TLS Workshop

German OWASP Day 2018

Achim Hoffmann, Damian Poddebniak, Sebastian Schinzel,

Einführungsrunde

Wer bin ich?

Was ist mein Hintergrund
und was weiß ich über TLS?

Was verspreche ich mir von diesem Workshop?

Einführung in das TLS Protokoll

Historie

- Ursprünglich bei Netscape entwickelt
- SSLv1 wurde niemals veröffentlicht
- SSLv2 wurde 1994 veröffentlicht
  - Entwickelt ohne Einbezug von Sicherheitsexperten
  - schwaches Protokoll mit kritischen Schwachstellen
  - ~19% der Alexa 1.000.000 Sites unterstützt noch SSLv2*
  - Einige wenige unterstützen *nur* SSLv2*

* https://jve.linuxwall.info/blog/index.php?post/TLS_Survey (Stand Januar 2014)

Historie

Verbreitung von SSLv2 – eigene Scans
(Stand November-Dezember 2015)

				Trusted certificates	
Protocol	**Port**	**SSL/TLS**	**SSLv2 support**	**SSL/TLS**	**SSLv2 support**
SMTP	25	3,357 K	936 K (28%)	1,083 K	190 K (18%)
POP3	110	4,193 K	404 K (10%)	1,787 K	230 K (13%)
IMAP	143	4,202 K	473 K (11%)	1,781 K	223 K (13%)
HTTPS	443	34,727 K	5,975 K (17%)	17,490 K	1,749 K (10%)
SMTPS	465	3,596 K	291 K (8%)	1,641 K	40 K (2%)
SMTP	587	3,507 K	423 K (12%)	1,657 K	133 K (8%)
IMAPS	993	4,315 K	853 K (20%)	1,909 K	260 K (14%)
POP3S	995	4,322 K	884 K (20%)	1,974 K	304 K (15%)
(Alexa 1M)	443	611 K	82 K (13%)	456 K	38 K (8%)

https://drownattack.com/drown-attack-paper.pdf

Historie

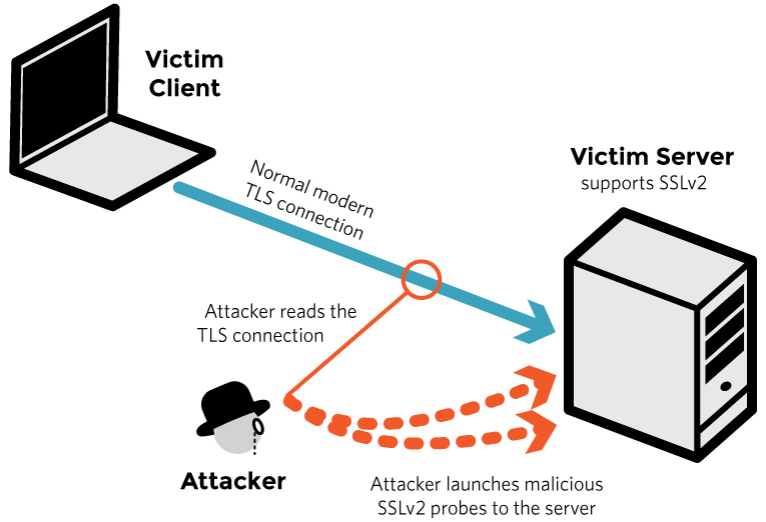
Sicherheitslücken von SSLv2

- Client wählt den Cipher aus einer vor Server bereitgestellten Liste von unterstützen Ciphers
  - → Die Liste der unterstützten Cipher ist nicht authentifiziert
- Truncation-Angriffe sind möglich
- Verschlüsselung und Nachrichten-Authentifizierung nutzen den selben Schlüssel
  - Bei „Export-Ciphers“ ist Authentizität genauso eingeschränkt wie Vertraulichkeit. Das hatte US-Gesetzgebung nicht vorgegeben.
- Anzahl der verwendeten Padding-Bytes ist nicht verschlüsselt
  - → Angreifer lernt exakte Länge des Klartextes selbst bei Block-Ciphern

Historie

Sicherheitslücken von SSLv2

- DROWN
- Angreifer bricht scheinbar sichere TLS-Verbindungen, wenn Server SSLv2 spricht



Historie

- 1995 wurde SSLv3 veröffentlicht
  - Neues Protokolldesign, das auch in späteren Versionen weiter gepflegt wurde
  - War bis Ende 2014 noch weit verbreitet
- 1999 wurde TLS 1.0 als RFC 2246 veröffentlicht
  - Geänderte „MUST“-Anforderungen an Protokolle und Ciphersuites:
    - Diffie-Hellman, Digital Signature Standard (DSS), 3DES
  - MACs wurden zu HMACS

http://yaksman.org/~lweith/ssl.pdf

Historie

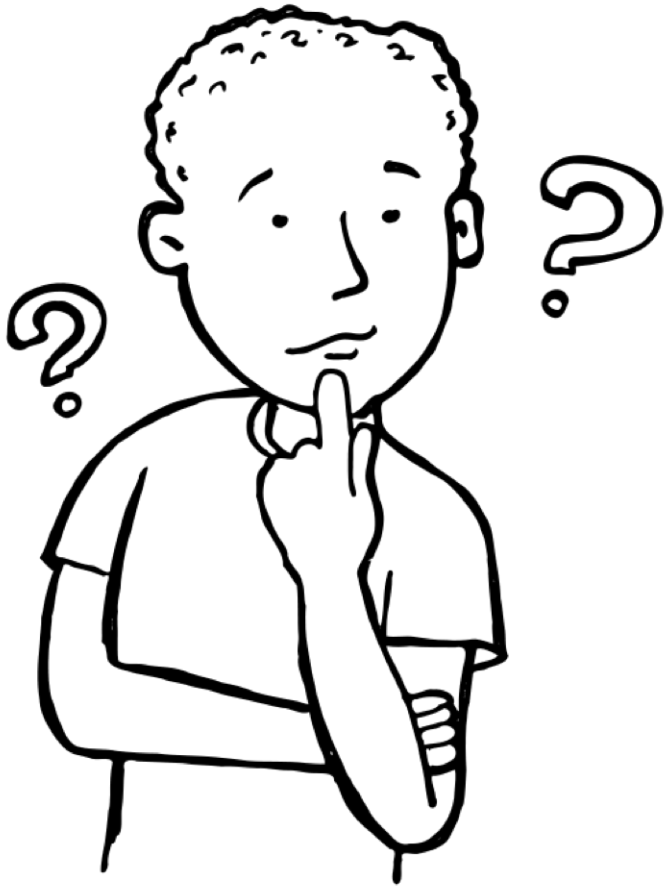
- TLS 1.1 wurde 2006 veröffentlicht
  - führte TLS extensions ein
  - „Export-grade-cryptography“ “MUST NOT“ be used
  - Explizite Initialisierungsvektoren
- TLS 1.2 wurde 2008 veröffentlicht
  - Unterstützung für Authenticated Encryption
- TLS 1.3 ist released unter RFC 8446 (August 2018)
  - Diskussionen in IETF-Mailingliste http://www.ietf.org/mail-archive/web/tls/current/maillist.html

TLS auf Netzwerkschicht

Vertraulichkeit und Integrität von Daten „in transport“

- IPSec – Layer 3 (VPN, VoIP, ...)
- SSL/TLS – Presentation-Layer (alle Netzwerkprotokolle)
- Ende-zu-Ende Verschlüsselung – Layer 7 (PGP, S/MIME, OTR, ...)

#	OSI Schicht	
7	Application	HTTP, SMTP, IMAP, GPG, S/MIME/, OTR
6	Presentation	SSL/TLS
5	Session	-
4	Transport	TCP, UDP
3	Network	IP, IPSec
2	Data layer	Ethernet
1	Physical	CAT5



Diskussion:

Reicht es, Chats nur SSL/TLS zu verschlüsseln?

Bietet SSL/TLS Ende-zu-Ende-Sicherheit?

Wem muss ich vertrauen?

Ziele von SSL/TLS

Nach Priorität:

1. Kryptografische Sicherheit
2. Interoperabilität
3. Erweiterbarkeit
4. Effizienz

Kryptografie

- Vertraulichkeit (Confidentiality)
- Integrität (Integrity)
- Verfügbarkeit (Availability)
- +Kompositionen (Nicht-Abstreitbarkeit, Authentizität, ...)

Kryptografie

Bausteine

- Symmetrische Kryptografie
- Kerckhoffs Prinzip
- Stream Cipher, Block Cipher
- Padding

Kryptografie

Bausteine

- Hash-Funktionen
  - Preimage resistance (find message from hash)
  - Second preimage resistance (generate second message with same hash)
  - Collision resistance (find any two messages with same hash)
- Message Authentication codes
  - „keyed hash“ für Integrität von Nachrichten

Kryptografie

Bausteine

- Block Cipher modes (ECB, CBC, GCM, ...)
- Asymmetrische Verschlüsselung (DH, RSA, ECC)
- Digitale Signaturen
- Generierung von Zufallszahlen
- Protokolle

Kryptografie

Bausteine

- Angriffe gegen kryptografische Protokolle
  - Angriff auf Protokollebene
  - Angriff auf Implementierungsebene
  - Angriff auf kryptografische Primitive

Kryptografie

Die „Stärke“ kryptografischer Primitive

#	Protection	Symm.	Asymm.	DH	ECC	Hash
1	Attacks in real time by individuals	32	-	-	-	-
2	Very short-term protection against small organizations	64	816	816	128	128
3	Short-term protection against medium organizations	72	1008	1008	144	144
4	Very short-termin protection against agencies	80	1248	1248	160	160
5	Short-term protection (10 years)	96	1776	1776	192	192
6	Medium-term protection (20 years)	112	2432	2432	224	224
7	Long-term protection (30 years)	128	3248	3248	256	256
8	Long-term protection and increased defense from quantum computers	256	15,424	15,424	512	512

http://www.ecrypt.eu.org/ecrypt2/documents/D.SPA.20.pdf

Kryptografie

Passive Angriffe

- Verschlüsselte Daten lassen sich im Nachhinein entschlüsseln
- Beispiel: Angreifer zeichnet verschlüsselten TLS-Verkehr auf, bekommt später den RSA-private key und kann damit den Verkehr entschlüsseln
- Fallstudie: Lavabit*

* https://en.wikipedia.org/wiki/Lavabit

Kryptografie

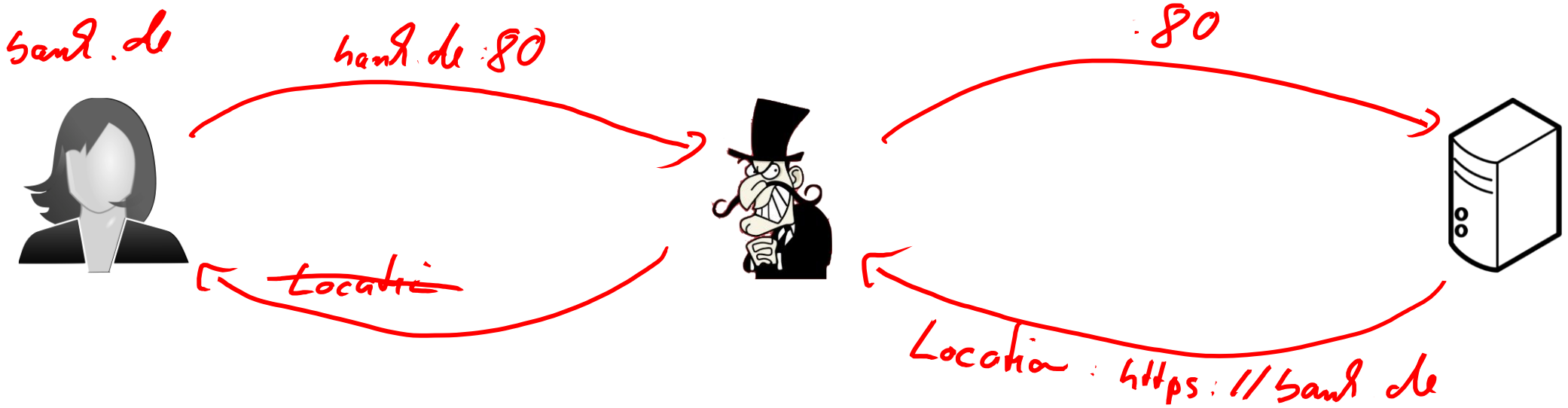
Aktive Angriffe

- Angreifer greift aktiv in den verschlüsselten Verkehr zwischen Client und Server ein
- Angreifer gibt sich gegenüber Client als Server aus und handelt als „man in the middle“

Kryptografie

Aktive Angriffe

- Beispiel: SSLStripping bei HTTPS

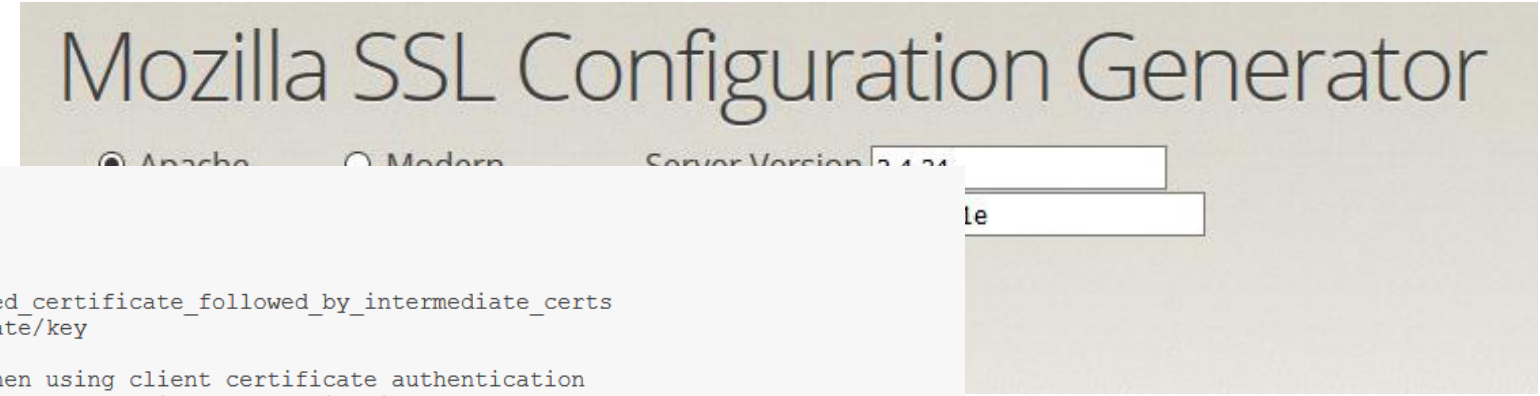


TLS in der Realität

- Was eine „sichere Konfiguration“ von TLS-Servern ist, hat sich in 2013-2016 mehrfach nach wenigen Monaten grundlegend geändert
- → Viele grundlegend neue Angriffe gegen TLS
- Problem: Admins sind sehr „konservativ“ was das Updaten von TLS-Bibliotheken und deren Konfiguration angeht.
- https://www.ssllabs.com/ssltest/analyze.html?d=fh-muenster.de&hideResults=on

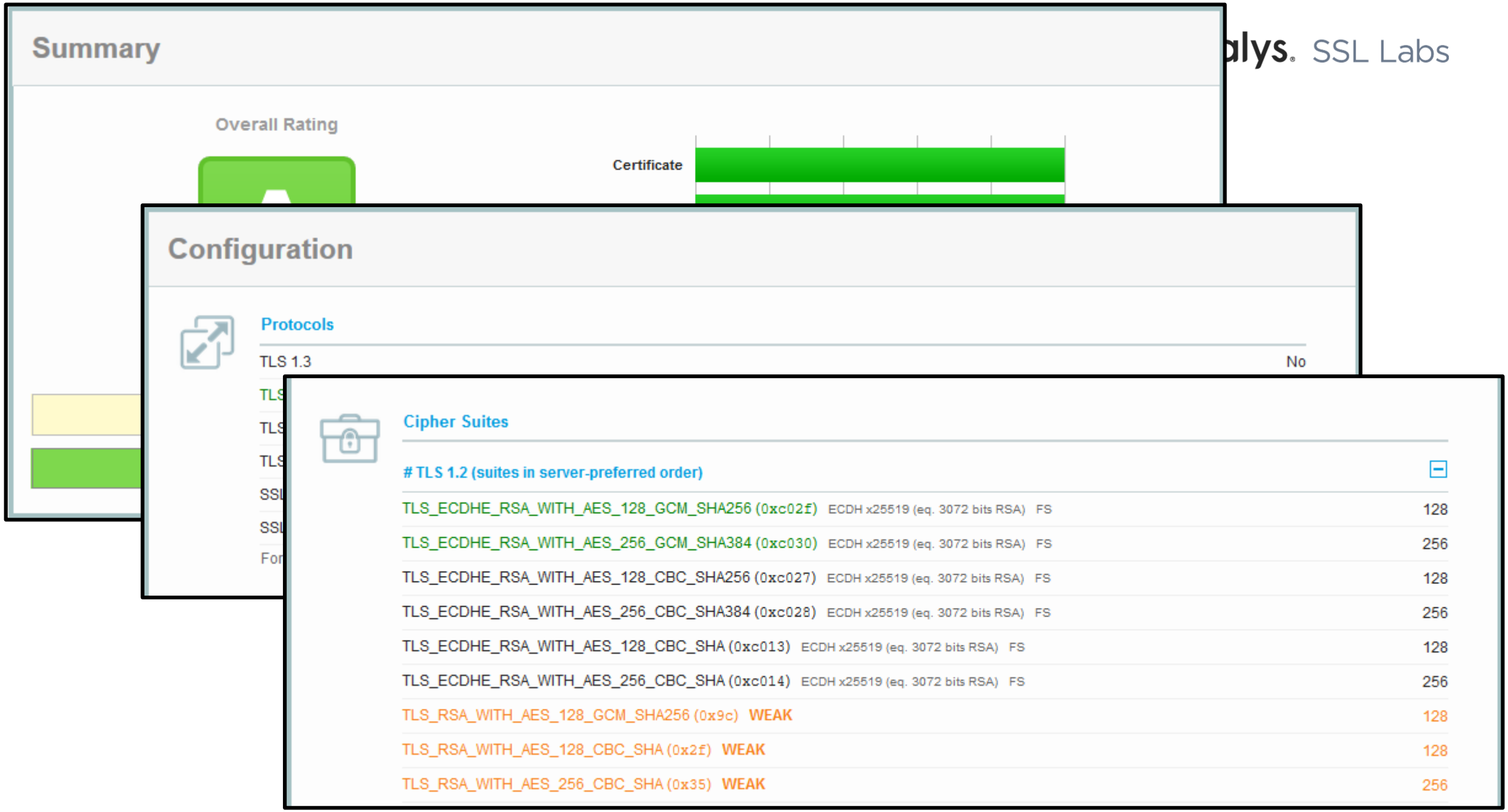
Einführung in TLS

Protokollversionen




```
<VirtualHost *:443>
    ...
    SSLEngine on
    SSLCertificateFile      /path/to/signed_certificate_followed_by_intermediate_certs
    SSLCertificateKeyFile   /path/to/private/key

    # Uncomment the following directive when using client certificate authentication
    #SSLCACertificateFile    /path/to/ca_certs_for_client_authentication


    # HSTS (mod_headers is required) (15768000 seconds = 6 months)
    Header always set Strict-Transport-Security "max-age=15768000"
    ...
</VirtualHost>

# modern configuration, tweak to your needs
SSLProtocol             all -SSLv3 -TLSv1 -TLSv1.1
SSLCipherSuite          ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:
SSLHonorCipherOrder     on
SSLCompression          off
SSLSessionTickets       off

# OCSP Stapling, only in httpd 2.3.3 and later
SSLUseStapling          on
SSLStaplingResponderTimeout 5
SSLStaplingReturnResponderErrors off
SSLStaplingCache        shmcb:/var/run/ocsp(128000)
```

https://wiki.mozilla.org/Security/Server_Side_TLS


Summary
Overall Rating
Certificate
Configuration
Protocols
TLS 1.3
No
Cipher Suites
TLS 1.2 (suites in server-preferred order)
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f) ECDH x25519 (eq. 3072 bits RSA) FS 128
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030) ECDH x25519 (eq. 3072 bits RSA) FS 256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027) ECDH x25519 (eq. 3072 bits RSA) FS 128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028) ECDH x25519 (eq. 3072 bits RSA) FS 256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013) ECDH x25519 (eq. 3072 bits RSA) FS 128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) ECDH x25519 (eq. 3072 bits RSA) FS 256
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c) WEAK 128
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f) WEAK 128
TLS_RSA_WITH_AES_256_CBC_SHA (0x35) WEAK 256

