black hat®
USA 2024

AUGUST 7-8, 2024
BRIEFINGS

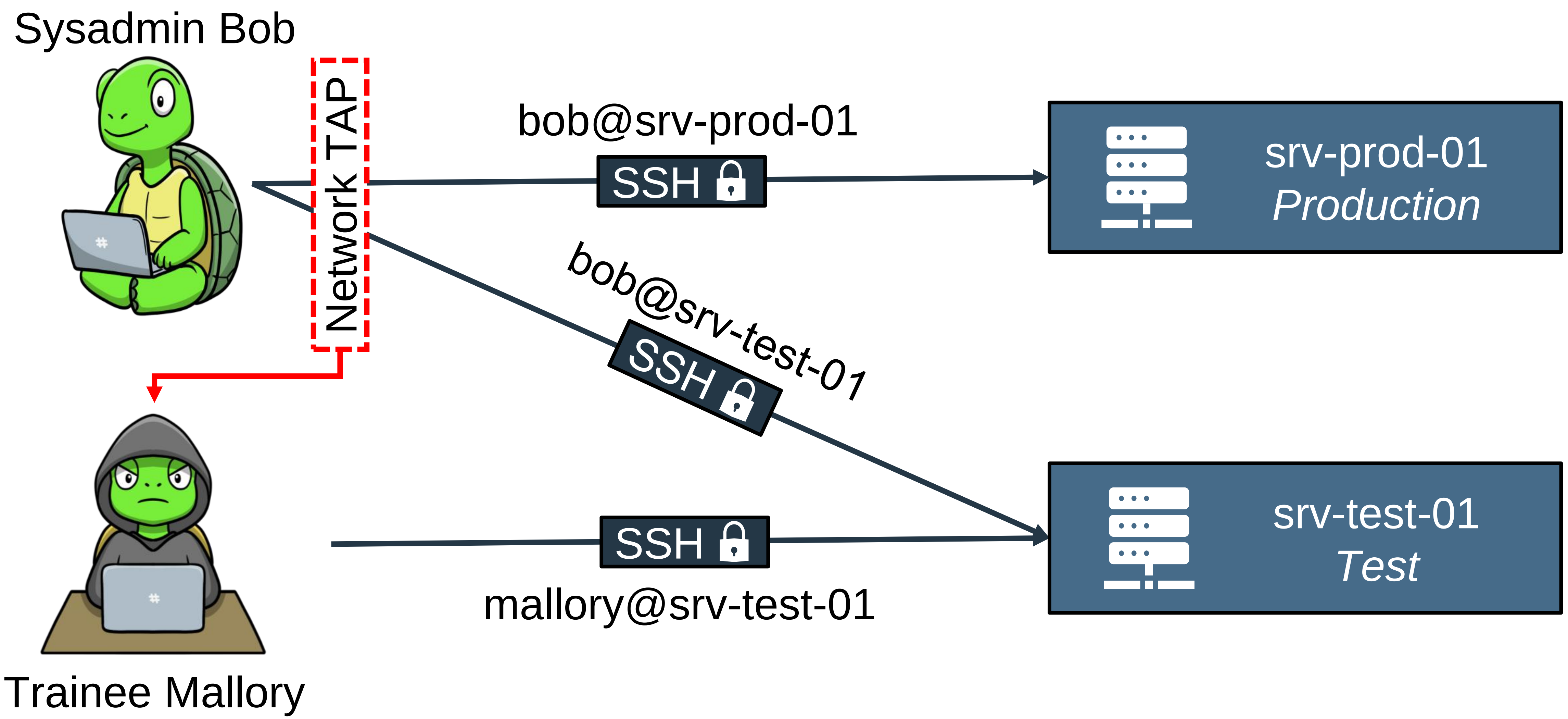
Terrapin Attack: Breaking SSH Channel Integrity by Sequence Number Manipulation

Fabian Bäumer
Ruhr University Bochum

Marcus Brinkmann
Ruhr University Bochum

Jörg Schwenk
Ruhr University Bochum

A Tale Of System Administration



Demo

- A ‘Normal’ Workday For Bob

In The Next 30 Minutes You Will Learn…

- … how Mallory was able to mess with Bob’s user authentication
- … which other attack variants Mallory can perform
- … the specific requirements for Mallory’s attack to work
- … how Bob can protect himself against Mallory’s attack

Beyond that,

- … how adding modern cryptography to older protocols can go wrong
- … how we handled a protocol-level responsible disclosure

Understanding SSH Is Key to Understanding Mallory’s Attack

SSH Connection Protocol (RFC 4254)

SSH Authentication Protocol (RFC 4252)

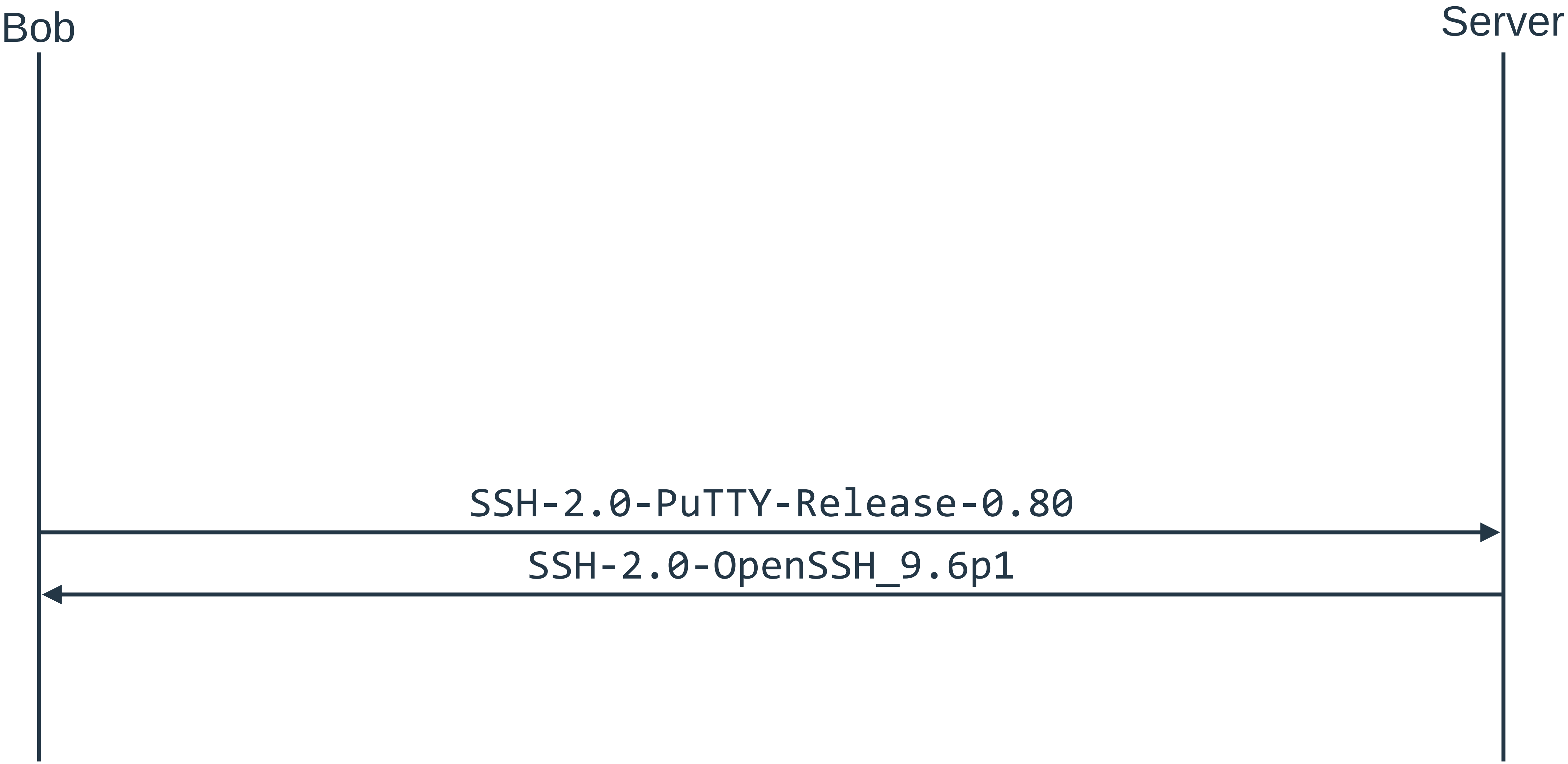
SSH Transport Layer Protocol (TLP) (RFC 4253)

