Protokolle
u23 2015

Florob

Chaos Computer Club Cologne e.V.
https://koeln.ccc.de

Cologne
2015-10-26

- Dateiformat zum Signieren und Verschlüsseln von Dateien
- RFC 4880: OpenPGP Message Format

Florob <florob@babelmonkeys.de>

Anforderungen

Was sind die Anforderungen an OpenPGP?

Anforderungen

Was sind die Anforderungen an OpenPGP?

- Vertraulichkeit (confidentiality)
- Integrität (integrity)
- Authentizität (authentication)
- Verbindlichkeit (non-repudiation)
- mehrere Empfänger
- ASCII-safe

- Plaintext
- Packet
- Signatur
- Integrität
- Verschlüsselung

```
The quick brown fox jumps over the lazy dog.
```

- Plaintext
- Packet
- Signatur
- Integrität
- Verschlüsselung

Literal Data Packet

```
The quick brown fox jumps over the lazy dog.
```

- Plaintext
- Packet
- Signatur
- Integrität
- Verschlüsselung

Literal Data Packet

```
The quick brown fox jumps over the lazy dog.
```

Signature Packet

Version: 4, Type: 1, PK-Algo: RSA, Hash-Algo: SHA-256

```
rsa_sign(sha256(signed_data), priv_key)
```

- Plaintext
- Packet
- Signatur
- Integrität
- Verschlüsselung

Literal Data Packet

```
The quick brown fox jumps over the lazy dog.
```

Signature Packet

Version: 4, Type: 1, PK-Algo: RSA, Hash-Algo: SHA-256

```
rsa_sign(sha256(signed_data), priv_key)
```

Modification Detection Code Packet

```
sha1(preceding_plaintext)
```

- Plaintext
- Packet
- Signatur
- Integrität
- Verschlüsselung

Sym. Enc. Integrity Protected Data Packet

Version: 1
`aes-cfb(K,`

Literal Data Packet

`The quick brown fox jumps over the lazy dog.`

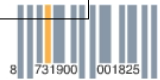
Signature Packet

Version: 4, Type: 1, PK-Algo: RSA, Hash-Algo: SHA-256
`rsa_sign(sha256(signed_data), priv_key)`

Modification Detection Code Packet

`sha1(preceding_plaintext)`

`)`



- Plaintext
- Packet
- Signatur
- Integrität
- Verschlüsselung

Public-Key Encrypted Session Key Packets

Version: 3, Key-ID: 0x32a5c9caf76ada3ad, PK-Algo: RSA
`rsa_enc(sym_cipher || K, pub_key)`

Sym. Enc. Integrity Protected Data Packet

Version: 1
`aes-cfb(K,`

Literal Data Packet

`The quick brown fox jumps over the lazy dog.`

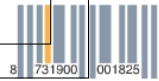
Signature Packet

Version: 4, Type: 1, PK-Algo: RSA, Hash-Algo: SHA-256
`rsa_sign(sha256(signed_data), priv_key)`

Modification Detection Code Packet

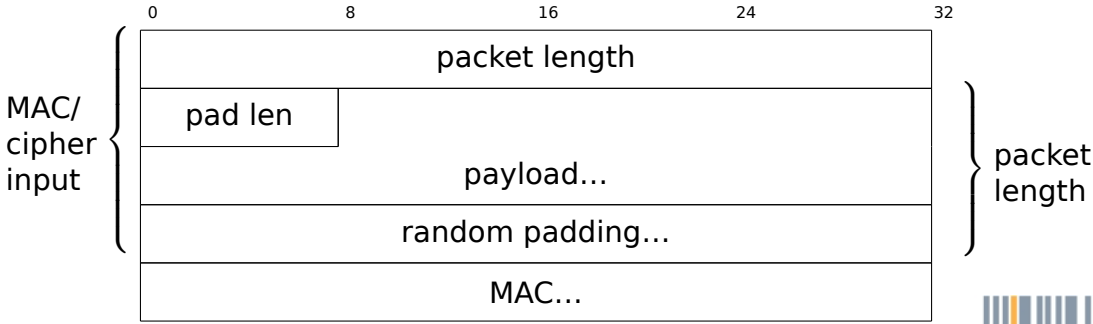
`sha1(preceding_plaintext)`

)