Ciphersuits

	Key-Exchange				Bulk Encryption		
TLS_	**ECDHE**	_	**RSA**	_WITH_	**AES_128_GCM**	_	**SHA256**
TLS_	**ECDHE**	_	**RSA**	_WITH_			
TLS_	**ECDHE**	_	**RSA**	_WITH_			
				…			
TLS_		**RSA**		_WITH_			

Hybride Verschlüsselung

- Schlüsselaustausch mit *asymmetrischer Kryptografie*
  - RSA
  - Diffie-Hellmann
  - Elliptische Kurven Diffie-Hellmann
- Nutzdatenverschlüsselung mit *symmetrischer Kryptografie*
  - 3DES
  - AES
  - CHACHA20

Ciphersuits

	Key-Exchange				Bulk Encryption		
TLS_	**ECDHE**	_	**RSA**	_WITH_	**AES_128_GCM**	_	**SHA256**
TLS_	**ECDHE**	_	**RSA**	_WITH_	**CHACHA20_POLY1305**	_	**SHA256**
TLS_	**ECDHE**	_	**RSA**	_WITH_	**AES_128_CBC**	_	**SHA256**
				…			
TLS_		**RSA**		_WITH_	**3DES_CBC**	_	**SHA**

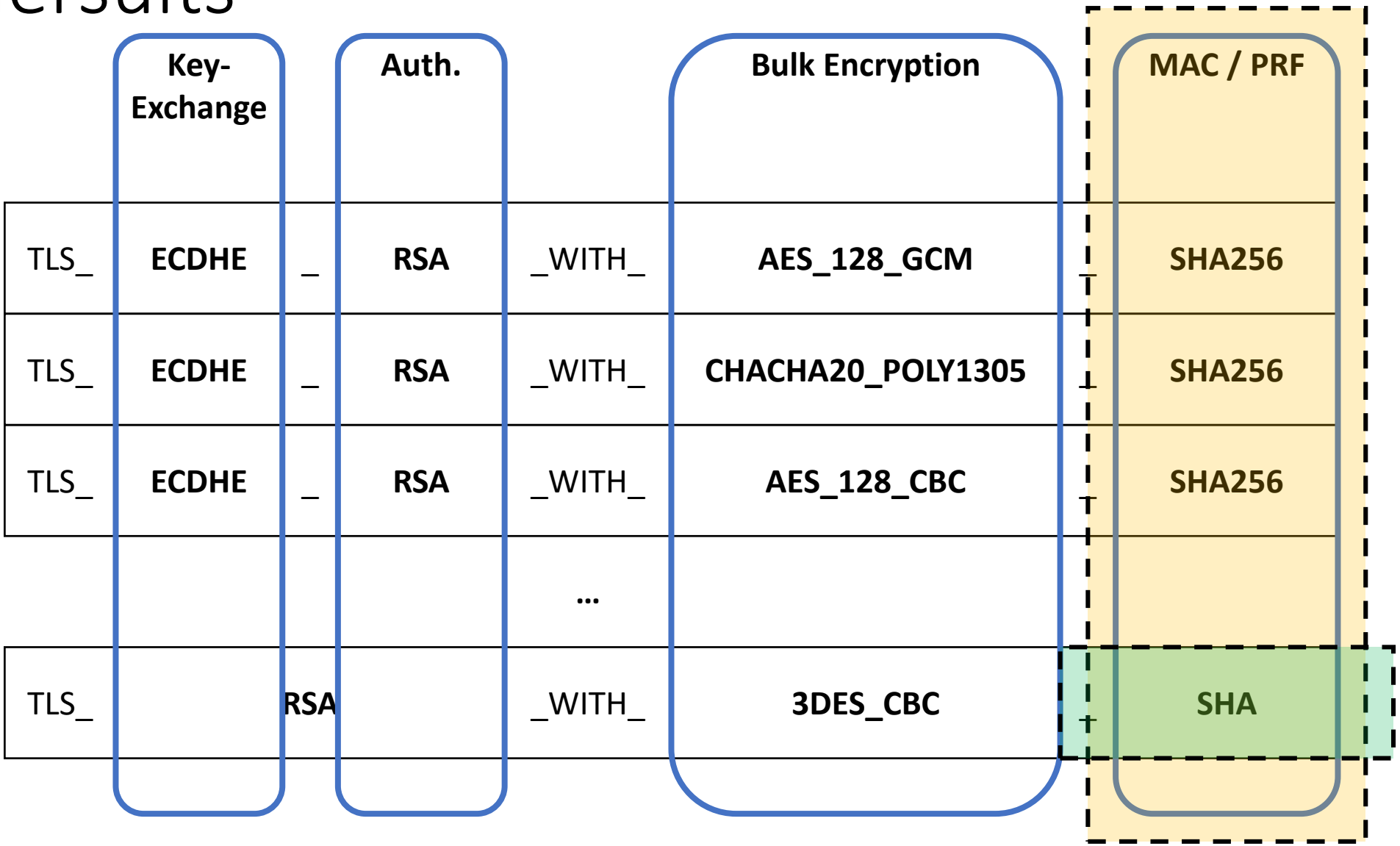
Ciphersuits

	Key-Exchange		Auth.		Bulk Encryption		
TLS_	**ECDHE**	_	**RSA**	_WITH_	**AES_128_GCM**	_	**SHA256**
TLS_	**ECDHE**	_	**RSA**	_WITH_	**CH**		
TLS_	**ECDHE**	_	**RSA**	_WITH_			
				…			
TLS_		**RSA**		_WITH_			

Zertifikatsbasierte Authentifizierung

- Server schickt Zertifikat mit öffentlichem Schlüssel zur
  - *Verschlüsselung* oder
  - Zur *Verifizierung einer Signatur*

Ciphersuits



Grundlagen Kryptografie

Es gibt zwei Phasen in TLS

- Schlüsselaustausch
  - Es wird ein gemeinsames Geheimnis ausgehandelt
- Verschlüsselte Kommunikation
  - Gemeinsames Geheimnis wird zur Authentifizierung und Verschlüsselung der ausgetauschten Nutzdaten verwendet

Bausteine

- Asymmetrische Veschlüsselung (RSA)
- Signaturen (RSA, DSA, ECDSA, EdDSA, Ed25519, …)
- Schlüsselaustauschverfahren (DH, ECDH, X25519, …)

- Ciphers
  - Streamcipher,
  - Blockcipher und
  - Verschlüsselungsmodi (CBC, GCM, …)

- Authentifizierung
  - Hashes und
  - MACs

RSA Verschlüsselung

Verschlüsselung

- Public-Key
  - pub = (N, e)

- c = RSA_enc(pub, m)

Entschlüsselung

- Secret-Key
  - sec = (N, d)

- m = RSA_dec(sec, c)

Ciphersuits

	Key-Exchange		Auth.		Bulk Encryption		MAC / PRF
TLS_	**ECDHE**	_	**RSA**	_WITH_	**AES_128_GCM**	_	**SHA256**
TLS_	**ECDHE**	_	**RSA**	_WITH_	**CHACHA20_POLY1305**	_	**SHA256**
TLS_	**ECDHE**	_	**RSA**	_WITH_	**AES_128_CBC**	_	**SHA256**
				…			
TLS_	**RSA**			_WITH_	**3DES_CBC**	_	**SHA**

RSA Schlüsselaustausch

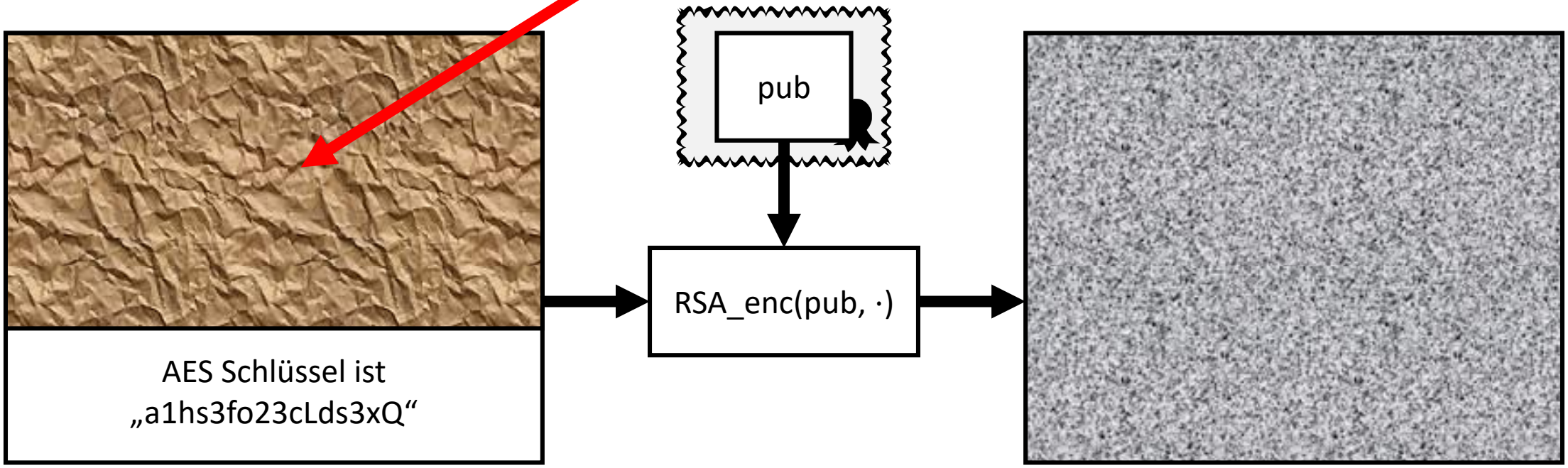
- Client generiert ein gemeinsames Geheimnis, dass s.g. „Premaster Secret“ und sendet es RSA-Verschlüsselt an den Server

- Server entschlüsselt den empfangenen RSA-Ciphertext und entnimmt das Premaster Secret.

→ Client und Server teilen sich ein gemeinsames Geheimnis

Verschlüsselung

PKCS#1 v1.5 Padding oder
„Bleichenbacher Padding“

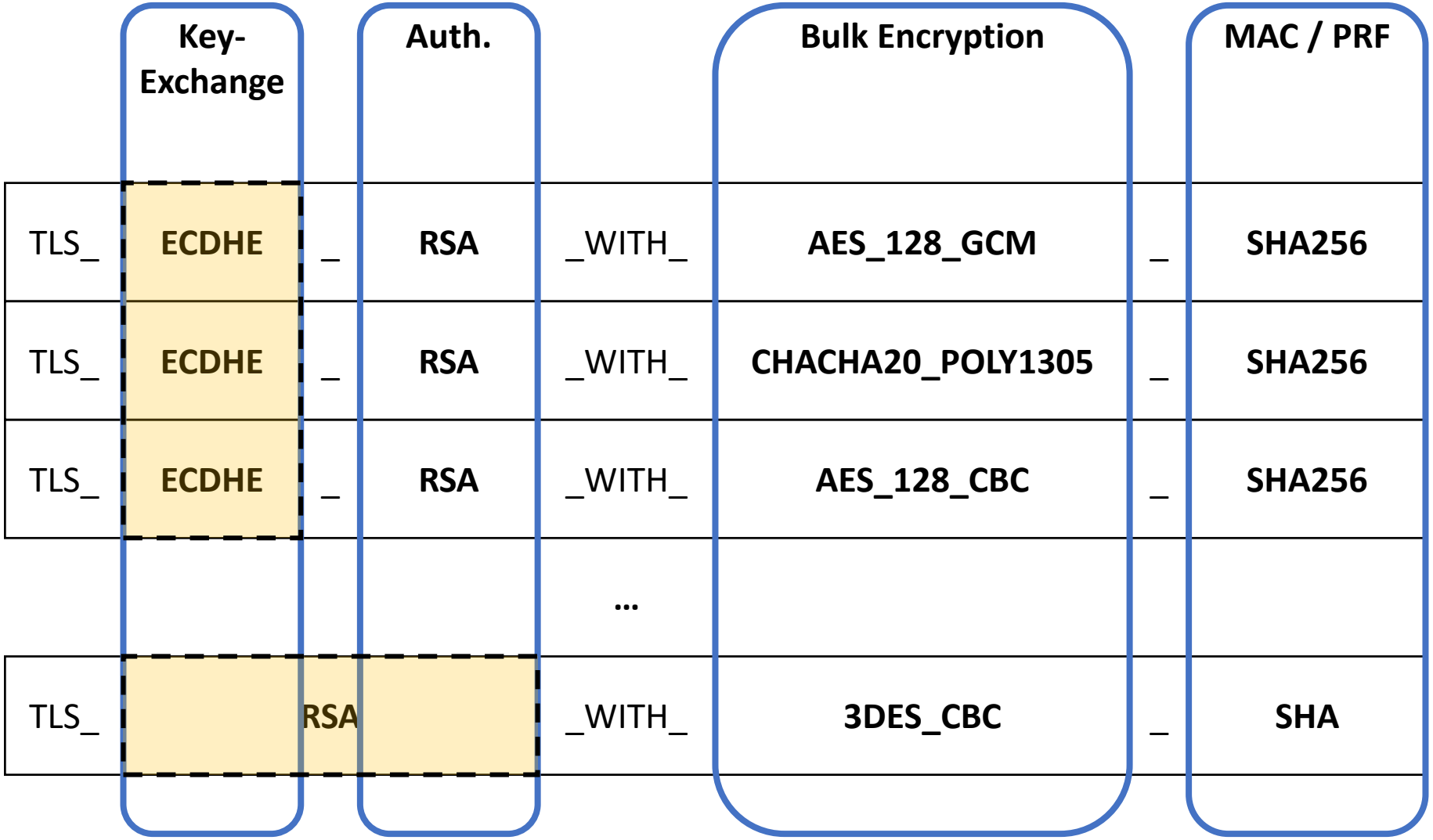


Schlüsselaustauschverfahren

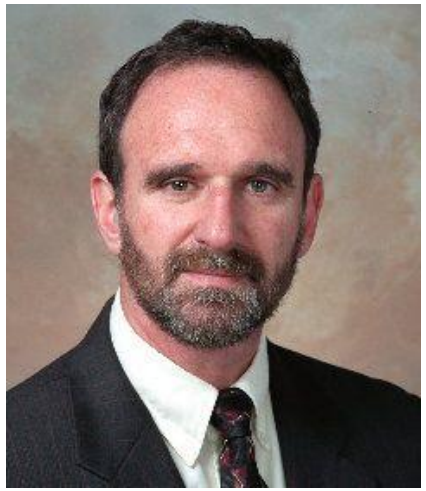
- RSA kann zum Schlüsselaustausch verwendet werden…
  - …sollte es aber nicht!
  - Einfachheit täuscht, Bleichenbacher, ROBOT, …
  - RSA Schlüsselaustausch bietet keine Forward-Secrecy!

- Es gibt bessere dedizierte Schlüsselaustauschverfahren
  - Diffie-Hellman
  - Elliptische Kurven Diffie-Hellman
  - …

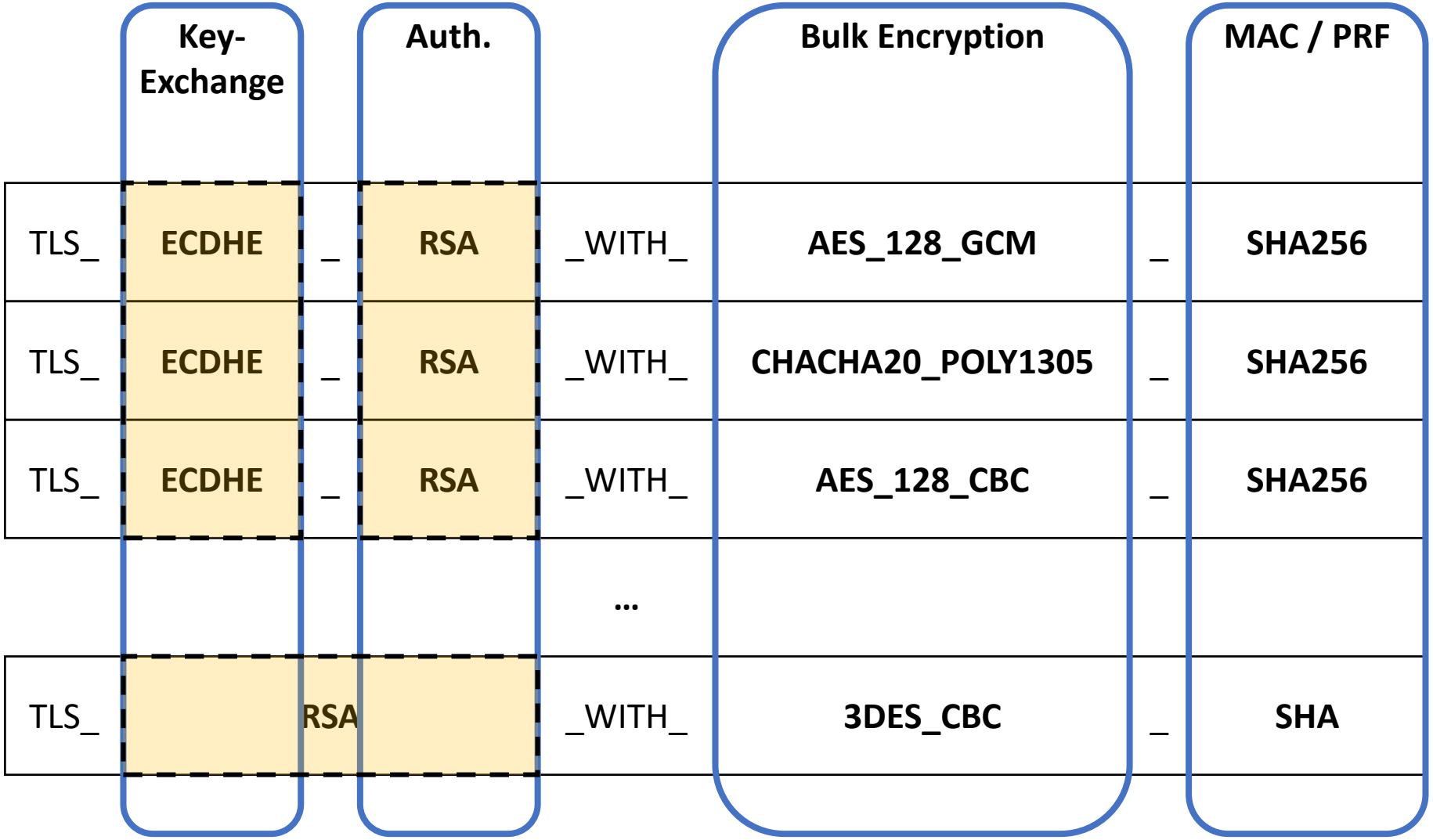
Ciphersuits



Diffie-Hellman Schlüsselaustausch

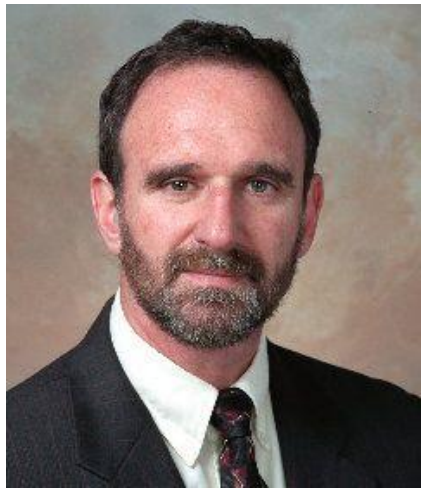


Ciphersuits



Diffie-Hellman Schlüsselaustausch

Whitfield		**Martin**
Wählt $x < p - 1$		Wählt $y < p - 1$
$X := g^x \mod p$	$\longrightarrow$	
	$\longleftarrow$	$Y := g^y \mod p$
$Y^x \mod p =$	$g^{xy} \mod p$	$= X^y \mod p$