=> Binary Packet Protocol

=> SSH Key Exchange

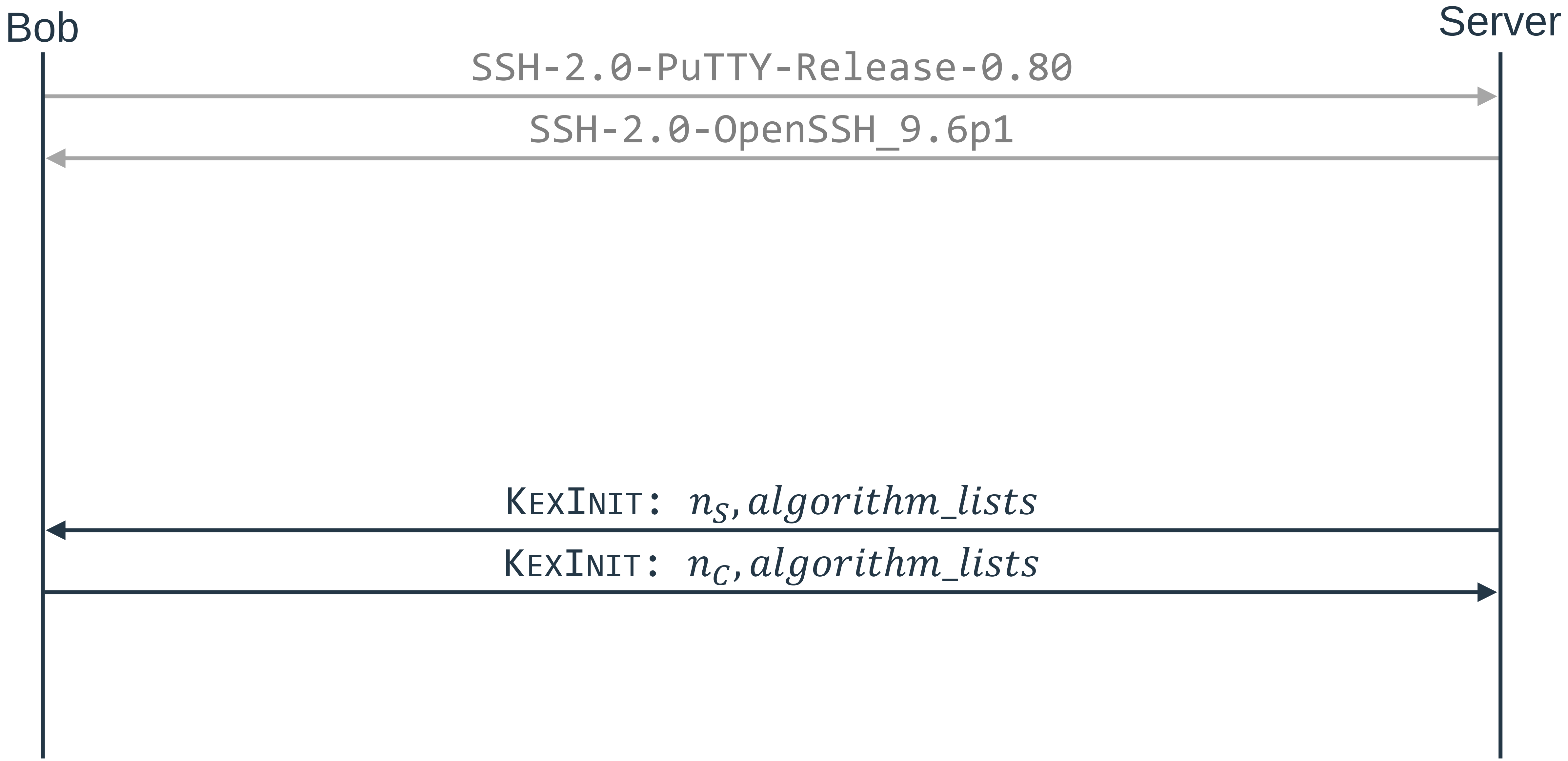
TCP / IP

Step 1: Exchange of Protocol Version



```
SSH-2.0-PuTTY-Release-0.80
SSH-2.0-OpenSSH_9.6p1
```

