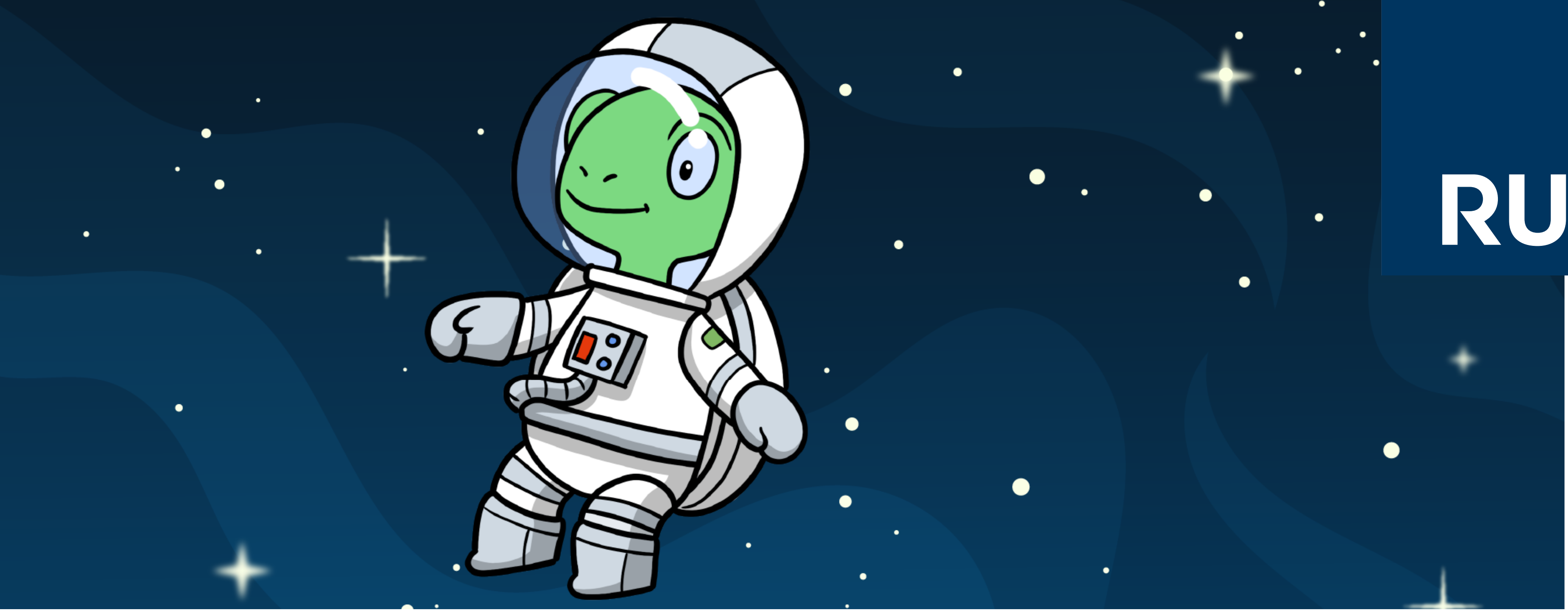
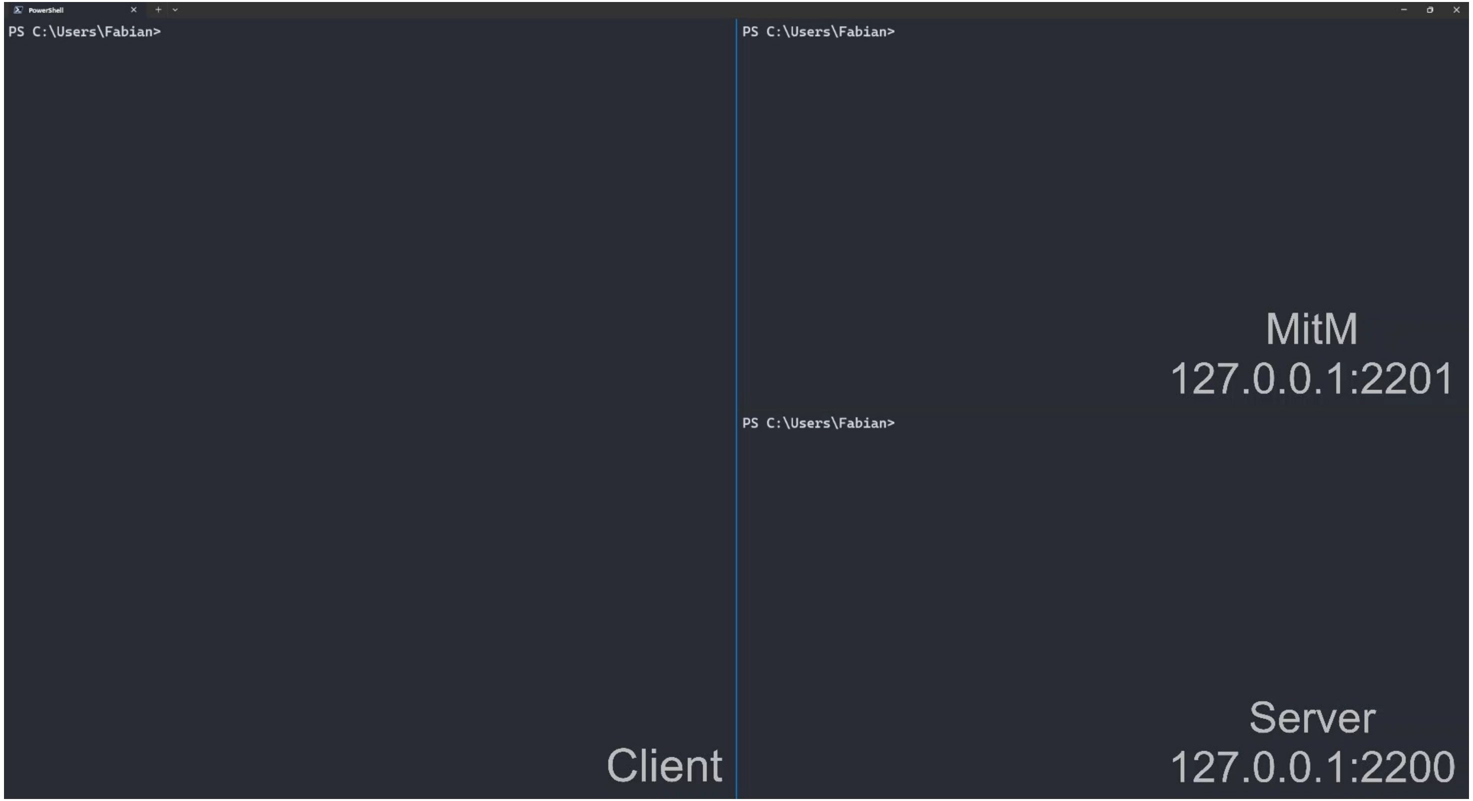


RUHR-UNIVERSITÄT BOCHUM

TERRAPIN ATTACK: BREAKING SSH CHANNEL INTEGRITY BY SEQUENCE NUMBER MANIPULATION

Fabian Bäumer, Marcus Brinkmann, Jörg Schwenk | RuhrSec 2025

Live Demo



```
PS C:\Users\Fabian>
```

```
PS C:\Users\Fabian>
```

```
PS C:\Users\Fabian>
```

In The Next 30 Minutes You Will Learn…

- … how an attacker was able to mess with the victim‘s user authentication
- … what a Terrapin attack is and how it is related to the live demo
- … the specific requirements for the attack to work
- … how you can protect yourself against similar attacks

Beyond that,

- … how the SSH protocol establishes connections
- … how adding modern cryptography to older protocols can go wrong

SSH
Protocol Flow

SSH Is Split Into Separate Layers

SSH Connection Protocol [RFC4254]

SSH Authentication Protocol [RFC4252]

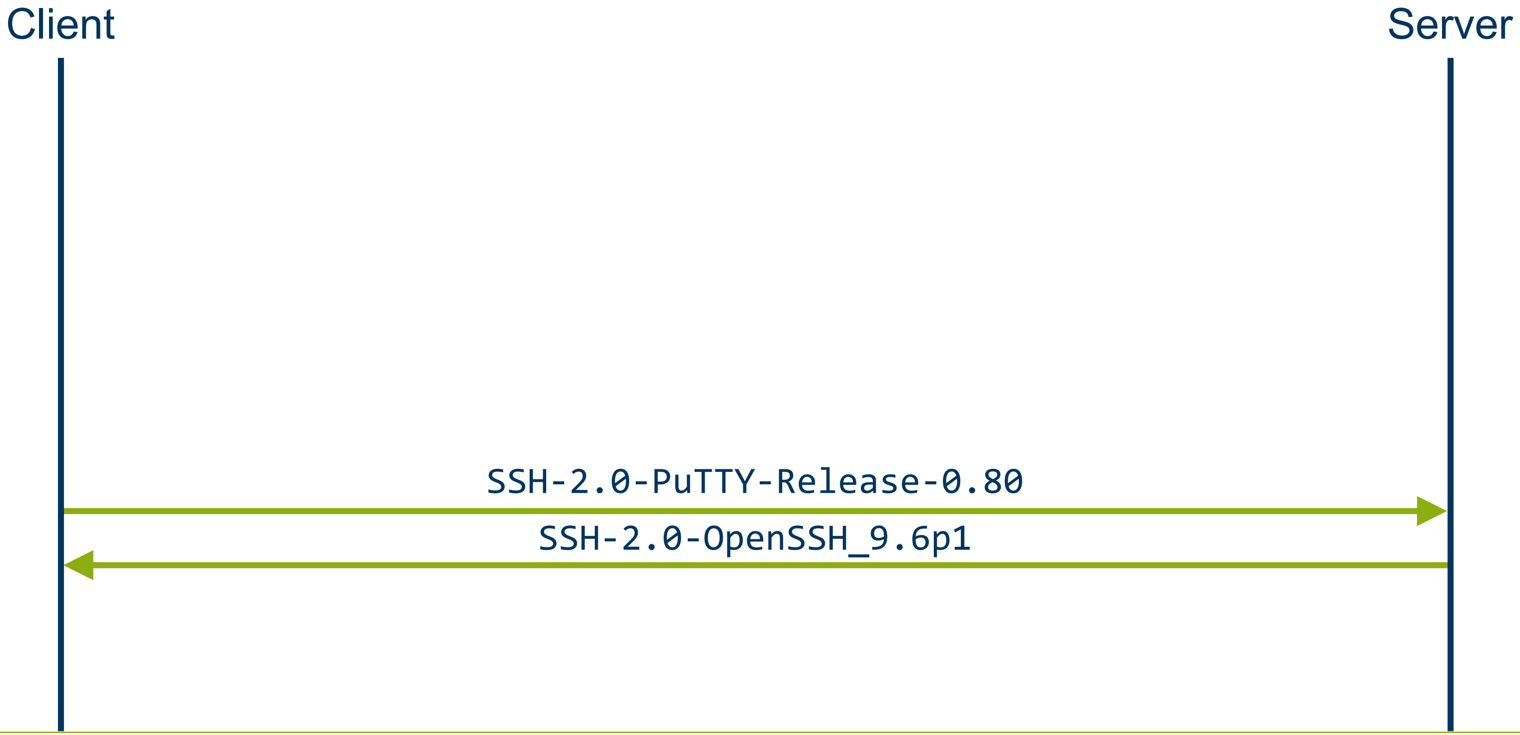
SSH Transport Layer Protocol (TLP) [RFC4253]

