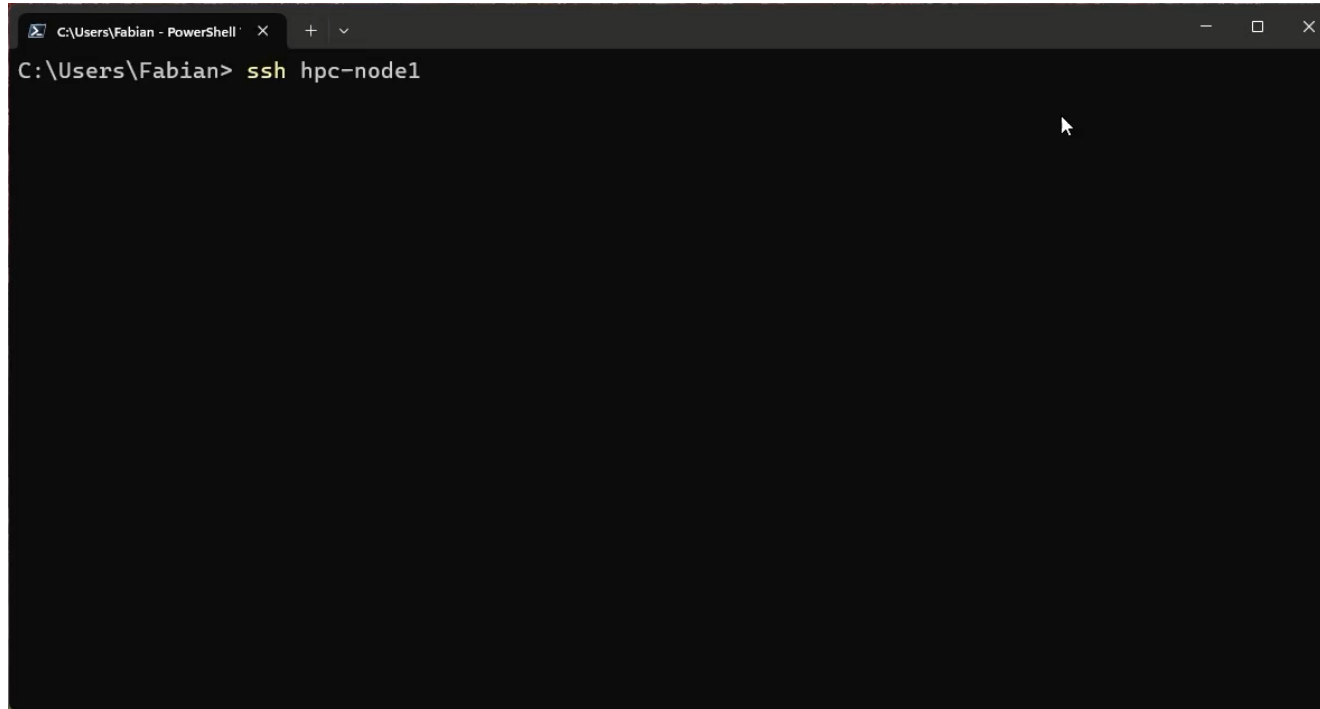


FINDING SSH STRICT KEY EXCHANGE VIOLATIONS BY STATE LEARNING

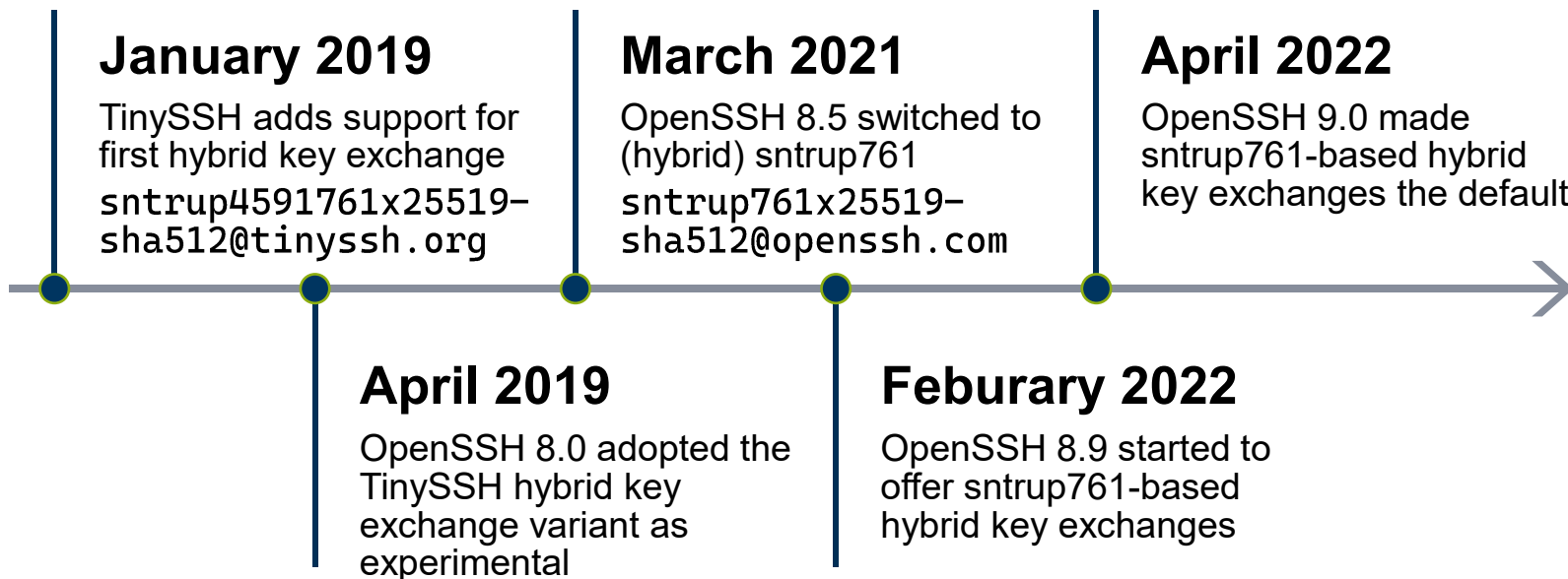
SPIQE'26 Workshop @ IEEE EuroS&P 2026

SSH – The Protocol for Secure Remote Access

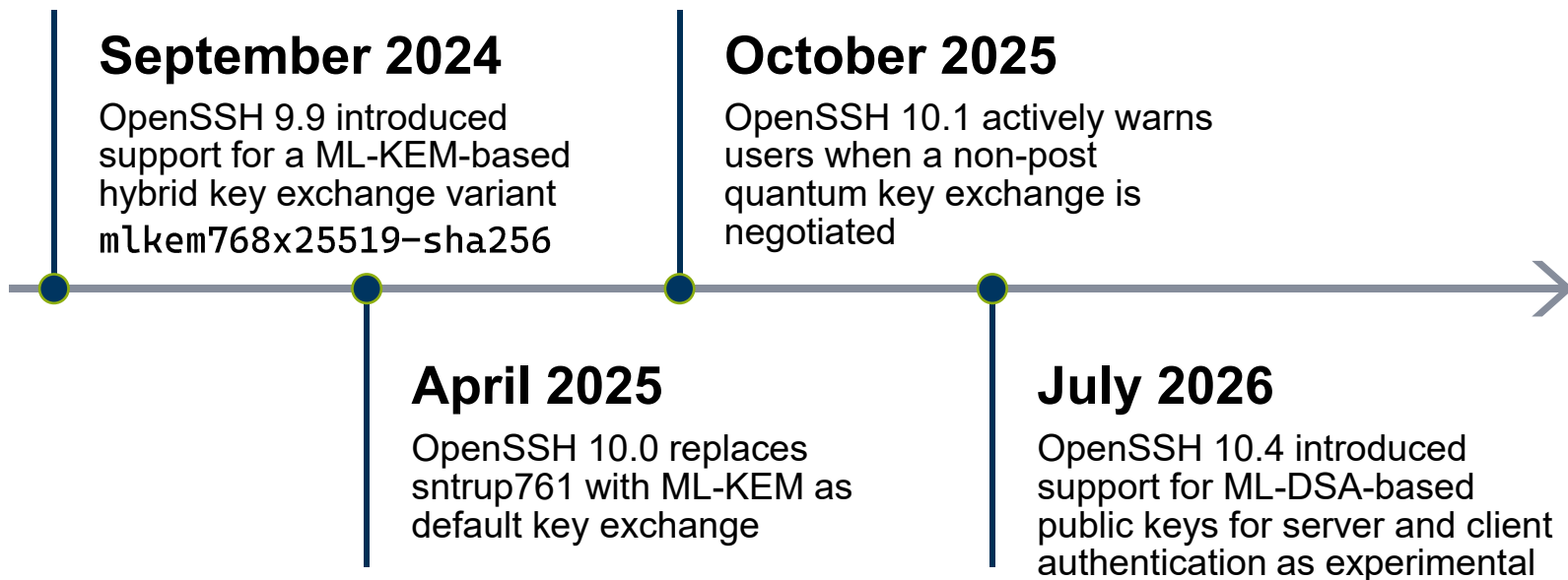


A screenshot of a Windows PowerShell terminal window. The title bar at the top reads "C:\Users\Fabian - PowerShell" and includes standard window controls (minimize, maximize, close). The terminal content shows the command prompt "C:\Users\Fabian>" followed by the command "ssh hpc-node1". The rest of the terminal area