Diffie-Hellman Schlüsselaustausch

Whitfield

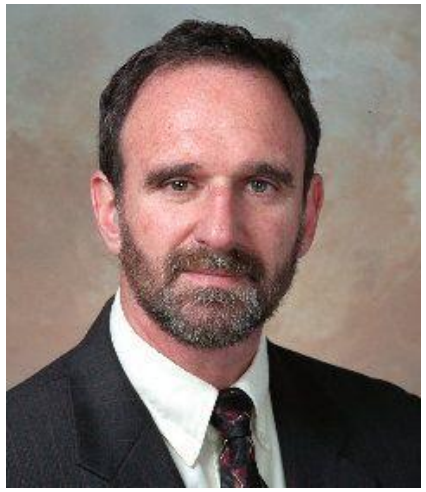
Wählt $x < p - 1$

Martin

Wählt $y < p - 1$

$X := g^x \mod p \longrightarrow$

$\longleftarrow Y := g^y \mod p$



RSA Signaturen

Verschlüsselung

- Public-Key
  - pub = (N, e)

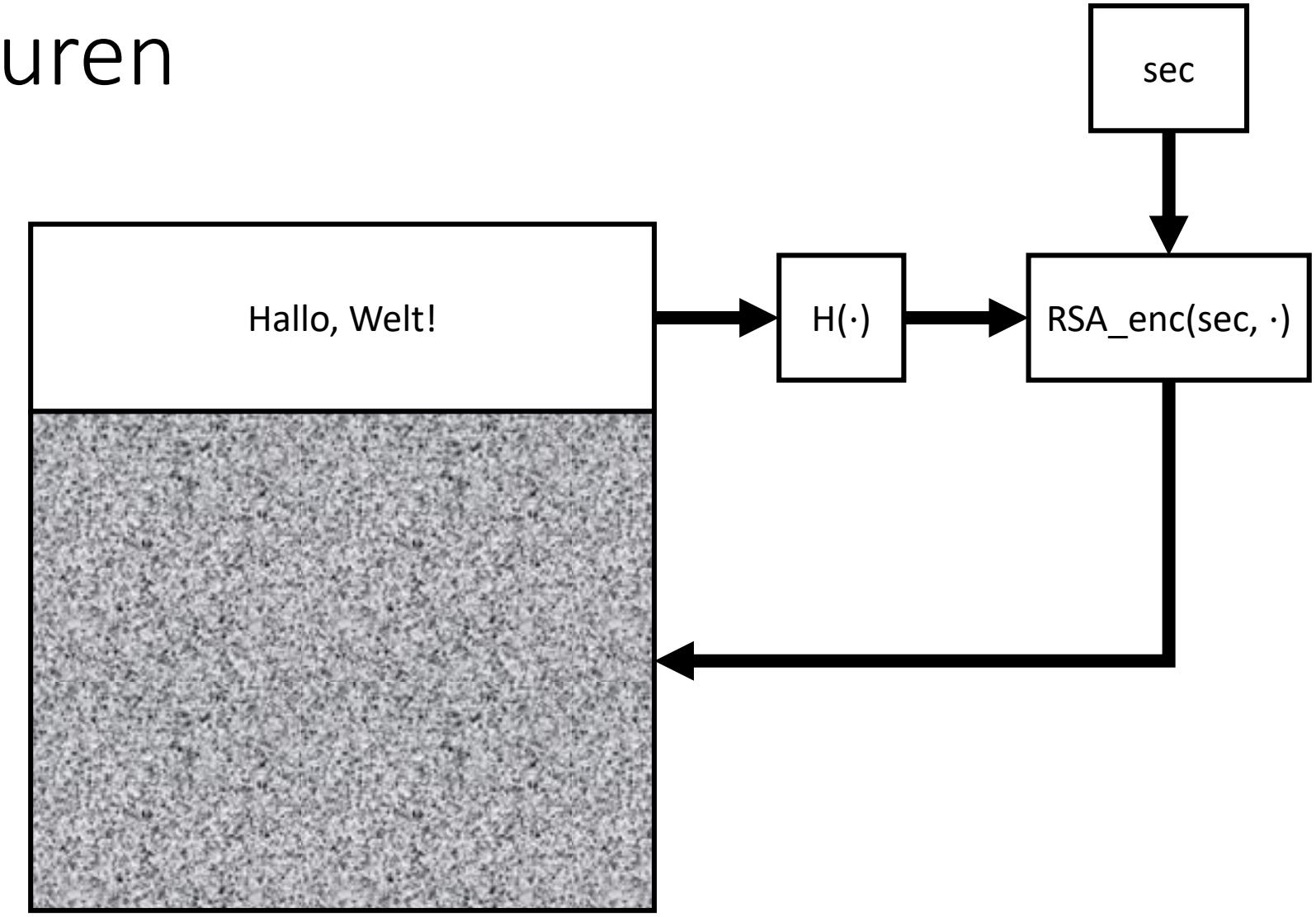
- c = RSA_enc(sec, m)

Entschlüsselung

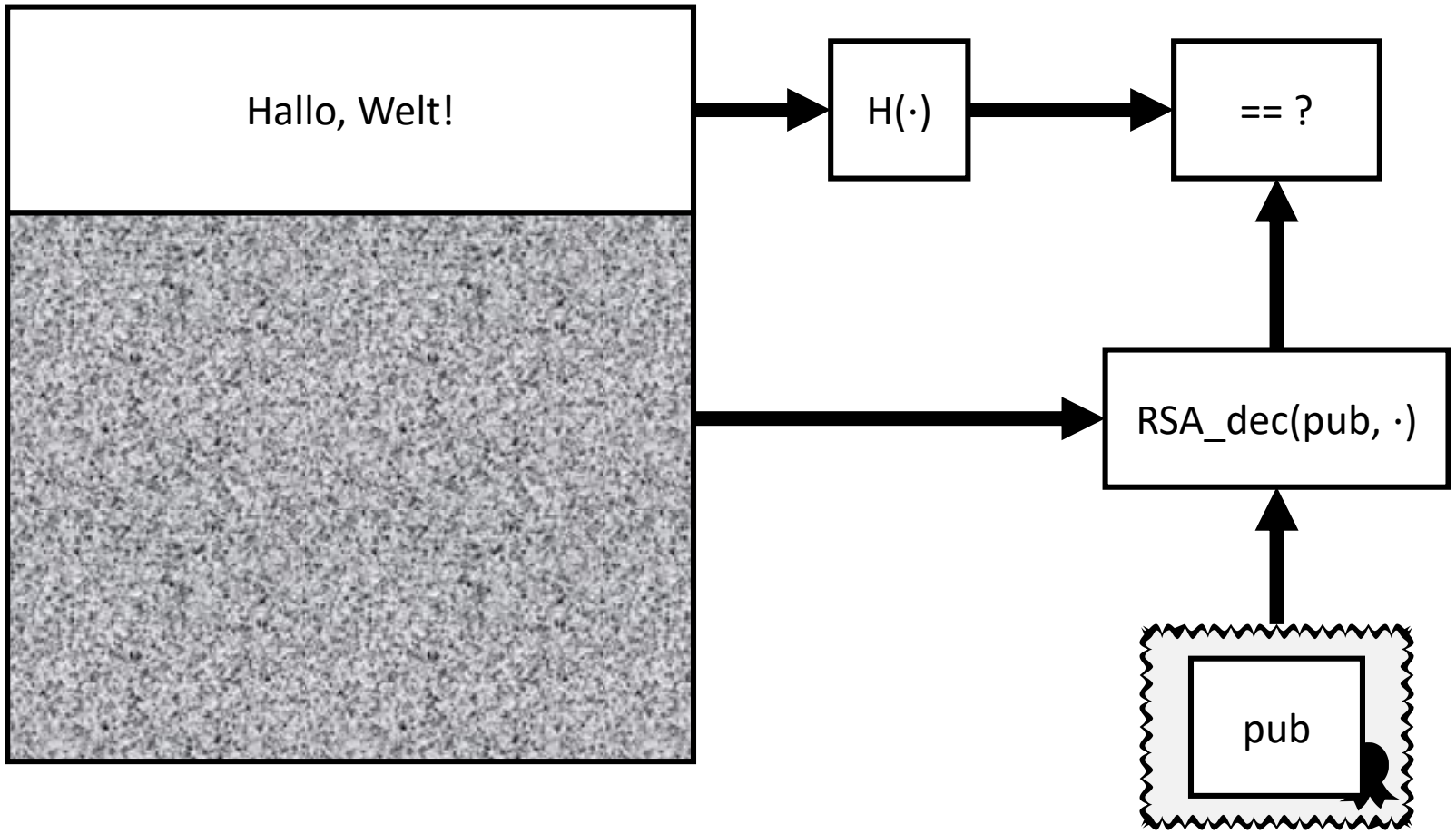
- Secret-Key
  - sec = (N, d)

- m = RSA_dec(pub, c)

Signaturen



Signaturen



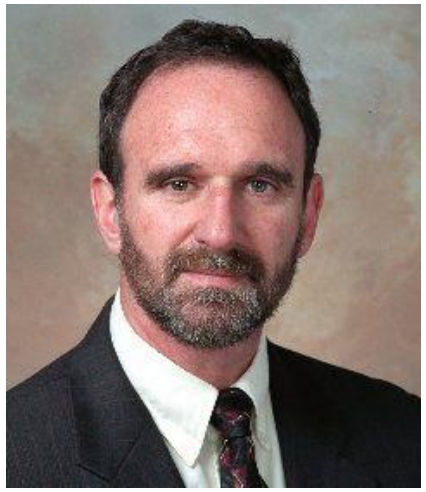
Diffie-Hellman Schlüsselaustausch

Whitfield

Wählt $x < p - 1$

Martin

Wählt $y < p - 1$



$X := g^x \mod p \longrightarrow$

$\longleftarrow Y := g^y \mod p$

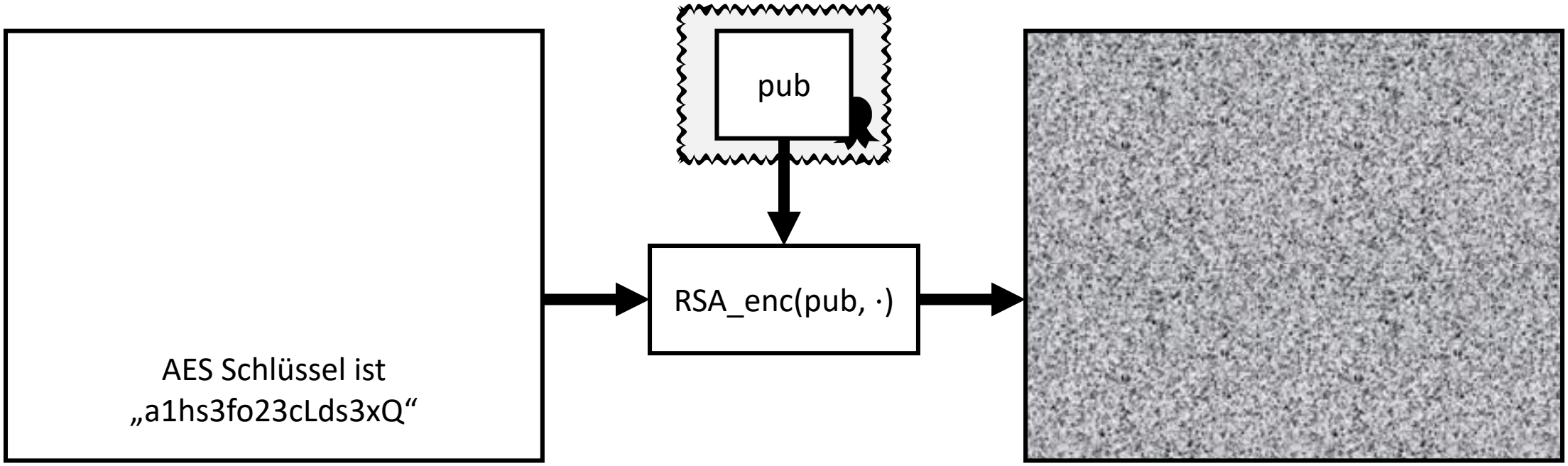
$Y^x \mod p = g^{xy} \mod p = X^y \mod p$

Diffie-Hellman und RSA Signaturen

- DHE_RSA bedeutet:
  - „RSA-signierter Diffie-Hellman Schlüsselaustausch“

- Das E bedeutet *Ephemeral*, d.h. die Schlüssel werden immer wieder neu ausgehandelt (Forward Secrecy)

RSA Schlüsselaustausch (keine Forward Secrecy)



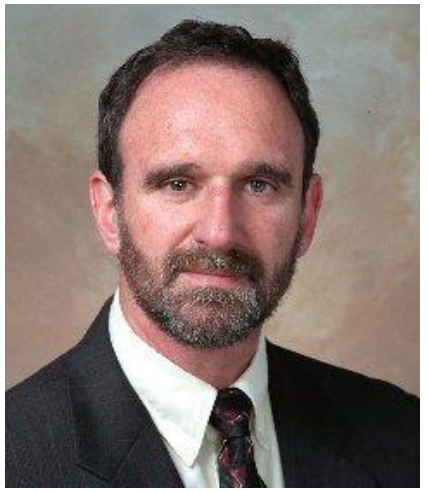
DHE_RSA Key Exchange Diskussion

- Vorteile
  - Bietet Forward Secrecy, d.h. passiver Angreifer kann in Zukunft selbst mit (N, d) die Session nicht mehr entschlüsseln
- Nachteile
  - Rechenintensiv
  - Es kann vorkommen, dass Forward Secrecy durch unsichere DH-Parameter eingeschränkt wird

Diffie-Hellman Schlüsselaustausch

Whitfield

Martin



Wählt $x < p - 1$

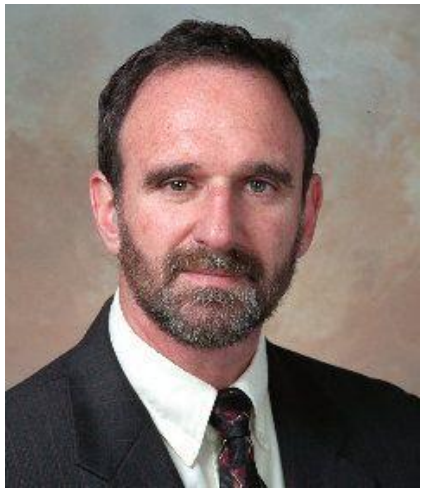
Wählt $y < p - 1$

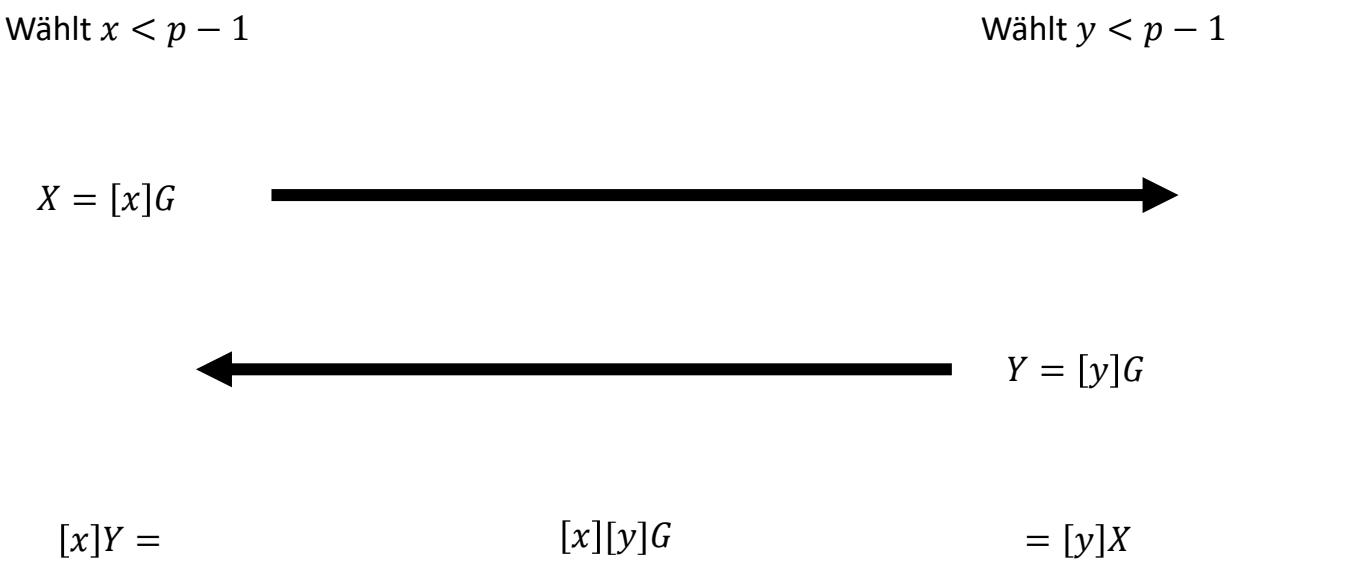
$X = g^x \ mod \ p$ →

← $Y = g^y \ mod \ p$

$Y^x \ mod \ p = \quad g^{xy} \ mod \ p \quad = X^y \ mod \ p$

Diffie-Hellman Schlüsselaustausch (ECC)





ECDHE_RSA und ECDHE_ECDSA

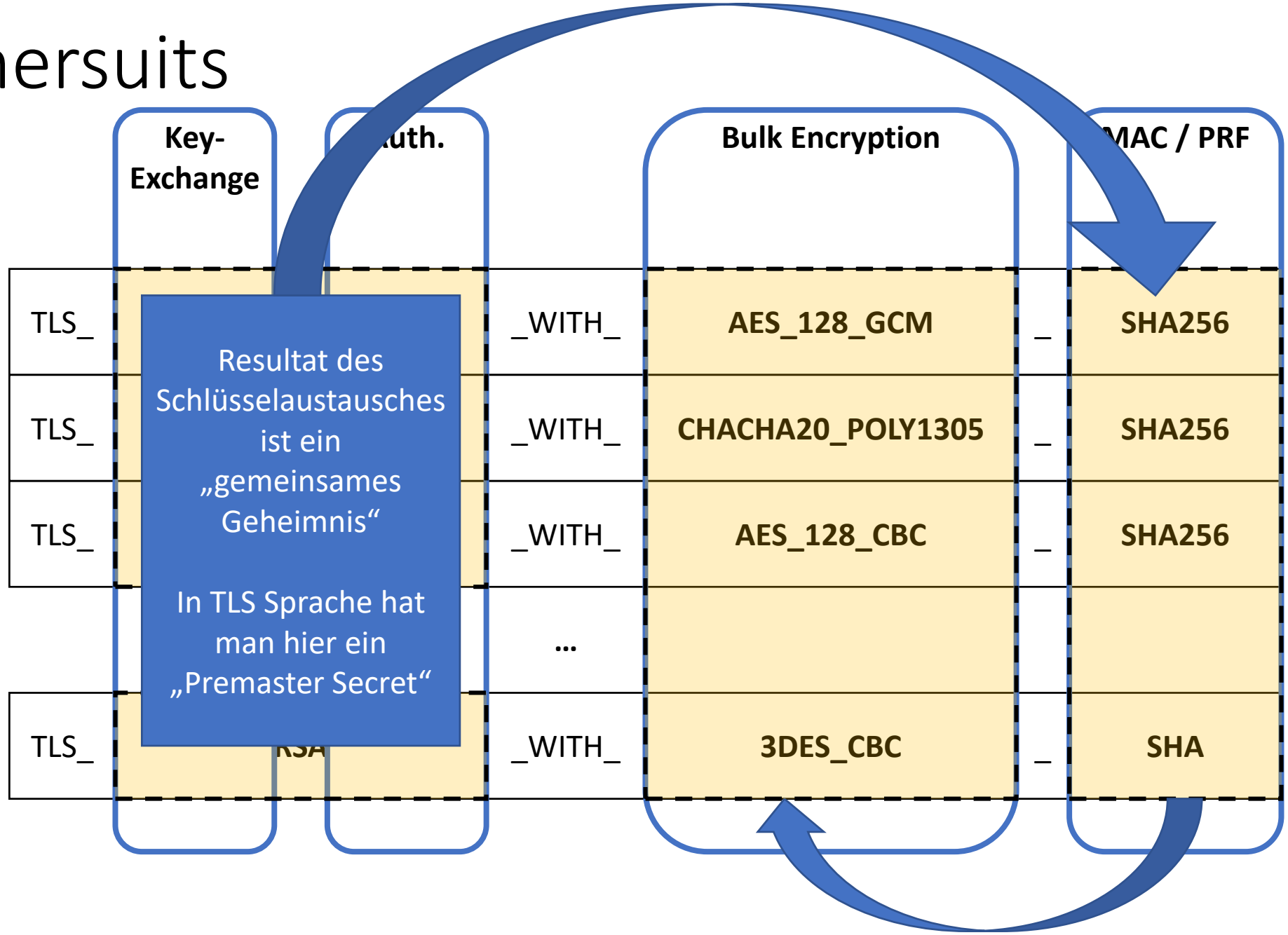
- *ECDHE* ist Diffie-Hellman-Schlüsselaustausch über Elliptic Curve-Cryptography (ECC)

- ECDSA bedeutet, dass die Authentifikation der Verbindung über ECC *Signaturen* (ECDSA) sichergestellt wird.