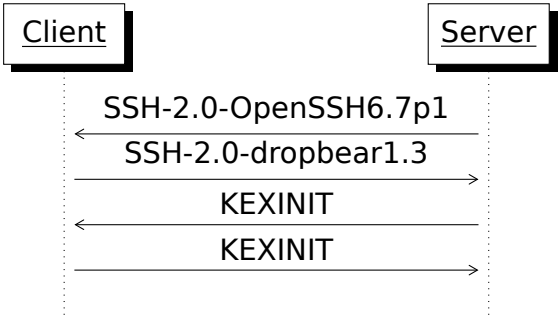


- Entferntes Login auf einem System, über eine unsichere Verbindung
- Meist für Kommandozeile
- Kann auch Datenübertragug (SFTP), etc.
- RFC 4253: The Secure Shell (SSH) Transport Layer Protocol

Paket


0
8
16
24
32
packet length
pad len
payload...
random padding...
MAC...
MAC/
cipher
input
packet
length

Setup


Client
Server
SSH-2.0-OpenSSH6.7p1
SSH-2.0-dropbear1.3
KEXINIT
KEXINIT

SSH_MSG_KEXINIT

- cookie (random bytes)
- key exchange algorithms
- server host key algorithms
- encryption algorithms C2S/S2C
- MAC algorithms C2S/S2C
- compression algorithms C2S/S2C
- languages algorithms C2S/S2C
- first key exchange packet follows
- room for extension

Algorithmenauswahl

- Erster Algorithmus in der Client Liste, der auch beim Server auftaucht
- Zum Beispiel:
  - key exchange: diffie-hellman-group14-sha1
  - server host key: ssh-rsa
  - encryption: aes128-ctr
  - MAC: hmac-sha2-256
  - compression: none

Key-exchange

DH Key-exchange

1. C: Berechne $e = g^x \mod p$, x zufällig
2. C: Sende e (`KEXDH_INIT`)
3. S: Berechne $f = g^y \mod p$, y zufällig
4. S: Berechne $K = e^y \mod p$
5. S: Berechne $H = \text{sha1}(V_C||V_S||I_C||I_S||K_S||e||f||K)$
6. S: Berechne Signatur $s = \text{rsa\_sign}(H)$
7. S: Sende K_S, f, s (`KEXDH_REPLY`)
8. C: Verifiziere K_S
9. C: Berechne $K = f^x \mod p$
10. C: Berechne H (wie oben)
11. C: Verifiziere s

K_S Hostkey

$V_{\{C,S\}}$ Identifikation String

$I_{\{C,S\}}$ `KEXINIT` payload

DH Key-exchange

1. C: Berechne $e = g^x \mod p$, x zufällig
2. C: Sende e (`KEXDH_INIT`)
3. S: Berechne $f = g^y \mod p$, y zufällig
4. S: Berechne $K = e^y \mod p$
5. S: Berechne $H = \text{sha1}(V_C||V_S||I_C||I_S||K_S||e||f||K)$
6. S: Berechne Signatur $s = \text{rsa\_sign}(H)$
7. S: Sende K_S, f, s (`KEXDH_REPLY`)
8. C: Verifiziere K_S
9. C: Berechne $K = f^x \mod p$
10. C: Berechne H (wie oben)
11. C: Verifiziere s

K_S Hostkey

$V_{\{C,S\}}$ Identifikation String

$I_{\{C,S\}}$ `KEXINIT` payload

DH Key-exchange

1. C: Berechne $e = g^x \mod p$, x zufällig
2. C: Sende e (`KEXDH_INIT`)
3. S: Berechne $f = g^y \mod p$, y zufällig
4. S: Berechne $K = e^y \mod p$
5. S: Berechne $H = \text{sha1}(V_C||V_S||I_C||I_S||K_S||e||f||K)$
6. S: Berechne Signatur $s = \text{rsa\_sign}(H)$
7. S: Sende K_S, f, s (`KEXDH_REPLY`)
8. C: Verifiziere K_S
9. C: Berechne $K = f^x \mod p$
10. C: Berechne H (wie oben)
11. C: Verifiziere s