is black, indicating the command has been entered but not yet executed or the output is not visible.

(Open)SSH Is a Case Study for the PQC Rollout



(Open)SSH Is a Case Study for the PQC Rollout



In this talk, you will learn about...

SSH and the strict KEX extension

Active state machine learning (SML)

Identifying strict KEX violations with SML

How SML can aid the PQC transition

SSH Is Split Into Separate Layers



SSH Connection Protocol [RFC4254]



SSH Authentication Protocol [RFC4252]



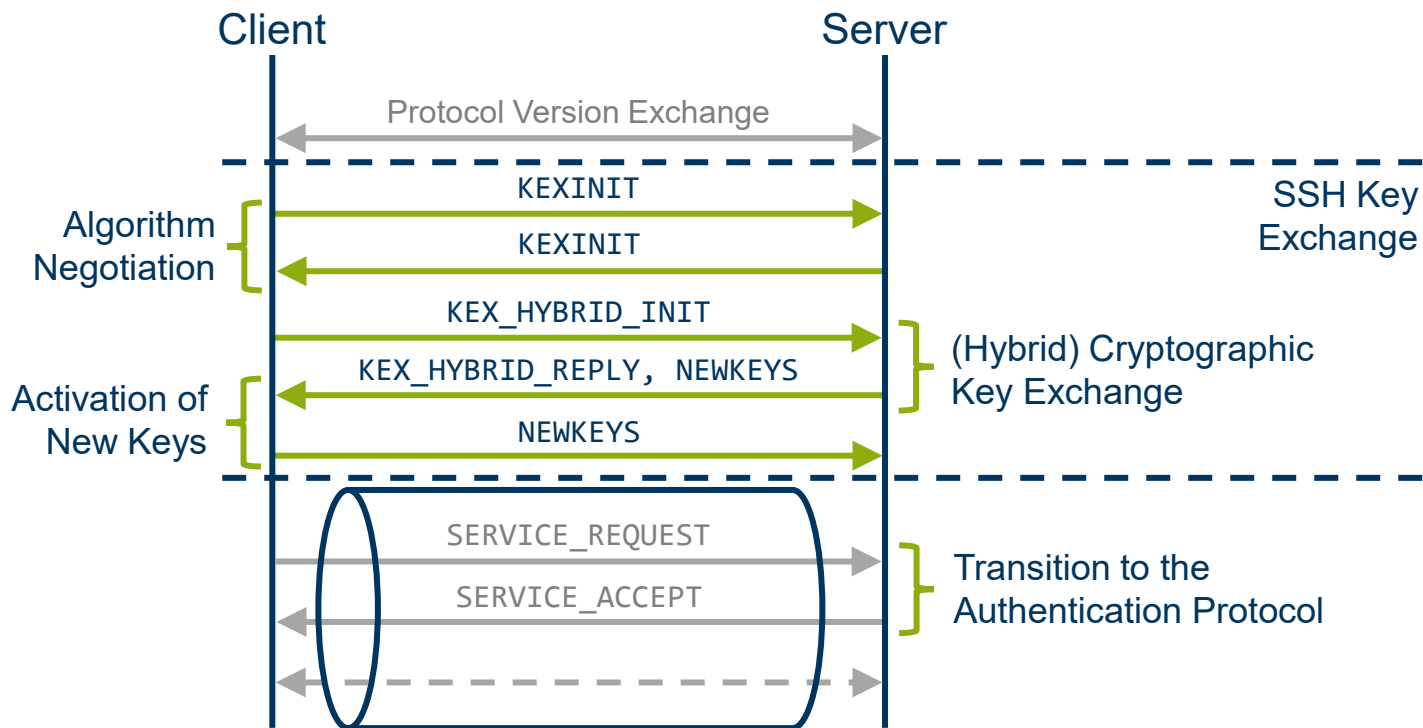
SSH Transport Layer Protocol
(TLP) [RFC4253]