Kurze Zusammenfassung

- *TLS führt einen Diffie-Hellman Key-Exchange durch*
- *Der Key-Exchange wird durch eine Signatur authentifiziert*
- *Der öffentliche Schlüssel zur Prüfung der Signatur ist im Serverzertifikat enthalten*

Ciphersuits

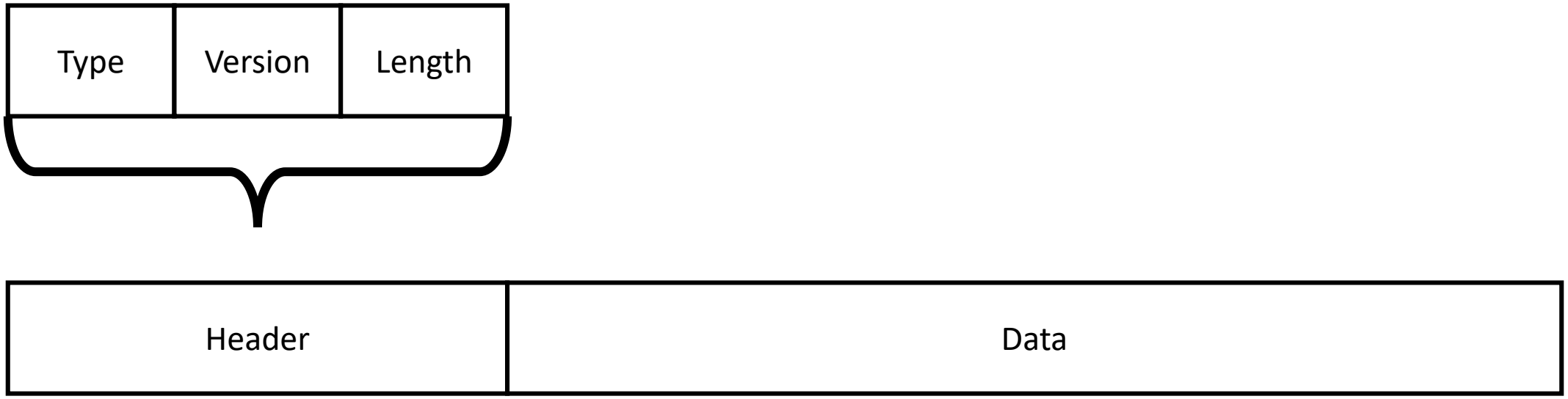


TLS Handshake im Detail

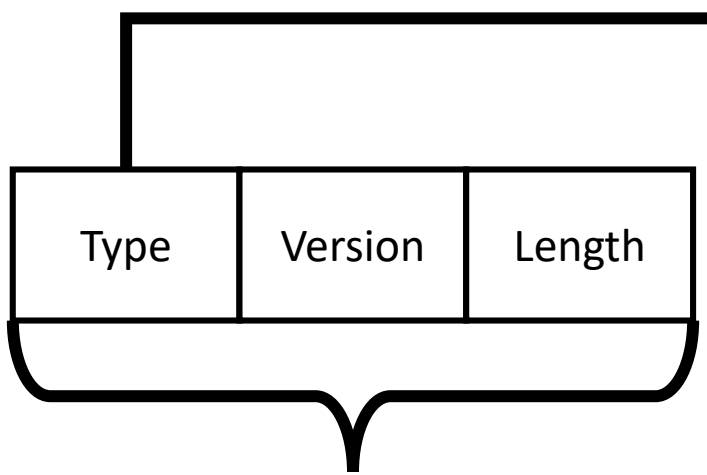
Aufbau

- TLS hat einen s.g. „Record Layer“ und Unterprotokolle
- Unterprotokolle sind:
  - Handshake
  - Change Cipher Spec.
  - Application Data
  - Alert
- Unter*protokoll* klingt komplizierter als es ist
  - Lies „Nachrichtengruppen“ + X
  - Change Cipher Spec. hat z.B. nur eine leere Signalisierungsnachricht

Record Layer



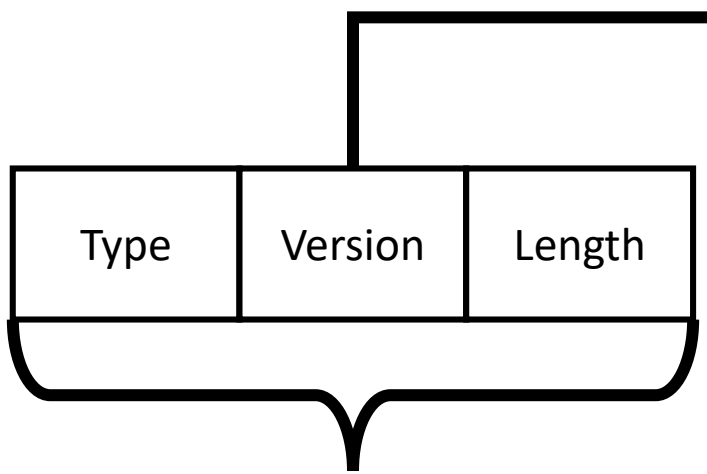
Record Layer



Type

- Bestimmt das Unterprotokoll
  - *Change Ciper Spec. (20)*
  - *Alert (21)*
  - *Handshake (22)*
  - *Application Data (23)*

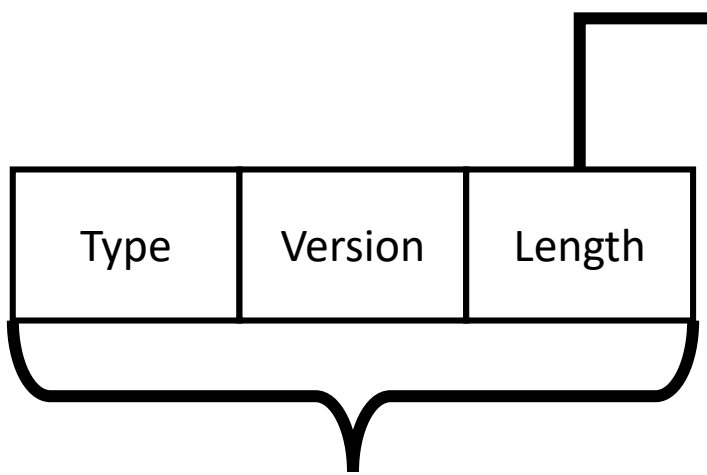
Record Layer



Version

- Major + Minor
- „Historisch gewachsen“
  - SSL 3.0 → 0x0300
  - TLS 1.0 → 0x0301
  - TLS 1.1 → 0x0302
  - …
- Praxis: „MUST accept any value {03, XX}“

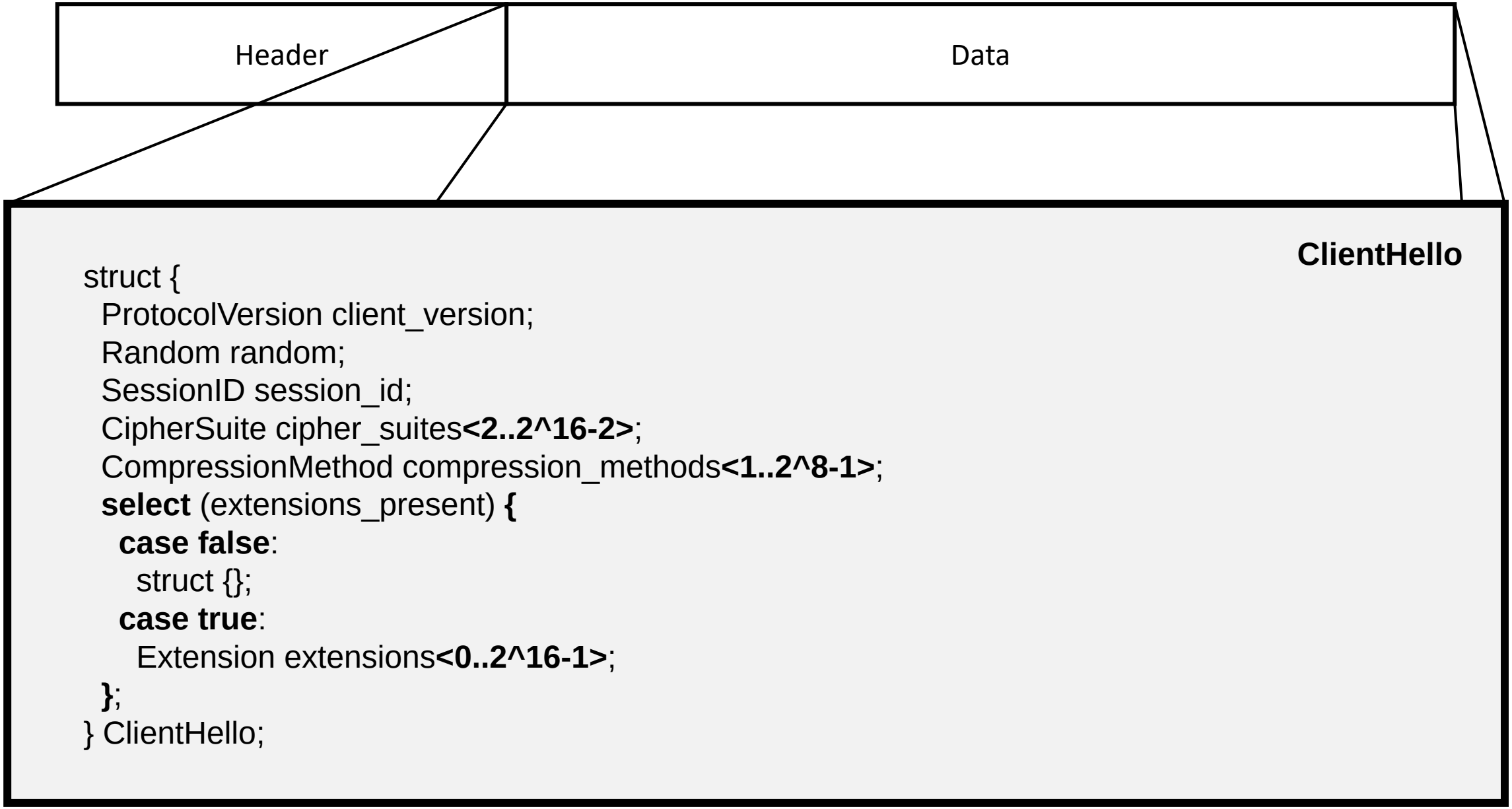
Record Layer



Length

- Maximale Länge eines *Records* beträgt 2^{14} Bytes
- Falls Nachrichten des Unterprotokolls länger sind, wird fragmentiert


Header
Data

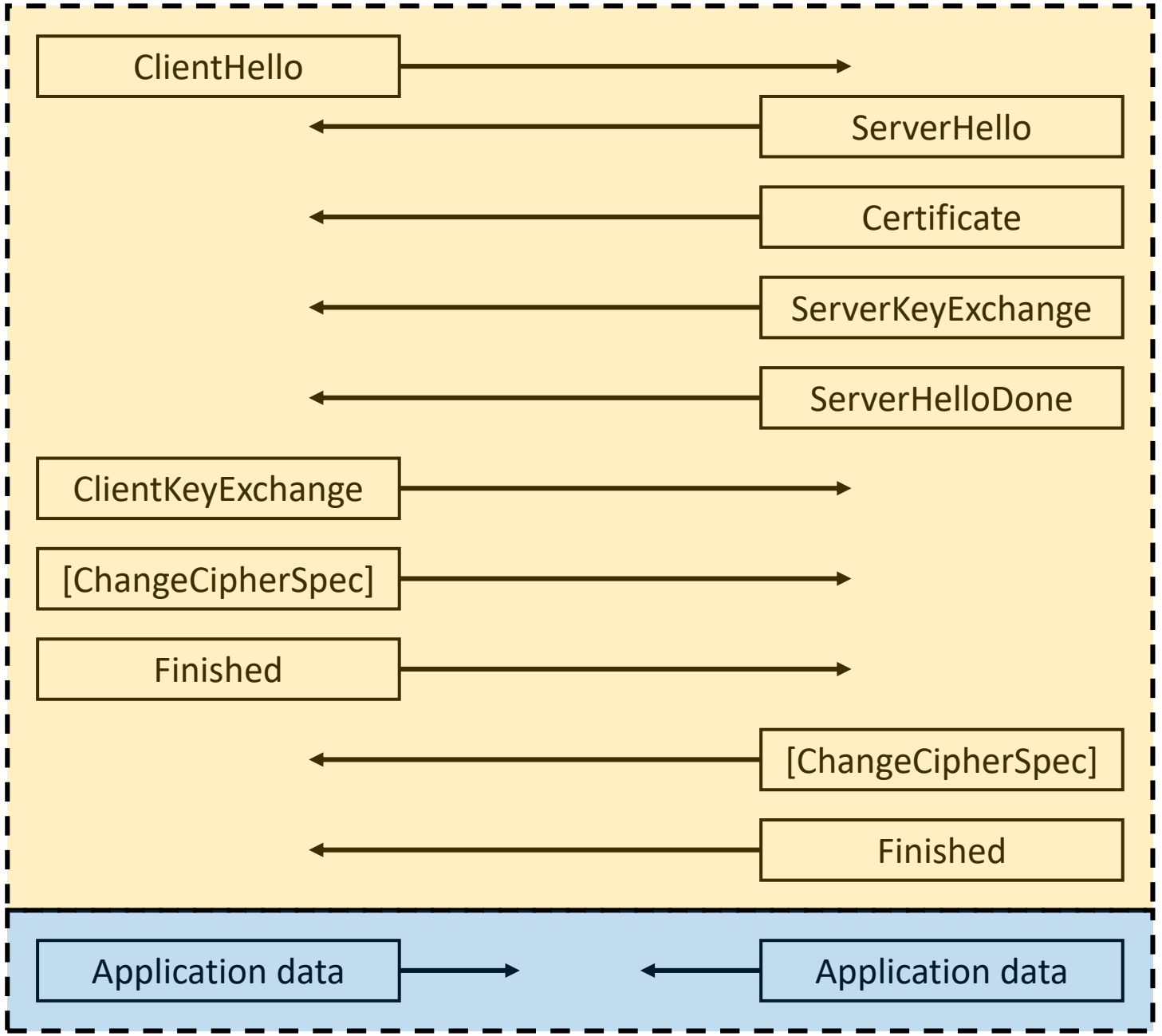

Header
Data
ClientHello
struct {
ProtocolVersion client_version;
Random random;
SessionID session_id;
CipherSuite cipher_suites<2..2^16-2>;
CompressionMethod compression_methods<1..2^8-1>;
select (extensions_present) {
case false:
struct {};
case true:
Extension extensions<0..2^16-1>;
};
} ClientHello;

Das TLS 1.2 Protokoll

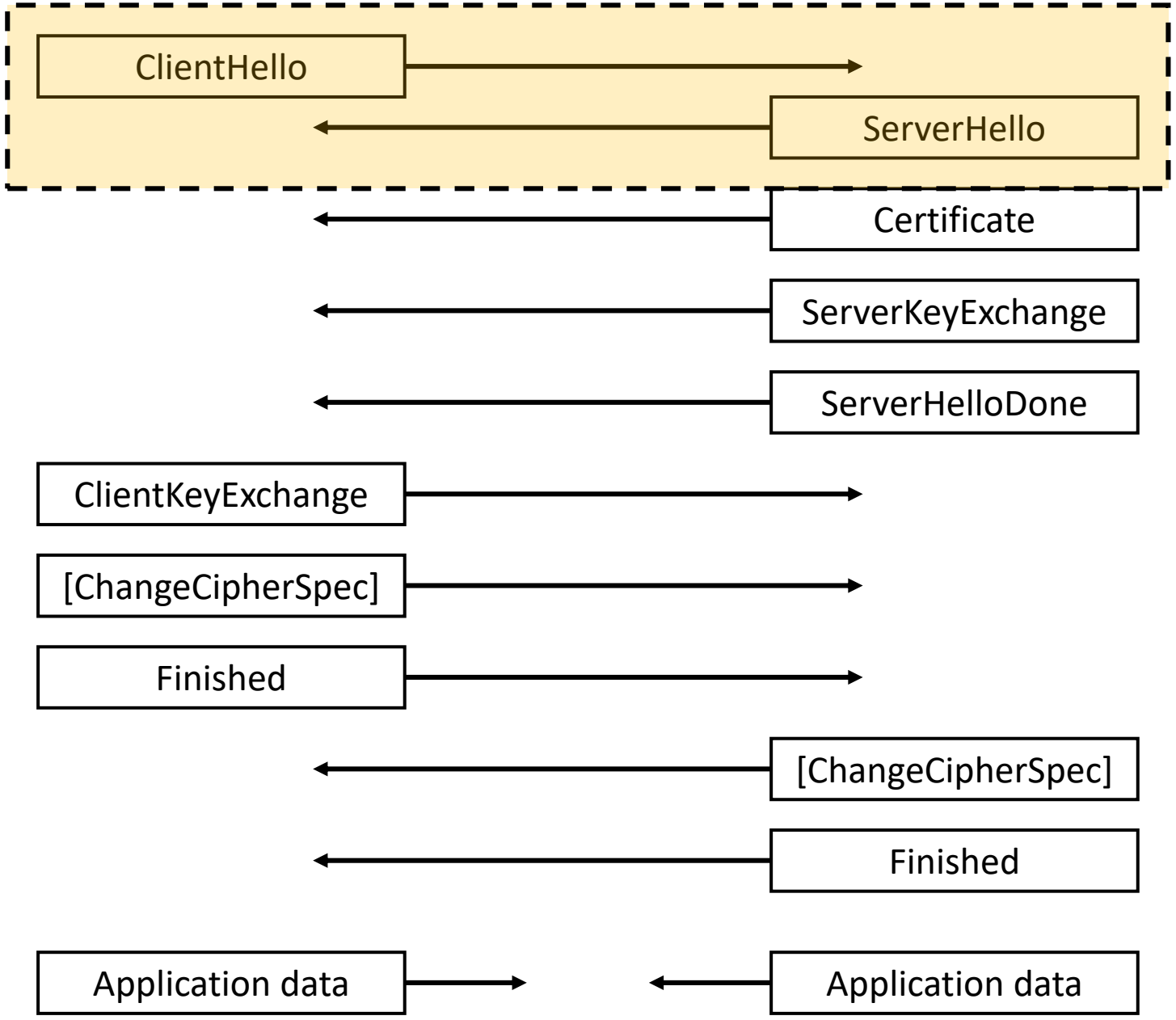
Drei gängige Workflows

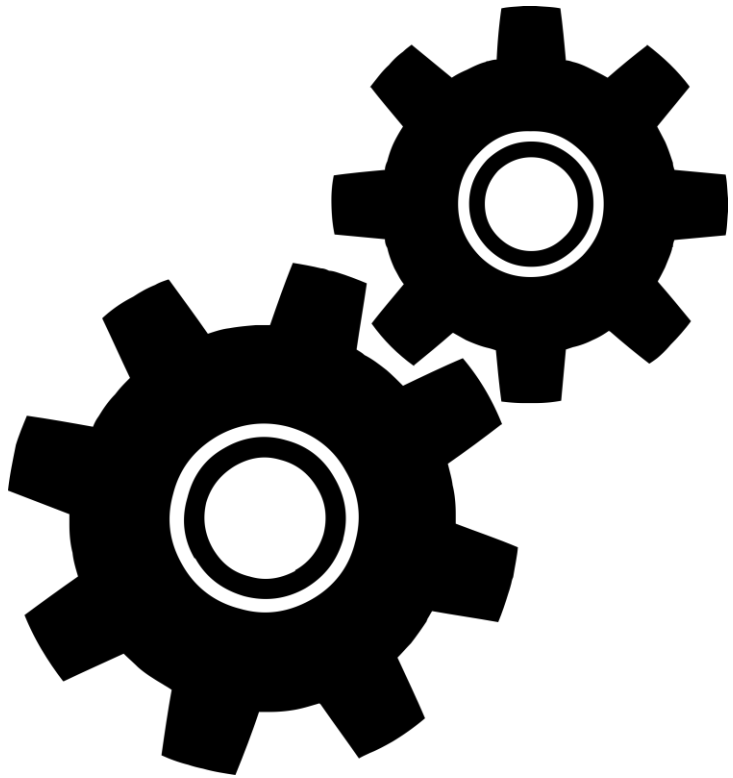
1. *Voller Handshake*
2. *Kurzer Handshake* (führt vorige Session fort)
3. Client und Server authentifizieren sich gegenseitig