Step 2: Exchange of Supported Algorithms



Bob

Server

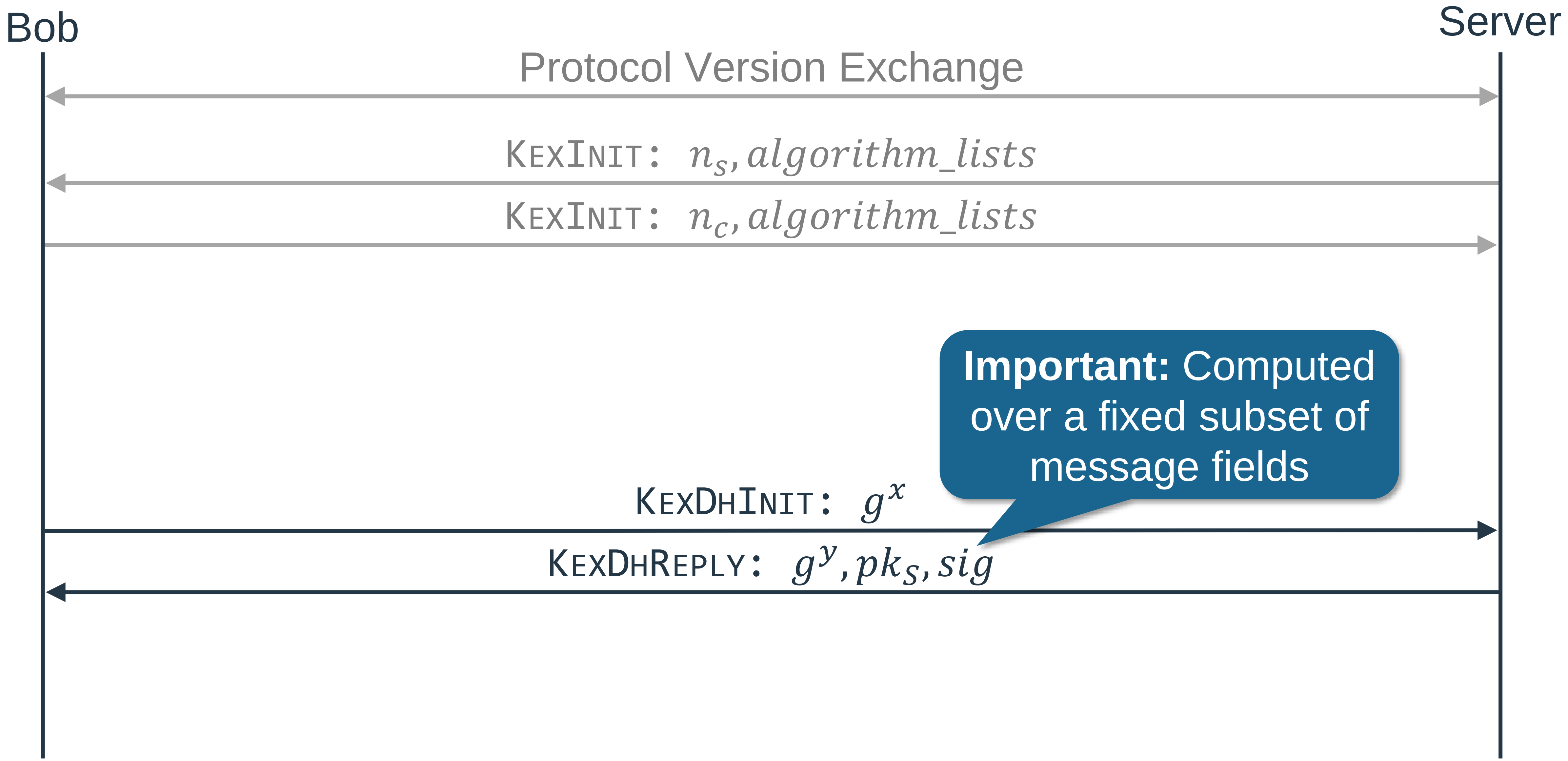
SSH-2.0-PuTTY-Release-0.80

SSH-2.0-OpenSSH_9.6p1

KEXINIT: $n_S, algorithm\_lists$

KEXINIT: $n_C, algorithm\_lists$

Step 3: Performing Key Exchange



Bob

Server

Protocol Version Exchange

KEXINIT: $n_S, algorithm\_lists$

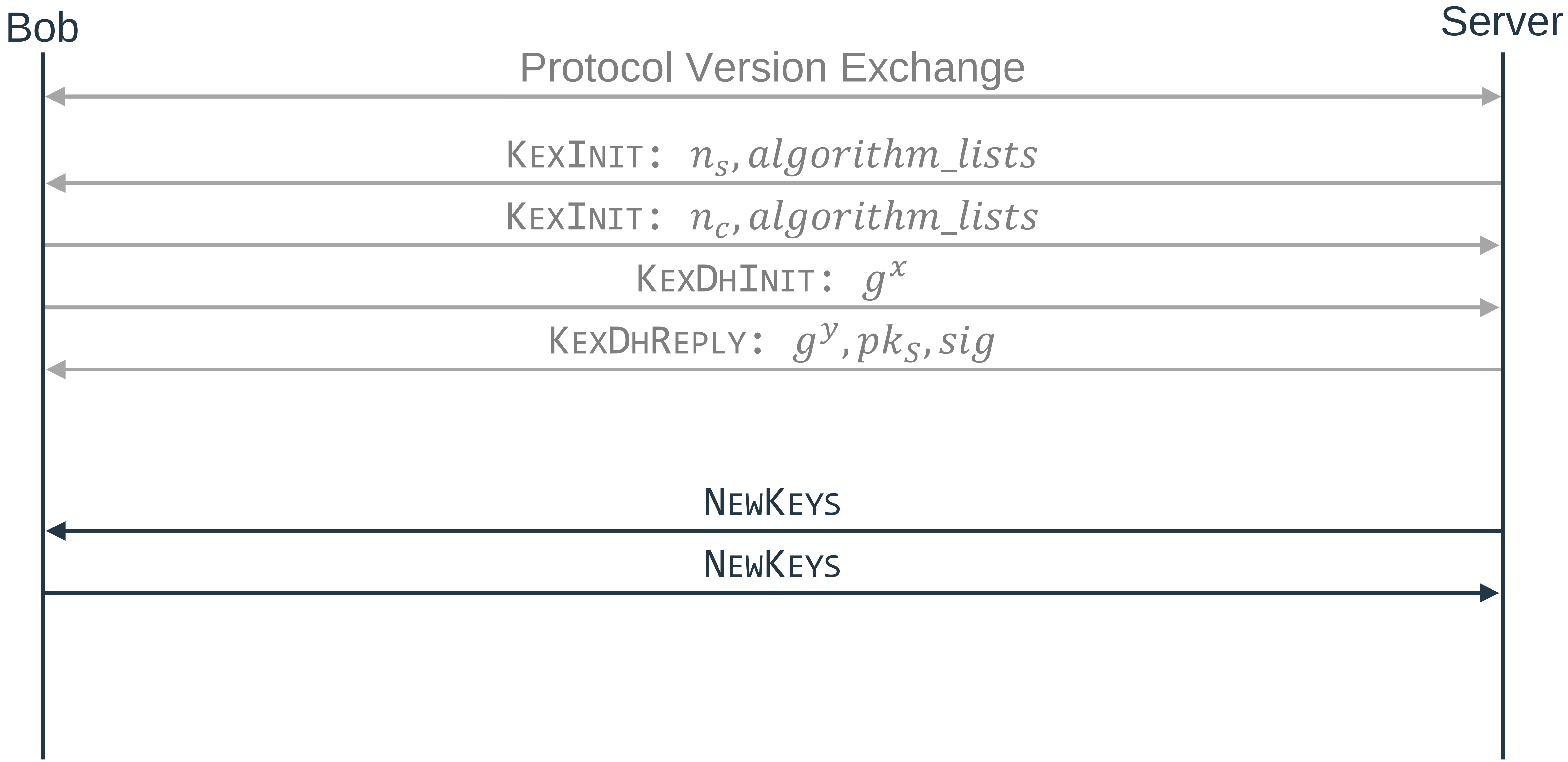
KEXINIT: $n_c, algorithm\_lists$

KEXDHINIT: g^x

KEXDHREPLY: g^y, pk_S, sig

Important: Computed over a fixed subset of message fields

Step 4: Activating the Secure Channel



Bob

Server

Protocol Version Exchange