→ Binary Packet Protocol

→ SSH Handshake

TCP / IP

Step 1: Exchange of Protocol Version



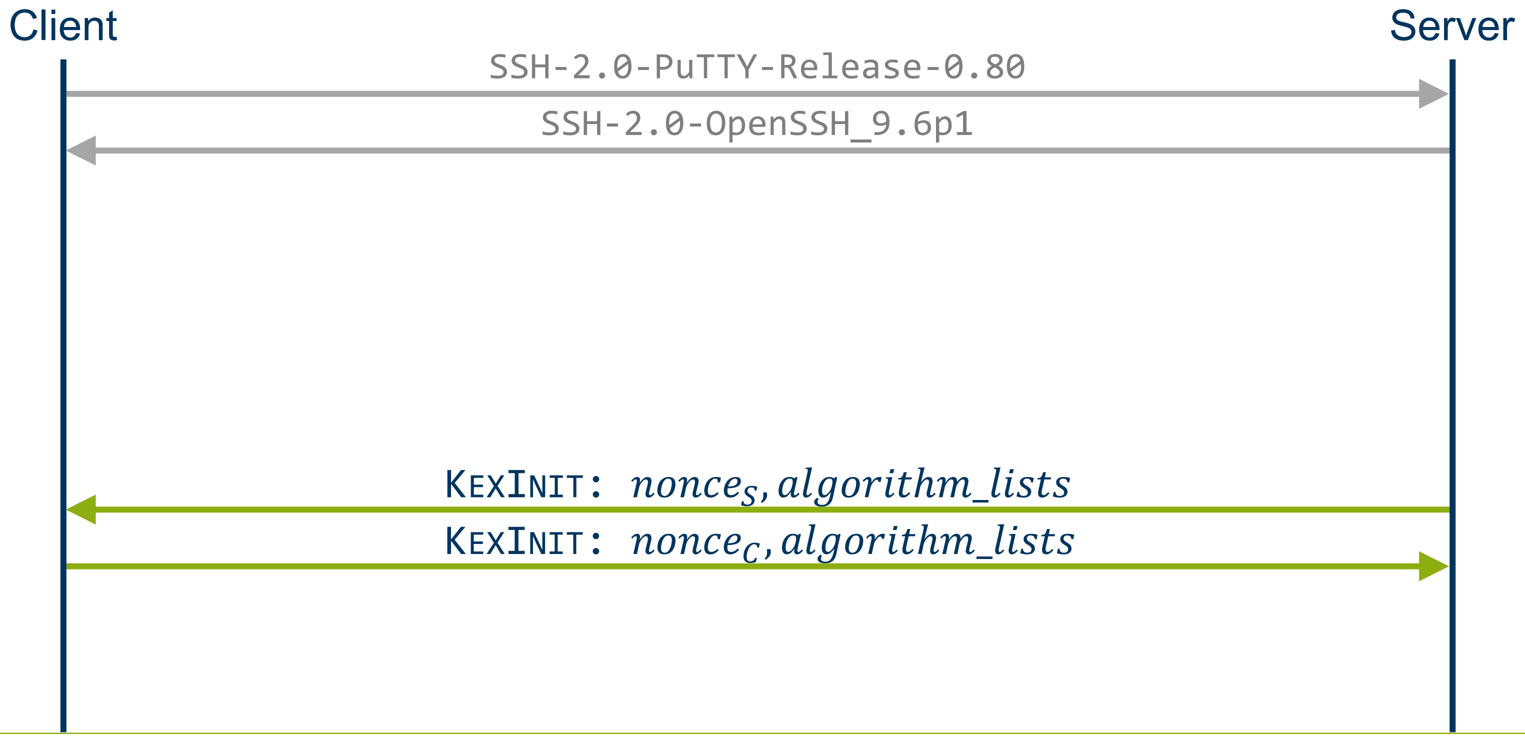
Client

Server

SSH-2.0-PuTTY-Release-0.80

SSH-2.0-OpenSSH_9.6p1

Step 2: Exchange of Supported Algorithms



Client

Server

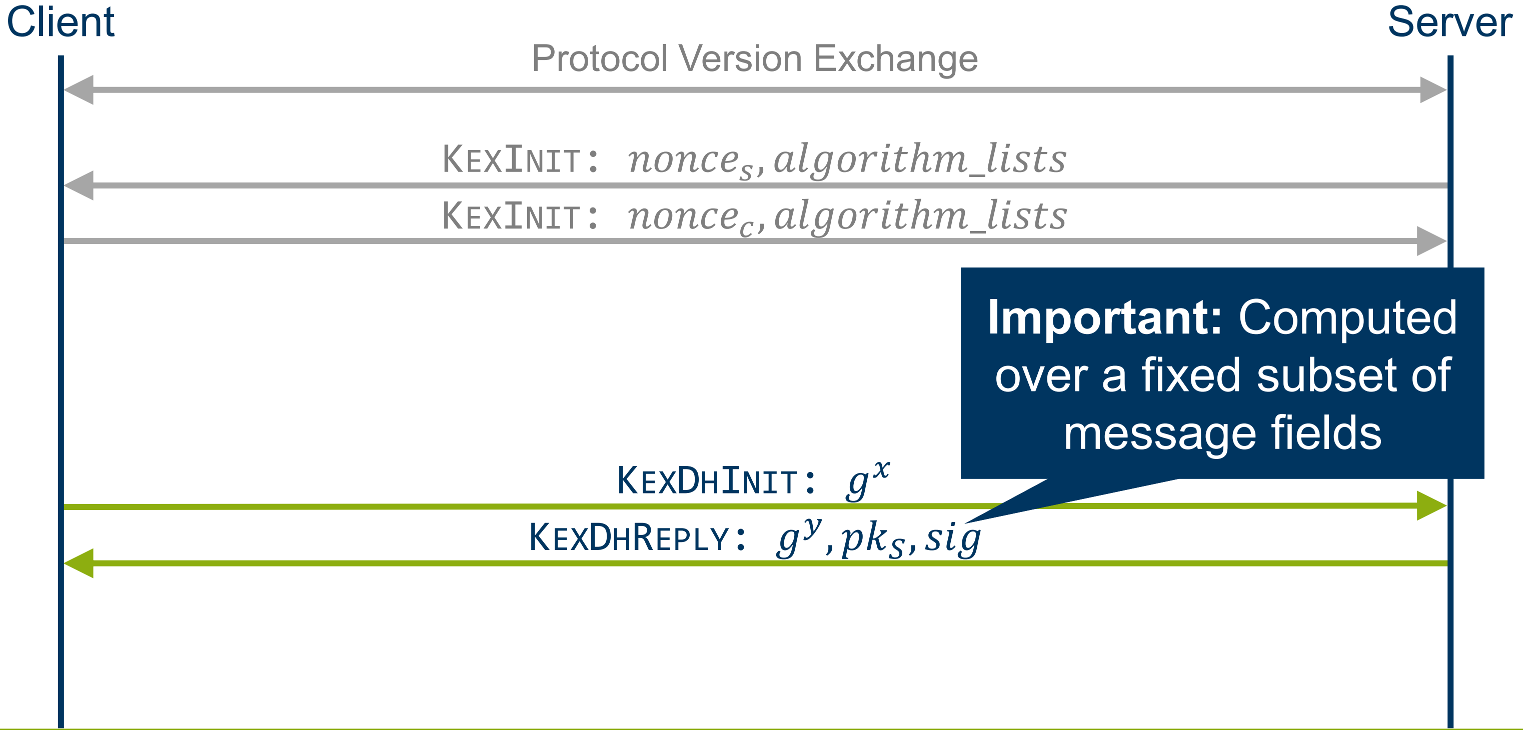
SSH-2.0-PuTTY-Release-0.80

SSH-2.0-OpenSSH_9.6p1

KEXINIT: $nonce_S, algorithm\_lists$

KEXINIT: $nonce_C, algorithm\_lists$

Step 3: Performing Key Exchange



Client

Server

Protocol Version Exchange

KexInit: $nonce_s, algorithm\_lists$

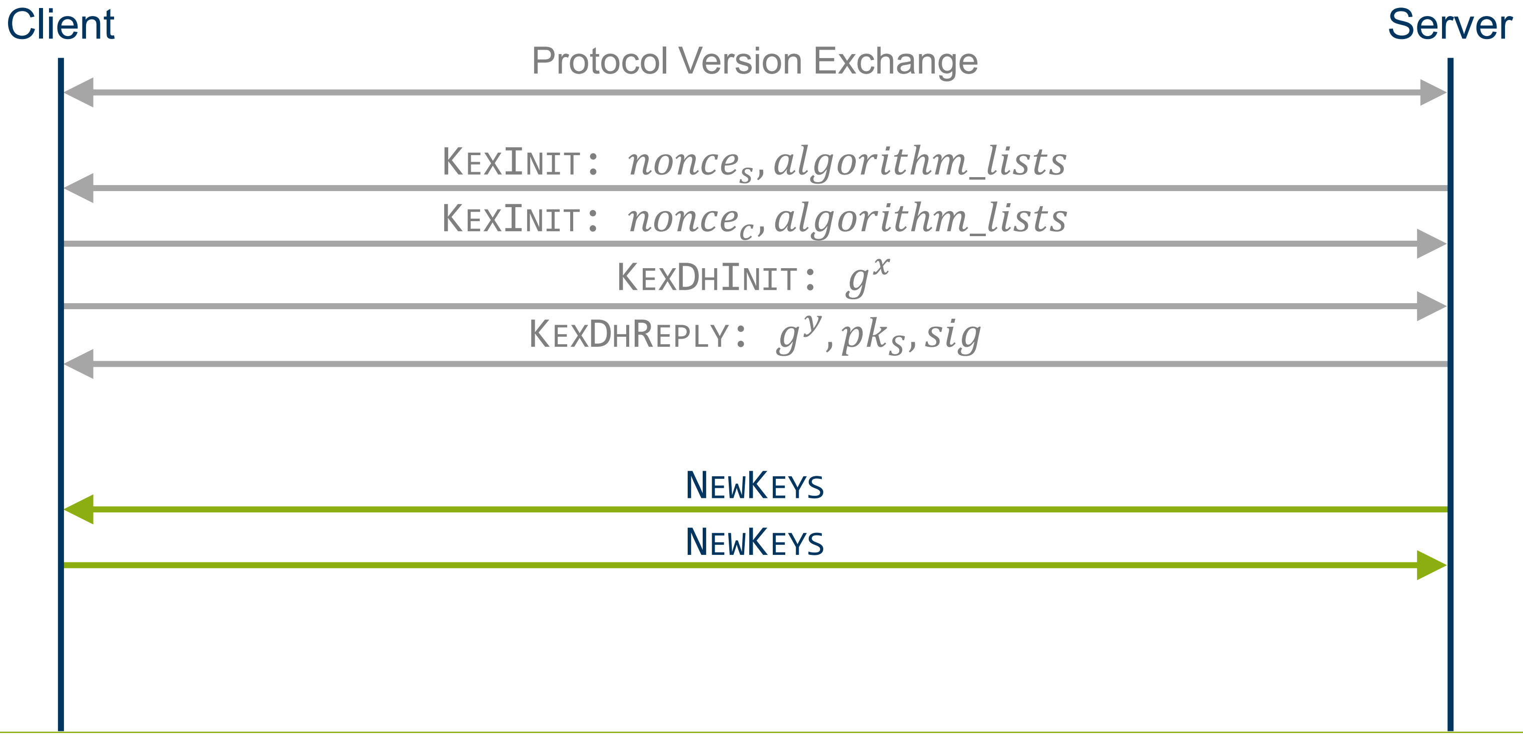
KexInit: $nonce_c, algorithm\_lists$

Important: Computed over a fixed subset of message fields

KexDhInit: g^x

KexDhReply: g^y, pk_S, sig

Step 4: Activating the Secure Channel



Client

Server

Protocol Version Exchange