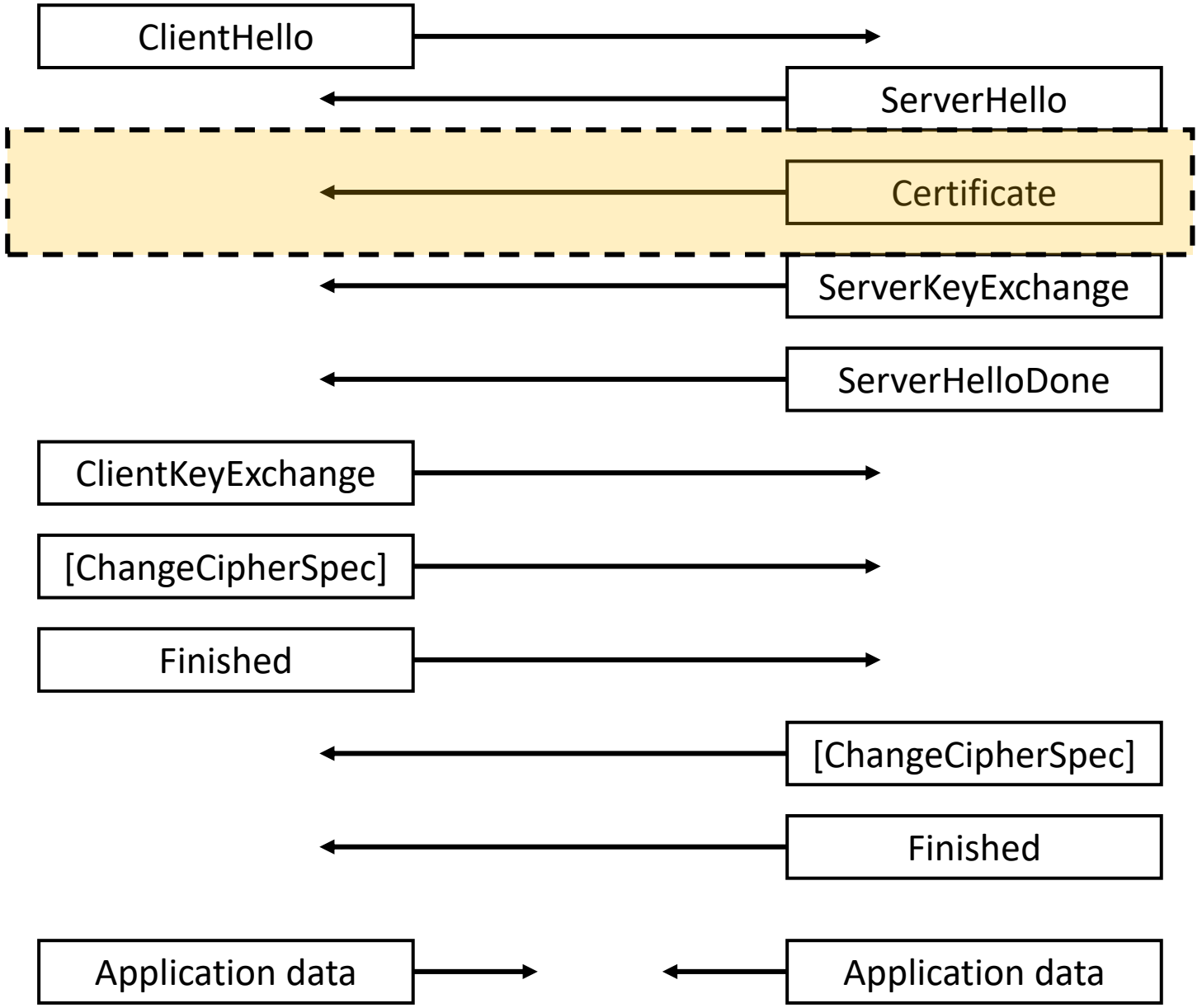
Handshake


ClientHello
ServerHello
Certificate
ServerKeyExchange
ServerHelloDone
ClientKeyExchange
[ChangeCipherSpec]
Finished
[ChangeCipherSpec]
Finished
Application data
Application data

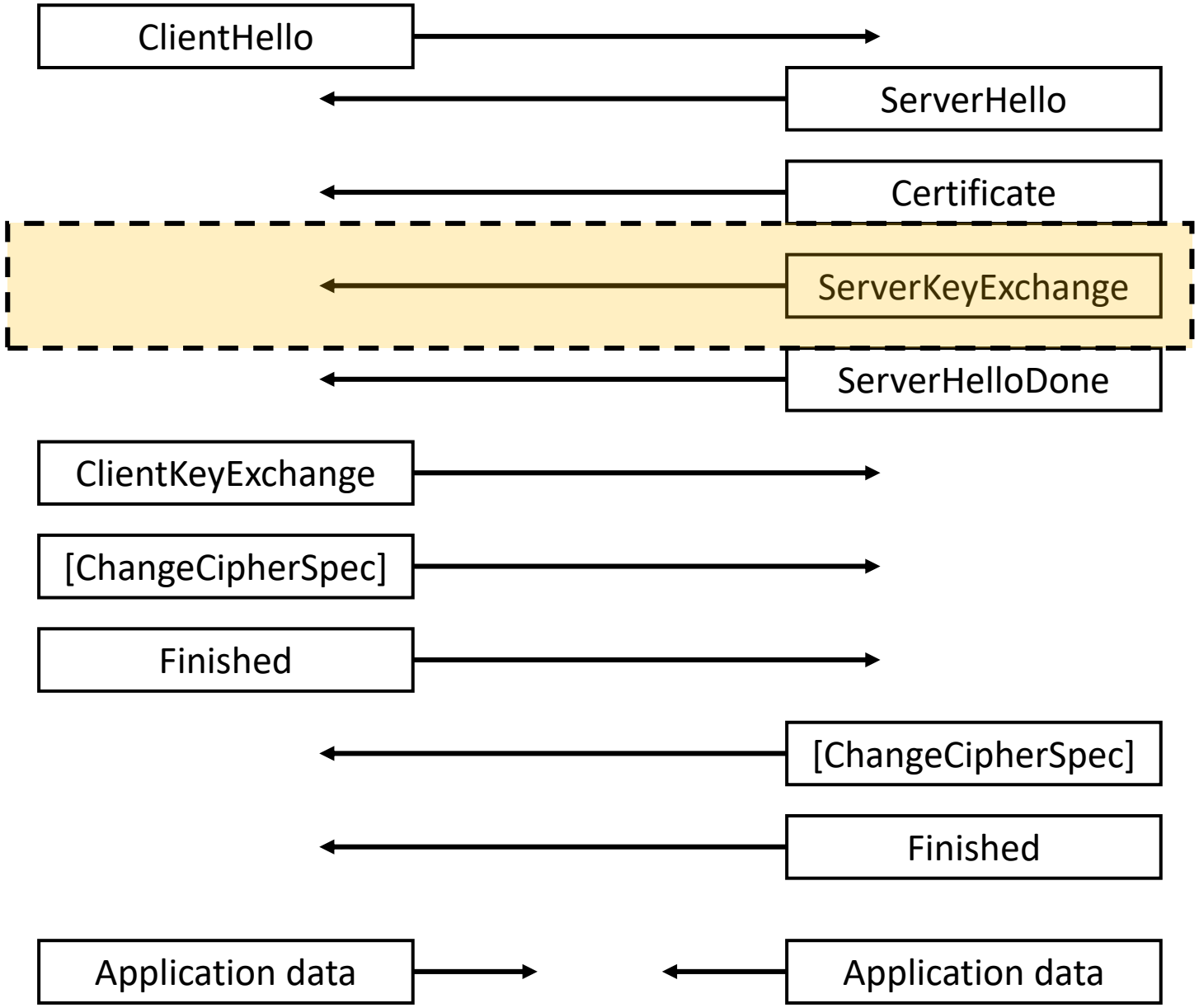
```
struct {
  ProtocolVersion server_version;
  Random random;
  SessionID session_id;
  CipherSuite cipher_suite;
  CompressionMethod
compression_method;
  select (extensions_present) {
    case false:
      struct {};
    case true:
      Extension extensions<0..2^16-1>;
  };
} ServerHello;
```



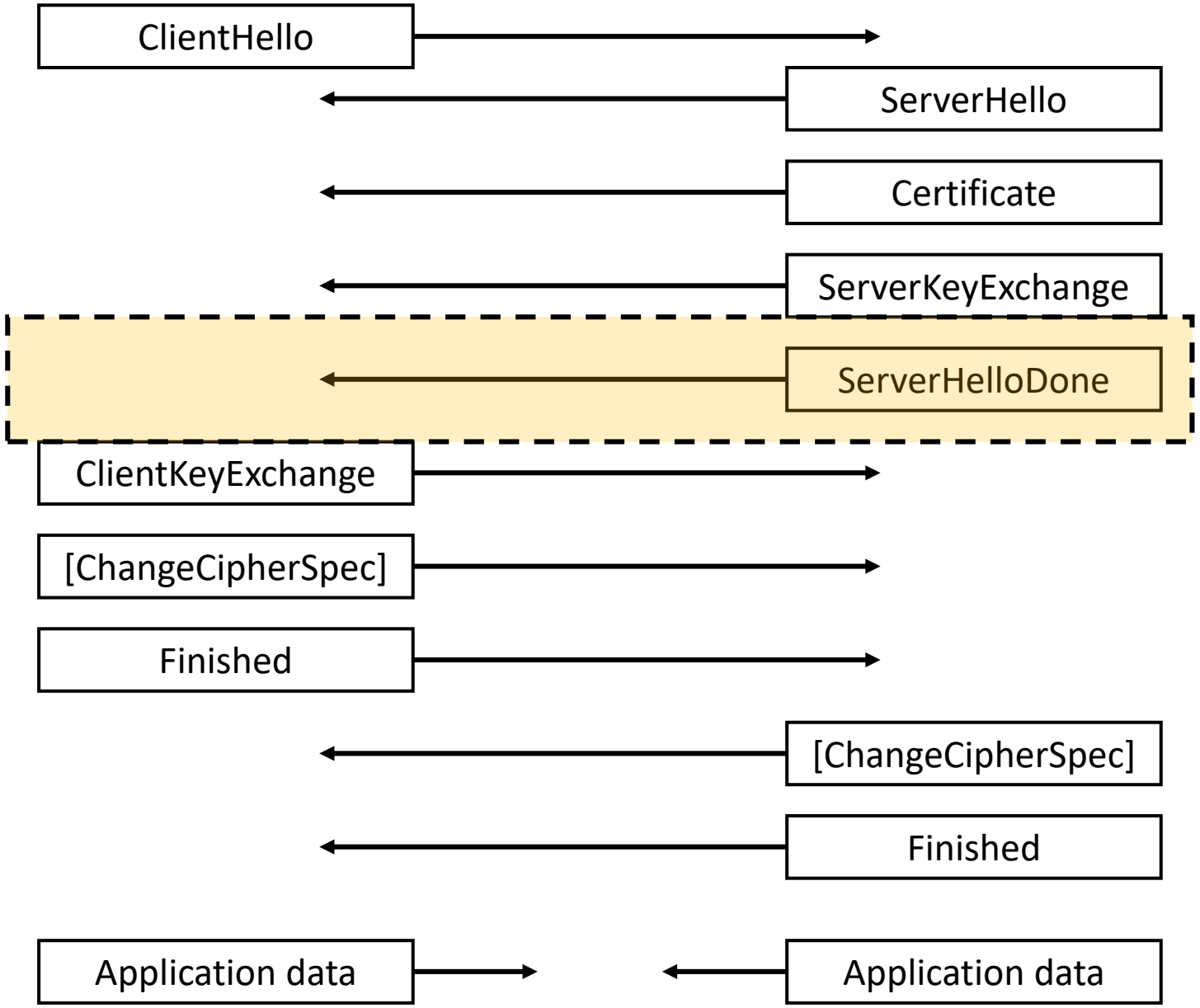



ClientHello
ServerHello
Certificate
ServerKeyExchange
ServerHelloDone
ClientKeyExchange
[ChangeCipherSpec]
Finished
[ChangeCipherSpec]
Finished
Application data
Application data

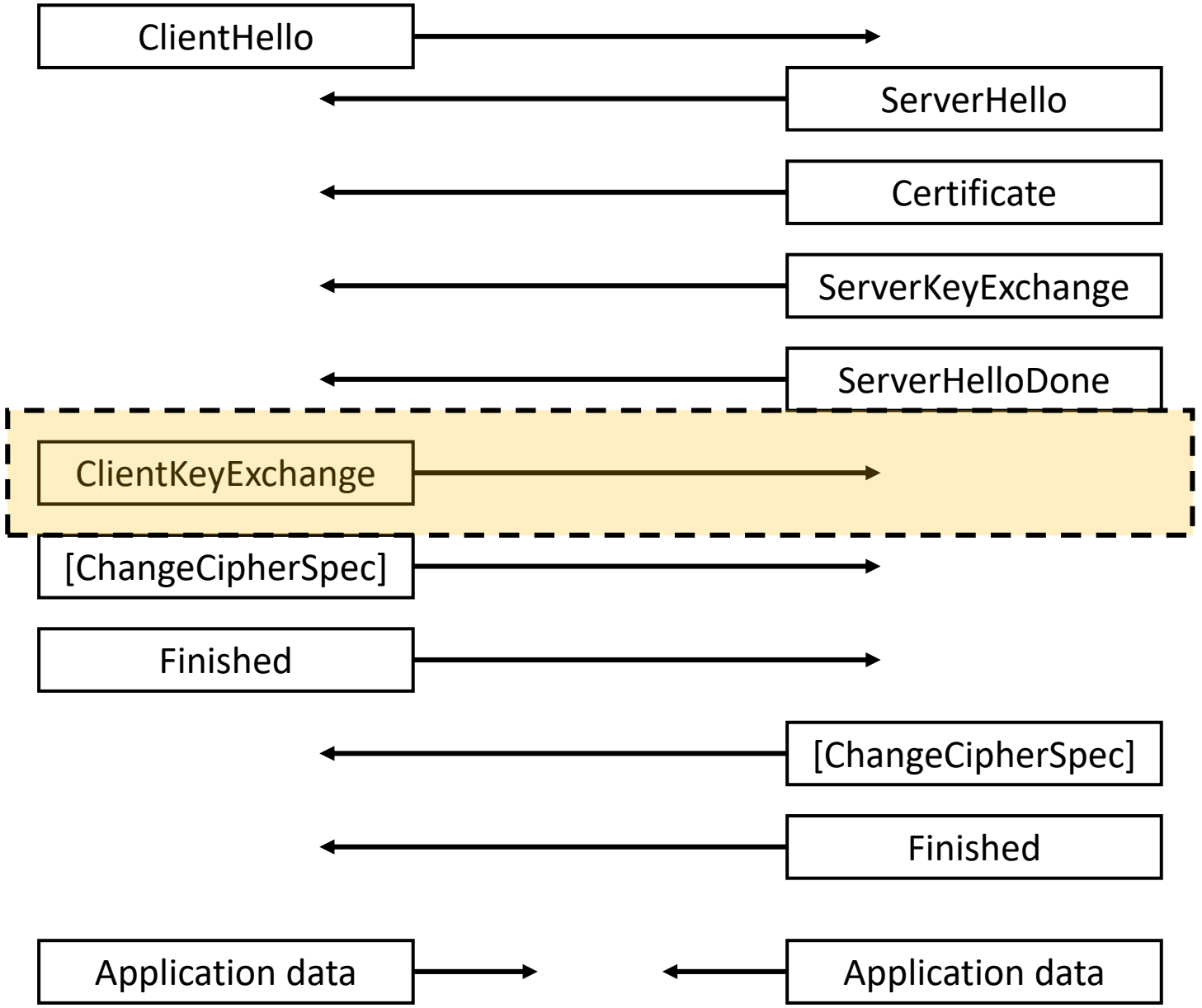
```
struct {
  select (KeyExchangeAlgorithm) {
    case dhe_rsa:
      ServerDHParams params;
      digitally-signed struct {
        opaque client_random[32];
        opaque server_random[32];
        ServerDHParams params;
      } signed_params;
    case ...:
      ...
  };
} ServerKeyExchange;
```