KEXINIT: $n_S, algorithm\_lists$

KEXINIT: $n_C, algorithm\_lists$

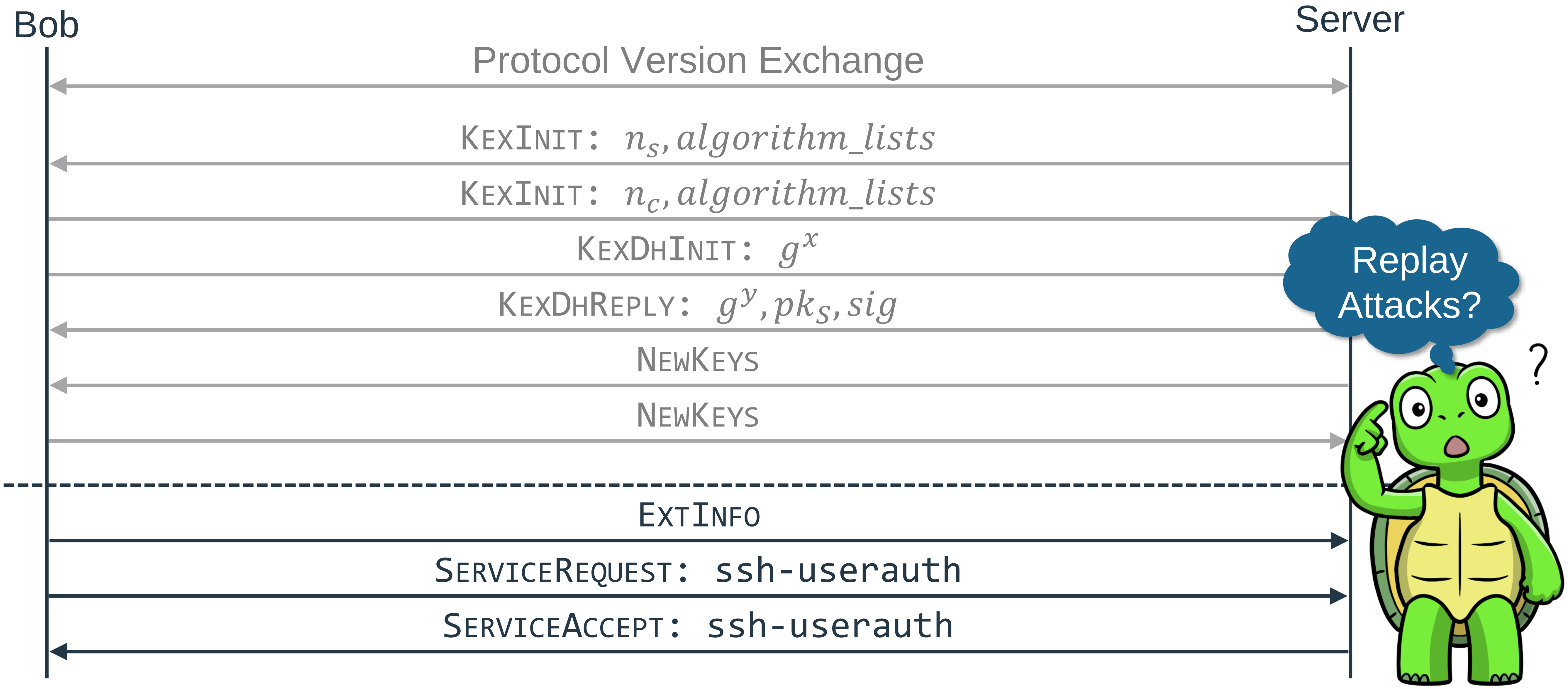
KEXDHINIT: g^x

KEXDHREPLY: g^y, pk_S, sig

NEWKEYS

NEWKEYS

Step 5: Request User Authentication Service



Bob

Server

Protocol Version Exchange

KexInit: $n_S, algorithm\_lists$

KexInit: $n_C, algorithm\_lists$

KexDhInit: g^x

KexDhReply: g^y, pk_S, sig

NewKeys

NewKeys

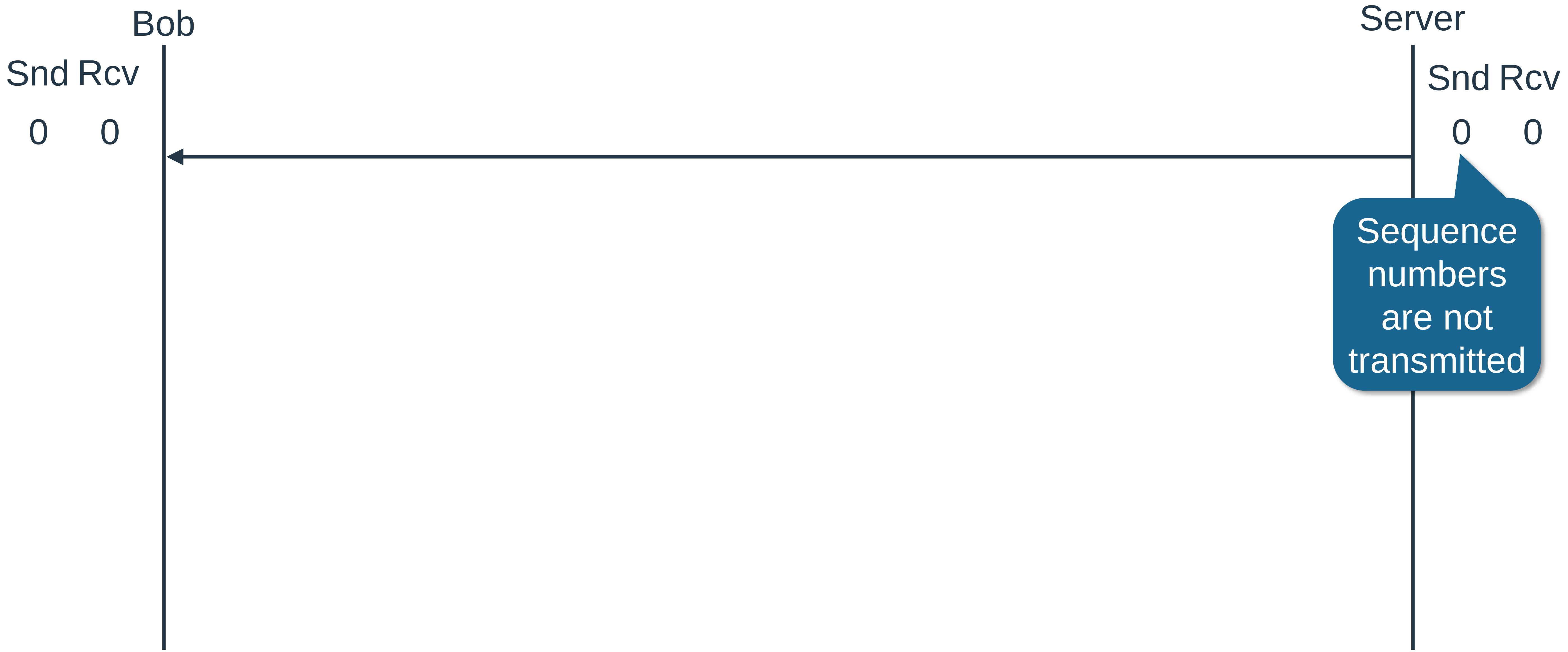
ExtInfo

ServiceRequest: `ssh-userauth`

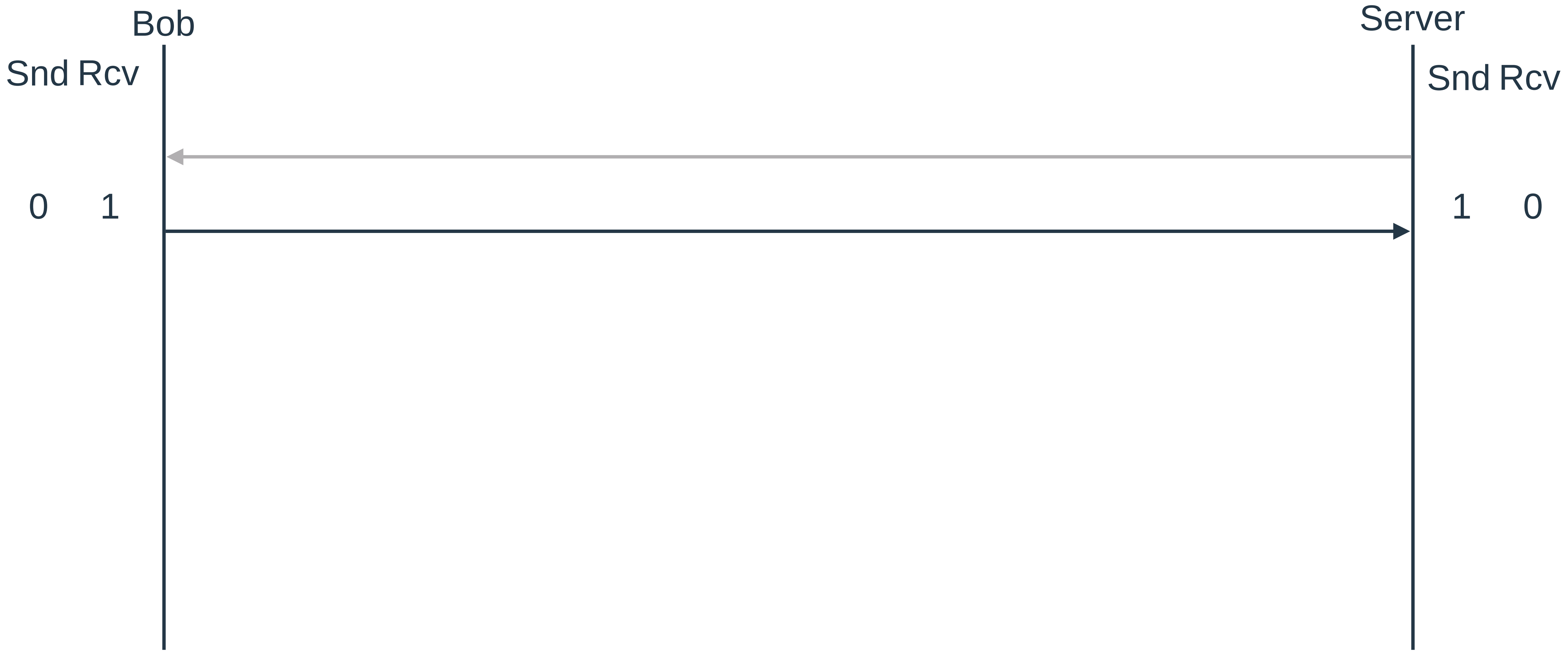
ServiceAccept: `ssh-userauth`

Replay Attacks?