- Binary Packet Protocol
- SSH Key Exchange

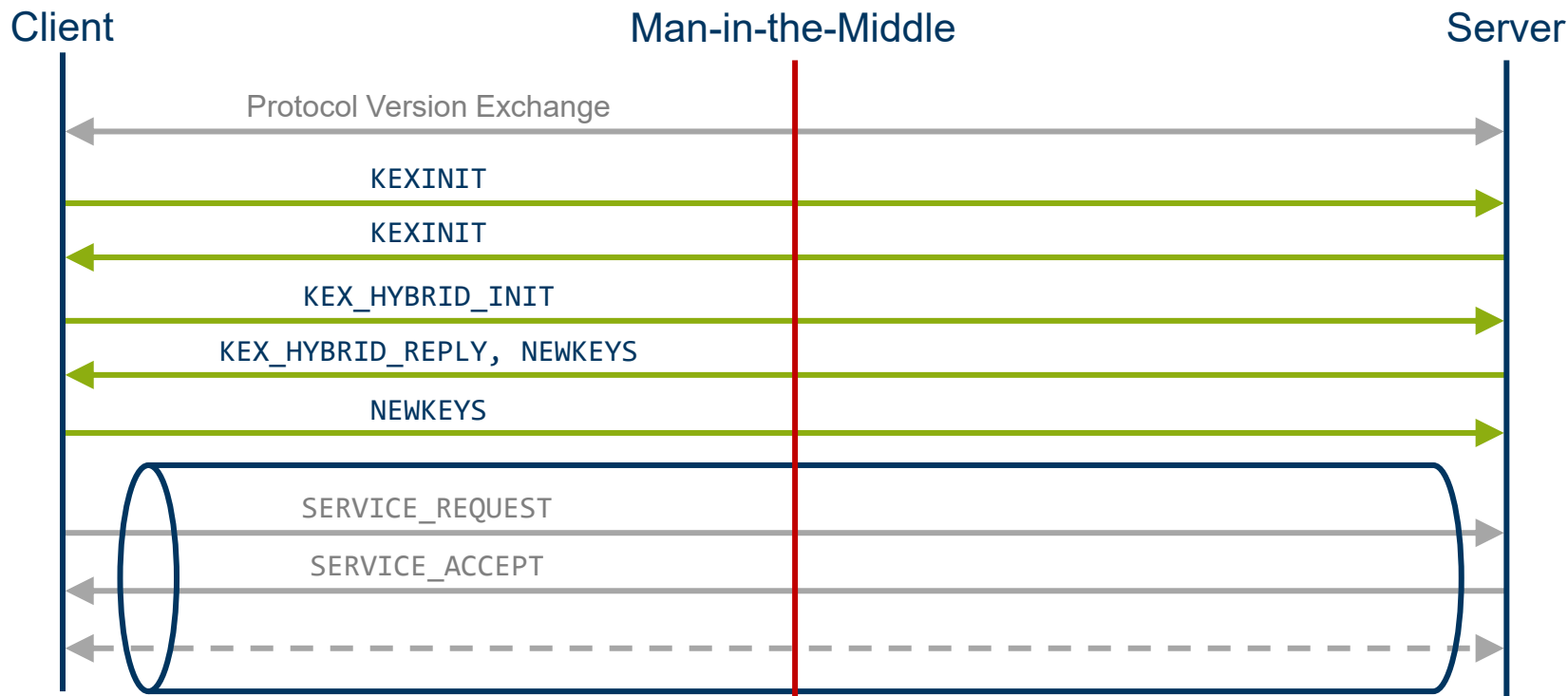


TCP / IP

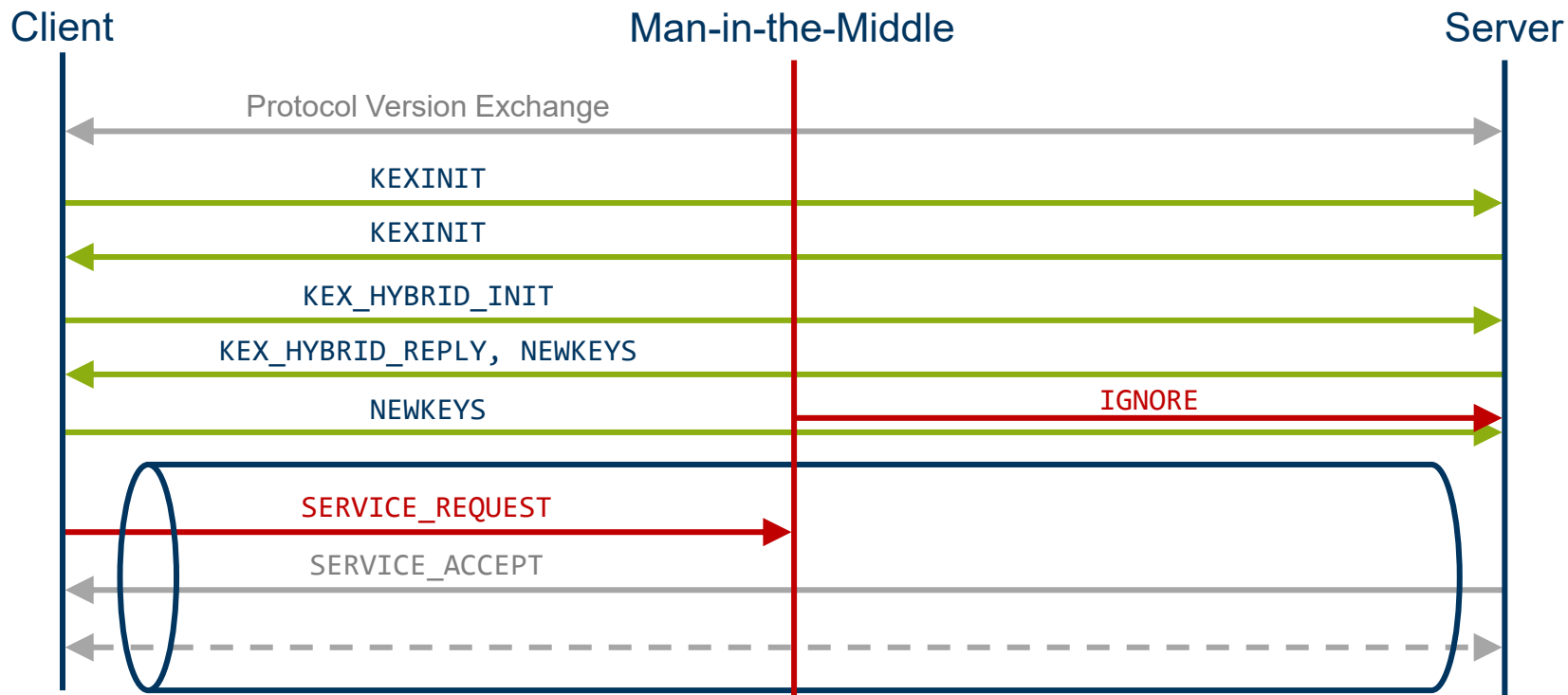
The SSH Transport Layer Consists of 5 Phases



The Terrapin Attack Broke SSH's Channel Integrity



The Terrapin Attack Broke SSH's Channel Integrity



Current Hybrid PQ KEX Are Vulnerable To Terrapin

The Terrapin attack has two root causes

1. Partial transcript authentication
2. No sequence number reset when activating new key material