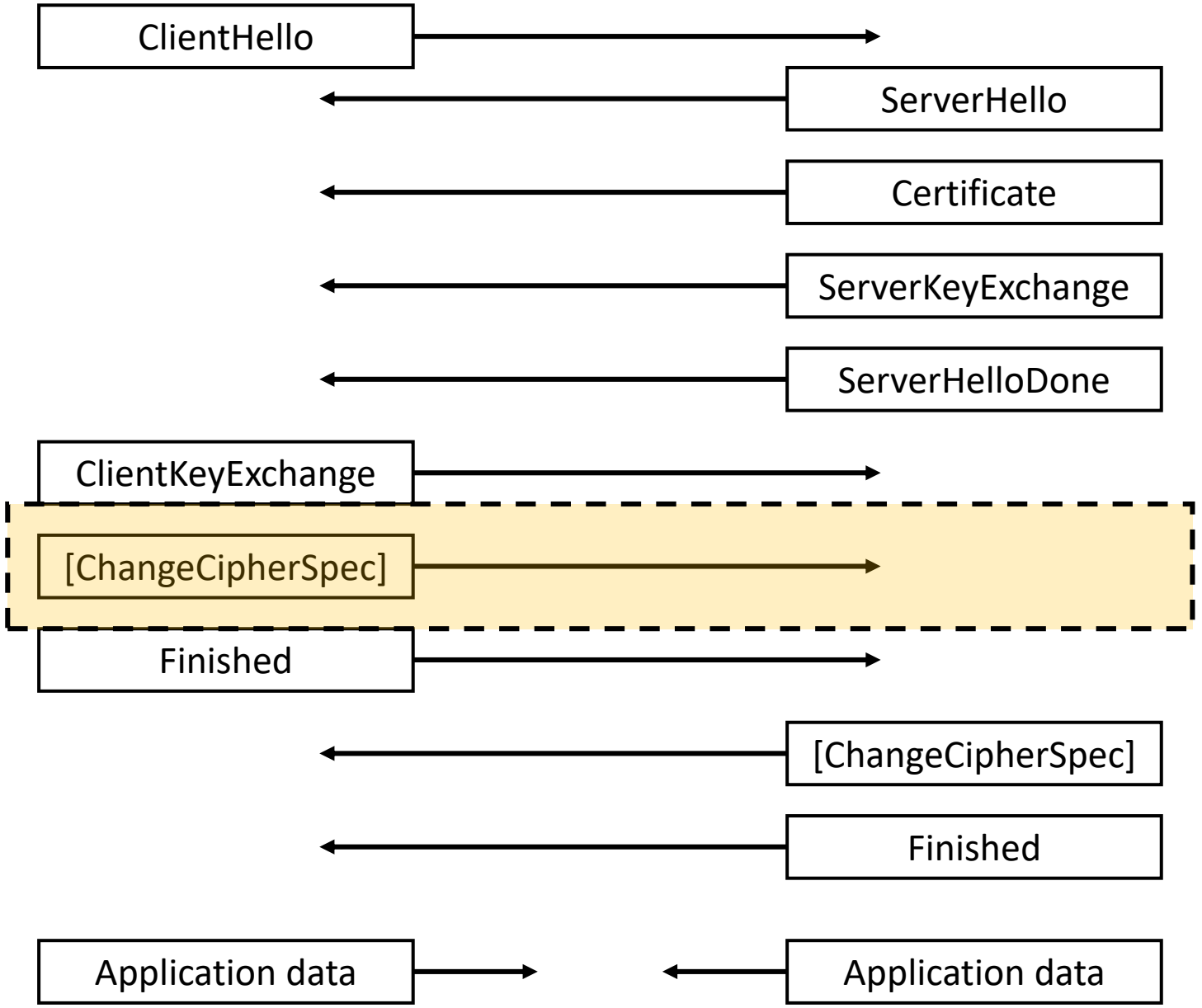


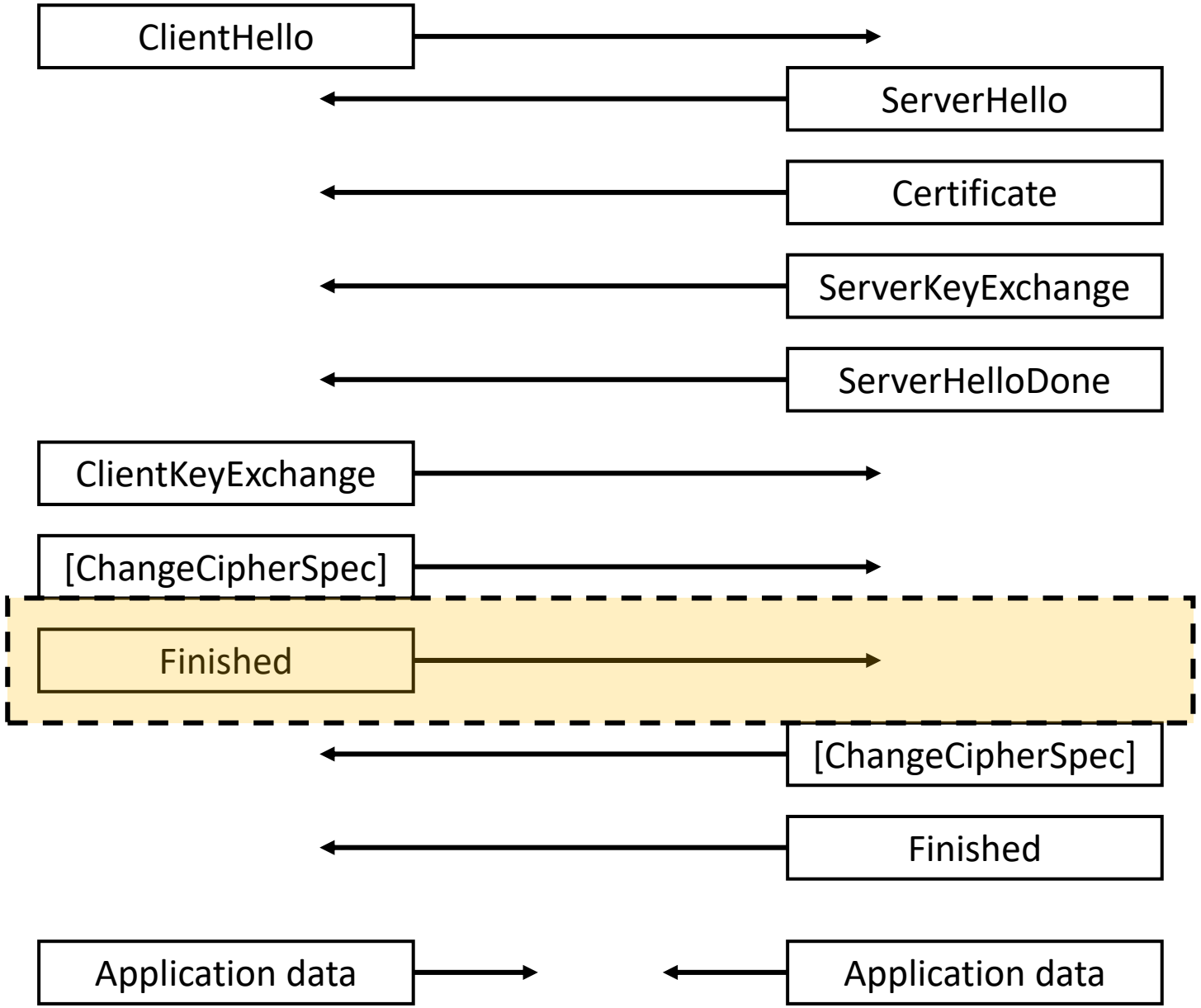
struct { } ServerHelloDone;

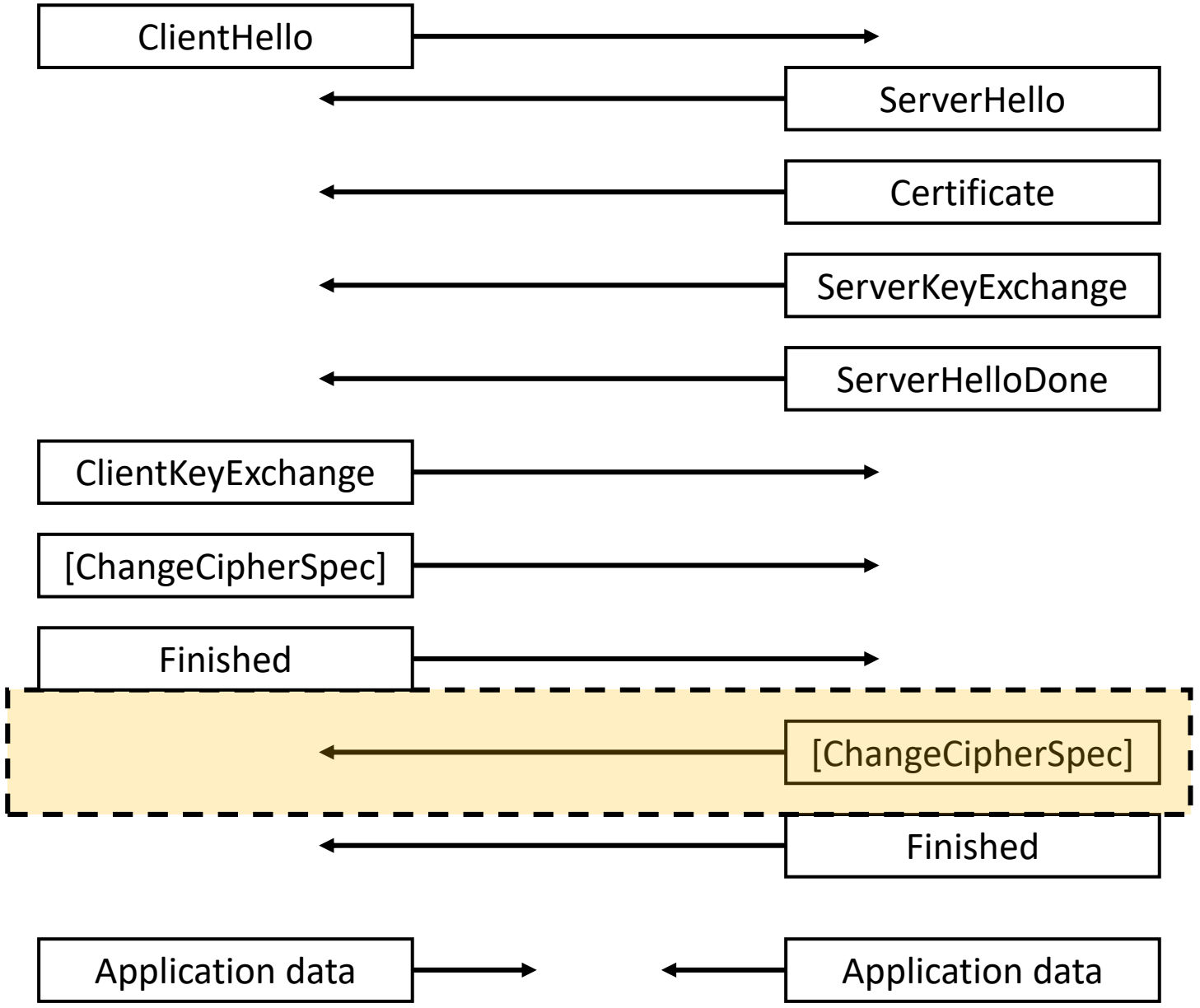


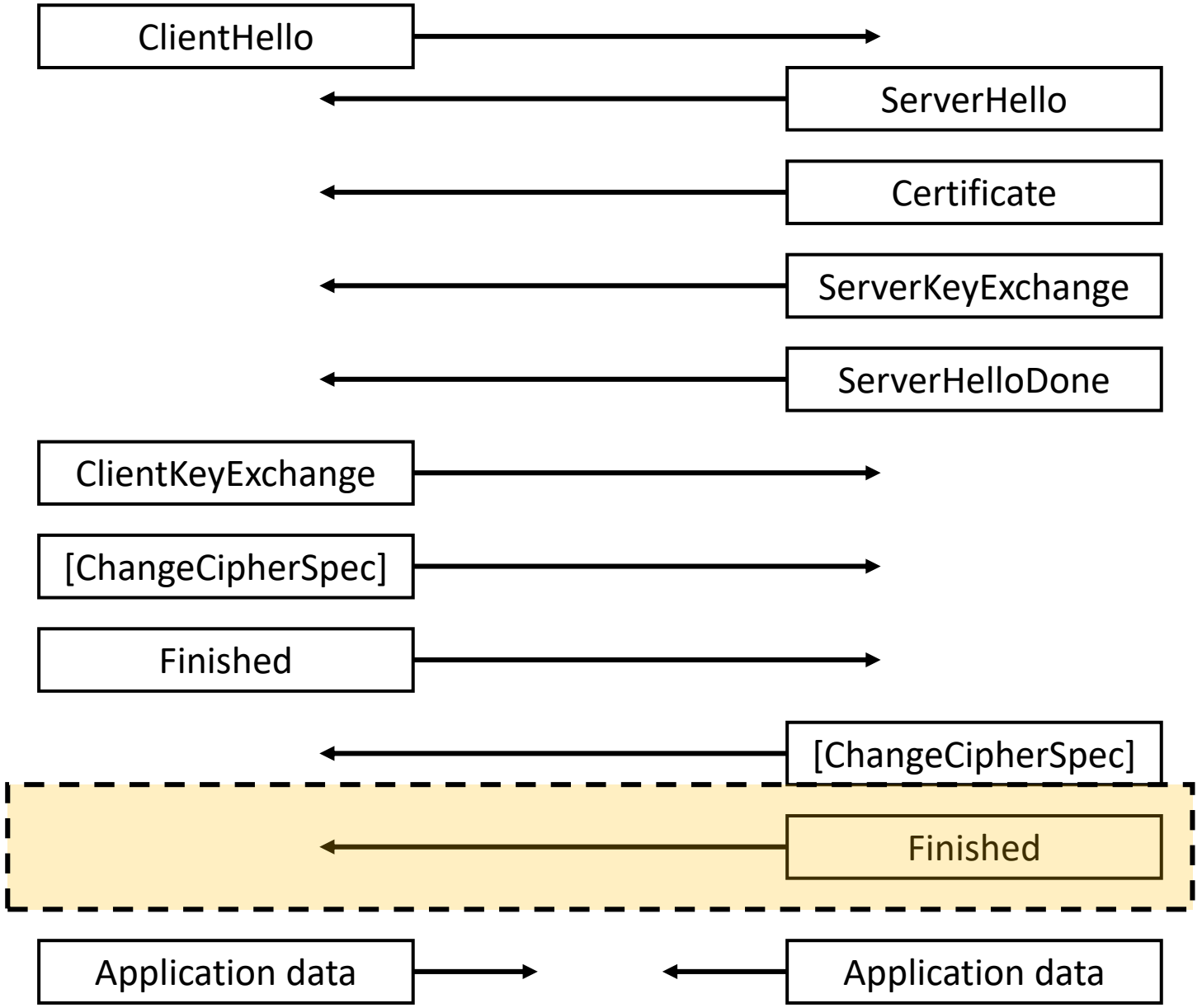
```
struct {
  select (KeyExchangeAlgorithm) {
    case rsa:
      EncryptedPreMasterSecret;
    case dhe_rsa:
      ClientDiffieHellmanPublic;
    case ...:
      ...
  } exchange_keys;
} ClientKeyExchange;
```




ClientHello
ServerHello
Certificate
ServerKeyExchange
ServerHelloDone
ClientKeyExchange
[ChangeCipherSpec]
Finished
[ChangeCipherSpec]
Finished
Application data
Application data


ClientHello
ServerHello
Certificate
ServerKeyExchange
ServerHelloDone
ClientKeyExchange
[ChangeCipherSpec]
Finished
[ChangeCipherSpec]
Finished
Application data
Application data

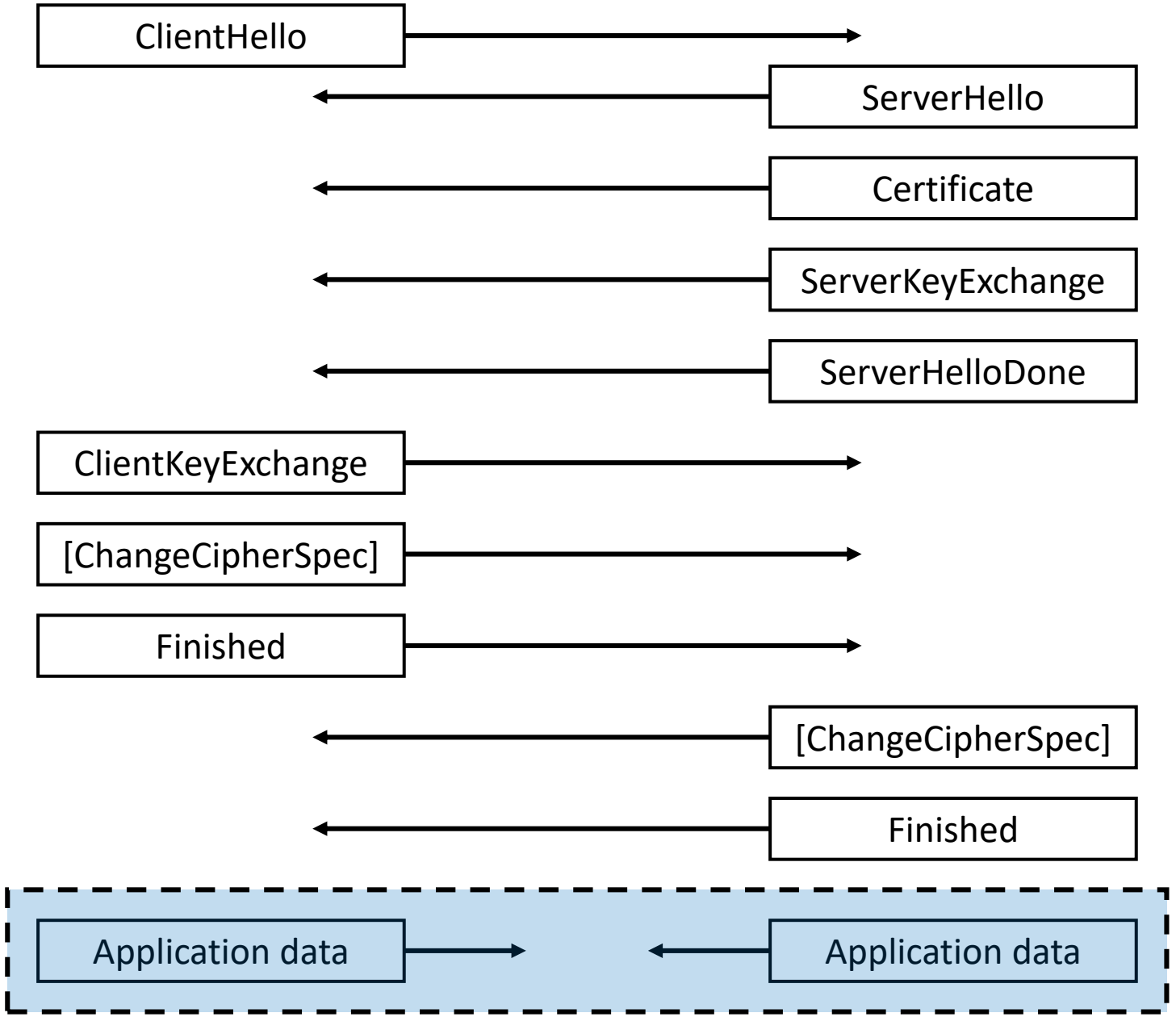

ClientHello
ServerHello
Certificate
ServerKeyExchange
ServerHelloDone
ClientKeyExchange
[ChangeCipherSpec]
Finished
[ChangeCipherSpec]
Finished
Application data
Application data


ClientHello
ServerHello
Certificate
ServerKeyExchange
ServerHelloDone
ClientKeyExchange
[ChangeCipherSpec]
Finished
[ChangeCipherSpec]
Finished
Application data
Application data