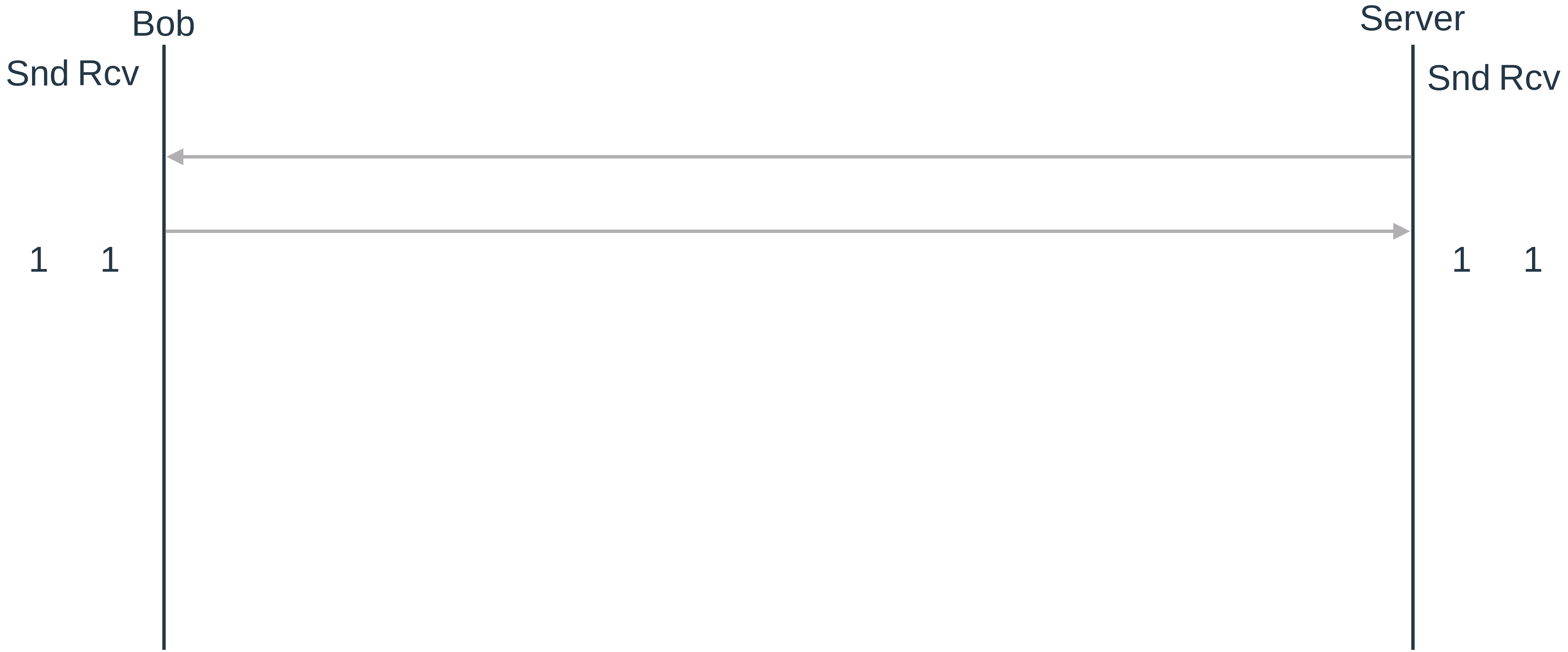
SSH Uses Implicit Sequence Numbers



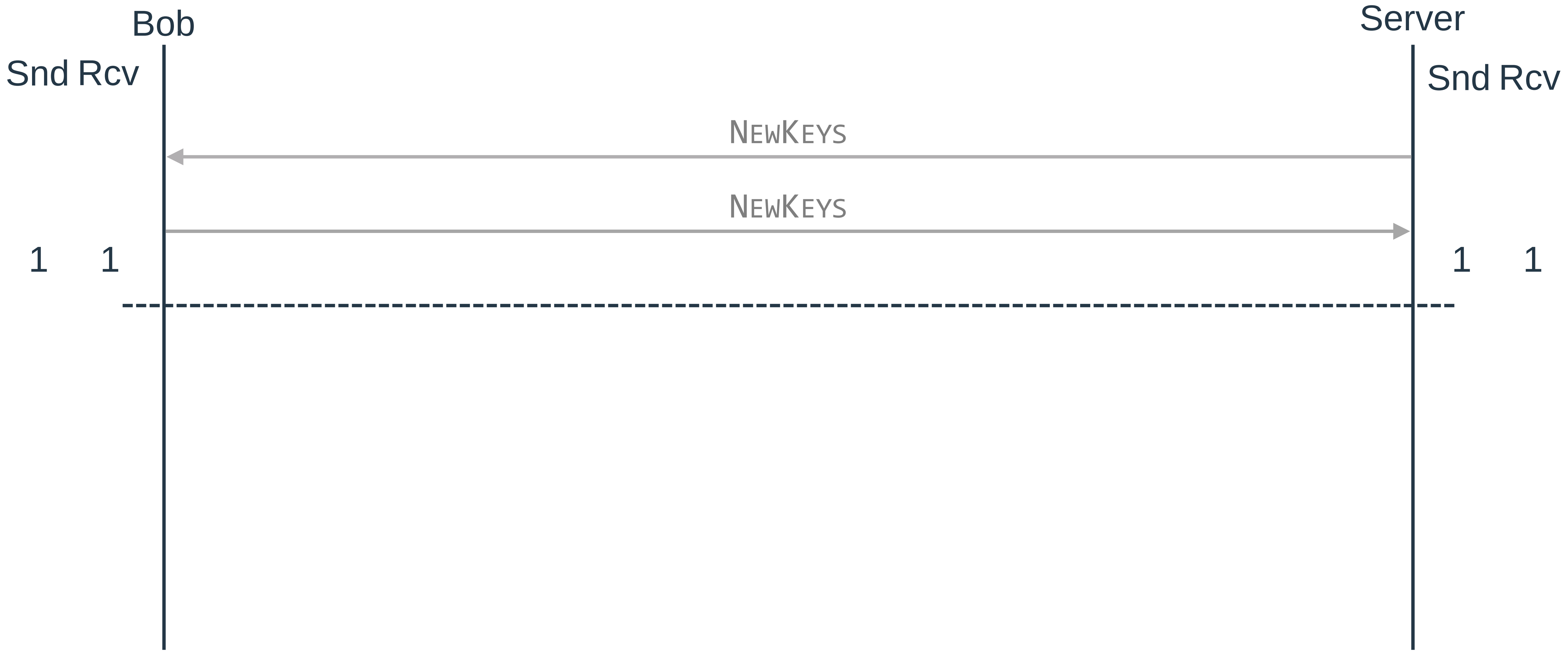
SSH Uses Implicit Sequence Numbers



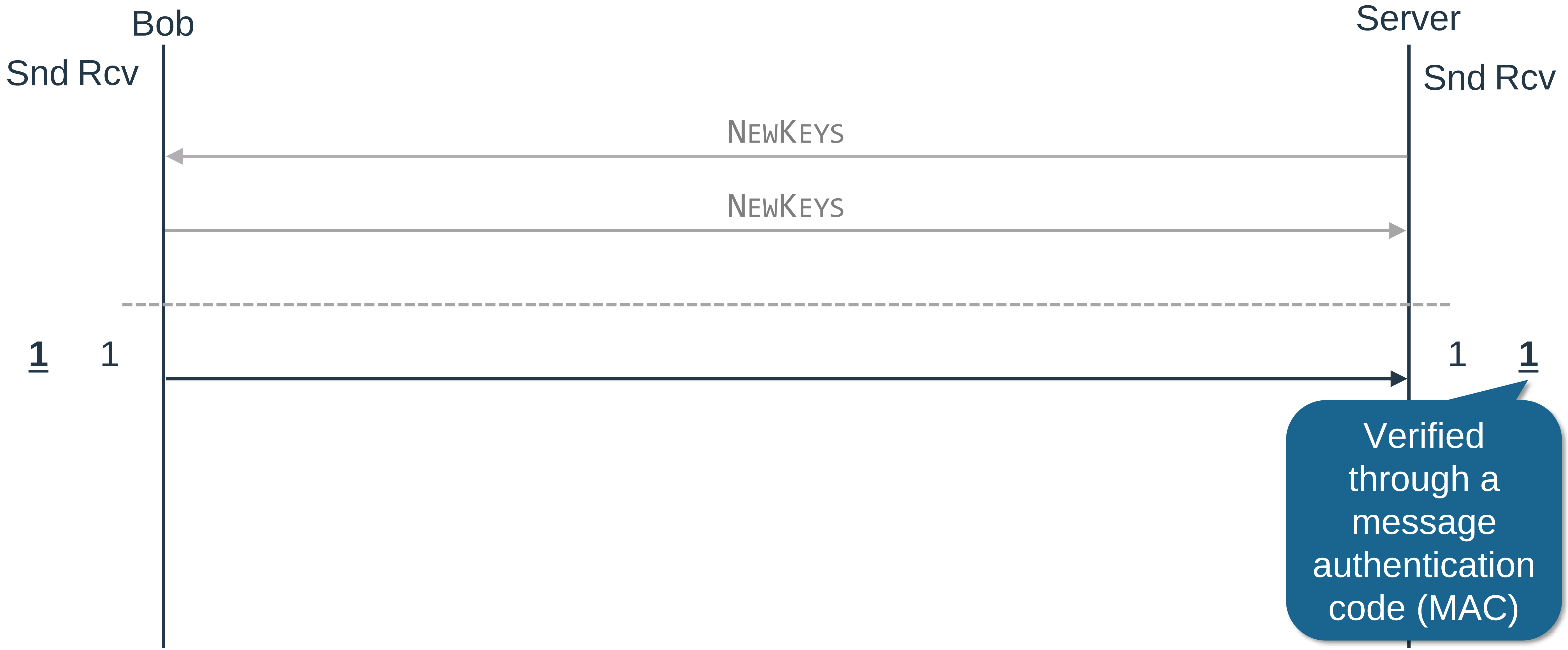
SSH Uses Implicit Sequence Numbers



SSH Uses Implicit Sequence Numbers

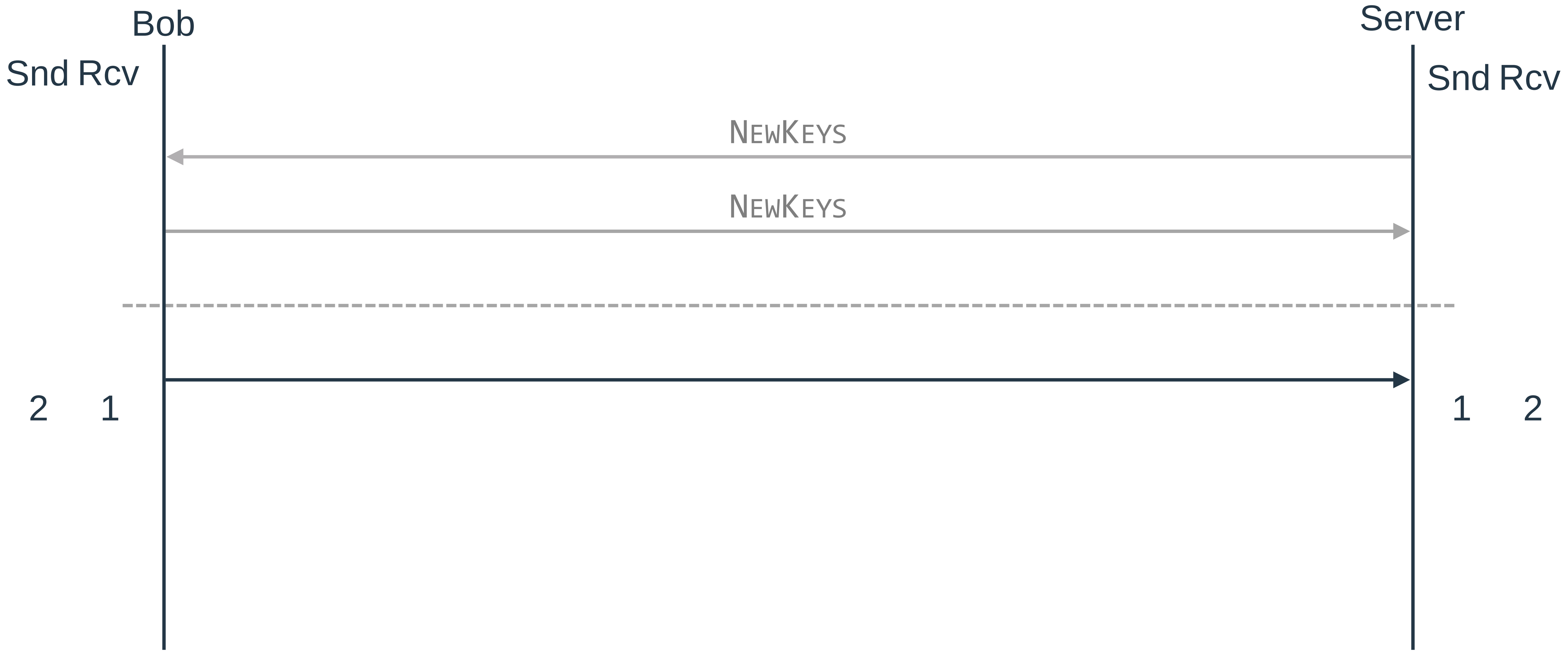


SSH Uses Implicit Sequence Numbers

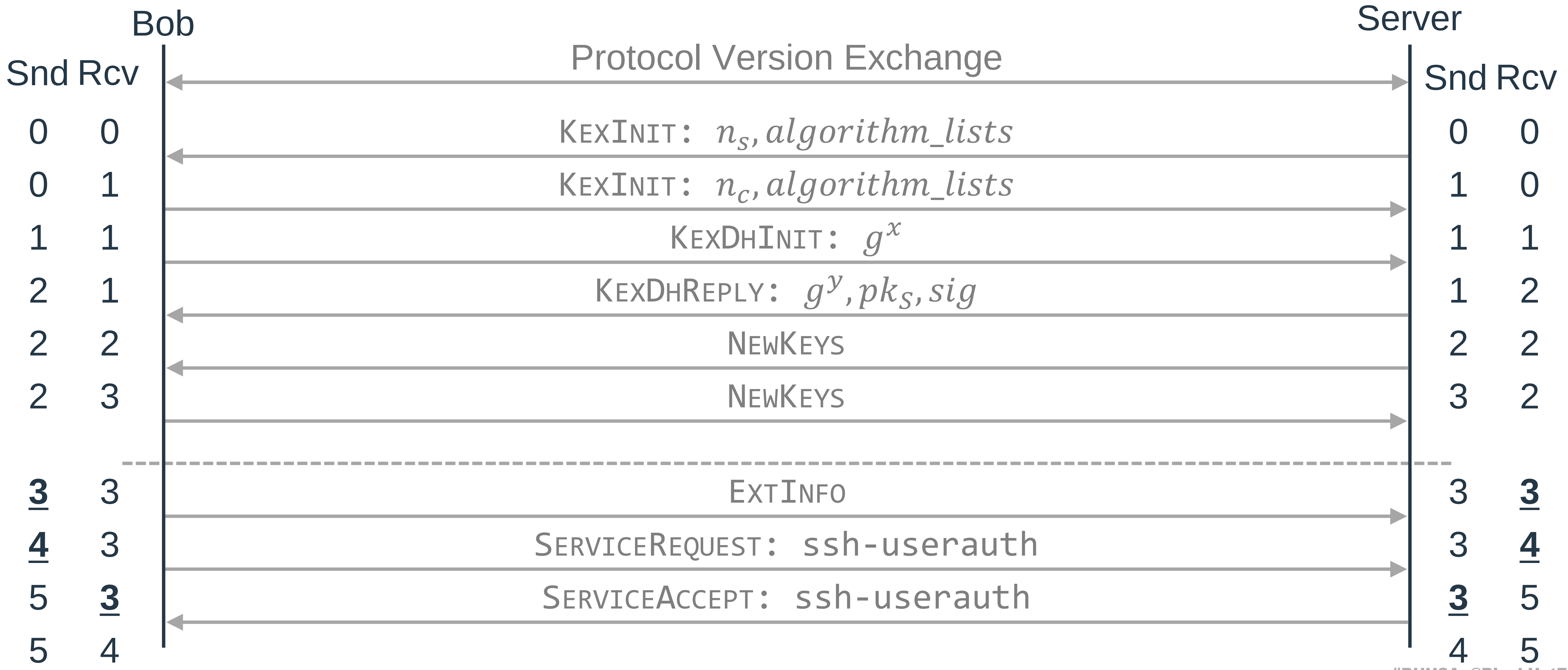


Verified through a message authentication code (MAC)