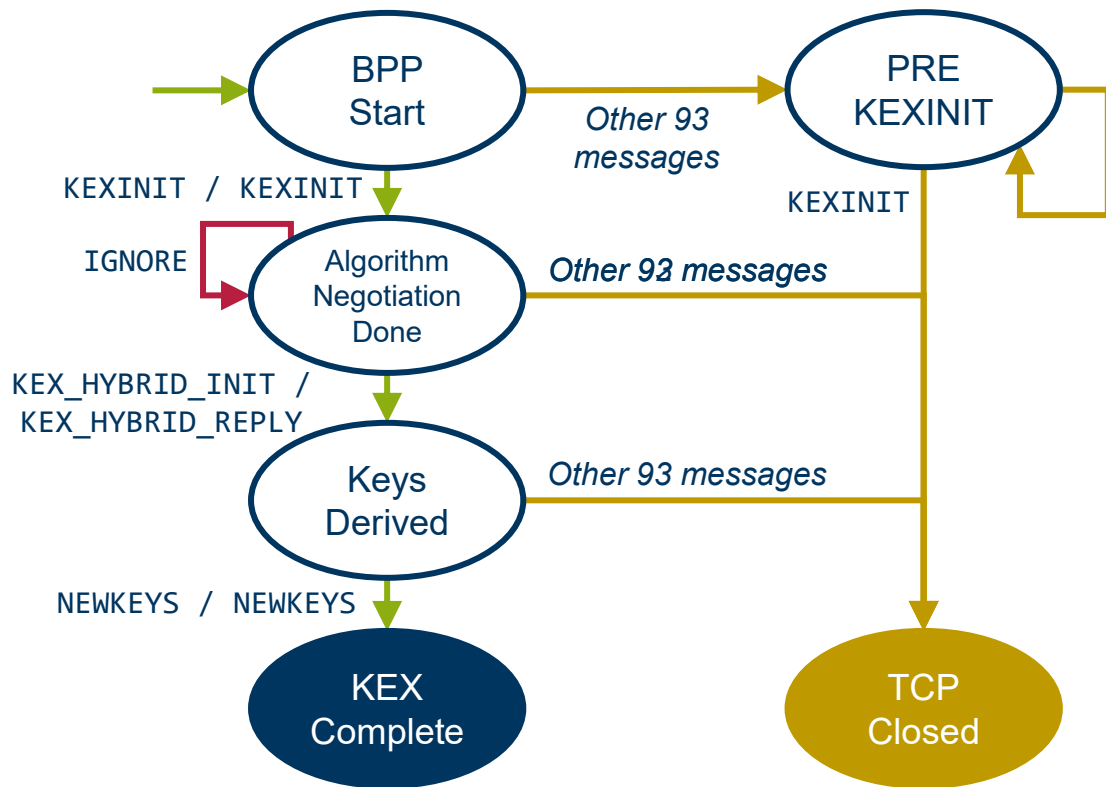
Hybrid PQ KEX inherit both root causes, enabling Terrapin attacks

```
string V_C, client identification string (CR and LF excluded)
string V_S, server identification string (CR and LF excluded)
string I_C, payload of the client's SSH_MSG_KEXINIT
string I_S, payload of the server's SSH_MSG_KEXINIT
string K_S, server's public host key
string C_INIT, client message octet string
string S_REPLY, server message octet string
string K, SSH shared secret
```

