sleep(Exit)
WNM-Sleep(Exit)
WNM-Sleep(Accept, GTK_0^z)

③
$Enc_{gtk}^{z+1}\{ GroupData(\dots) \}$
$Enc_{gtk}^{z+1}\{ GroupData(\dots) \}$

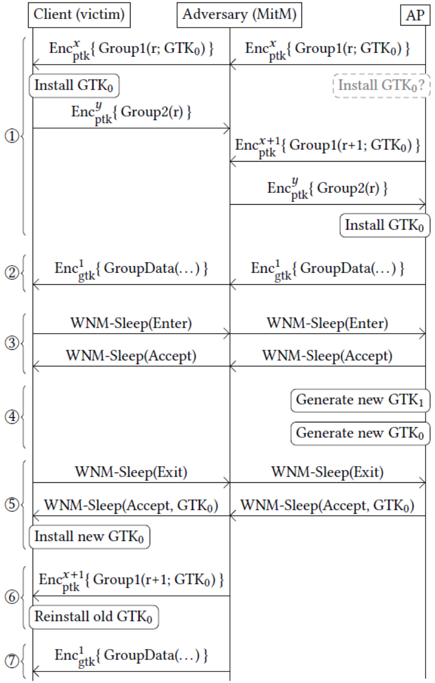
④
Generate new GTK_1
group key handshake
Install new GTK_1

⑤
Generate new GTK_0
group key handshake
Install new GTK_0

⑥
WNM-Sleep(Accept, GTK_0^z)
(Re)install old GTK_0^z

⑦
$Enc_{gtk}^{z+1}\{ GroupData(\dots) \}$

group HS & WNM



Client (victim)
Adversary (MitM)
AP

①
$Enc_{ptk}^{x}\{ Group1(r; GTK_0) \}$
$Enc_{ptk}^{x}\{ Group1(r; GTK_0) \}$
Install GTK_0
Install GTK_0?
$Enc_{ptk}^{y}\{ Group2(r) \}$
$Enc_{ptk}^{x+1}\{ Group1(r+1; GTK_0) \}$
$Enc_{ptk}^{y}\{ Group2(r) \}$
Install GTK_0

②
$Enc_{gtk}^{1}\{ GroupData(\dots) \}$
$Enc_{gtk}^{1}\{ GroupData(\dots) \}$

③
WNM-Sleep(Enter)
WNM-Sleep(Enter)
WNM-Sleep(Accept)
WNM-Sleep(Accept)

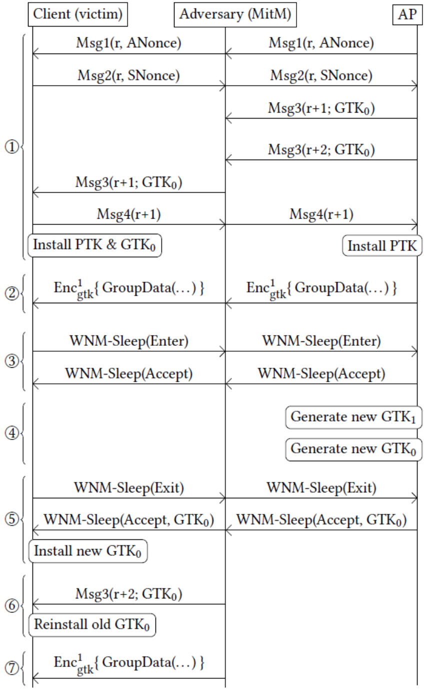
④
Generate new GTK_1
Generate new GTK_0

⑤
WNM-Sleep(Exit)
WNM-Sleep(Exit)
WNM-Sleep(Accept, GTK_0)
WNM-Sleep(Accept, GTK_0)
Install new GTK_0

⑥
$Enc_{ptk}^{x+1}\{ Group1(r+1; GTK_0) \}$
Reinstall old GTK_0

⑦
$Enc_{gtk}^{1}\{ GroupData(\dots) \}$

4-way HS & WNM



Client (victim)
Adversary (MitM)
AP

①
Msg1(r, ANonce)
Msg1(r, ANonce)
Msg2(r, SNonce)
Msg2(r, SNonce)
Msg3(r+1; GTK_0)
Msg3(r+2; GTK_0)
Msg3(r+1; GTK_0)
Msg4(r+1)
Msg4(r+1)
Install PTK & GTK_0
Install PTK