SSH Uses Implicit Sequence Numbers

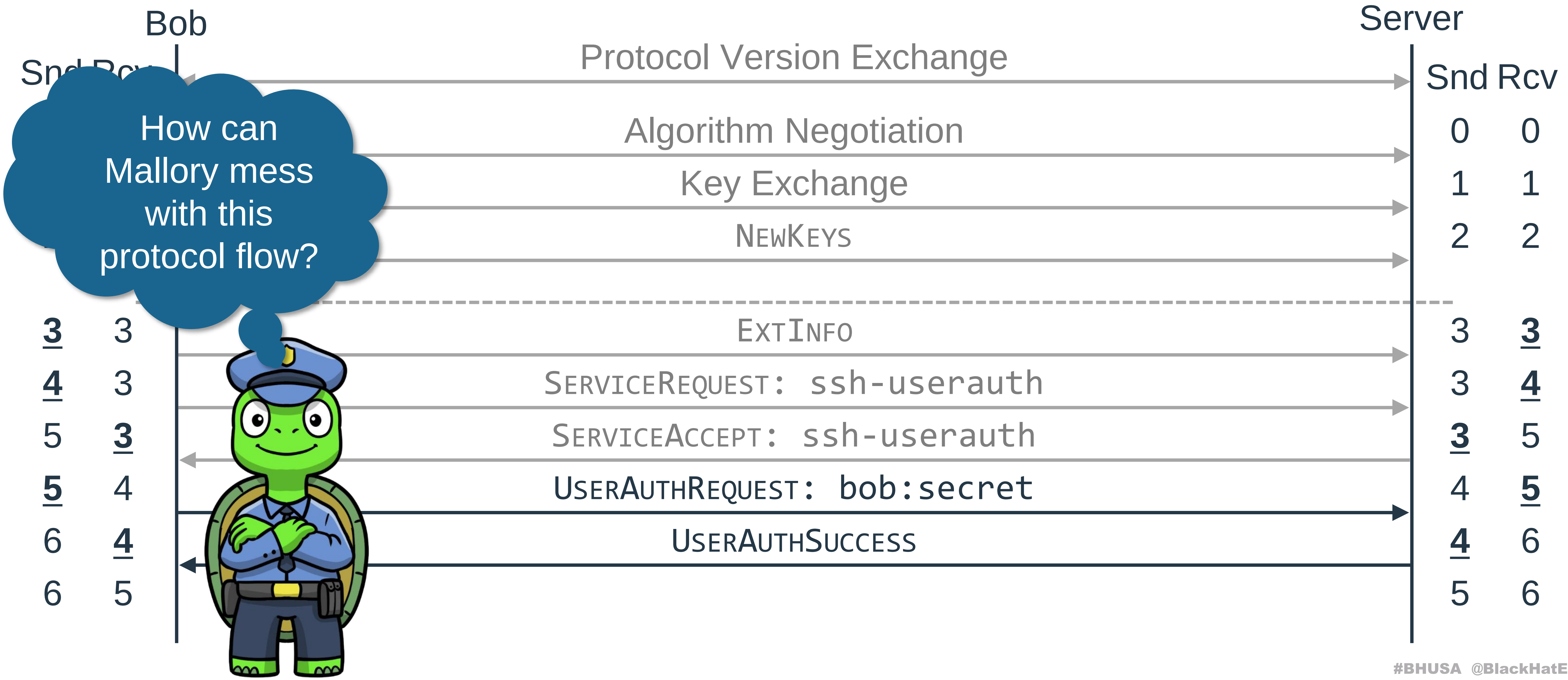


Introducing Sequence Numbers to the Flow



Bob Snd	Bob Rcv	Bob		Server	Server Snd	Server Rcv
		↔	Protocol Version Exchange	↔		
0	0	←	KexInit: $n_S, algorithm\_lists$	←	0	0
0	1	→	KexInit: $n_c, algorithm\_lists$	→	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**
4	3	→	ServiceRequest: ssh-userauth	→	3	**4**
5	**3**	←	ServiceAccept: ssh-userauth	←	**3**	5
5	4				4	5

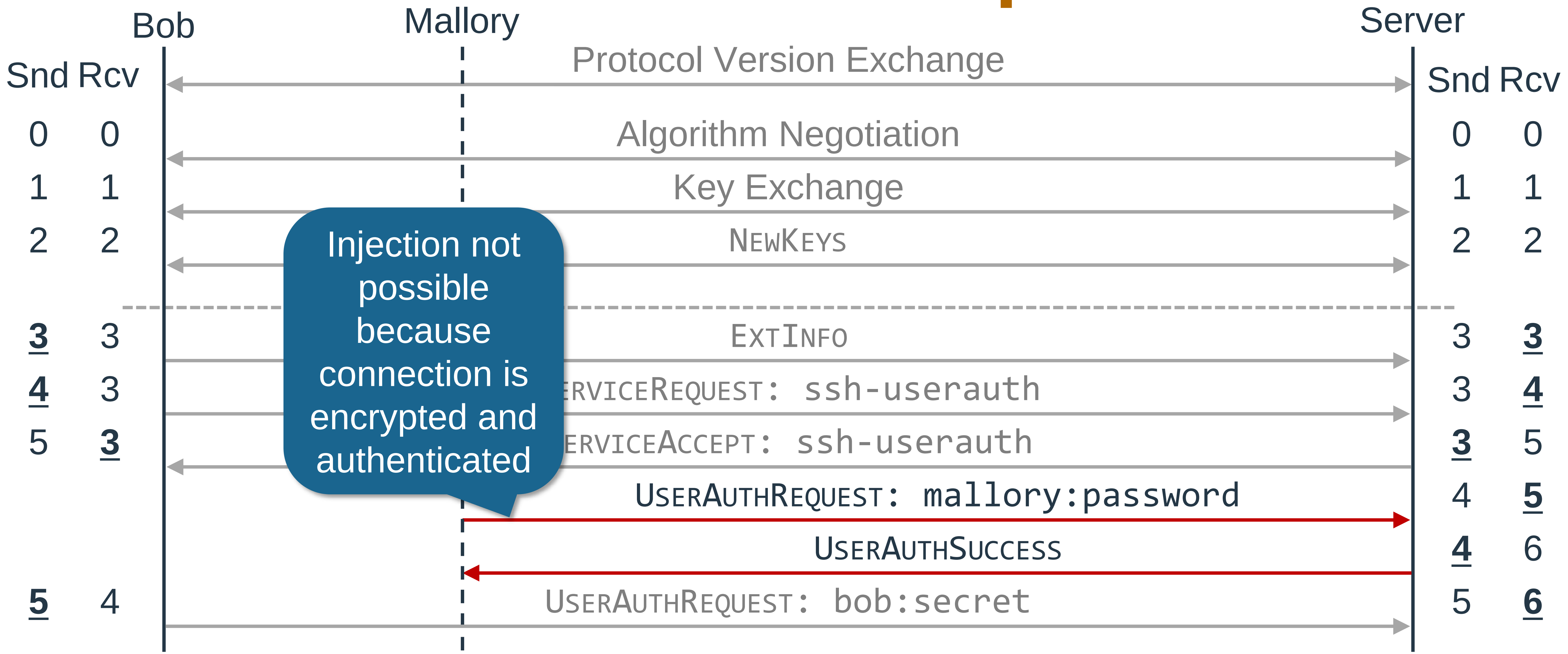
Step 6: Authenticating the User



Bob			Server	
Snd	Rcv		Snd	Rcv
		Protocol Version Exchange		
		Algorithm Negotiation	0	0
		Key Exchange	1	1
		NewKeys	2	2
3	3	ExtInfo	3	**3**
4	3	ServiceRequest: ssh-userauth	3	**4**
5	**3**	ServiceAccept: ssh-userauth	**3**	5
5	4	UserAuthRequest: bob:secret	4	**5**
6	**4**	UserAuthSuccess	**4**	6
6	5		5	6

How can Mallory mess with this protocol flow?

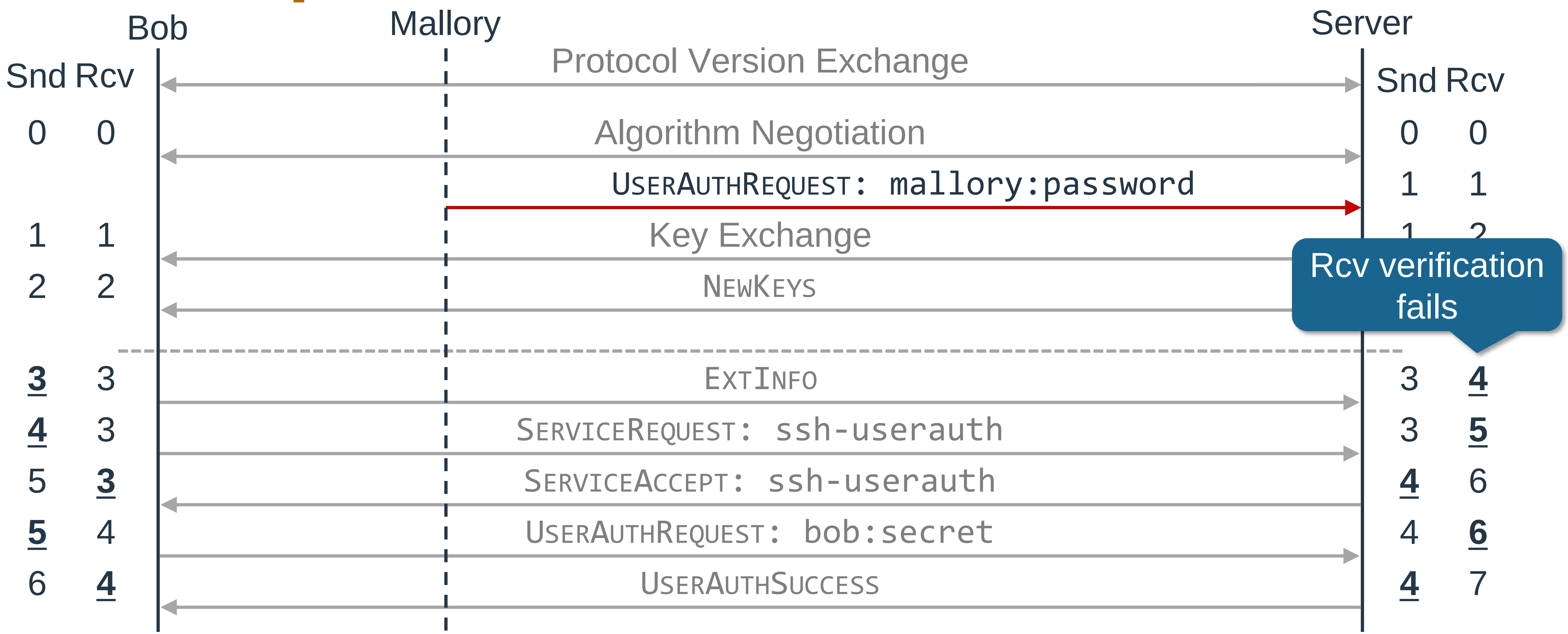
Mallory‘s Ultimate Goal: Inject Forged Authentication Request