Strict KEX is the Countermeasure against Terrapin

- Described in *draft-ietf-sshm-strict-kex-01* and consists of two parts:
 1. Implicit sequence number reset
 2. Hardened message handling during initial key exchange
- **Assumption:** Sequence number reset is implemented correctly
 - Otherwise, the implementation would not be interoperable
- **RQ: Are SSH implementations accepting optional messages during initial KEX despite strict KEX enabled?**

Strict KEX Can Be Described as a Mealy Machine



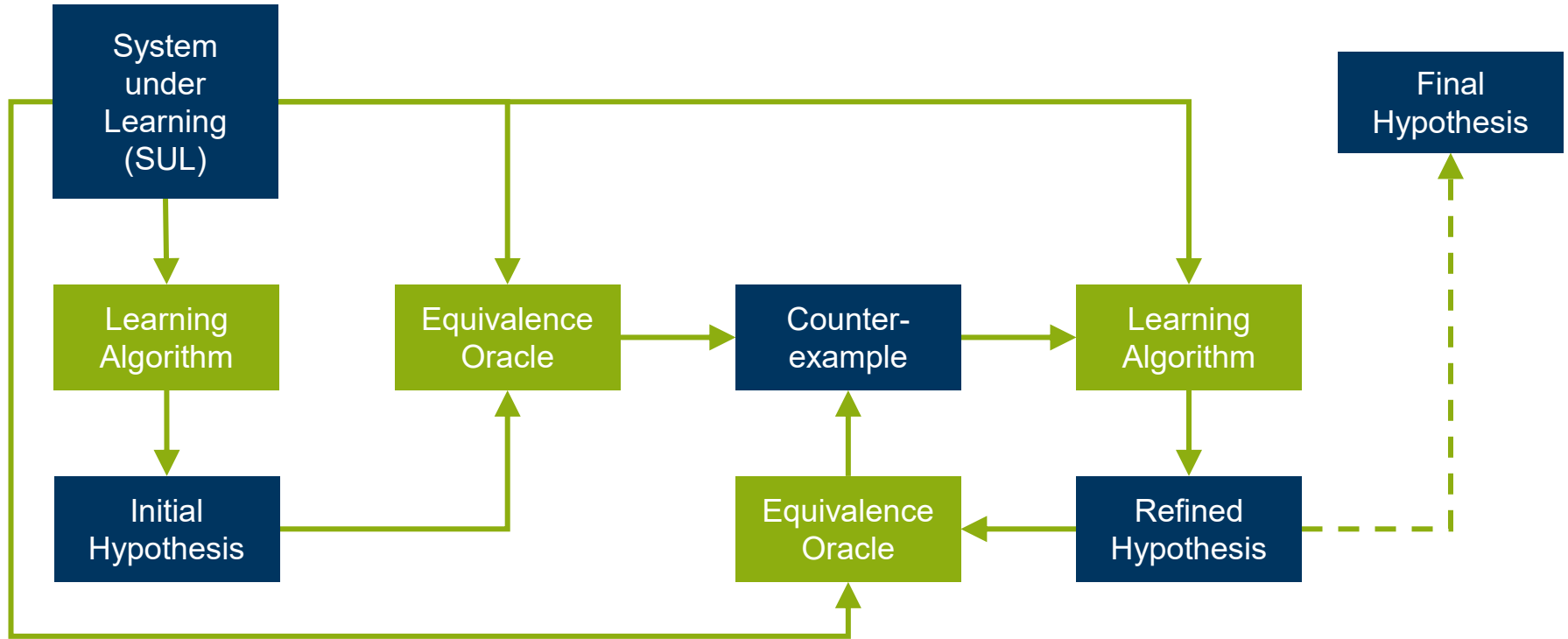
- **Insight:**

Strict KEX violations are additional paths.

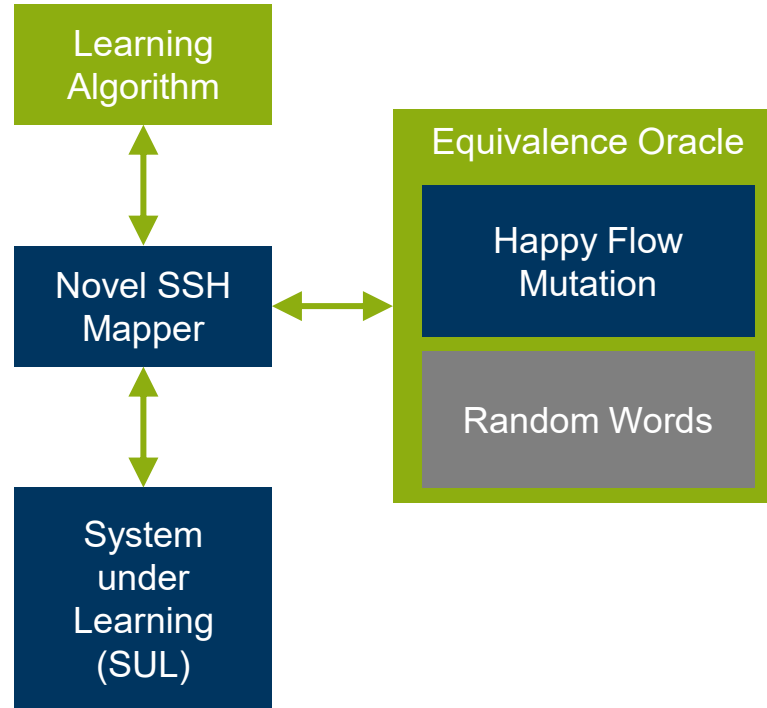
- **Problem:**

How to derive the state machine of an SSH implementation?