KexInit: $nonce_s, algorithm\_lists$

KexInit: $nonce_c, algorithm\_lists$

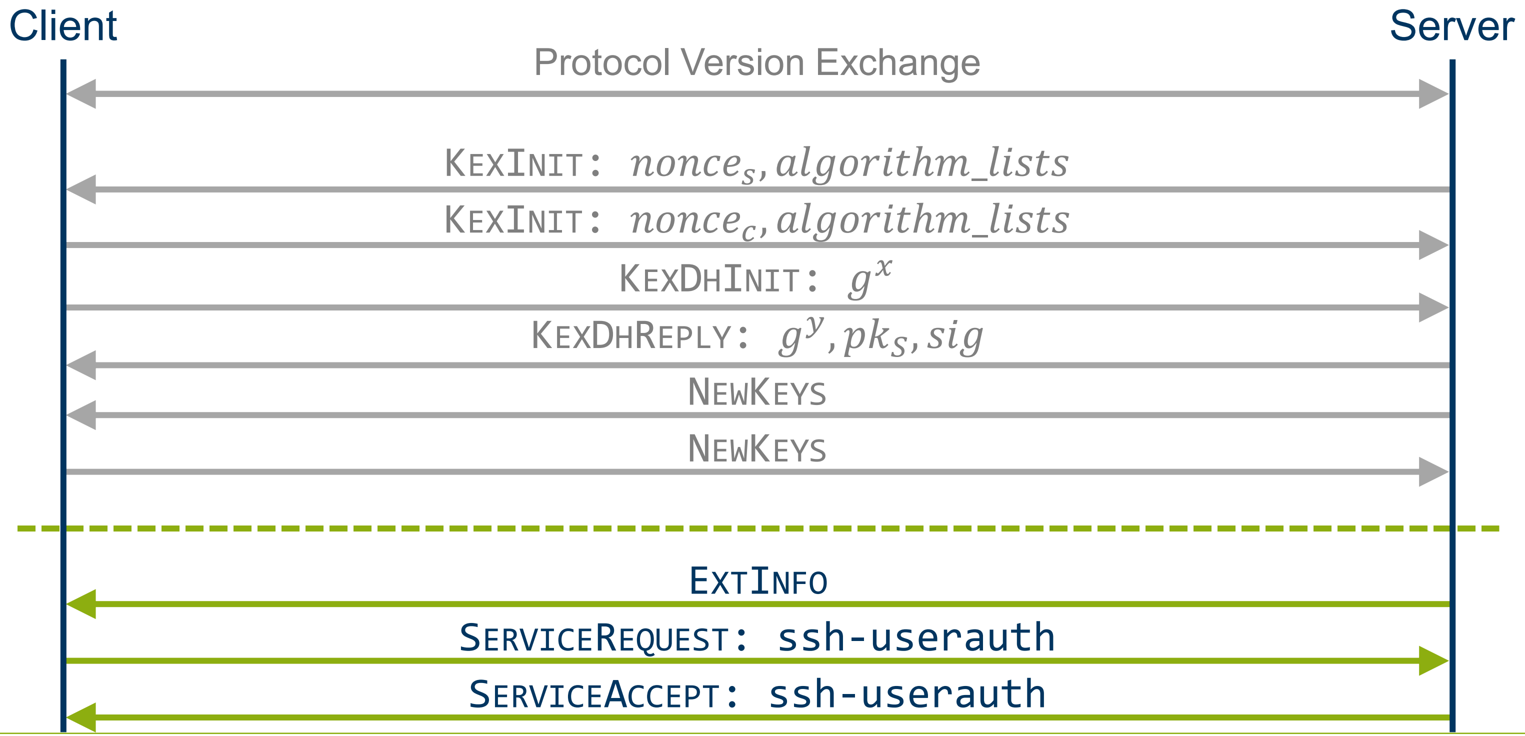
KexDhInit: g^x

KexDhReply: g^y, pk_S, sig

NewKeys

NewKeys

Step 5: Request User Authentication Service



Client

Server

Protocol Version Exchange

KexInit: $nonce_s, algorithm\_lists$

KexInit: $nonce_c, algorithm\_lists$

KexDhInit: g^x

KexDhReply: g^y, pk_S, sig

NewKeys

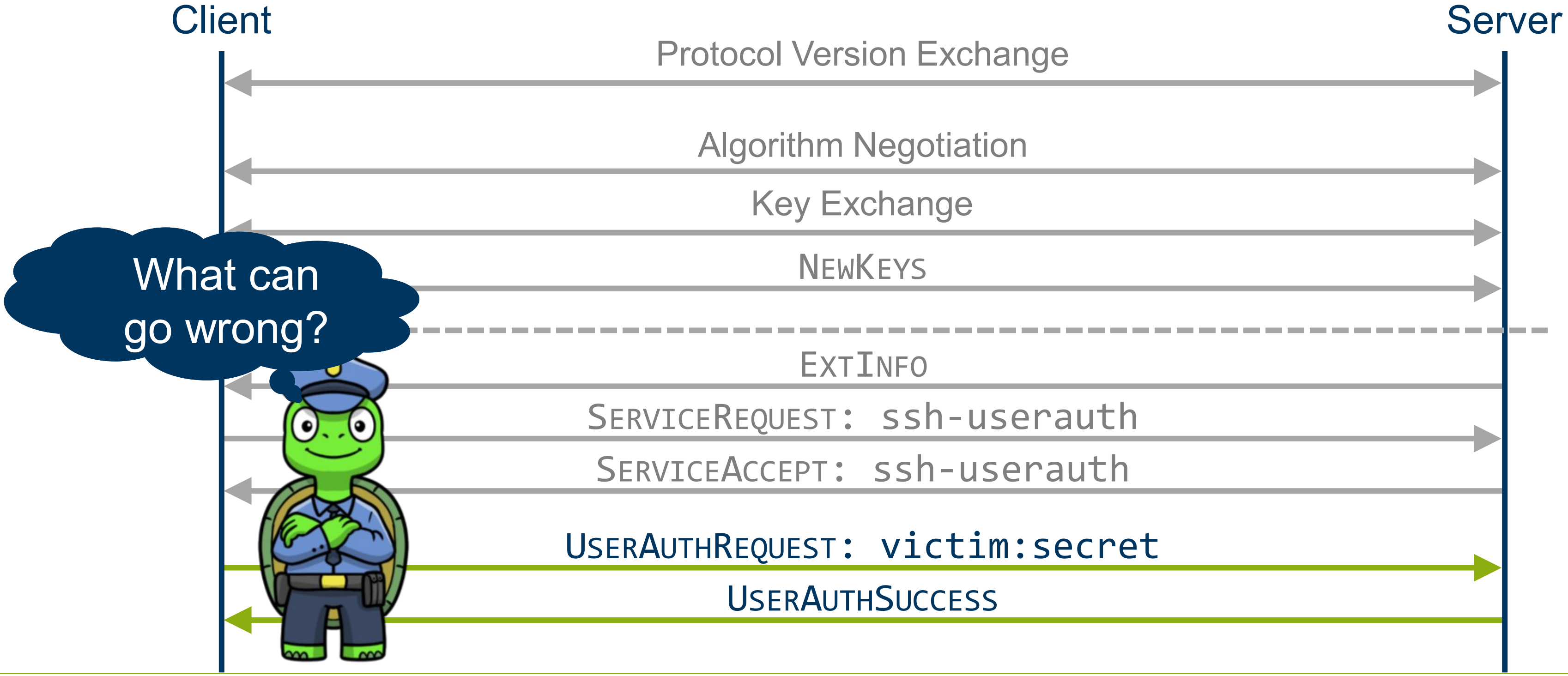
NewKeys

ExtInfo

ServiceRequest: ssh-userauth

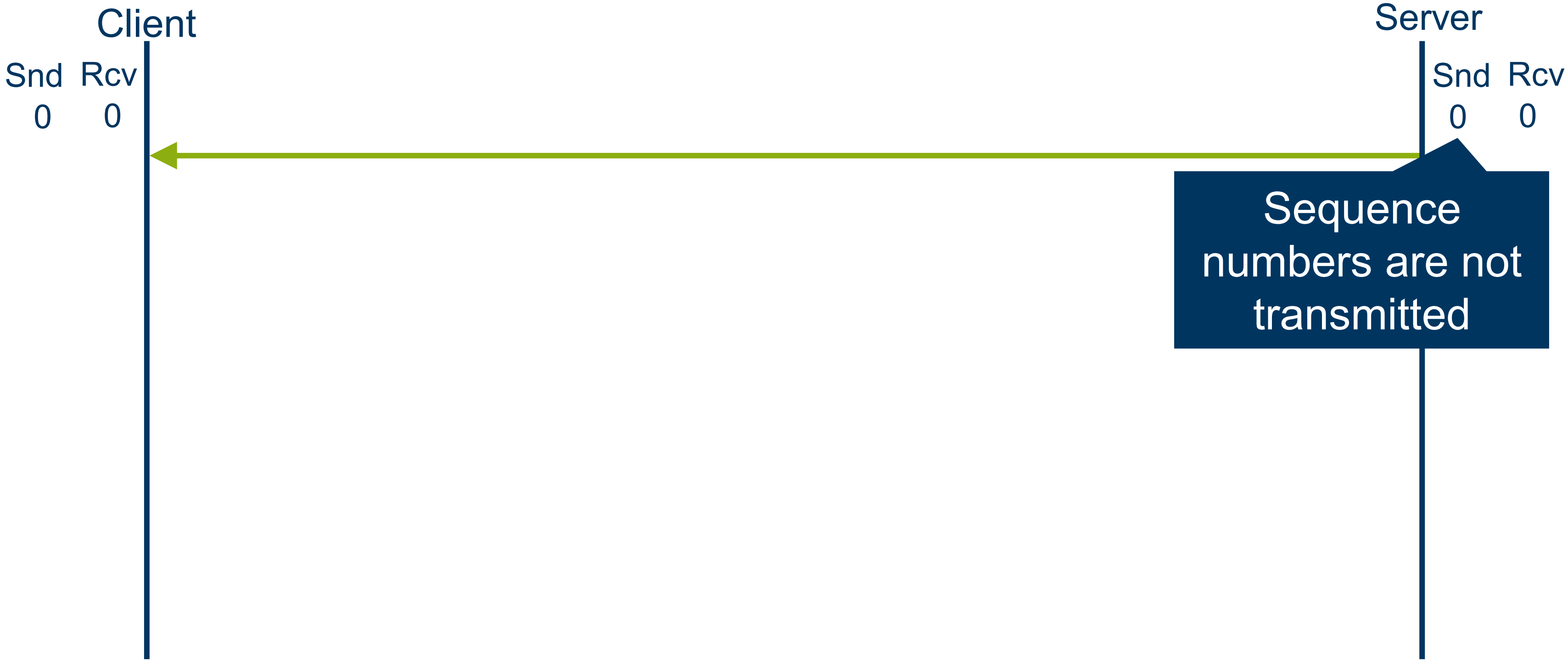
ServiceAccept: ssh-userauth

Step 6: Authenticating the User


Client
Server
Protocol Version Exchange
Algorithm Negotiation
Key Exchange
NEWKEYS
EXTINFO
SERVICEREQUEST: ssh-userauth
SERVICEACCEPT: ssh-userauth
USERAUTHREQUEST: victim:secret
USERAUTHSUCCESS
What can go wrong?