1. Der Server hat sich authentifiziert (Zertifikat)

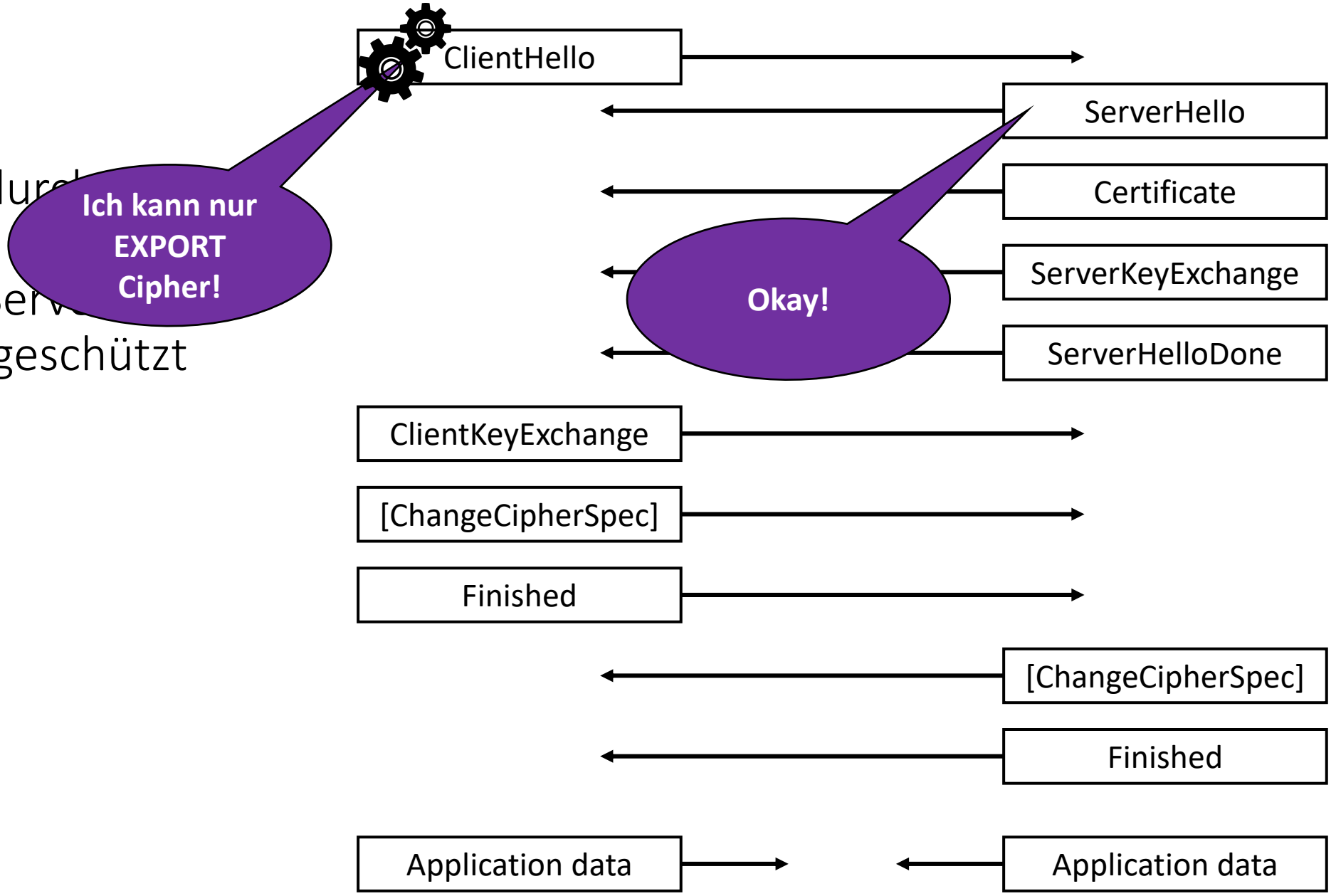
2. Ein gemeinsames Geheimnis wurde (sicher) ausgetauscht (Diffie-Hellman)

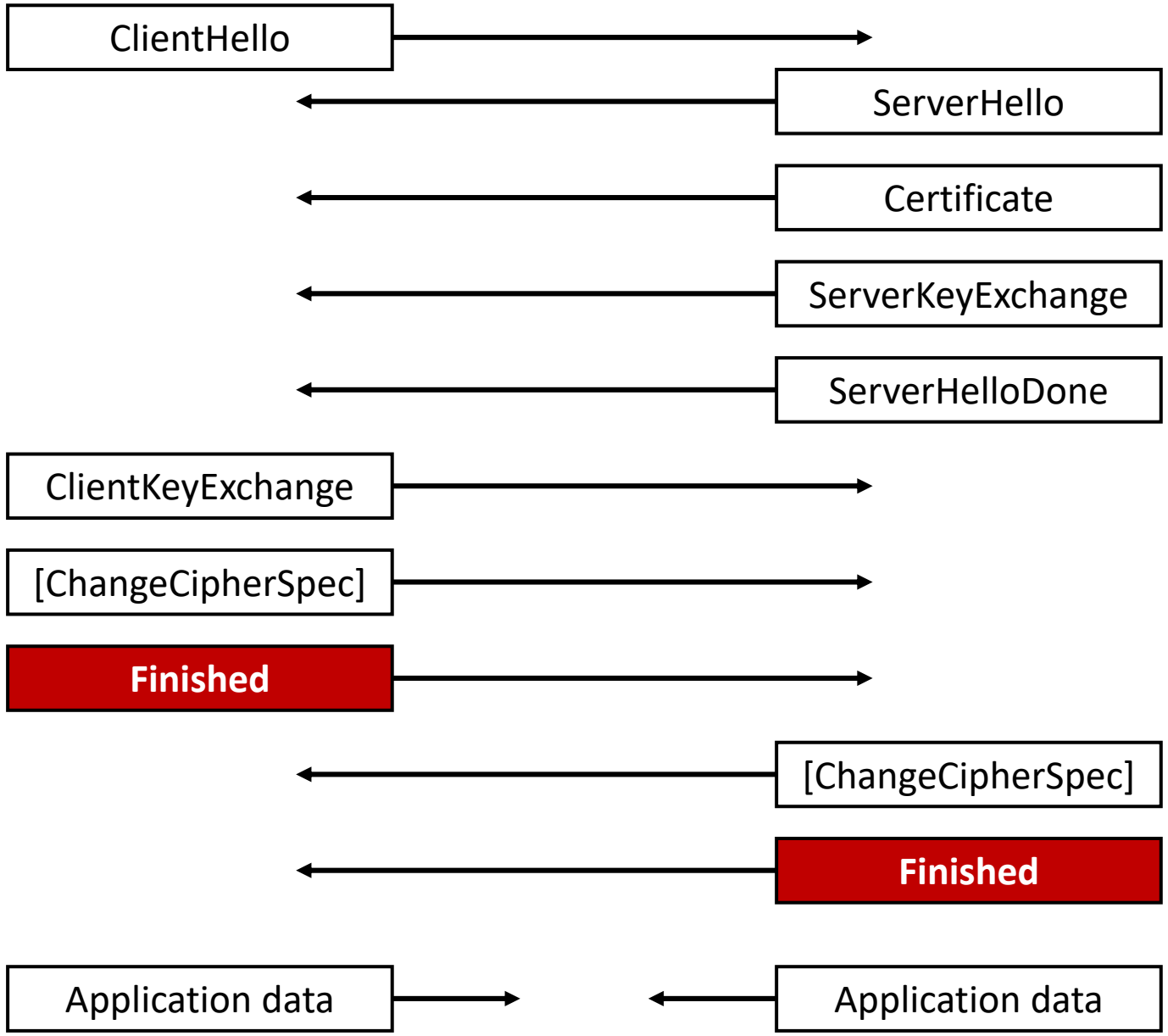
3. Beide Kommunikationspartner sind im Status „Verschlüsselung eingeschaltet“

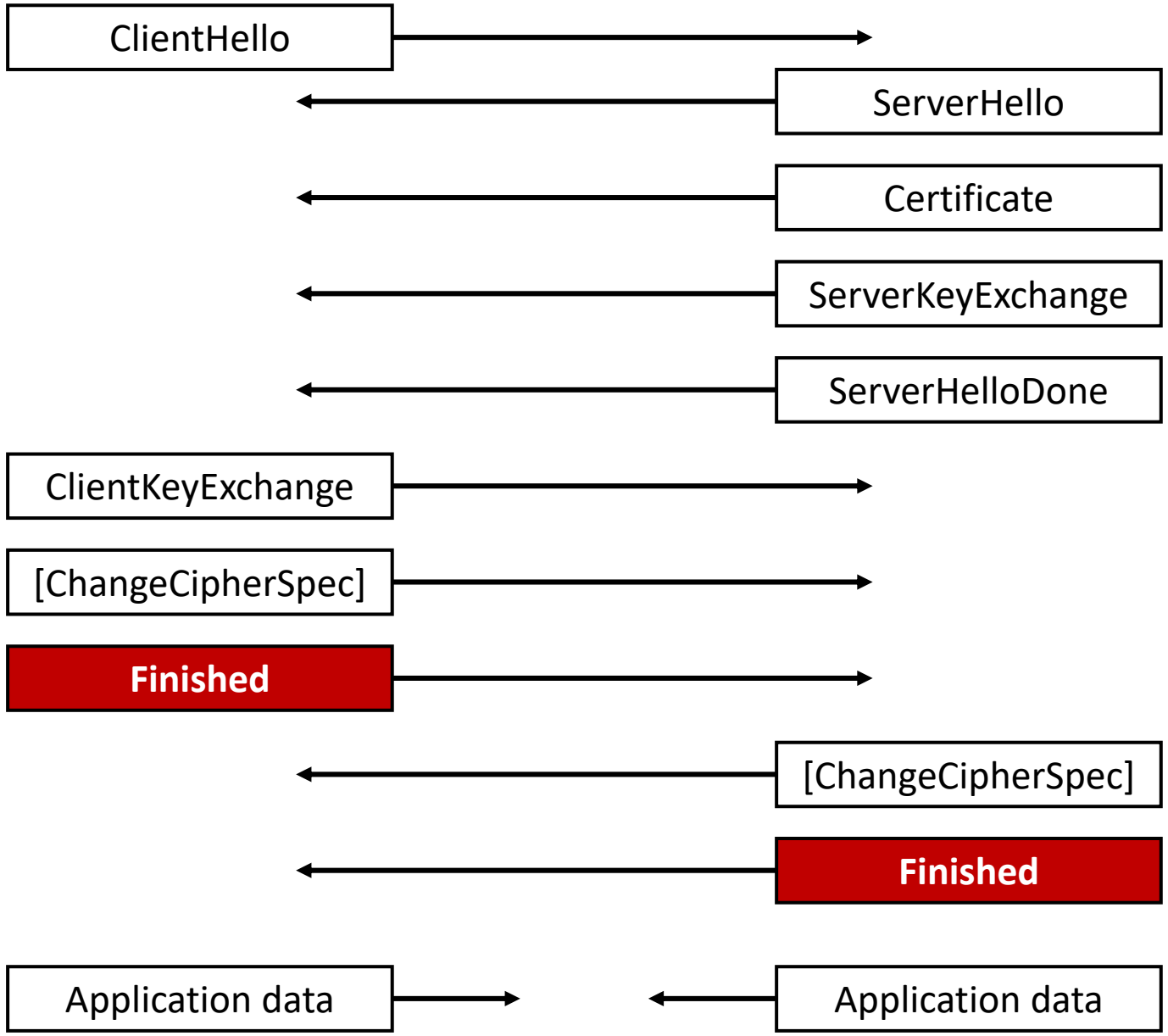


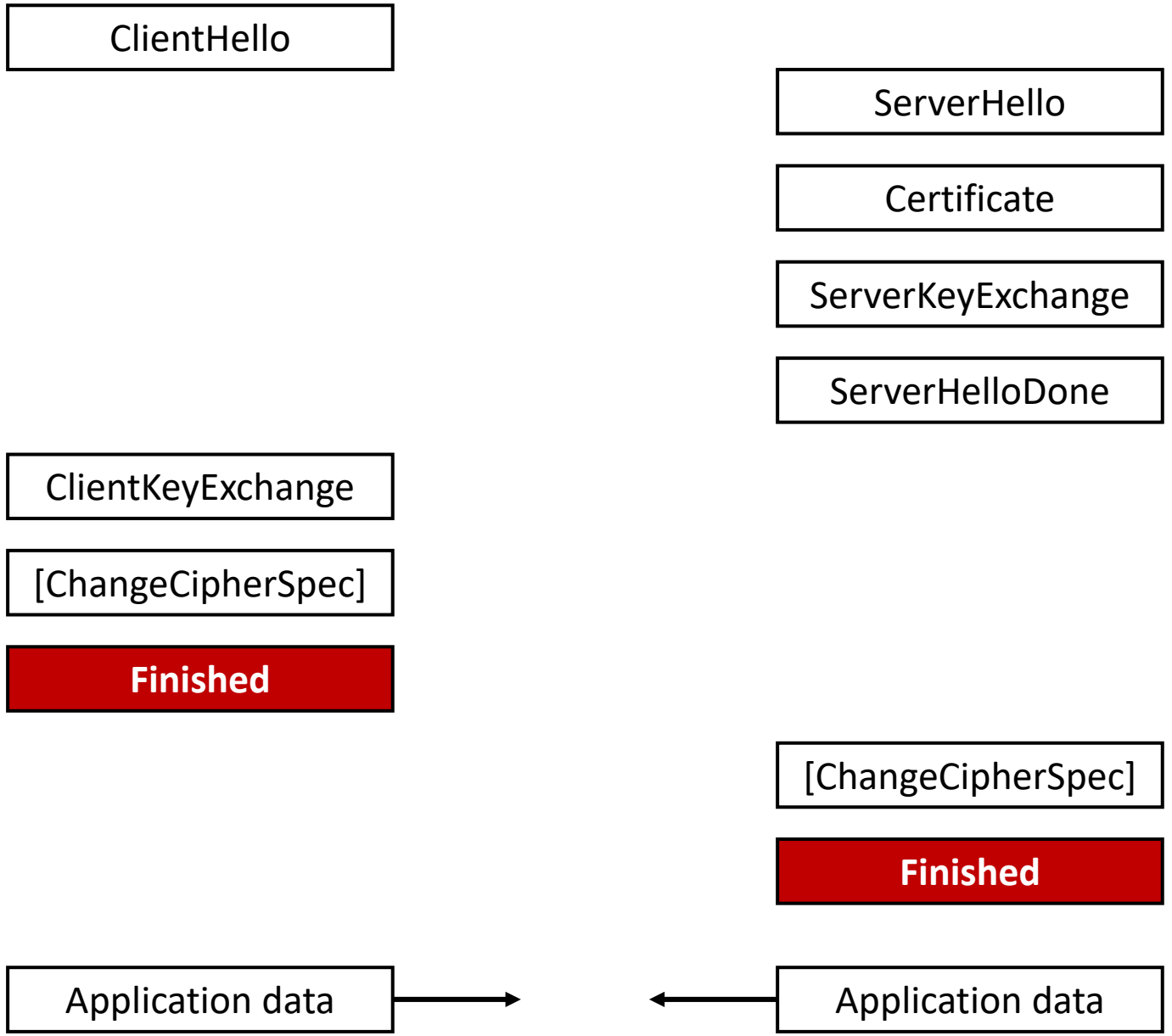
Problem:

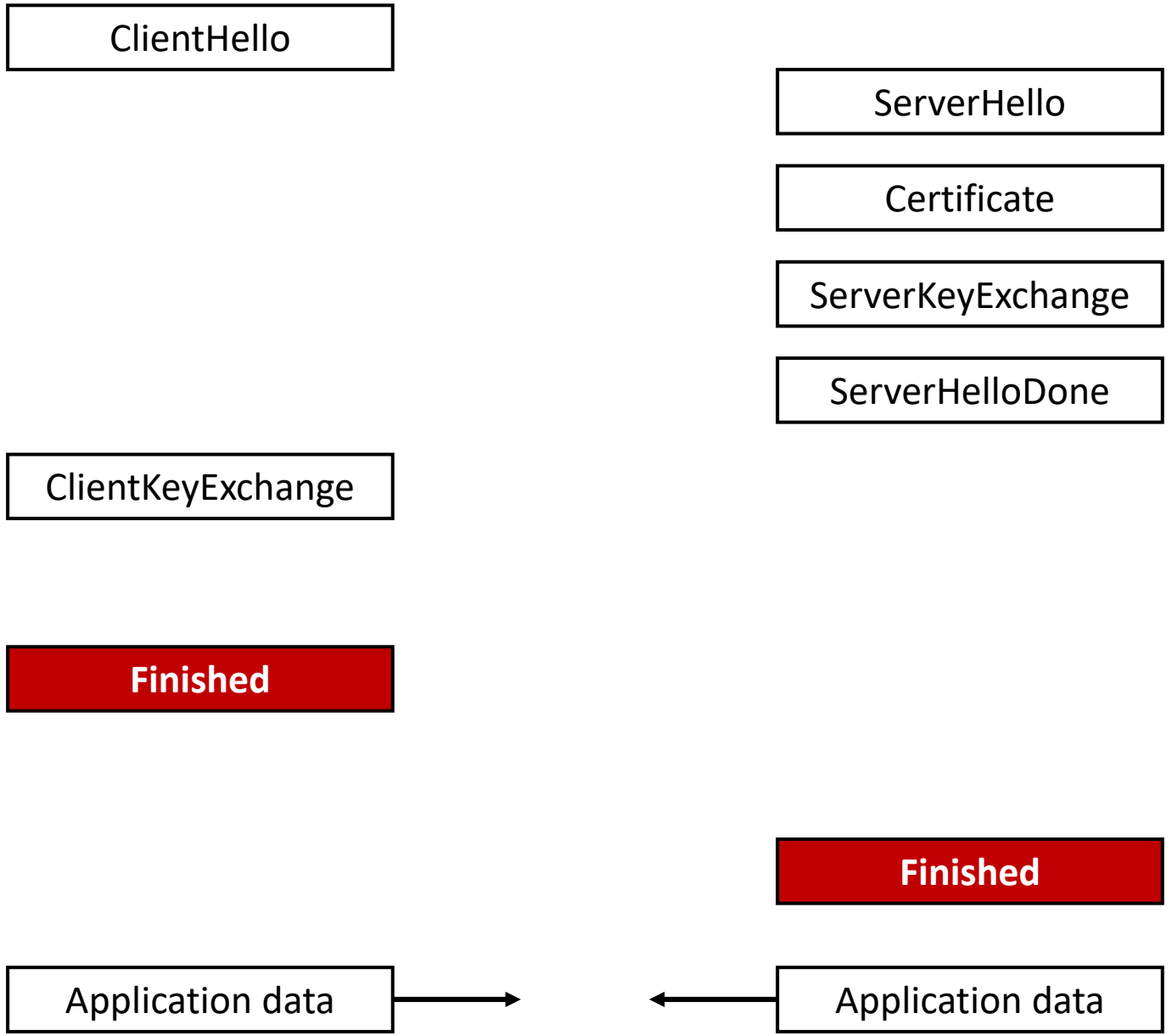
- Downgrade durch MitM?
- ClientHello, Server… etc. völlig ungeschützt