K_S Hostkey

$V_{\{C,S\}}$ Identifikation String

$I_{\{C,S\}}$ `KEXINIT` payload

DH Key-exchange

1. C: Berechne $e = g^x \mod p$, x zufällig
2. C: Sende e (`KEXDH_INIT`)
3. S: Berechne $f = g^y \mod p$, y zufällig
4. S: Berechne $K = e^y \mod p$
5. S: Berechne $H = \text{sha1}(V_C||V_S||I_C||I_S||K_S||e||f||K)$
6. S: Berechne Signatur $s = \text{rsa\_sign}(H)$
7. S: Sende K_S, f, s (`KEXDH_REPLY`)
8. C: Verifiziere K_S
9. C: Berechne $K = f^x \mod p$
10. C: Berechne H (wie oben)
11. C: Verifiziere s

K_S Hostkey

$V_{\{C,S\}}$ Identifikation String

$I_{\{C,S\}}$ `KEXINIT` payload

DH Key-exchange

1. C: Berechne $e = g^x \mod p$, x zufällig
2. C: Sende e (`KEXDH_INIT`)
3. S: Berechne $f = g^y \mod p$, y zufällig
4. S: Berechne $K = e^y \mod p$
5. S: Berechne $H = \text{sha1}(V_C||V_S||I_C||I_S||K_S||e||f||K)$
6. S: Berechne Signatur $s = \text{rsa\_sign}(H)$
7. S: Sende K_S, f, s (`KEXDH_REPLY`)
8. C: Verifiziere K_S
9. C: Berechne $K = f^x \mod p$
10. C: Berechne H (wie oben)
11. C: Verifiziere s

K_S Hostkey

$V_{\{C,S\}}$ Identifikation String

$I_{\{C,S\}}$ `KEXINIT` payload

DH Key-exchange

1. C: Berechne $e = g^x \mod p$, x zufällig
2. C: Sende e (`KEXDH_INIT`)
3. S: Berechne $f = g^y \mod p$, y zufällig
4. S: Berechne $K = e^y \mod p$
5. S: Berechne $H = \text{sha1}(V_C||V_S||I_C||I_S||K_S||e||f||K)$
6. S: Berechne Signatur $s = \text{rsa\_sign}(H)$
7. S: Sende K_S, f, s (`KEXDH_REPLY`)
8. C: Verifiziere K_S
9. C: Berechne $K = f^x \mod p$
10. C: Berechne H (wie oben)
11. C: Verifiziere s

K_S Hostkey

$V_{\{C,S\}}$ Identifikation String

$I_{\{C,S\}}$ `KEXINIT` payload

DH Key-exchange

1. C: Berechne $e = g^x \mod p$, x zufällig
2. C: Sende e (`KEXDH_INIT`)
3. S: Berechne $f = g^y \mod p$, y zufällig
4. S: Berechne $K = e^y \mod p$
5. S: Berechne $H = \text{sha1}(V_C||V_S||I_C||I_S||K_S||e||f||K)$
6. S: Berechne Signatur $s = \text{rsa\_sign}(H)$
7. S: Sende K_S, f, s (`KEXDH_REPLY`)
8. C: Verifiziere K_S
9. C: Berechne $K = f^x \mod p$
10. C: Berechne H (wie oben)
11. C: Verifiziere s

K_S Hostkey

$V_{\{C,S\}}$ Identifikation String

$I_{\{C,S\}}$ `KEXINIT` payload

DH Key-exchange

1. C: Berechne $e = g^x \mod p$, x zufällig
2. C: Sende e (`KEXDH_INIT`)
3. S: Berechne $f = g^y \mod p$, y zufällig
4. S: Berechne $K = e^y \mod p$
5. S: Berechne $H = \text{sha1}(V_C||V_S||I_C||I_S||K_S||e||f||K)$
6. S: Berechne Signatur $s = \text{rsa\_sign}(H)$
7. S: Sende K_S, f, s (`KEXDH_REPLY`)
8. C: Verifiziere K_S
9. C: Berechne $K = f^x \mod p$
10. C: Berechne H (wie oben)
11. C: Verifiziere s