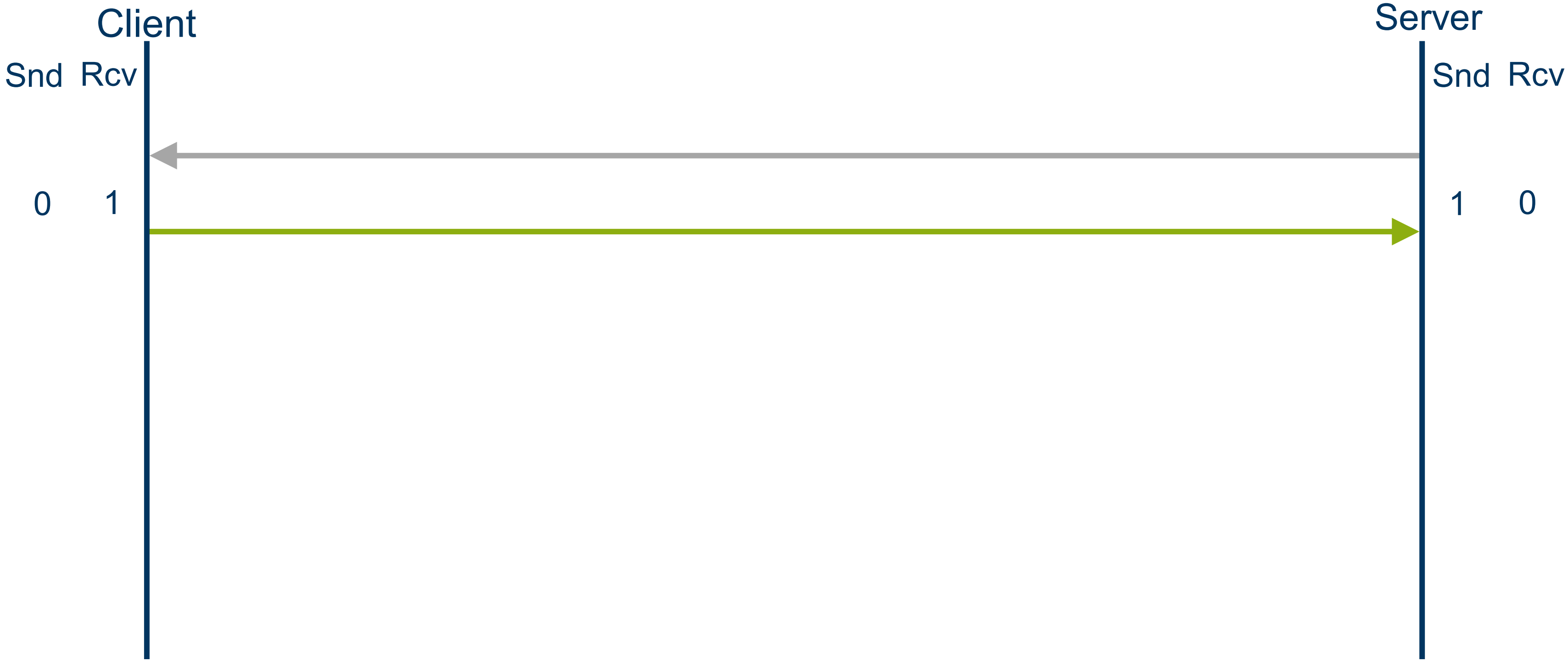
SSH
Sequence Numbers

SSH Uses Implicit Sequence Numbers

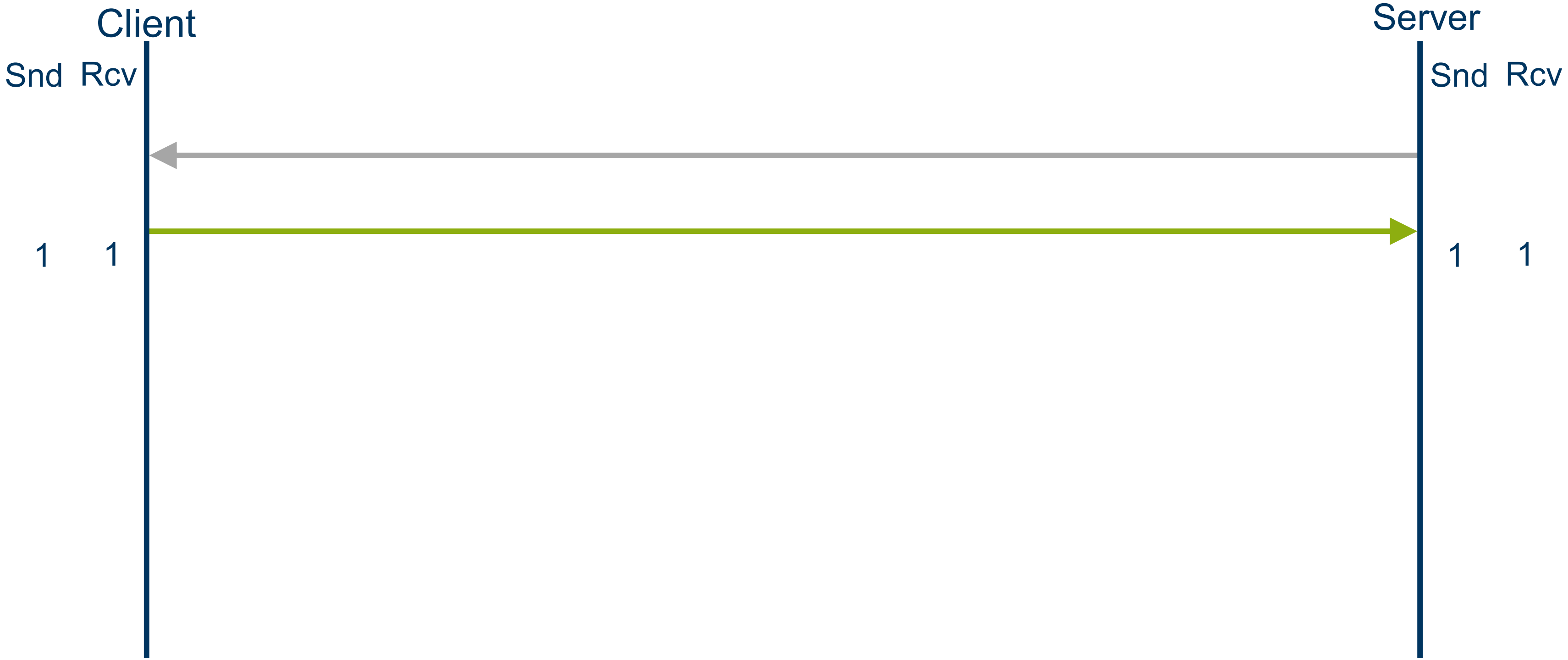


Sequence numbers are not transmitted

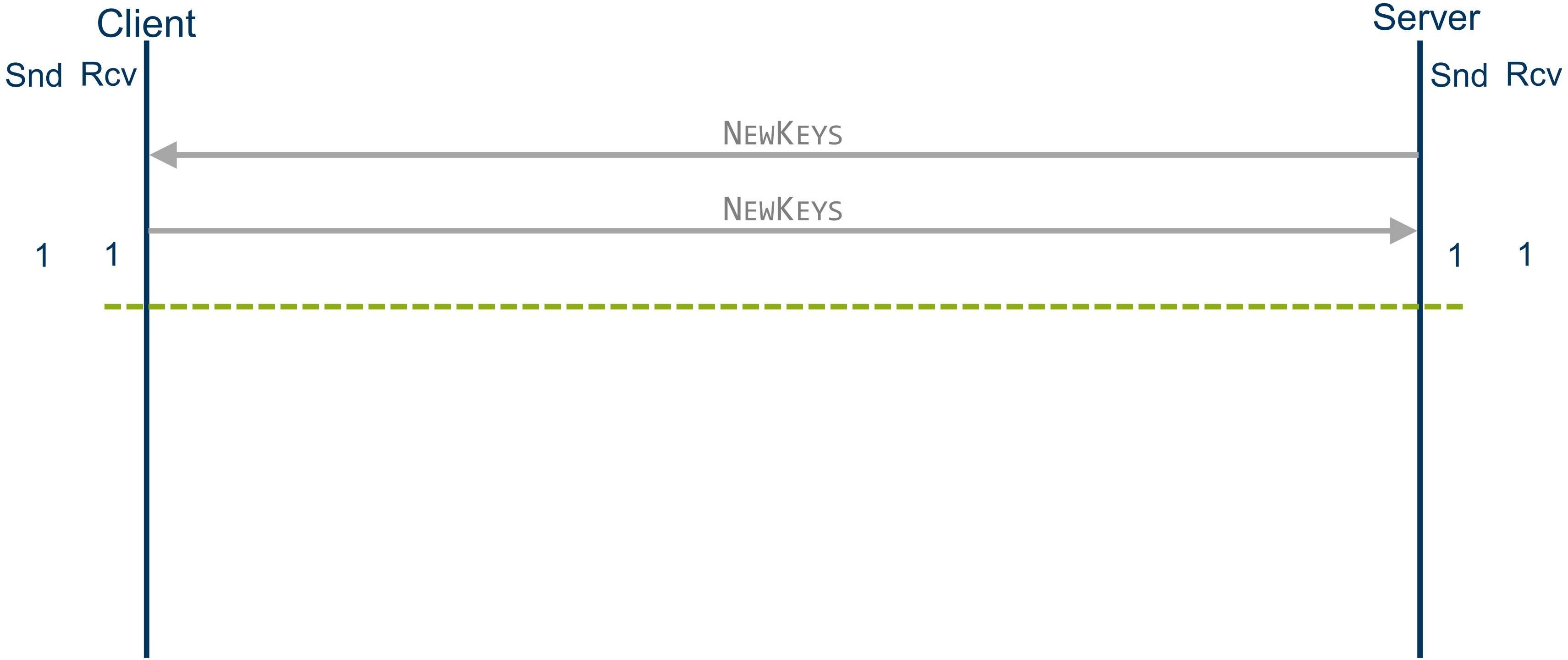
SSH Uses Implicit Sequence Numbers



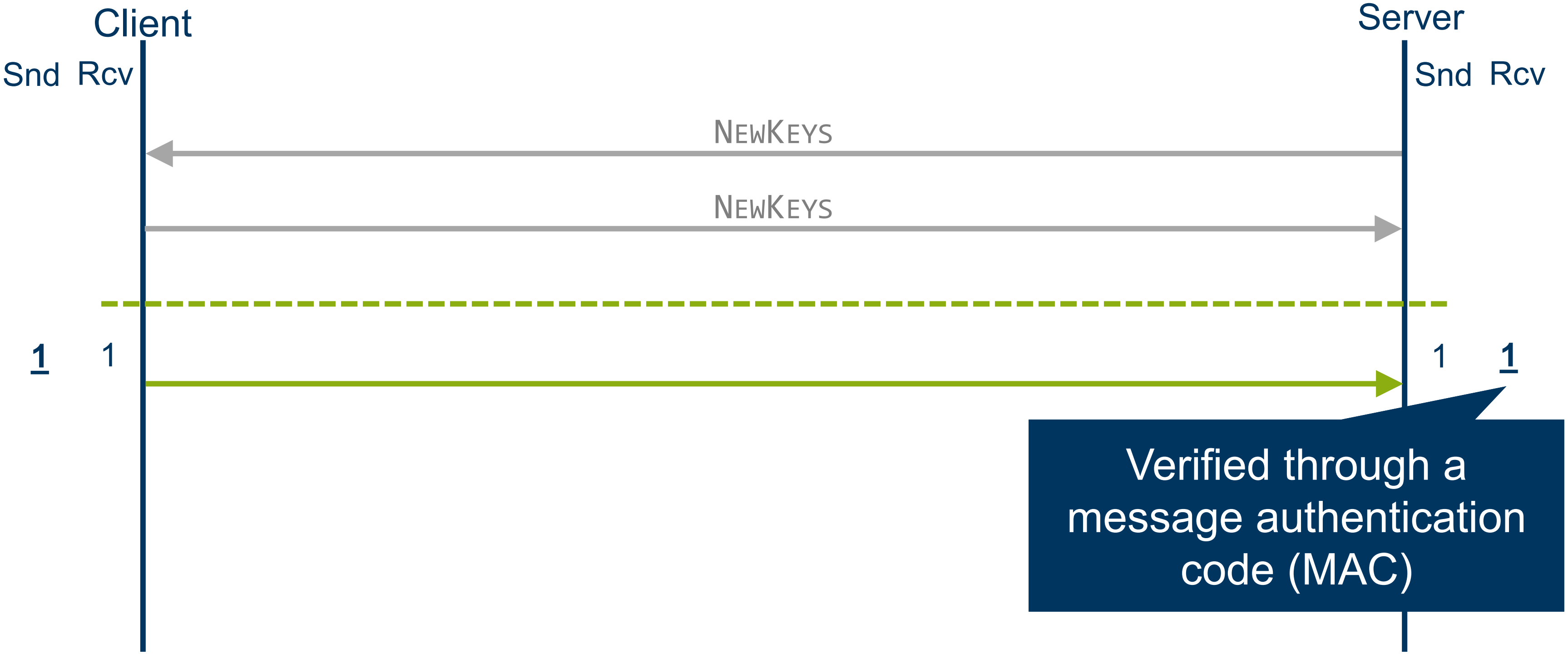
SSH Uses Implicit Sequence Numbers



SSH Uses Implicit Sequence Numbers

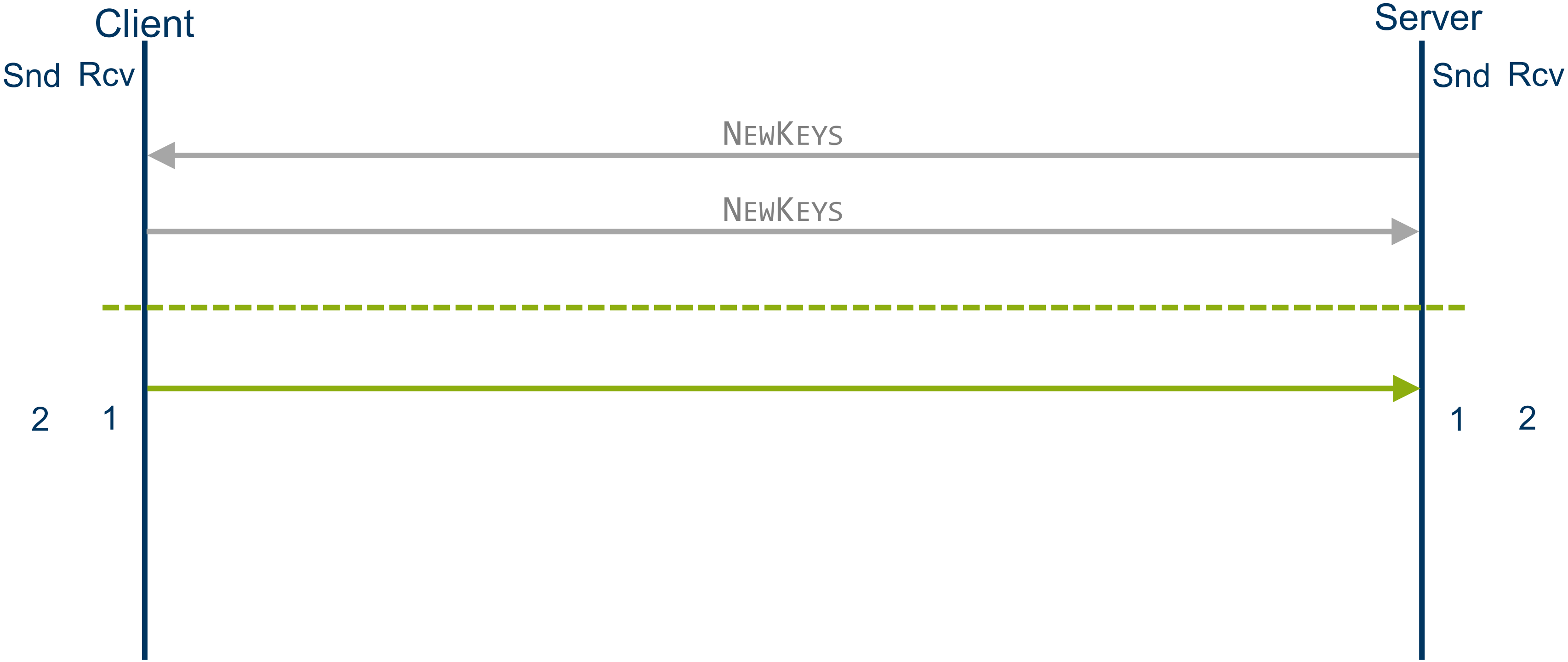


SSH Uses Implicit Sequence Numbers

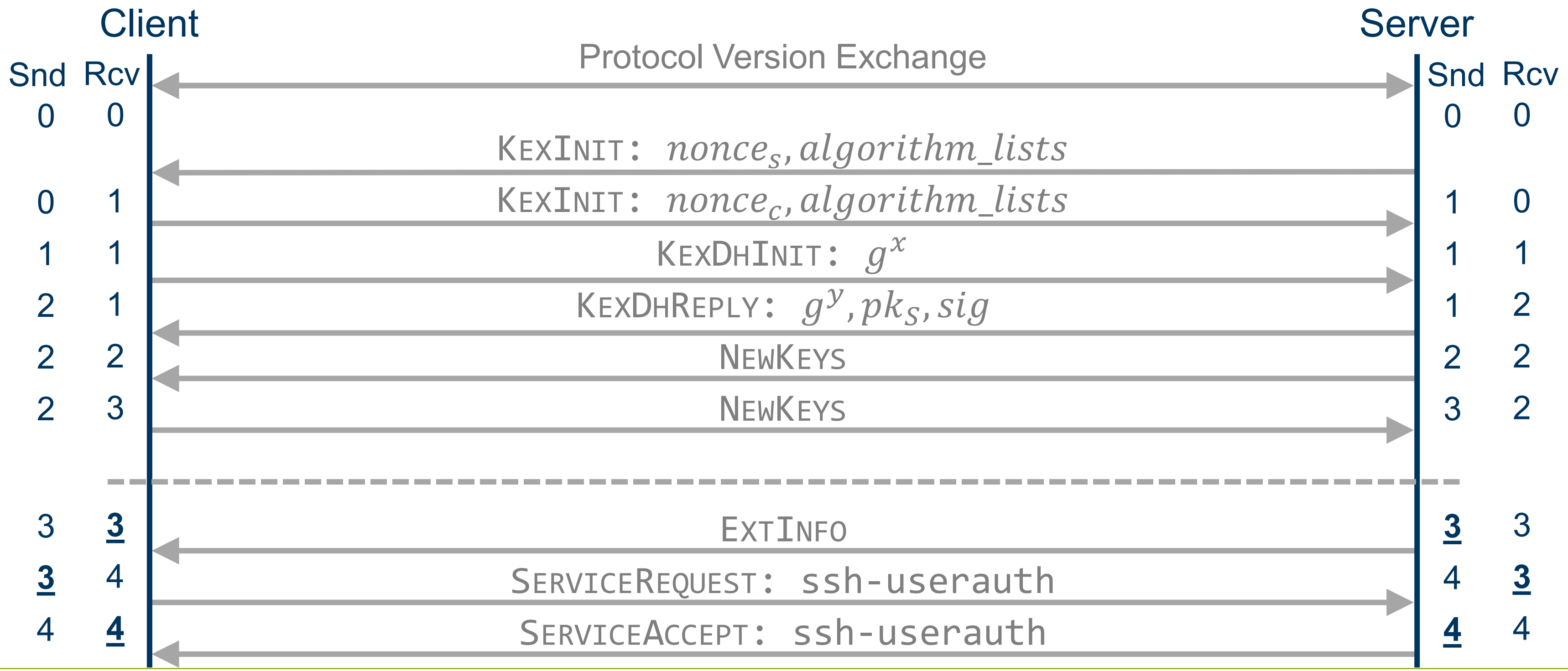


Verified through a message authentication code (MAC)

SSH Uses Implicit Sequence Numbers



Introducing Sequence Numbers to the Flow

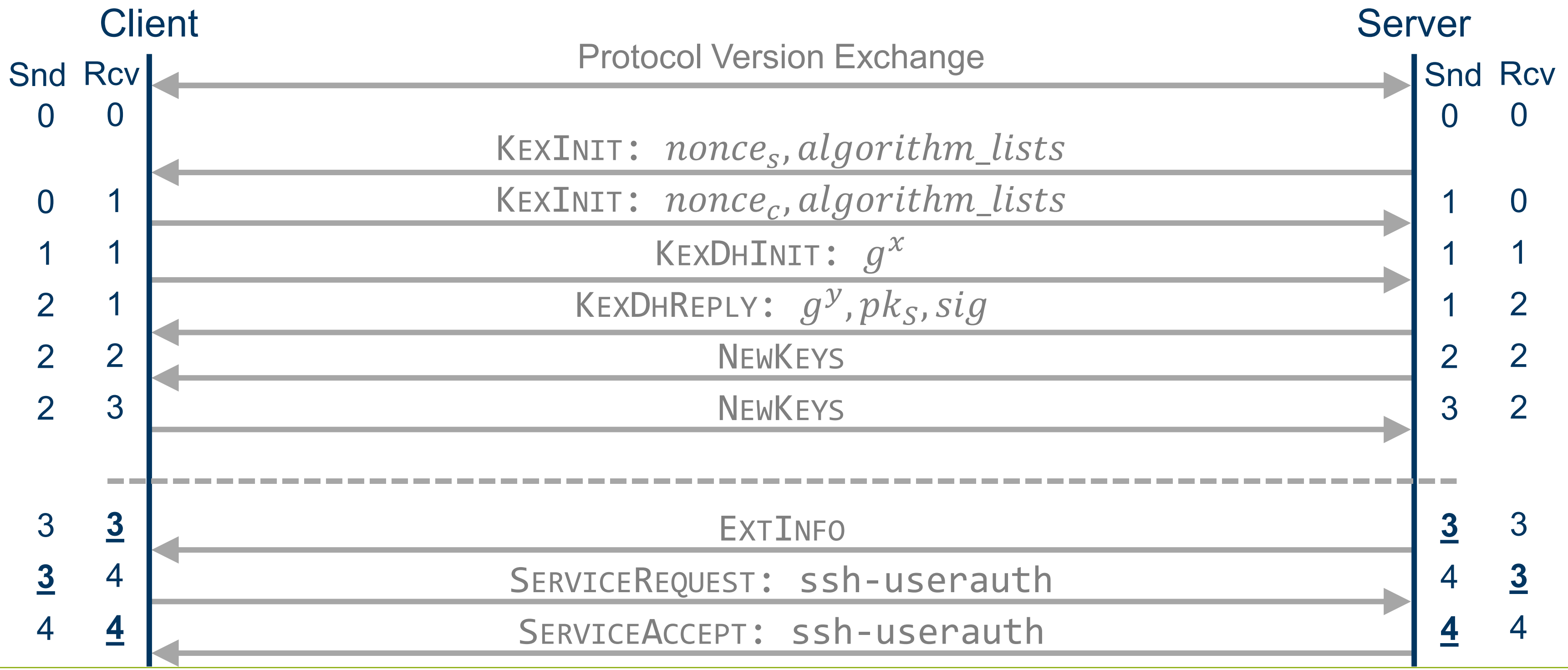


Client Snd	Client Rcv	Message	Server Snd	Server Rcv
		↔ Protocol Version Exchange		
0	0	← KEXINIT: $nonce_s, algorithm\_lists$	0	0
0	1	→ KEXINIT: $nonce_c, algorithm\_lists$	1	0
1	1	→ KEXDHINIT: g^x	1	1