Active State Machine Learning Can Derive a State Machine by Interaction



Our SSH State Learner Features a Novel SSH Mapper and Equivalence Oracle



The Happy Flow Equivalence Oracle Explores Input Sequences Close to a Protocol's Happy Flow

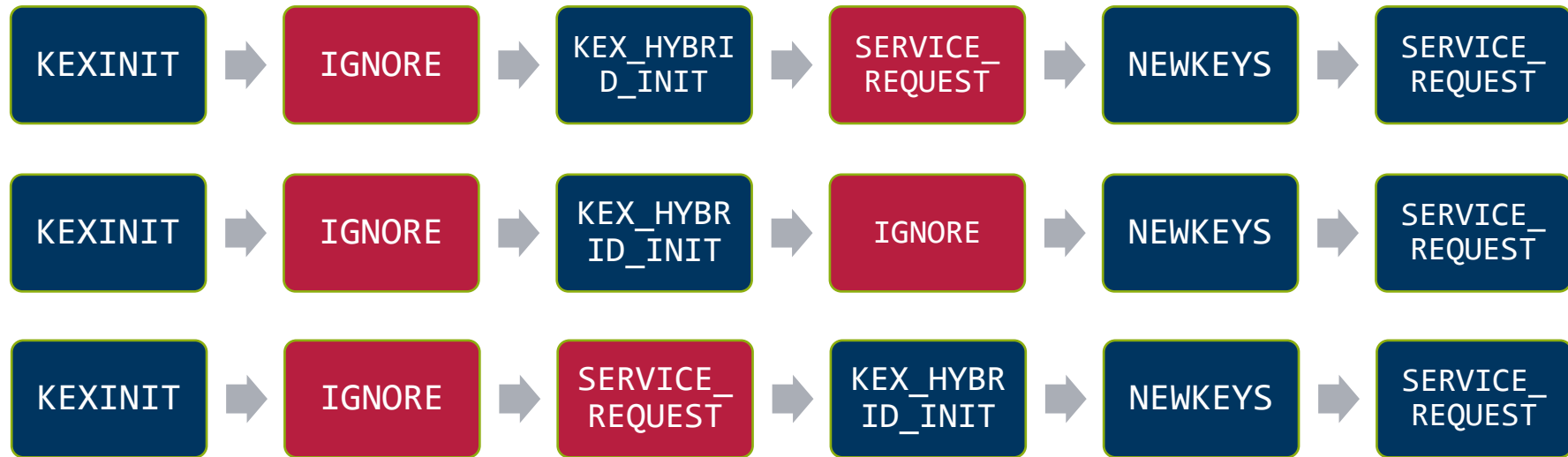


The Happy Flow Equivalence Oracle Explores Input Sequences Close to a Protocol's Happy Flow



The Happy Flow Equivalence Oracle Explores Input Sequences Close to a Protocol's Happy Flow

Full exploration for up to n insertions (here $n = 2$)

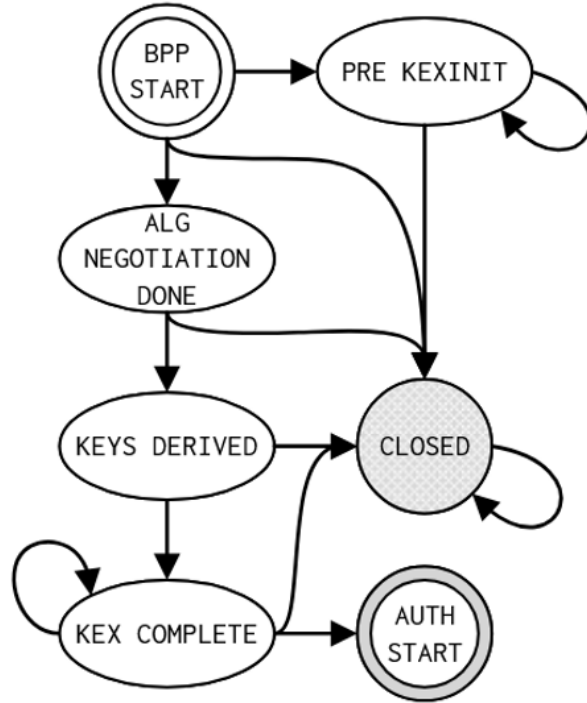


Happy Flow Mutation: 133,011 test sequences <<< Full Exploration: 689,869,781,056 test sequences

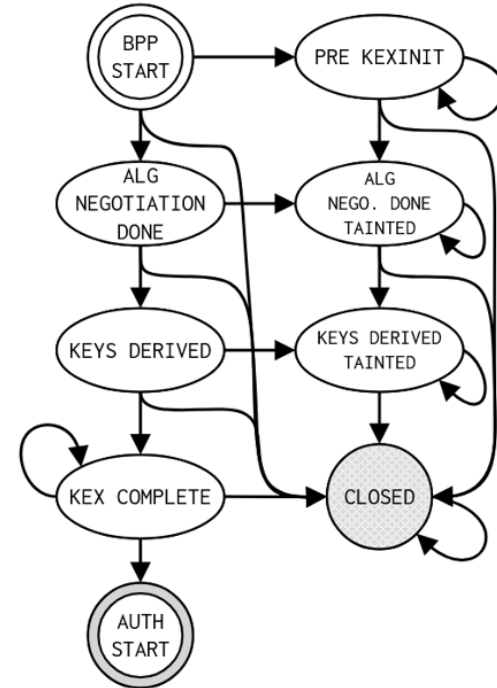
Our Equivalence Oracle Reduces Learning Time from Hours to Minutes

Algorithm	Time [s]	# States		# Queries	
		Total	Key Ex.	Total	Final
Random words	3,189	8	6	45,466	21,163
Random Wp	37,367	38	6	792,680	745,542
Wp-method	128,832 [†]	18	6	4,533,240	369,529
This work	206	9	6	8,906	3,769