K_S Hostkey

$V_{\{C,S\}}$ Identifikation String

$I_{\{C,S\}}$ `KEXINIT` payload

DH Key-exchange

1. C: Berechne $e = g^x \mod p$, x zufällig
2. C: Sende e (`KEXDH_INIT`)
3. S: Berechne $f = g^y \mod p$, y zufällig
4. S: Berechne $K = e^y \mod p$
5. S: Berechne $H = \text{sha1}(V_C||V_S||I_C||I_S||K_S||e||f||K)$
6. S: Berechne Signatur $s = \text{rsa\_sign}(H)$
7. S: Sende K_S, f, s (`KEXDH_REPLY`)
8. C: Verifiziere K_S
9. C: Berechne $K = f^x \mod p$
10. C: Berechne H (wie oben)
11. C: Verifiziere s

K_S	Hostkey
$V_{\{C,S\}}$	Identifikation String
$I_{\{C,S\}}$	`KEXINIT` payload

DH Key-exchange

1. C: Berechne $e = g^x \mod p$, x zufällig
2. C: Sende e (`KEXDH_INIT`)
3. S: Berechne $f = g^y \mod p$, y zufällig
4. S: Berechne $K = e^y \mod p$
5. S: Berechne $H = \text{sha1}(V_C||V_S||I_C||I_S||K_S||e||f||K)$
6. S: Berechne Signatur $s = \text{rsa\_sign}(H)$
7. S: Sende K_S, f, s (`KEXDH_REPLY`)
8. C: Verifiziere K_S
9. C: Berechne $K = f^x \mod p$
10. C: Berechne H (wie oben)
11. C: Verifiziere s

K_S	Hostkey
$V_{\{C,S\}}$	Identifikation String
$I_{\{C,S\}}$	`KEXINIT` payload

DH Key-exchange

1. C: Berechne $e = g^x \mod p$, x zufällig
2. C: Sende e (`KEXDH_INIT`)
3. S: Berechne $f = g^y \mod p$, y zufällig
4. S: Berechne $K = e^y \mod p$
5. S: Berechne $H = \text{sha1}(V_C||V_S||I_C||I_S||K_S||e||f||K)$
6. S: Berechne Signatur $s = \text{rsa\_sign}(H)$
7. S: Sende K_S, f, s (`KEXDH_REPLY`)
8. C: Verifiziere K_S
9. C: Berechne $K = f^x \mod p$
10. C: Berechne H (wie oben)
11. C: Verifiziere s

K_S Hostkey

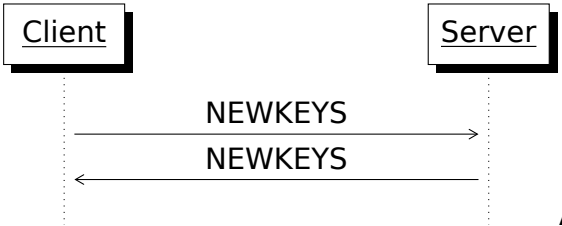
$V_{\{C,S\}}$ Identifikation String

$I_{\{C,S\}}$ `KEXINIT` payload

Key-derivation

- $IV_{C2S} = \text{sha1}(K||H||\text{“A”}||\text{session\_id})$
- $IV_{S2C} = \text{sha1}(K||H||\text{“B”}||\text{session\_id})$
- $\text{encK}_{C2S} = \text{sha1}(K||H||\text{“C”}||\text{session\_id})$
- $\text{encK}_{S2C} = \text{sha1}(K||H||\text{“D”}||\text{session\_id})$
- $\text{macK}_{C2S} = \text{sha1}(K||H||\text{“E”}||\text{session\_id})$
- $\text{macK}_{S2C} = \text{sha1}(K||H||\text{“F”}||\text{session\_id})$

Activate new keys

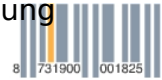


Ab der folgenden Nachricht werden die gerade verhandelten Schlüssel verwendet

Integrität/Authentizität

$$\mathrm{HMAC}(K, P) = \mathrm{sha2}((K \oplus \mathrm{opad}) || \mathrm{sha2}((K \oplus \mathrm{ipad}) || P))$$

- “Keyed-Hashing”
- verifiziert Integrität und Authentizität
- ipad = 0x36 Blockgröße mal
- opad = 0x5c Blockgröße mal
- Berechnet über das ganze Packet (ohne mac) vor Verschlüsselung



Verschlüsselung

- Verschlüsselt wird das ganze Packet, außgenommen der MAC
- Padding so gewählt, das dieser Teil ein vielfaches der Blockgröße ist
- Verschlüsselte Daten als fortlaufender Strom betrachtet