2	1	← KEXDHREPLY: g^y, pk_S, sig	1	2
2	2	← NEWKEYS	2	2
2	3	→ NEWKEYS	3	2
3	$\underline{3}$	← EXTINFO	$\underline{3}$	3
$\underline{3}$	4	→ SERVICEREQUEST: ssh-userauth	4	$\underline{3}$
4	$\underline{4}$	← SERVICEACCEPT: ssh-userauth	$\underline{4}$	4

Terrapin Attack

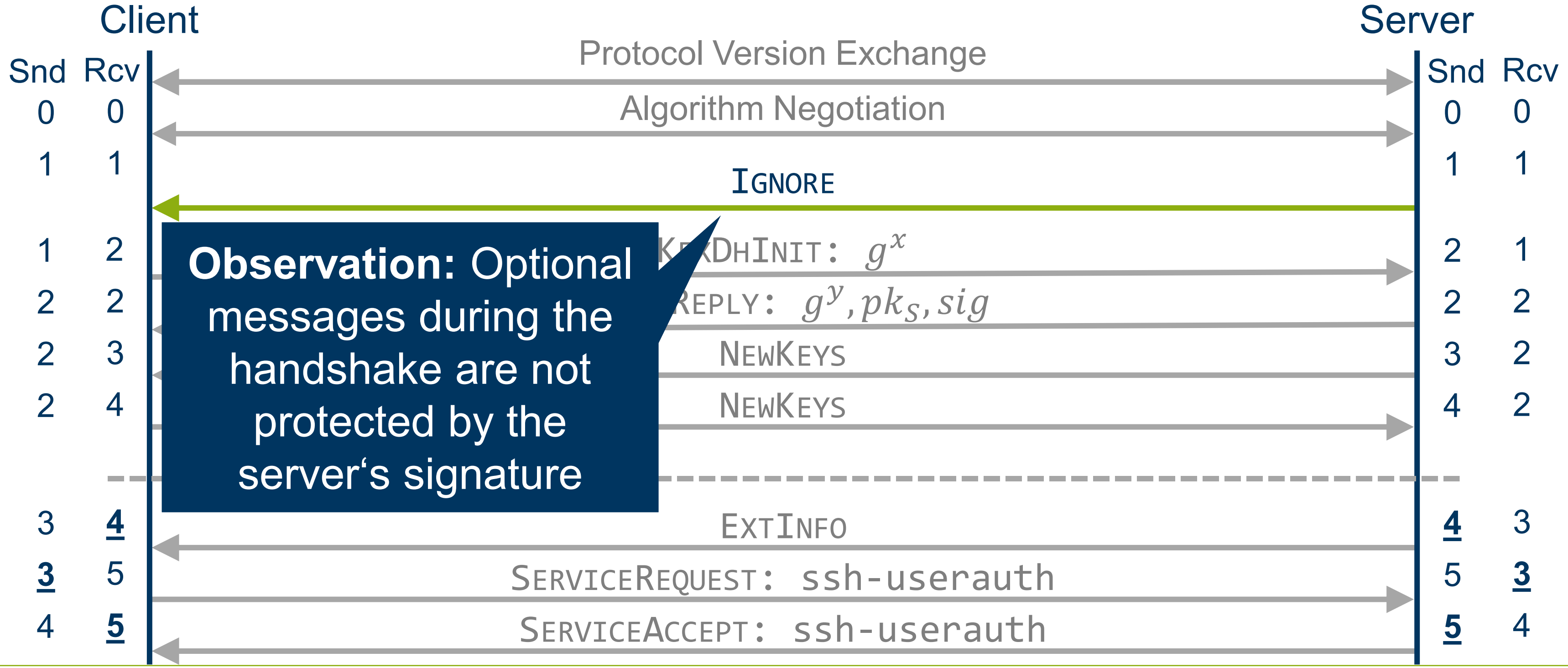
Standard-Compliant Attack

SSH Allows for Optional Messages in Handshakes



Client			Server	
Snd	Rcv		Snd	Rcv
		Protocol Version Exchange ↔		
0	0		0	0
		KEXINIT: $nonce_s, algorithm\_lists$ ←		
0	1	KEXINIT: $nonce_c, algorithm\_lists$ →	1	0
1	1	KEXDHINIT: g^x →	1	1
2	1	KEXDHREPLY: g^y, pk_S, sig ←	1	2
2	2	NEWKEYS ←	2	2
2	3	NEWKEYS →	3	2
3	**3**	EXTINFO ←	**3**	3
3	4	SERVICEREQUEST: ssh-userauth →	4	**3**
4	**4**	SERVICEACCEPT: ssh-userauth ←	**4**	4

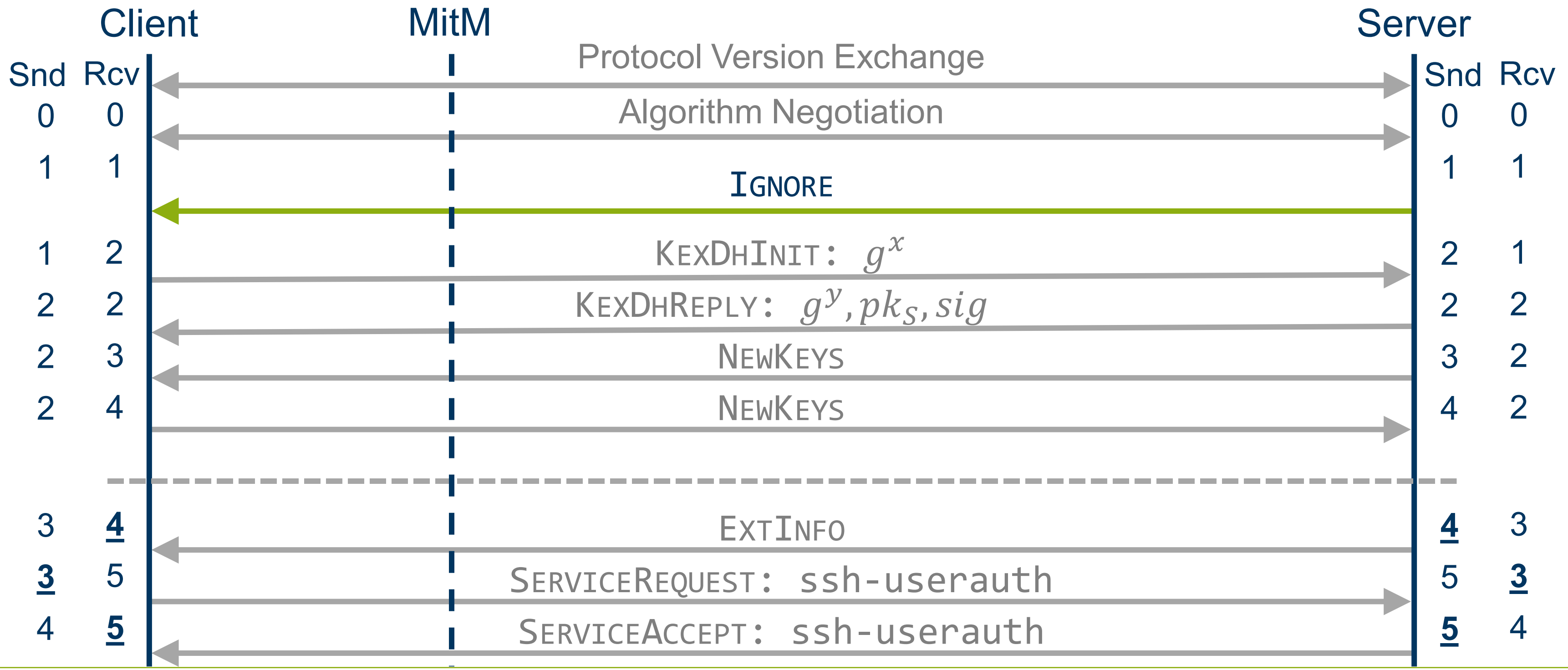
SSH Allows for Optional Messages in Handshakes