Bob Snd	Bob Rcv		Server Snd	Server Rcv
		Protocol Version Exchange		
0	0	Algorithm Negotiation	0	0
1	1	Key Exchange	1	1
2	2	NEWKEYS	2	2
3	3	EXTINFO	3	**3**
4	3	ERVICEREQUEST: ssh-userauth	3	**4**
5	**3**	ERVICEACCEPT: ssh-userauth	**3**	5
		USERAUTHREQUEST: mallory:password	4	**5**
		USERAUTHSUCCESS	**4**	6
5	4	USERAUTHREQUEST: bob:secret	5	**6**

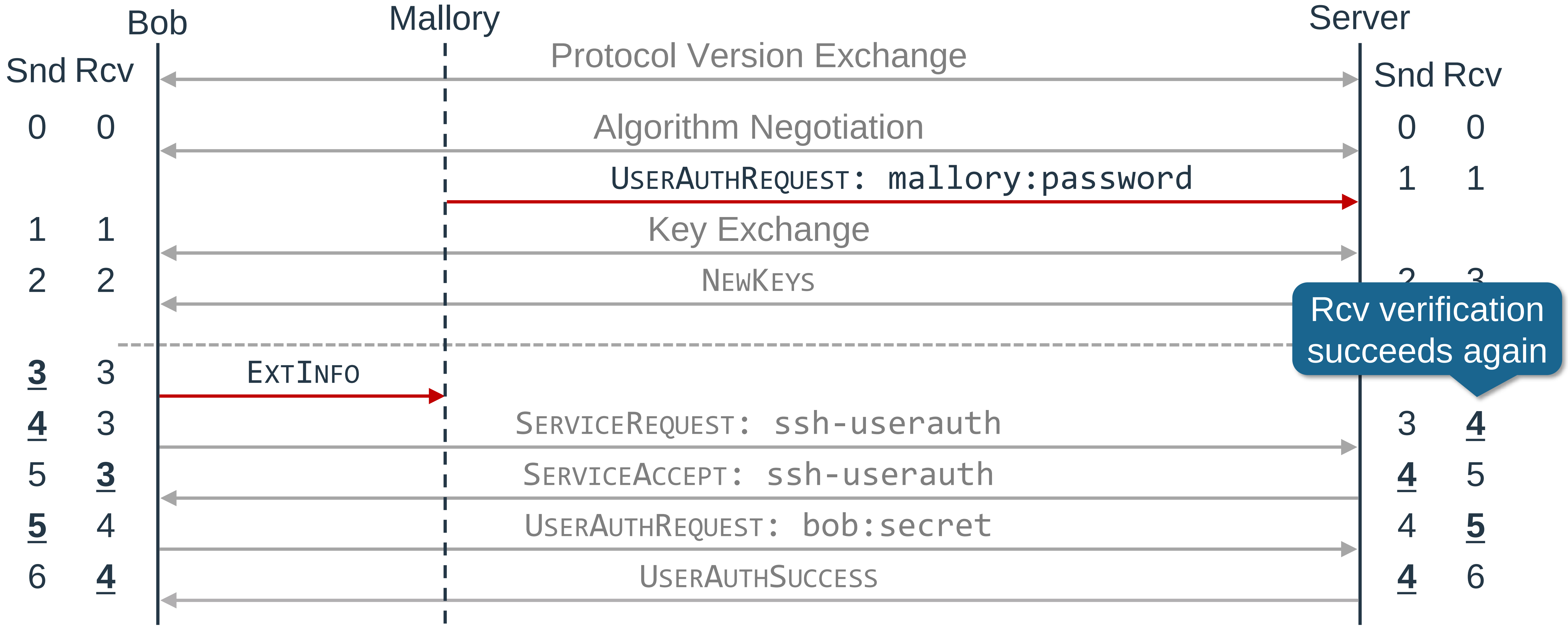
Injection not possible because connection is encrypted and authenticated

Mallory Tries To Move The Authentication Request Into Unauthenticated Context…



Bob Snd	Bob Rcv		Server Snd	Server Rcv
		Protocol Version Exchange		
0	0	Algorithm Negotiation	0	0
		USERAUTHREQUEST: mallory:password	1	1
1	1	Key Exchange	1	2
2	2	NEWKEYS		
3	3	EXTINFO	3	**4**
4	3	SERVICEREQUEST: ssh-userauth	3	**5**
5	**3**	SERVICEACCEPT: ssh-userauth	**4**	6
5	4	USERAUTHREQUEST: bob:secret	4	**6**
6	**4**	USERAUTHSUCCESS	**4**	7

… And Drops the First Authenticated Message to Realign Sequence Numbers

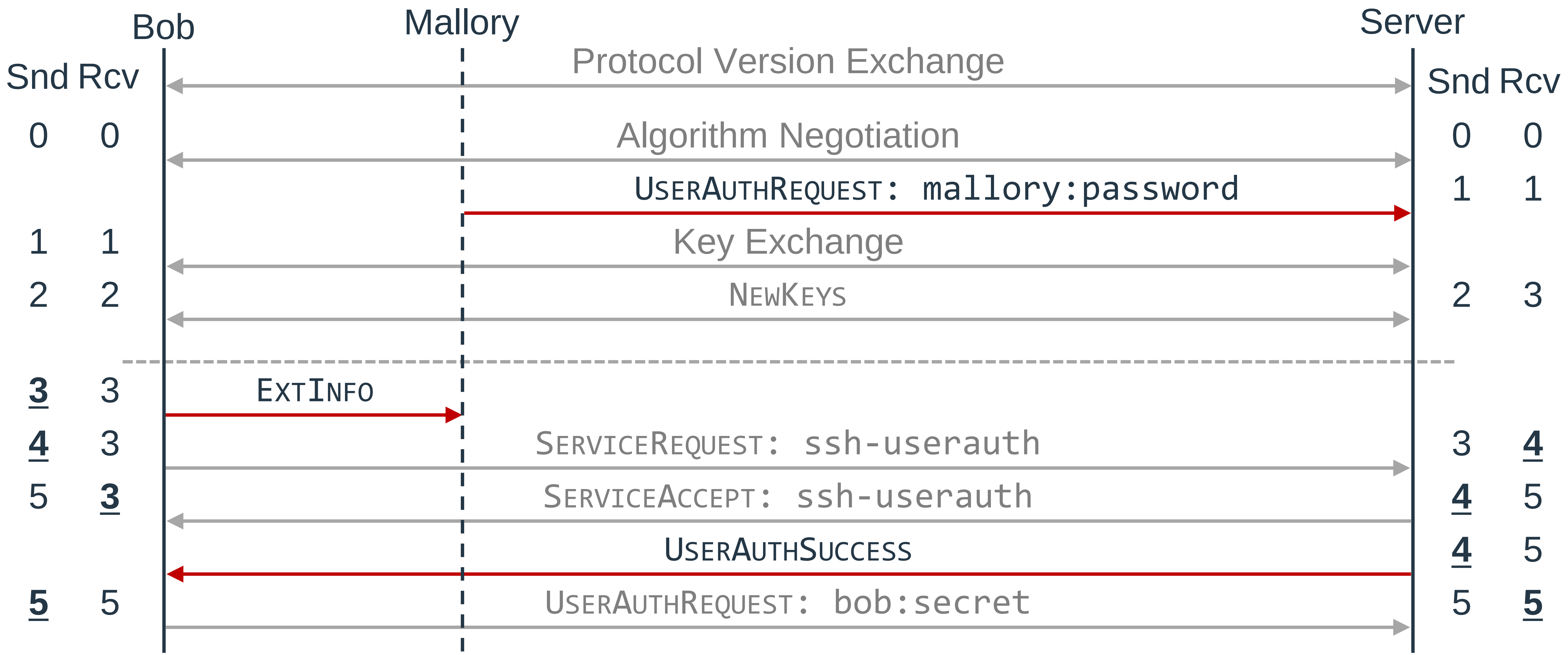


Bob | Mallory | Server

Snd	Rcv		Snd	Rcv
		Protocol Version Exchange		
0	0	Algorithm Negotiation	0	0
		USERAUTHREQUEST: mallory: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
5	4	USERAUTHREQUEST: bob:secret	4	**5**
6	**4**	USERAUTHSUCCESS	**4**	6