②
$Enc_{gtk}^{1}\{ GroupData(\dots) \}$
$Enc_{gtk}^{1}\{ GroupData(\dots) \}$

③
WNM-Sleep(Enter)
WNM-Sleep(Enter)
WNM-Sleep(Accept)
WNM-Sleep(Accept)

④
Generate new GTK_1
Generate new GTK_0

⑤
WNM-Sleep(Exit)
WNM-Sleep(Exit)
WNM-Sleep(Accept, GTK_0)
WNM-Sleep(Accept, GTK_0)
Install new GTK_0

⑥
Msg3(r+2; GTK_0)
Reinstall old GTK_0

⑦
$Enc_{gtk}^{1}\{ GroupData(\dots) \}$

Implementation Specific Flaws

Can we replay Message 4?

- Yes, certain MediaTek Drivers accept replayed Msg4's
- Used in 100+ devices → **many vulnerable products**

ASUS RT-AC51U

TP-Link RE370K

Are PTK rekeys implemented properly?

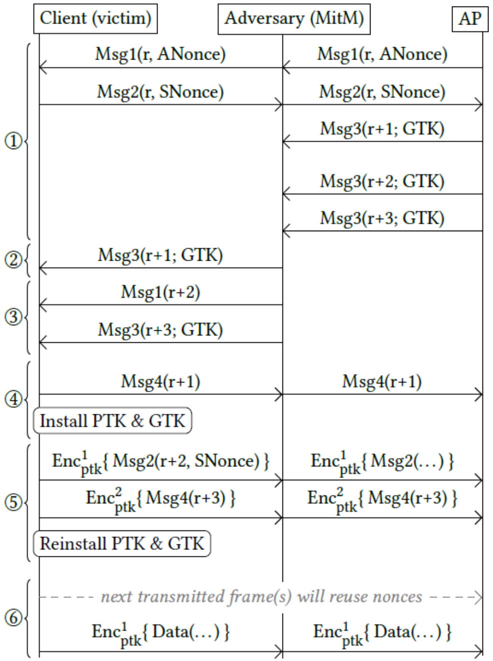
Rekey is a new 4-way handshake

- Same messages exchanged as in initial 4-way handshake
- But new ANonce and SNonce is used

macOS:

- Patched default KRACK attack
- But **reused the SNonce during a rekey**
- SNonce reuse patched in macOS 10.13.3

Exploiting macOS’s SNonce reuse


Client (victim)
Adversary (MitM)
AP
①
Msg1(r, ANonce)
Msg1(r, ANonce)
Msg2(r, SNonce)
Msg2(r, SNonce)
Msg3(r+1; GTK)
Msg3(r+2; GTK)
Msg3(r+3; GTK)
②
Msg3(r+1; GTK)
③
Msg1(r+2)
Msg3(r+3; GTK)
④
Msg4(r+1)
Msg4(r+1)
Install PTK & GTK
⑤
$\text{Enc}^1_{\text{ptk}}\{$ Msg2(r+2, SNonce) $\}$
$\text{Enc}^1_{\text{ptk}}\{$ Msg2(…) $\}$
$\text{Enc}^2_{\text{ptk}}\{$ Msg4(r+3) $\}$
$\text{Enc}^2_{\text{ptk}}\{$ Msg4(r+3) $\}$
Reinstall PTK & GTK
⑥
next transmitted frame(s) will reuse nonces
$\text{Enc}^1_{\text{ptk}}\{$ Data(…) $\}$
$\text{Enc}^1_{\text{ptk}}\{$ Data(…) $\}$

Adversary can replay old handshake

- Need to inject **encrypted message 1**
- Feasible under specific conditions
- Causes **key reinstallation**

Conclusion

- We made attacks more practical
- Bypassed official countermeasure
- Handling group keys is hard
- Keep auditing devices & protocols!

Thank you!

Questions?

krackattacks.com/followup.html