Client Snd	Client Rcv	Message	Server Snd	Server Rcv
		Protocol Version Exchange		
0	0	Algorithm Negotiation	0	0
1	1		1	1
		IGNORE		
1	2	KexDhInit: g^x	2	1
2	2	KexDhReply: g^y, pk_S, sig	2	2
2	3	NewKeys	3	2
2	4	NewKeys	4	2
3	4	ExtInfo	4	3
3	5	ServiceRequest: ssh-userauth	5	3
4	5	ServiceAccept: ssh-userauth	5	4

Observation: Optional messages during the handshake are not protected by the server‘s signature

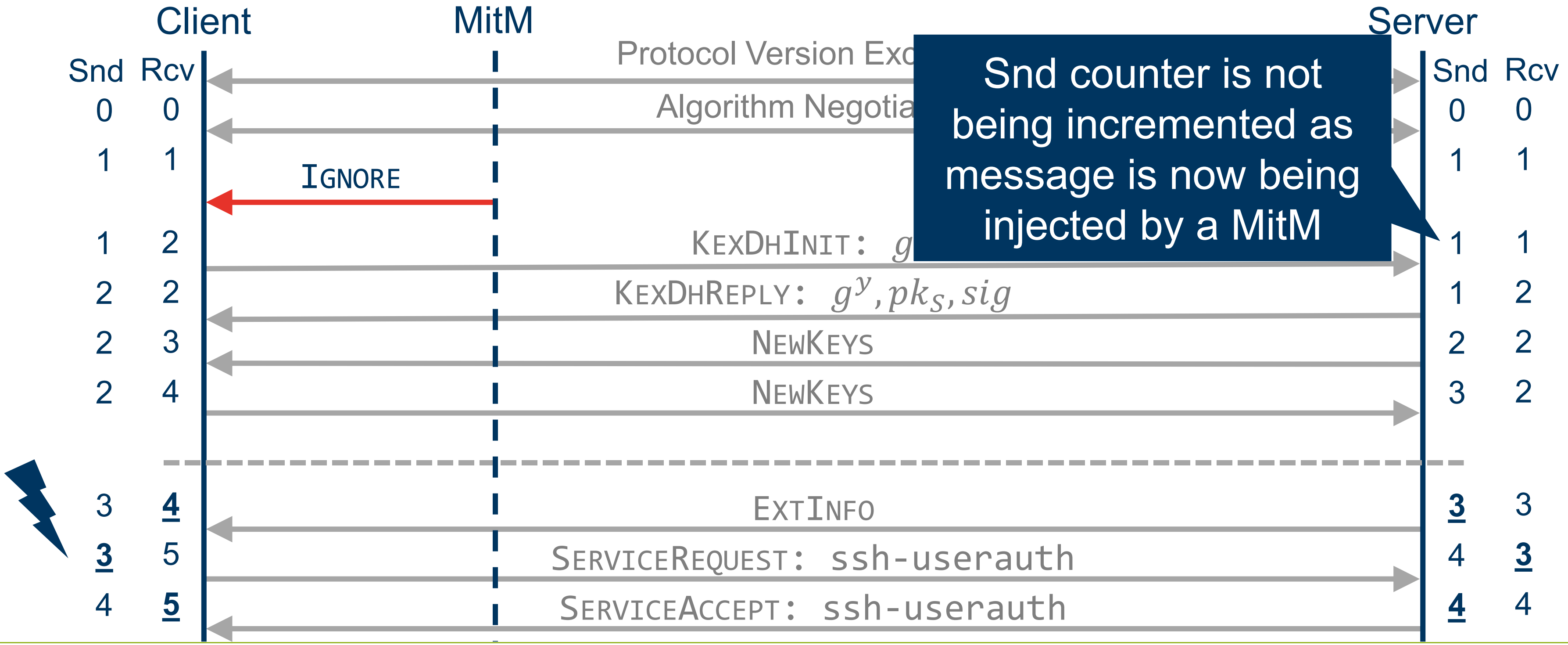
SSH Allows for Optional Messages in Handshakes



Client | MitM | Server

Snd	Rcv		Snd	Rcv
		Protocol Version Exchange		
0	0	Algorithm Negotiation	0	0
1	1		1	1
		Ignore		
1	2	KexDhInit: g^x	2	1
2	2	KexDhReply: g^y, pk_S, sig	2	2
2	3	NewKeys	3	2
2	4	NewKeys	4	2
3	**4**	ExtInfo	**4**	3
3	5	ServiceRequest: ssh-userauth	5	**3**
4	**5**	ServiceAccept: ssh-userauth	**5**	4

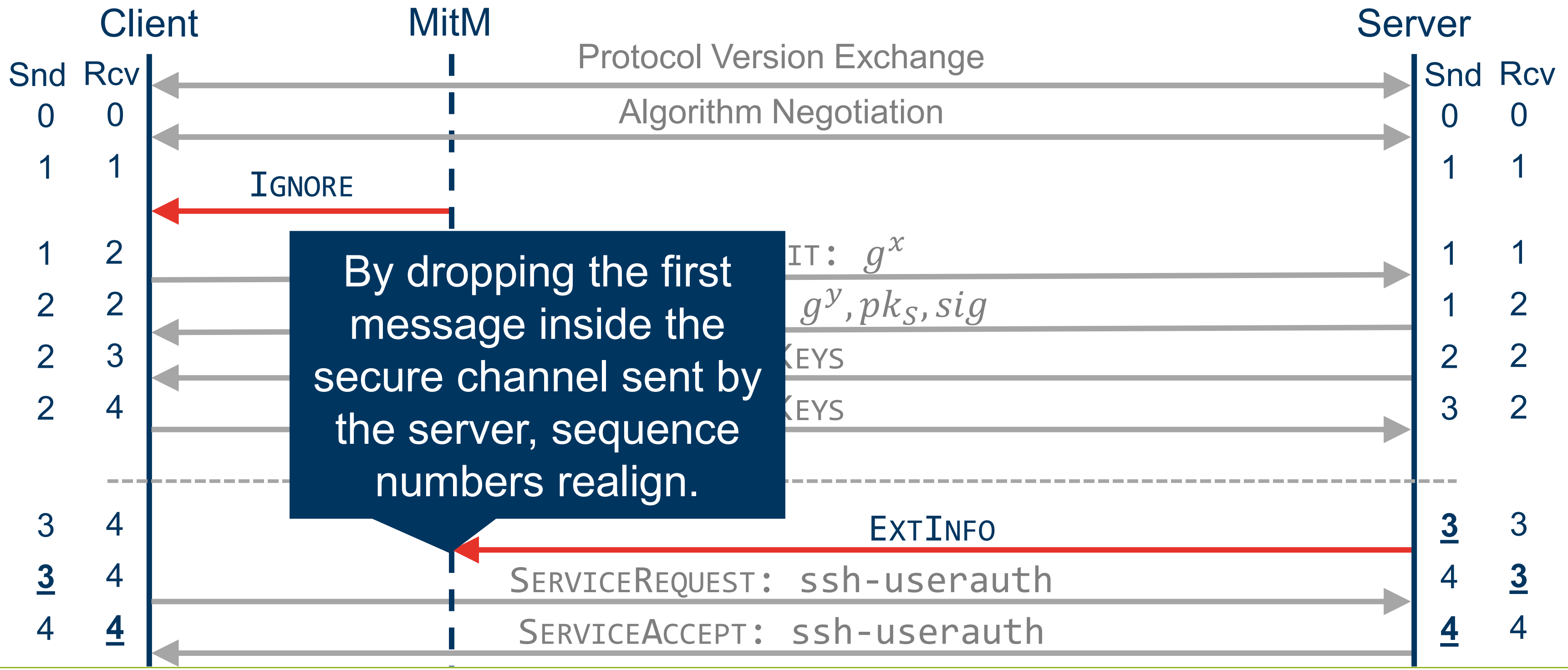
MitM Attackers Can Inject Messages Into Handshake…



Snd counter is not being incremented as message is now being injected by a MitM

… And Drop Messages Inside The Secure Channel

CVE-2023-48795
(CVE-2024-41909)



By dropping the first message inside the secure channel sent by the server, sequence numbers realign.

The ExtInfo Message Contains Extensions as Key-Value Pairs

server-sig-algs

- List of public key algorithms for user authentication
- Enables RSA-SHA2 support

ping@openssh.com

- Like Heartbeat extension in TLS
- Can be used to obscure keystroke timings

Other Extensions

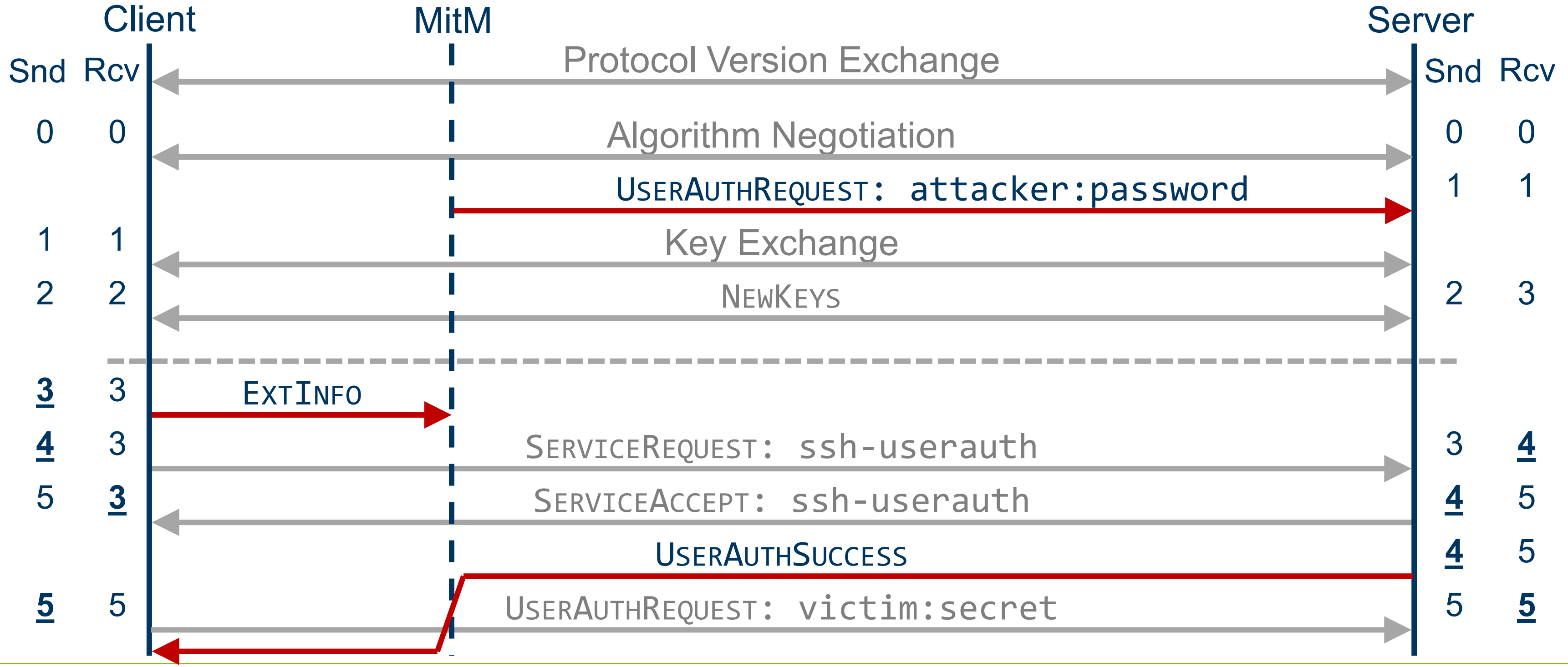
- Not considered because no security impact