We Observed Two Implementation Flavors

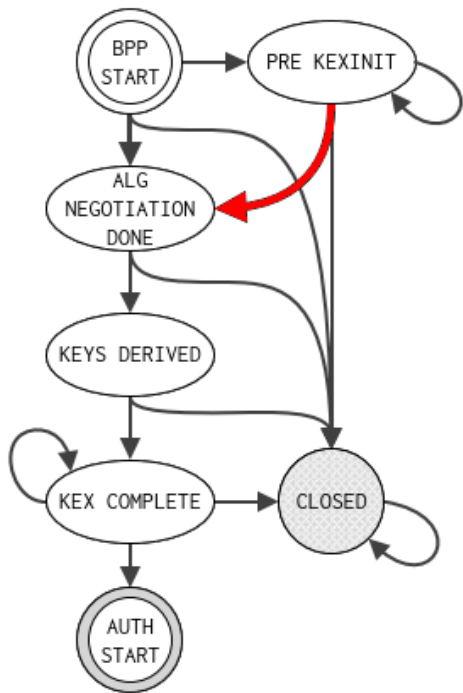


Immediate Termination Implementation

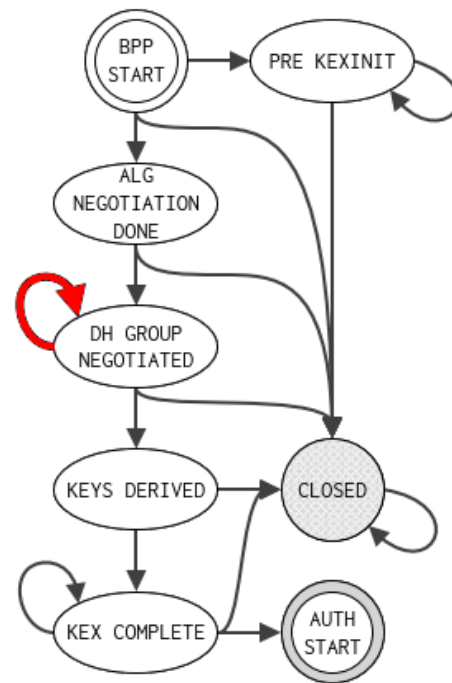


Taint-based Implementation

We Identified 4 Different Violation Categories

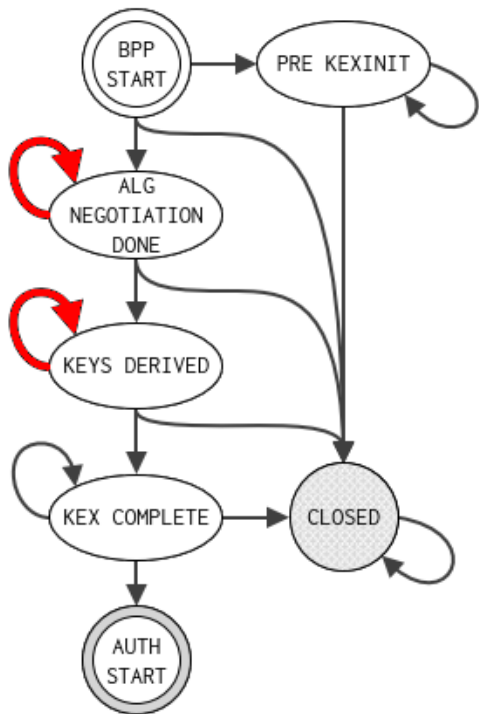


Category C1 – Early Message Injection

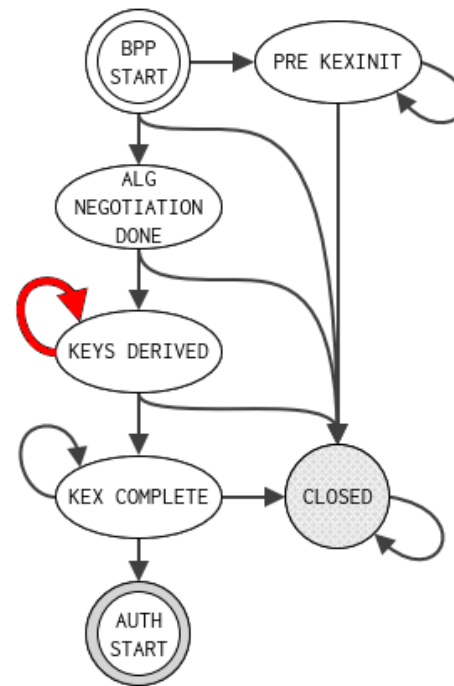


Category C2 – Flow-dependent Injection

We Identified 4 Different Violation Categories



Category C3 – Partial Hardening



Category C4 – Late Message Injection

7 out of 10 Implementations Violated Strict KEX

Implementation	Version	OS	Observed Violations			
			C1	C2	C3	C4
AsyncSSH	2.20.0		—	✗	—	—
Bitvise SSH	9.43		—	—	—	—
Dropbear	2025.87		—	—	—	✗
Erlang SSH	5.2.8		—	—	✗	—
Golang x/crypto/ssh	0.37.0		—	—	—	—
LANCOM LCOS	10.90.0126 Rel		—	—	✗	—
libssh	0.11.1		—	✗	—	—
OpenSSH	9.9p2		—	—	—	—
Tectia SSH	6.6.5.353		✗	—	—	—
TinySSH	20250201		—	—	—	—

All violations except C2 were present in hybrid key exchange flows as well!

We Uncovered Two High-Severity Vulnerabilities

Rogue Session Attack against Tectia SSH (CVE-2025-32942)

- Attacker can log a victim into an attacker-controlled account
- CVSS 7.2 (High)

Unauthenticated Remote Code Execution in Erlang SSH (CVE-2025-32433)

- Erlang SSH accepted channel requests during the initial key exchange
- CVSS 10.0 (Critical)

More details can be
found in our paper!

The PQC Transition Is Not a Simple Swap in Math

In theory, transitioning to PQC is “simple”

“Find all occurrences of classical asymmetric cryptography, replace it with its post-quantum-secure counterpart.”

In practice, transitioning to PQC involves a lot of protocol engineering

- New algorithm identifiers
- Changes to algorithm negotiation and possible fallbacks
- New / adapted protocol flows with new code paths
- Hybrid constructions
- Or even completely new constructions

The PQC Transition Is More Complex Than Strict KEX

Strict KEX was a minor change

- Sequence number reset
- Hardened message handling during the initial key exchange



The PQC transition introduces entirely new protocol flows

- New algorithms (identifiers)
- New protocol flows
- Hybrid constructions

A minor change broke in 7 of 10 implementations; PQC changes far more.