Rcv verification succeeds again

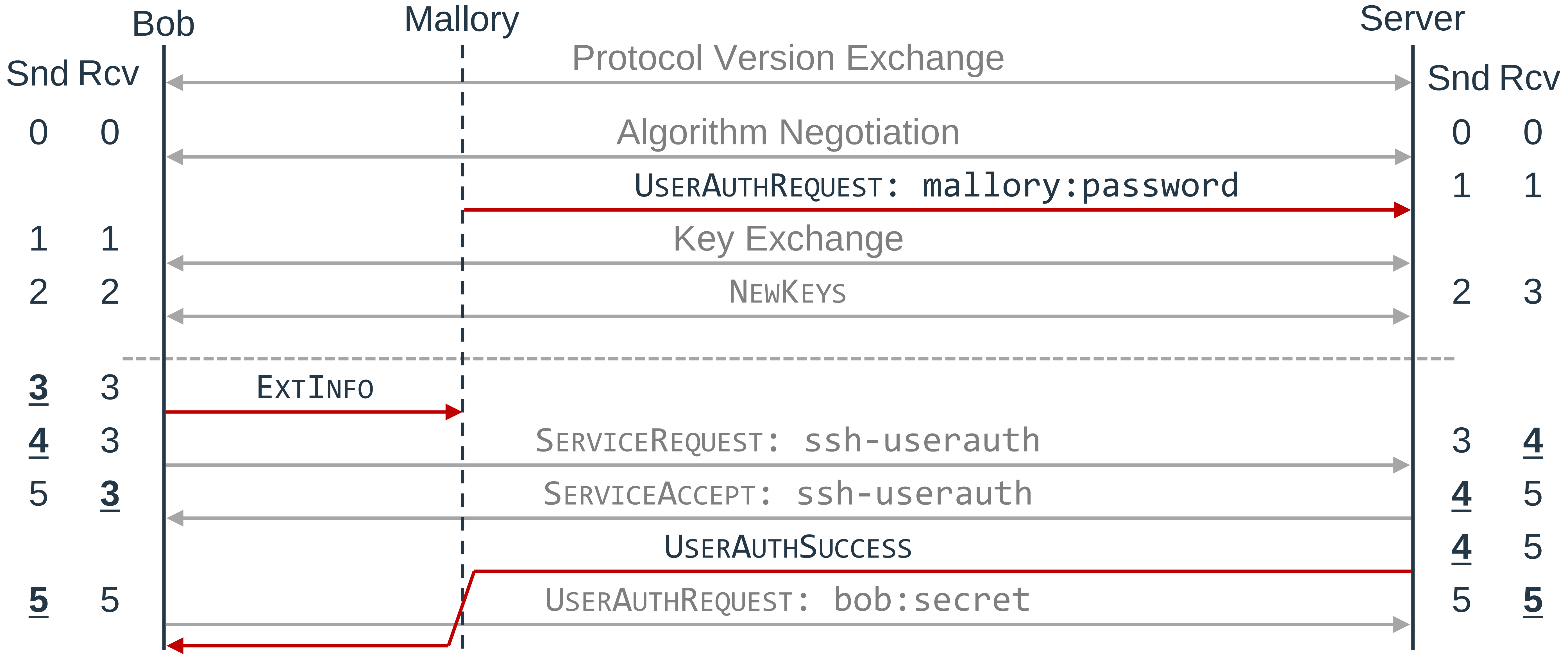
Authentication Succeeds Earlier Than Expected



Bob | Mallory | Server

Bob Snd	Bob Rcv		Server Snd	Server Rcv
		Protocol Version Exchange		
0	0	Algorithm Negotiation	0	0
		UserAuthRequest: mallory: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: bob:secret	5	**5**

Mallory’s Attack Can Succeed by Delaying Authentication Success



Bob | Mallory | Server

Snd Rcv | | Snd Rcv

Protocol Version Exchange

0 0 | Algorithm Negotiation | 0 0

USERAUTHREQUEST: mallory: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: bob:secret | 5 **5**

What Went Wrong Here?

Lax Server State Machine

Server accepted user authentication in unauthenticated context.

Implementation Flaw

Fixed Subset Host Key Signature

Signature fails to detect message injection during handshake.

Specification Flaw

Linked Sequence Numbers

Sqn numbers are maintained across different encryption contexts.

Specification Flaw

Let’s Talk About Attack Variants

Lax Server State Machine

Server accepted user authentication in unauthenticated context.

Implementation Flaw

Fixed Subset Host Key Signature

Signature fails to detect message injection during handshake.

Specification Flaw

Linked Sequence Numbers

Sqn numbers are maintained across different encryption contexts.

Specification Flaw

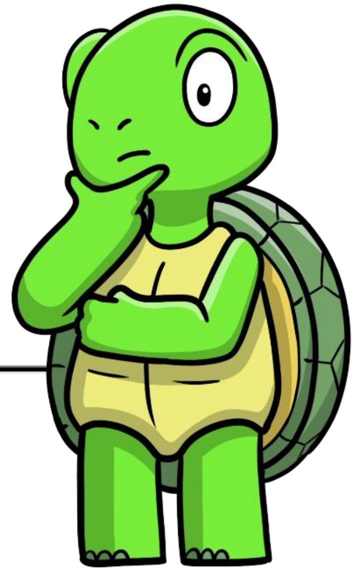
What if the server accepts other messages as well?

Message truncation inside the secure channel is a (cryptographically) successful attack in itself.

Removing `ExtInfo` can negatively impact user authentication!

Caveat: Truncating Encrypted Messages May Hinder Subsequent Message's Decryption

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



But: 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%		

How Can Bob Protect Himself?

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 vendors support “strict kex”

~ 11 million servers offer “strict kex”

We Contacted 31 Vendors During Disclosure

Oct 2023
Initial contact with OpenSSH and AsyncSSH

Nov 2023
AsyncSSH published patch to fix implementation bugs
Initial contact with 29 additional vendors of SSH implementations

Dec 2023
Public Disclosure

Thanks to all involved parties for the smooth responsible disclosure process!

Lessons Learned

1. **Terrapin is a novel cryptographic attack targeting SSH channel integrity**

- Exploitable in practice to downgrade connection‘s security (w/o implementation flaws)
- Enables exploitation of certain implementation flaws as a MitM

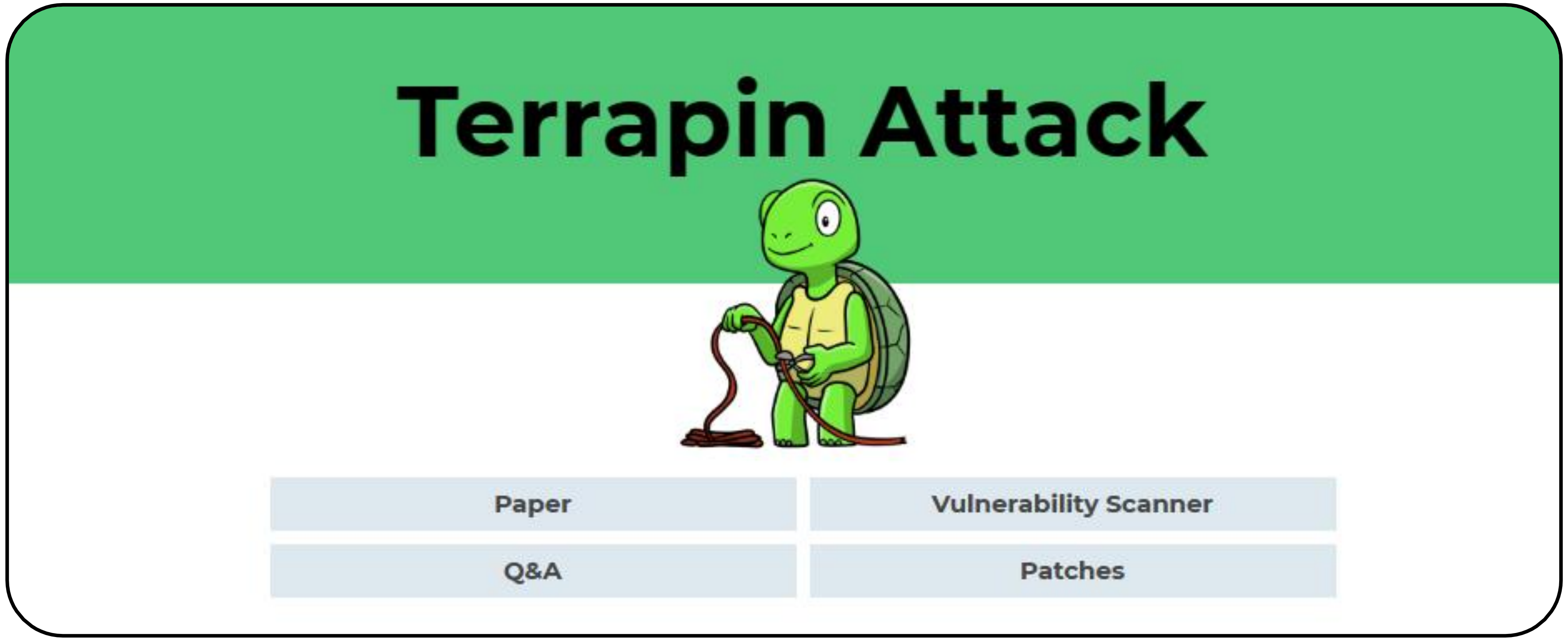
2. **Widespread encryption modes are affected**

- ChaCha20-Poly1305
- CTR / CBC ciphers alongside Encrypt-then-MAC

3. **“Strict Kex” as a protocol-level countermeasure**

- Requires support from client and server to take effect

Thanks! Questions?



Terrapin Attack

https://terrapin-attack.com/

E-Mail:	fabian.baeumer@rub.de
X (formerly Twitter):	@TrueSkrillor
Mastodon:	@Skrillor@infosec.exchange