Terrapin Attack

Exploiting Implementation Flaws

Implementation Bugs Can Escalate Impact

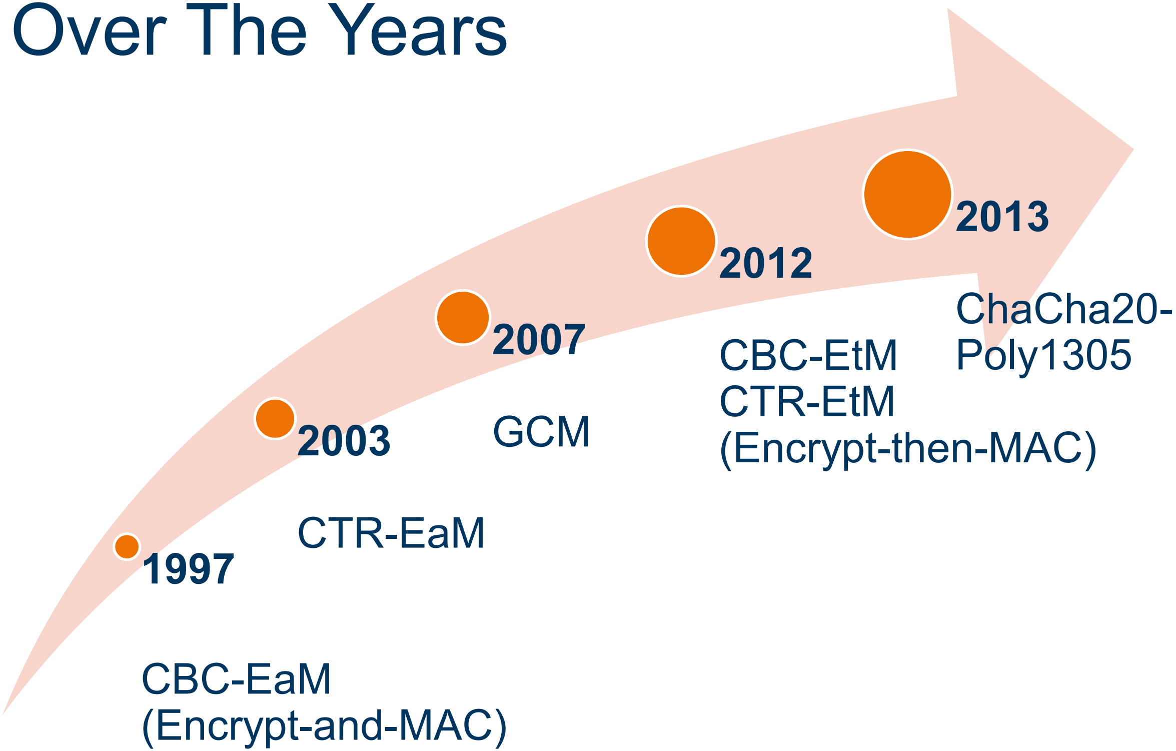
CVE-2023-46445
CVE-2023-46446


Client
MitM
Server
Snd Rcv
Snd Rcv
Protocol Version Exchange
0 0
Algorithm Negotiation
0 0
USERAUTHREQUEST: attacker:password
1 1
1 1
Key Exchange
2 2
NEWKEYS
2 3
3 3
EXTINFO
4 3
SERVICEREQUEST: ssh-userauth
3 4
5 3
SERVICEACCEPT: ssh-userauth
4 5
USERAUTHSUCCESS
4 5
5 5
USERAUTHREQUEST: victim:secret
5 5

Terrapin Attack

Encryption Modes

SSH Adopted Various Authenticated Encryption Modes Over The Years



1997

CBC-EaM
(Encrypt-and-MAC)

2003

CTR-EaM

2007

GCM

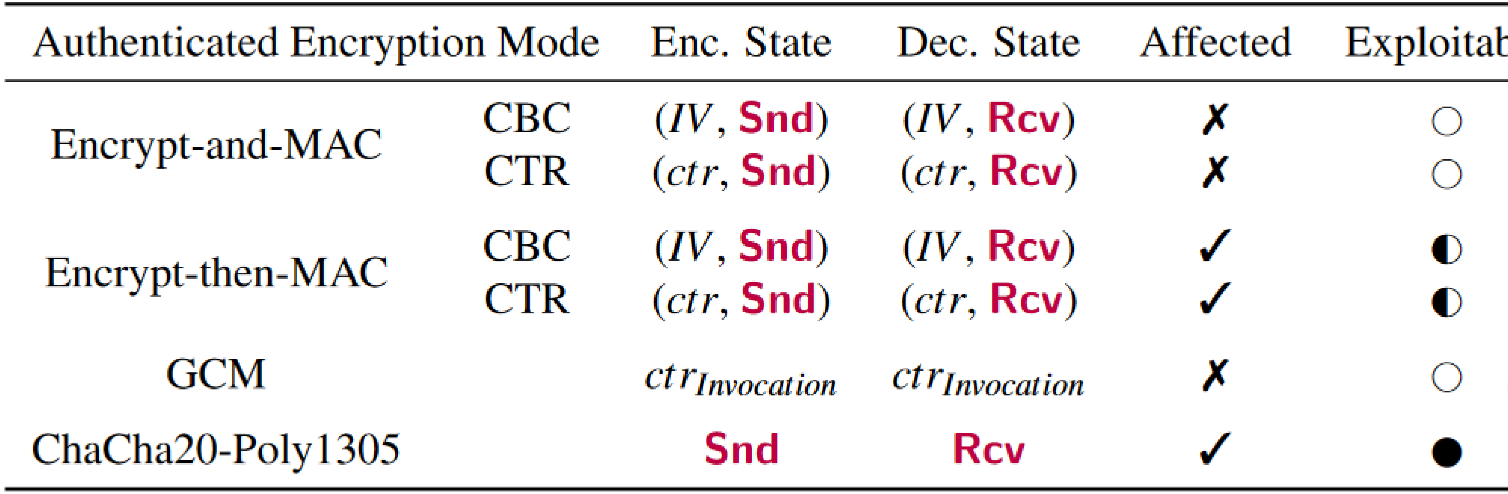
2012

CBC-EtM
CTR-EtM
(Encrypt-then-MAC)

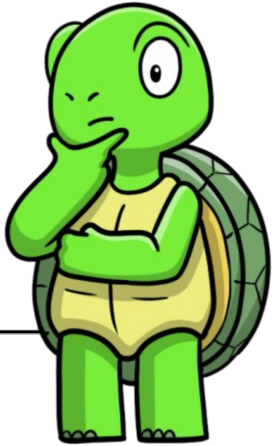
2013

ChaCha20-Poly1305

A Successful Terrapin Attack Depends on Authenticated Encryption Mode

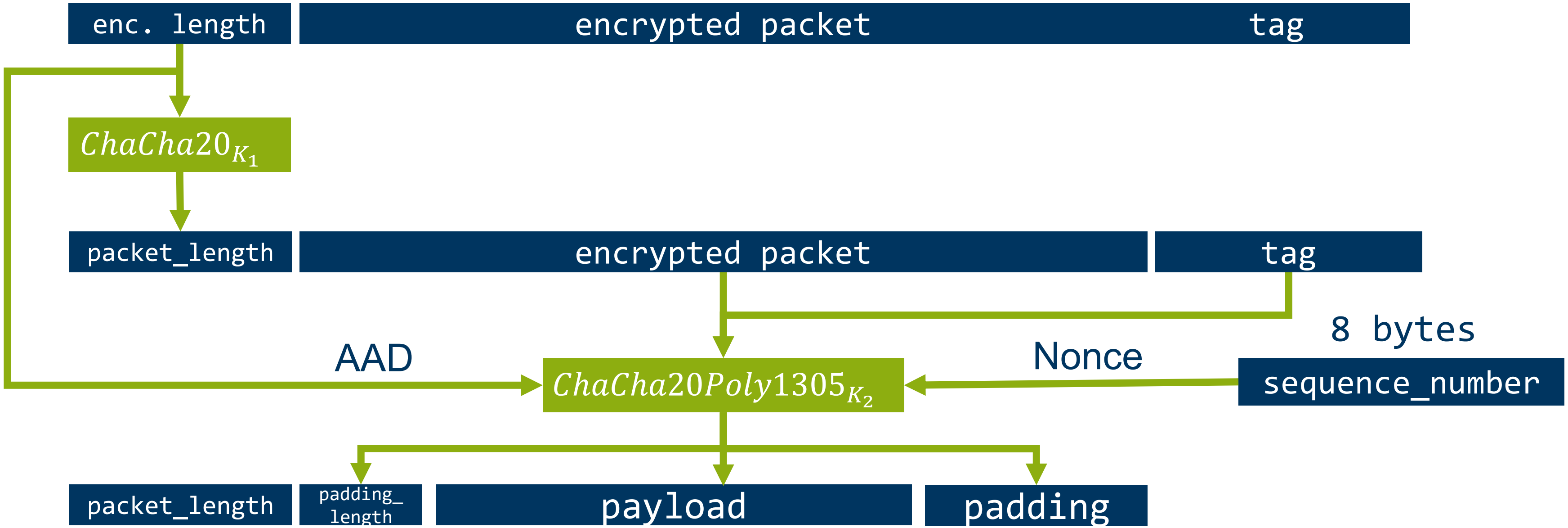


Authenticated Encryption Mode		Enc. State	Dec. State	Affected	Exploitable
Encrypt-and-MAC	CBC	(IV, Snd)	(IV, Rcv)	✗	○
	CTR	(ctr, Snd)	(ctr, Rcv)	✗	○
Encrypt-then-MAC	CBC	(IV, Snd)	(IV, Rcv)	✓	◐
	CTR	(ctr, Snd)	(ctr, Rcv)	✓	◐
GCM		$ctr_{Invocation}$	$ctr_{Invocation}$	✗	○
ChaCha20-Poly1305		Snd	Rcv	✓	●