State Learning Can Guard the PQC Transition



Black-box approach, no source code access required



Systematically explores behavior beyond the happy flow



Learned models can be diffed against the specification, across versions, and across implementations



Re-running the learner is cheap whenever a new algorithm or extension lands

There are Plenty of Valuable PQ Learning Targets

- PQ/Hybrid KEX negotiation and fallback
- Framing and error handling of large PQ ciphertexts and keys
- PQ authentication will change the authentication state machine
- For SSH: Interaction of new KEX methods with strict KEX
 - Do sequence number resets and message restrictions still apply?
 - We already verified this for `sntrup761x25519-sha512@openssh.com`

Goal: Identify possible deviations in implementations before large-scale deployment

Conclusion

- **We implement a state learner to find strict KEX violations**
 - Each violation represents an additional path in the implementation's state machine
- **At least 7 out of 10 implementations violate strict KEX**
 - Across 4 different violation categories
 - Two additional high-severity vulnerabilities
- **State learning can aid the PQC transition**
 - The PQC transition is highly complex with a lot of changes to existing protocols
 - Try to catch deviations before rollout

Thanks! Questions?

Link to our paper



<https://doi.org/10.1145/3719027.3765208>

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

Finding SSH Strict Key Exchange Violations by State Learning

Fabian Baeumer
Ruhr University Bochum
Bochum, Germany
fabian.baeumer@rub.de

Marcus Brinkmann
Ruhr University Bochum
Bochum, Germany
marcus.brinkmann@rub.de

Marcel Maehren
Ruhr University Bochum
Bochum, Germany
marcel.maehren@rub.de

Jörg Schwenk
Ruhr University Bochum
Bochum, Germany
joerg.schwenk@rub.de

Abstract

SSH is an important protocol for secure remote shell access to servers on the Internet. At USENIX 2024, Baeumer et al. presented the Terrapin attack on SSH [6], which relies on the attacker injecting optional messages during the key exchange. To mitigate this attack, SSH vendors adopted an extension developed by OpenSSH called *strict key exchange* ("strict KEX"). With strict KEX, optional messages are forbidden during the handshake, preventing the attack. In practice, this should simplify the state machine of an SSH handshake to a linear message flow similar to that of TLS.

In this work, we analyze the design, implementation, and security of strict KEX in popular SSH servers, using black-box state learning, which can uncover the hidden state machine of an implementation. In practice, it is limited by the number of learned messages and the complexity of the state machine. Thus, learning the complete state machine of SSH is infeasible. Previous research on SSH, therefore, excluded optional messages, learning only a partial state machine. However, these messages are a critical part of the Terrapin attack. We propose to instead learn the complete state machine of the handshake phase of an SSH server, but with strict KEX enabled.

We investigate the security of ten SSH implementations supporting strict KEX for up to five key exchange algorithms. In total, we learn 33 state machines, revealing significant differences in the implementations. We show that seven implementations violate the strict KEX specification and find two critical security vulnerabilities. One results in a rogue session attack in the proprietary Tectia SSH implementation. Another affects the official SSH implementation of the Erlang Open Telecom Platform, and enables unauthenticated remote code execution in the security context of the SSH server.

CCS Concepts

• Networks → Transport protocols; Protocol testing and verification; Security protocols.

Keywords

SSH, State Learning, Strict Key Exchange, Protocol Violations, Message Injection, Man-in-the-Middle



This work is licensed under a Creative Commons Attribution 4.0 International License.
CCS '24, Taipei
© 2024 Copyright held by the owner(s)/author(s).
ACM ISBN 978-4-401-1525-0/2024/10
<https://doi.org/10.1145/3719027.3765208>

ACM Reference Format:

Fabian Baeumer, Marcel Maehren, Marcus Brinkmann, and Jörg Schwenk. 2025. Finding SSH Strict Key Exchange Violations by State Learning. In *Proceedings of the 2025 ACM SIGSAC Conf. on Computer and Communications Security (CCS '25)*, October 21–27, 2025, Taipei, ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3719027.3765208>

1 Introduction

SSH, The Secure Shell (SSH) protocol was developed to prevent sniffing and active man-in-the-middle (MitM) attacks when an administrator executes commands on a remote server. Such servers can be stand-alone Linux or Windows systems, cloud-based virtual machines, or embedded devices such as routers, switches, and management systems. The security of SSH servers is thus of critical importance to the Internet and telecommunication networks. Since the release of the SSHv2 standard in 2006 by the IETF as a series of RFCs [33–37], SSH has continuously evolved through a series of extensions, some published as separate RFCs, some developed by the community, and some proprietary to a single product. In 2024, the IETF formed the Secure Shell Maintenance (ssm) working group, intending to revitalize the standardization of SSH under its leadership. These efforts were also motivated by the worldwide transition to post-quantum cryptography and the countermeasures against the Terrapin attack that require SSH standard changes.

Terrapin Attack and Strict KEX. At USENIX 2024, Baeumer et al. presented the Terrapin attack [6] against the integrity of the encrypted and authenticated channel in an SSH connection. The attacker can remove an arbitrary number of messages from the beginning of the channel, for example, drop an EXT_INFO message. The attack exploits that SSH allows optional messages in the handshake (e.g., the IGNORE messages in the partial state machine in Figure 2), and the fact that sequence numbers are not reset before switching to authenticated encryption. As mitigation, OpenSSH proposed a new protocol extension called *strict KEX* [39]. The client and server can negotiate this protocol extension by sending a pseudo-algorithm identifier in their KEXINIT messages. If proposed by both parties, any optional or invalid messages must cause the handshake to fail, effectively banning these messages from this phase. This restriction is also applied retroactively to any messages sent before KEXINIT. Additionally, the two sequence number counters maintained by each peer will be reset, for each of the two unidirectional channels, after sending and receiving the NEWKEYS message.

RUHR
UNIVERSITÄT
BOCHUM

RUB