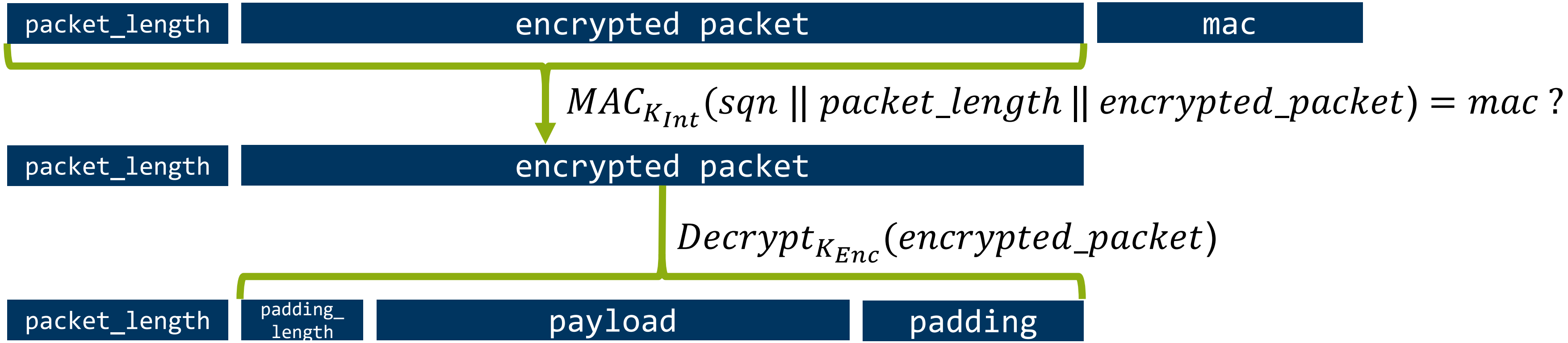


ChaCha20-Poly1305 Allows Perfect Prefix Truncation

$K_{Enc} = K_2 || K_1$

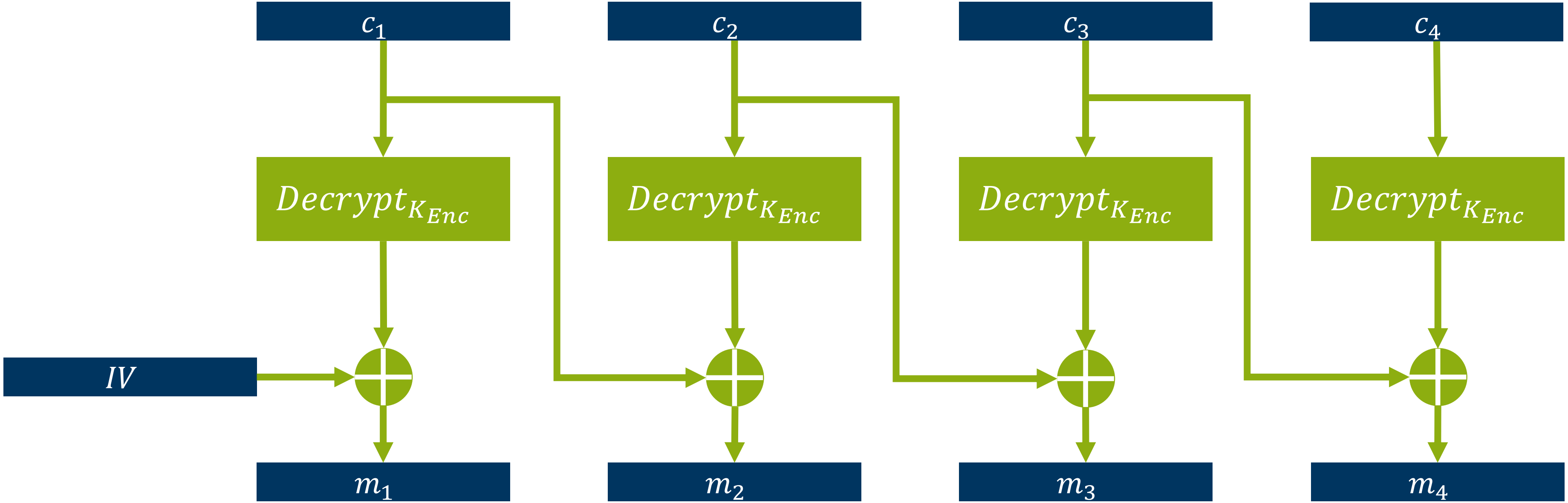


CBC-EtM Allows Probabilistic Truncation Attacks

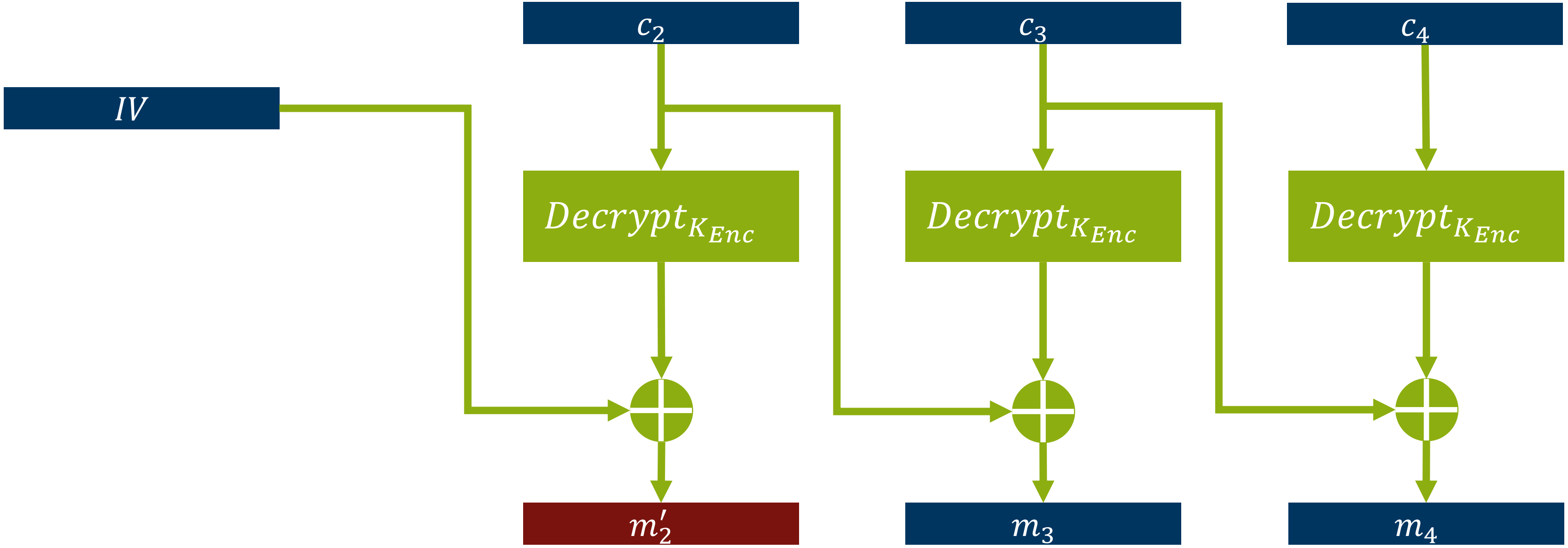

packet_length
encrypted packet
mac
$MAC_{K_{Int}}(sqn \mid\mid packet\_length \mid\mid encrypted\_packet) = mac\,?$
packet_length
encrypted packet
$Decrypt_{K_{Enc}}(encrypted\_packet)$
packet_length
padding_ length
payload
padding

- **Observation:** Truncation of first message causes first block of second message to become pseudorandom
- MAC protects integrity of ciphertext allowing MAC verification to succeed

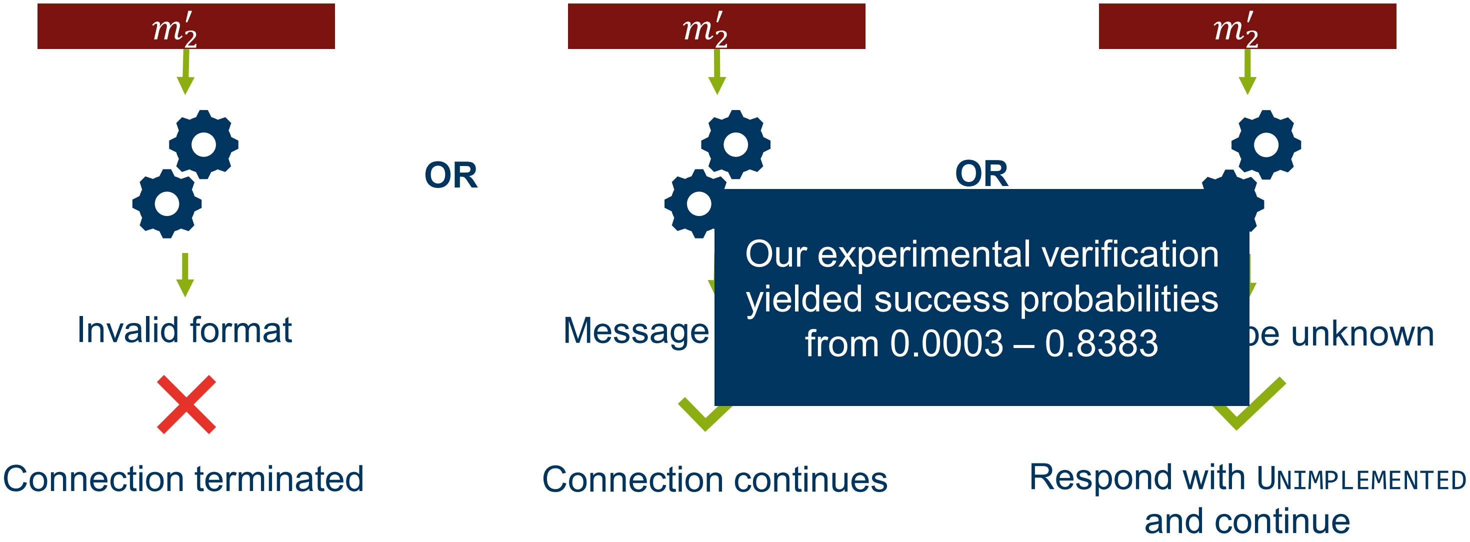
Truncation in CBC Encryption Modes Causes One Pseudorandom Block



Truncation in CBC Encryption Modes Causes One Pseudorandom Block



The Attack’s Success Depends on How Peers Handle The Corrupt Message Block



m'_2

Invalid format

Connection terminated

OR

m'_2

Message

Connection continues

OR

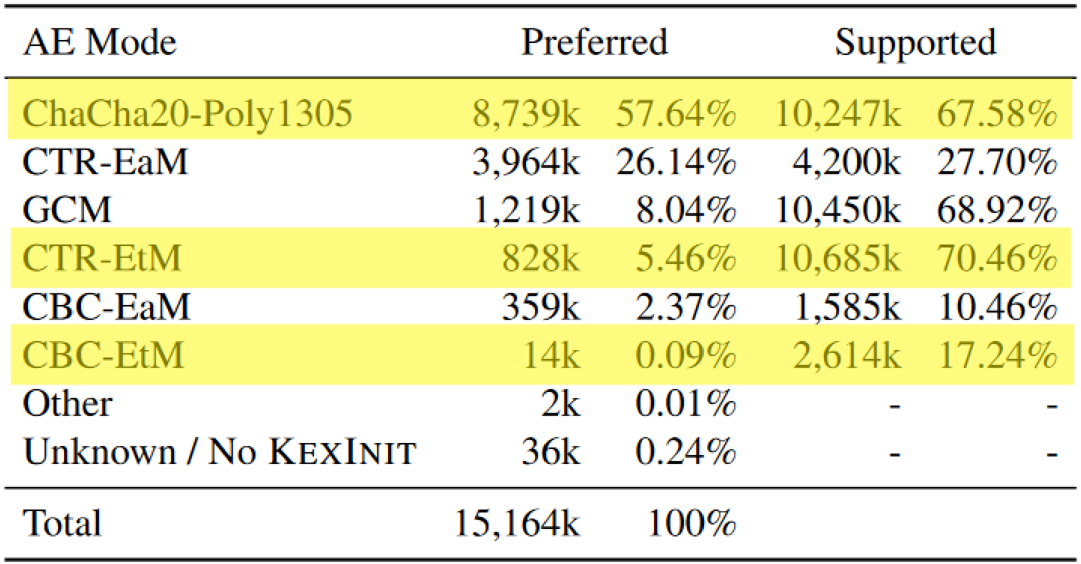
m'_2

pe unknown

Respond with UNIMPLEMENTED and continue

Our experimental verification yielded success probabilities from 0.0003 – 0.8383

ChaCha20-Poly1305 And EtM Are Popular



AE Mode	Preferred		Supported	
ChaCha20-Poly1305	8,739k	57.64%	10,247k	67.58%
CTR-EaM	3,964k	26.14%	4,200k	27.70%
GCM	1,219k	8.04%	10,450k	68.92%
CTR-EtM	828k	5.46%	10,685k	70.46%
CBC-EaM	359k	2.37%	1,585k	10.46%
CBC-EtM	14k	0.09%	2,614k	17.24%
Other	2k	0.01%	-	-
Unknown / No KEXINIT	36k	0.24%	-	-
Total	15,164k	100%		

Terrapin Attack

Countermeasures & Conclusion

Mitigating Our Attack Is Difficult

Countermeasure	Our Suggestion	"Strict KEX" (OpenSSH)
Reset sequence numbers at key installation	✓	✓
Authenticate the entire handshake transcript (hash)	✓	
Harden handshake to disallow unexpected messages		✓

- **> 30 unique implementations support "strict kex"**
- **~ 11 million servers offer "strict kex"**

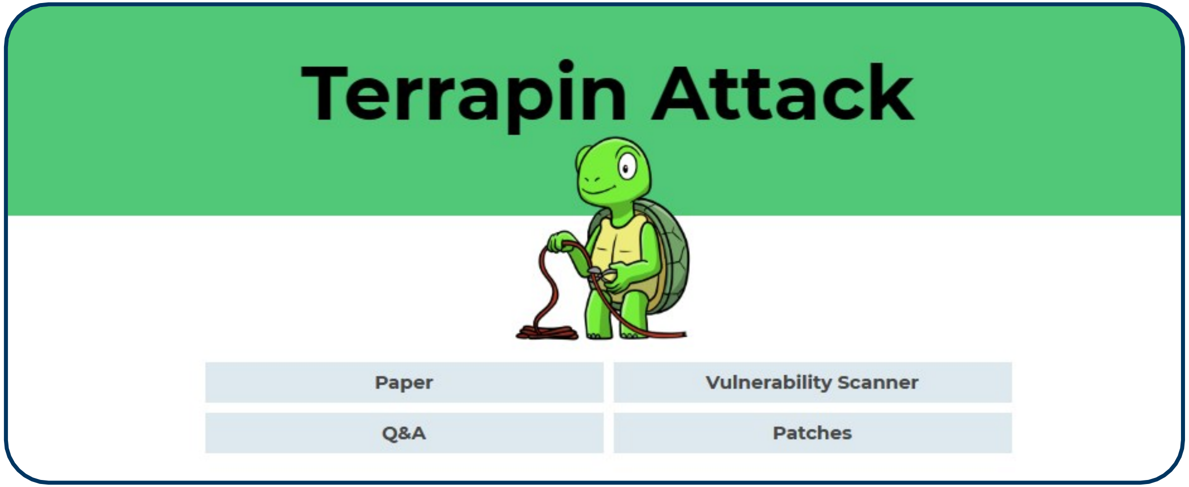
Lessons Learned

- **Terrapin is a novel cryptographic attack targeting SSH channel integrity**
  - Can be exploited in practice to downgrade the connection‘s security
  - May lead to more severe vulnerabilities if combined with state machine flaws

- **Widespread encryption modes are affected**
  - ChaCha20-Poly1305 (67.58%)
  - CBC-EtM (17.24%)
  - CTR-EtM (70.46%)

- **“Strict Kex” as a protocol-level countermeasure**
  - Requires support from client and server to take effect

Thanks! Questions?



https://terrapin-attack.com/

E-Mail:	fabian.baeumer@rub.de
Bluesky:	@skrillor.bsky.social
Mastodon:	@Skrillor@infosec.exchange