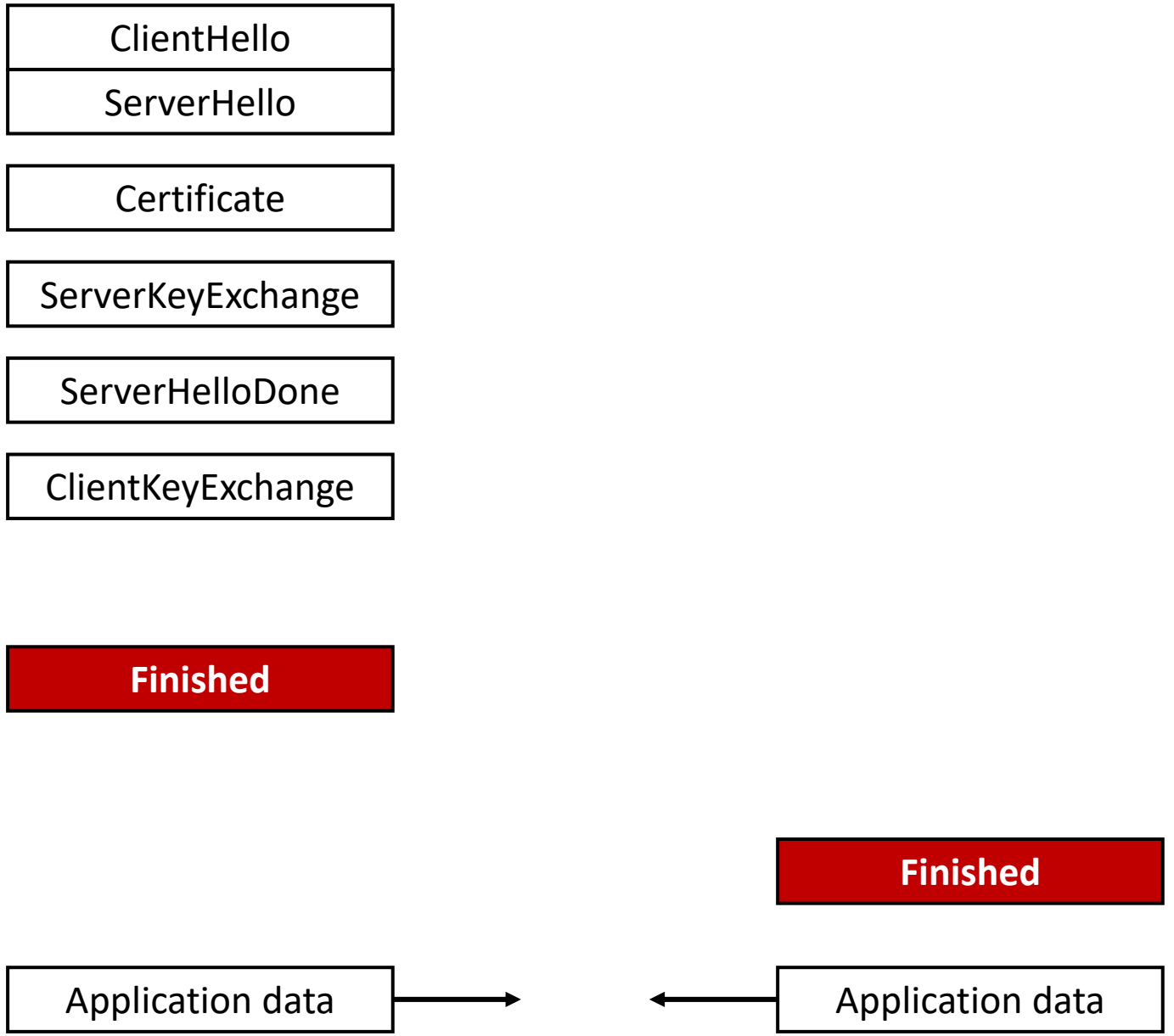


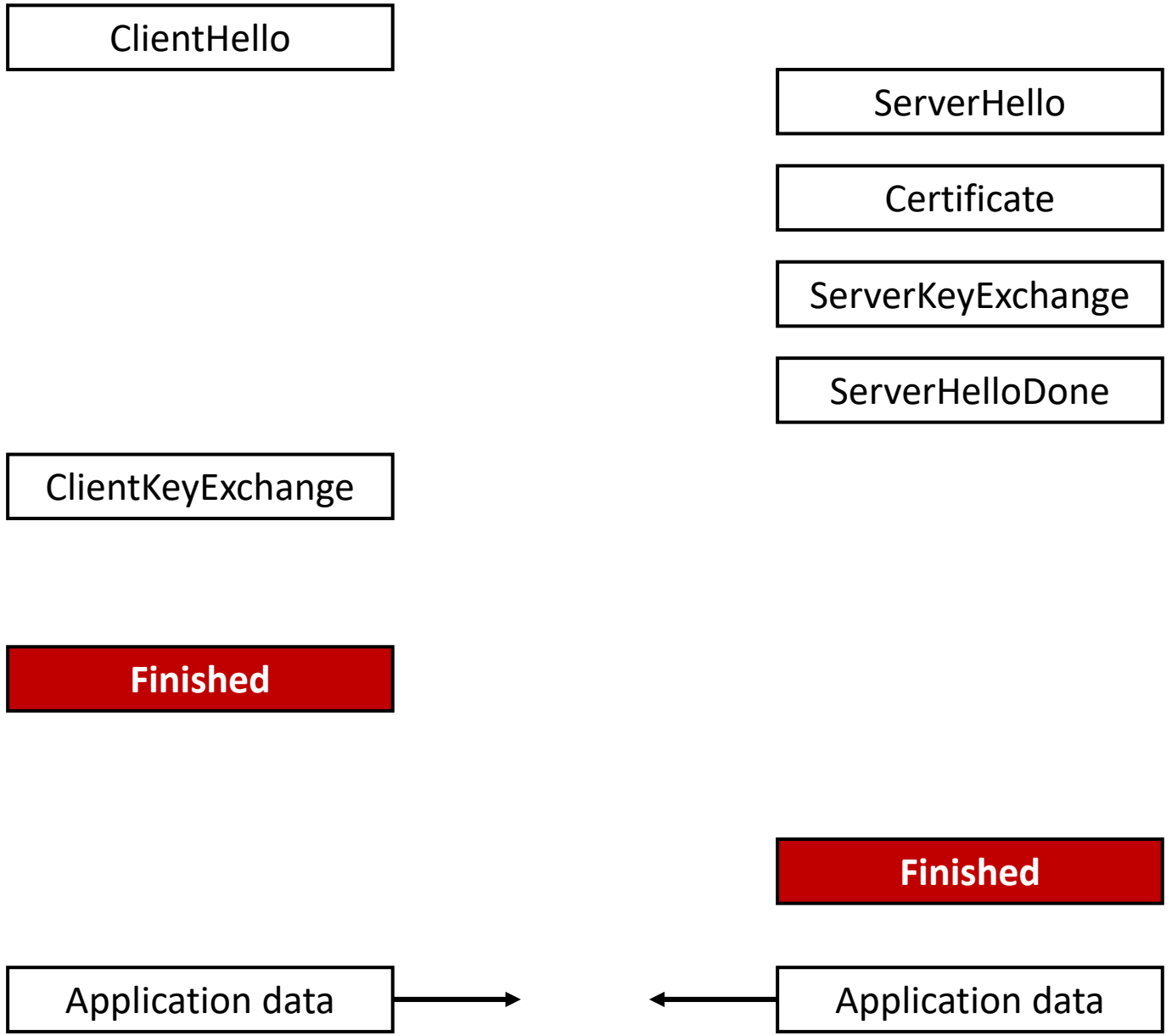

ClientHello
ServerHello
Certificate
ServerKeyExchange
ServerHelloDone
ClientKeyExchange
[ChangeCipherSpec]
Finished
[ChangeCipherSpec]
Finished
Application data
Application data

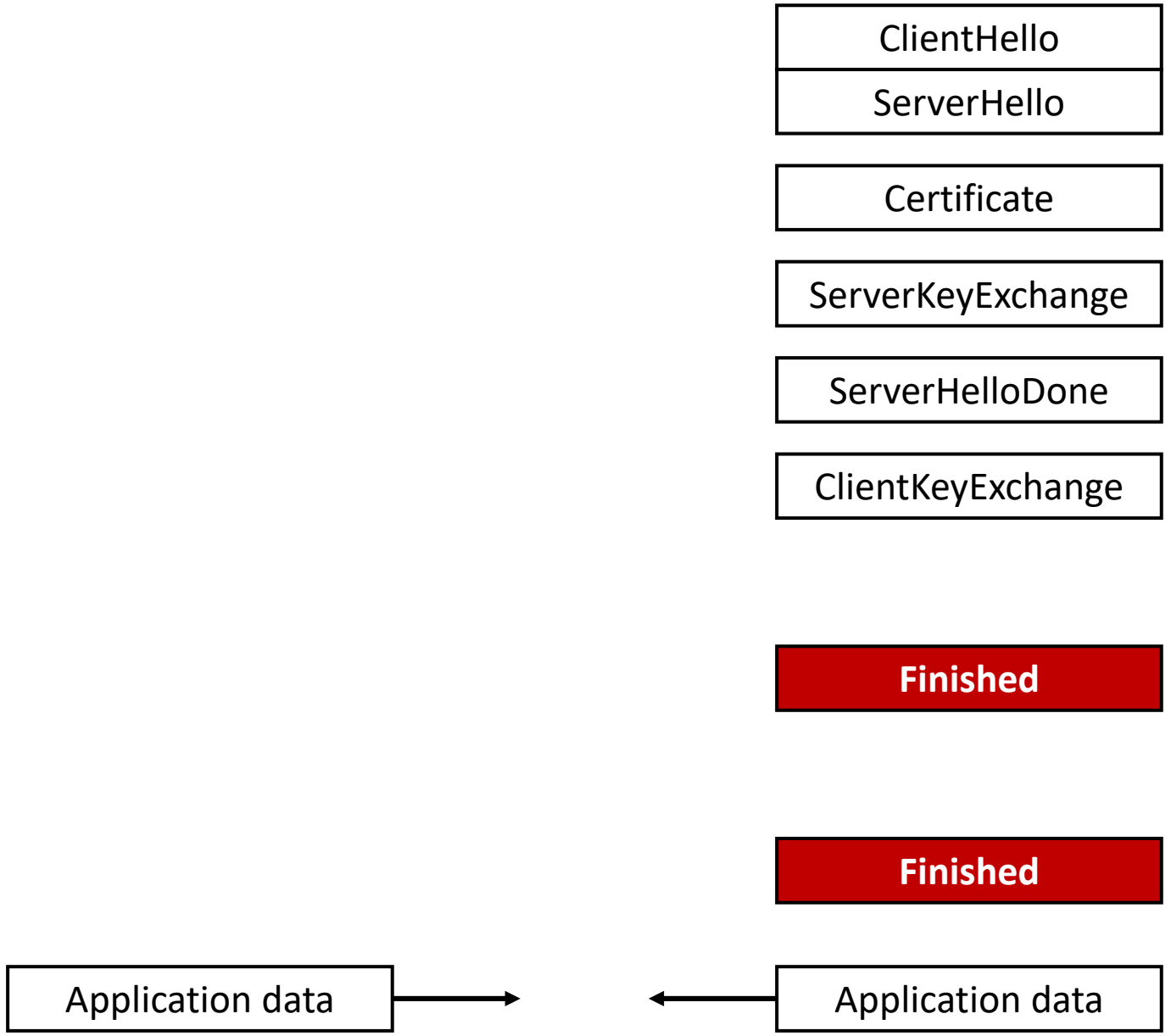

ClientHello
ServerHello
Certificate
ServerKeyExchange
ServerHelloDone
ClientKeyExchange
[ChangeCipherSpec]
Finished
[ChangeCipherSpec]
Finished
Application data
Application data

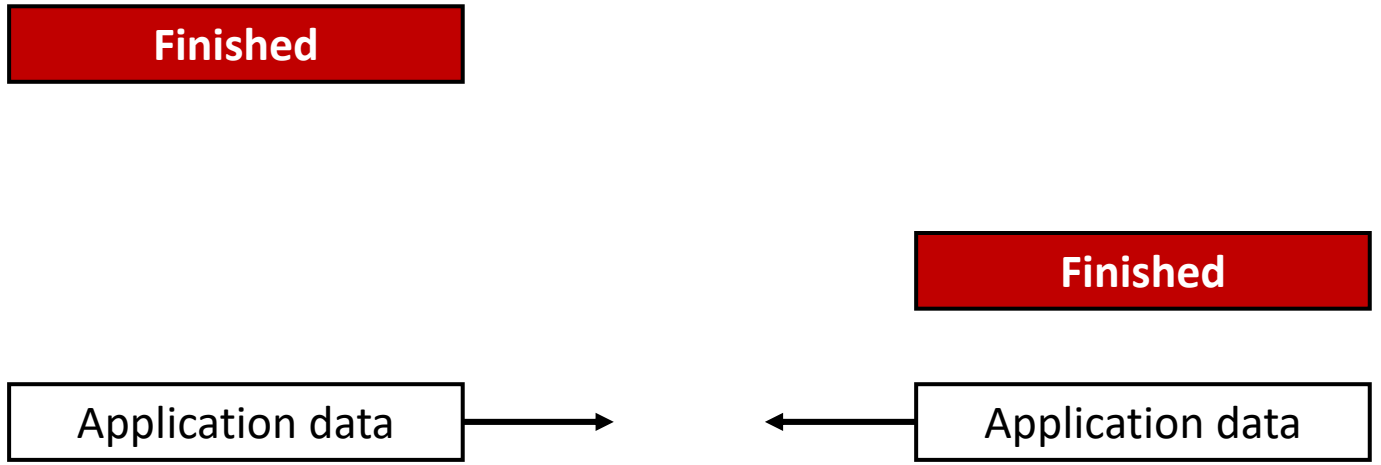

ClientHello
ServerHello
Certificate
ServerKeyExchange
ServerHelloDone
ClientKeyExchange
[ChangeCipherSpec]
Finished
[ChangeCipherSpec]
Finished
Application data
Application data

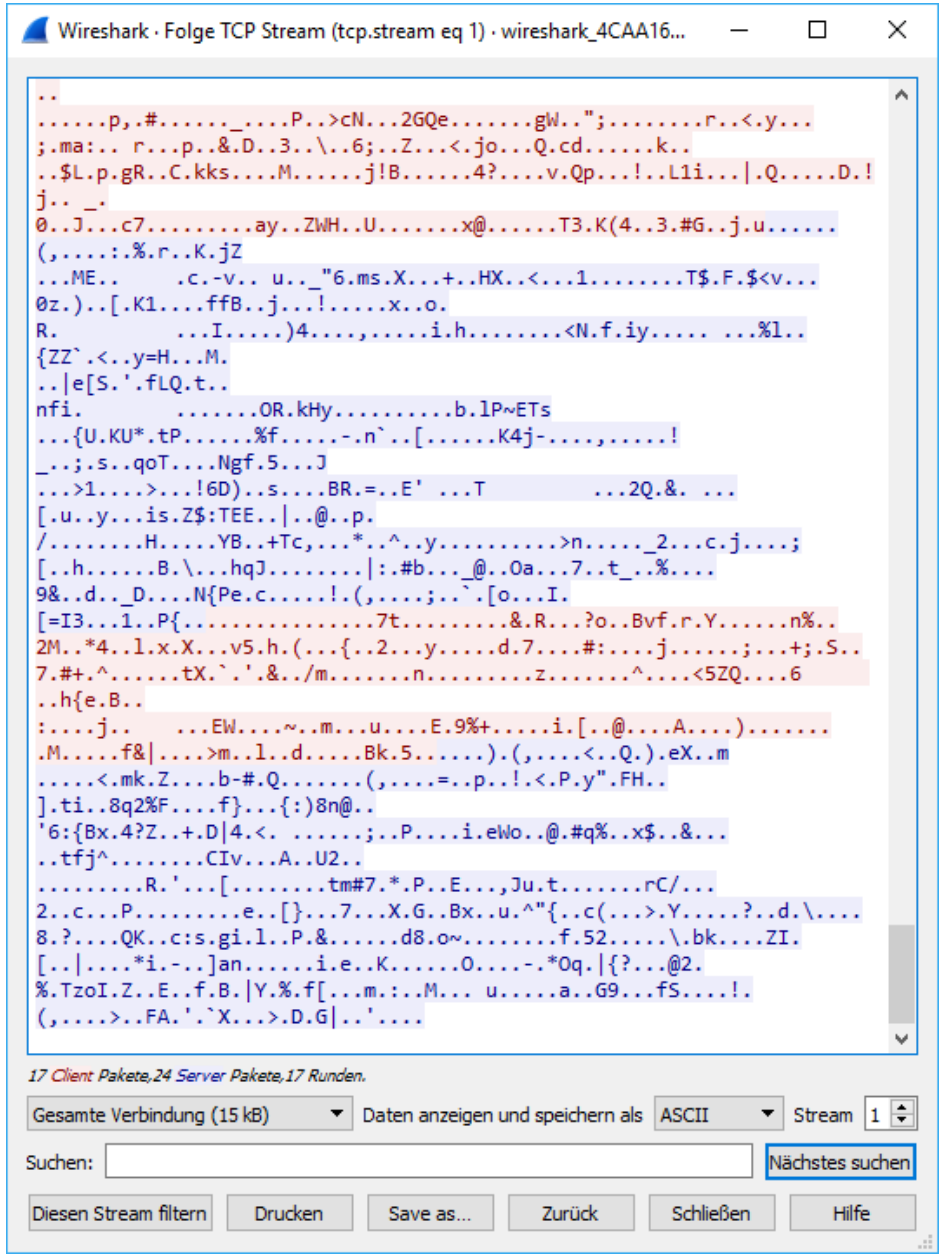

ClientHello
ServerHello
Certificate
ServerKeyExchange
ServerHelloDone
ClientKeyExchange
Finished
Finished
Application data
Application data

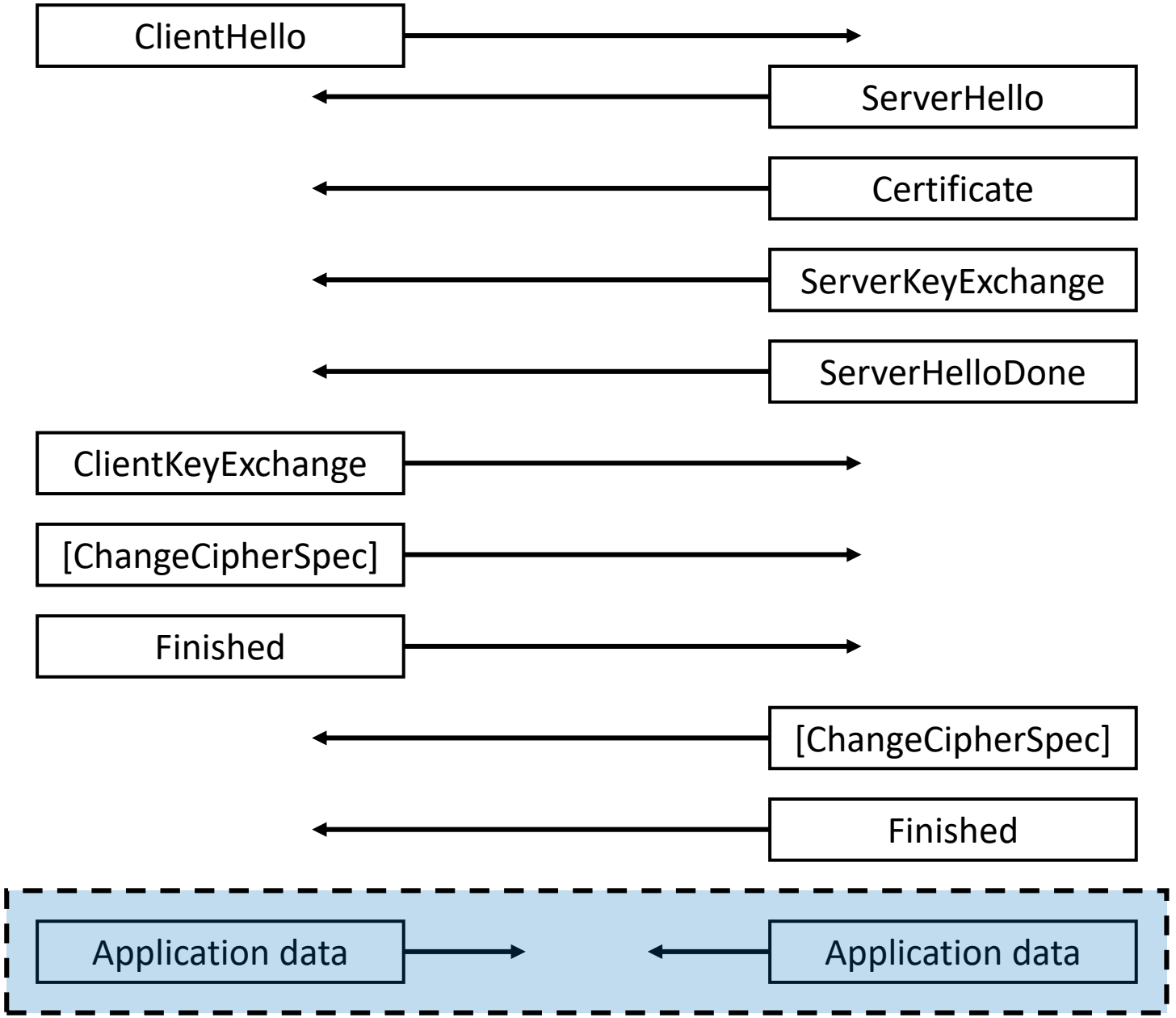

ClientHello
ServerHello
Certificate
ServerKeyExchange
ServerHelloDone
ClientKeyExchange
Finished
Finished
Application data
Application data


ClientHello
ServerHello
Certificate
ServerKeyExchange
ServerHelloDone
ClientKeyExchange
Finished
Finished
Application data
Application data

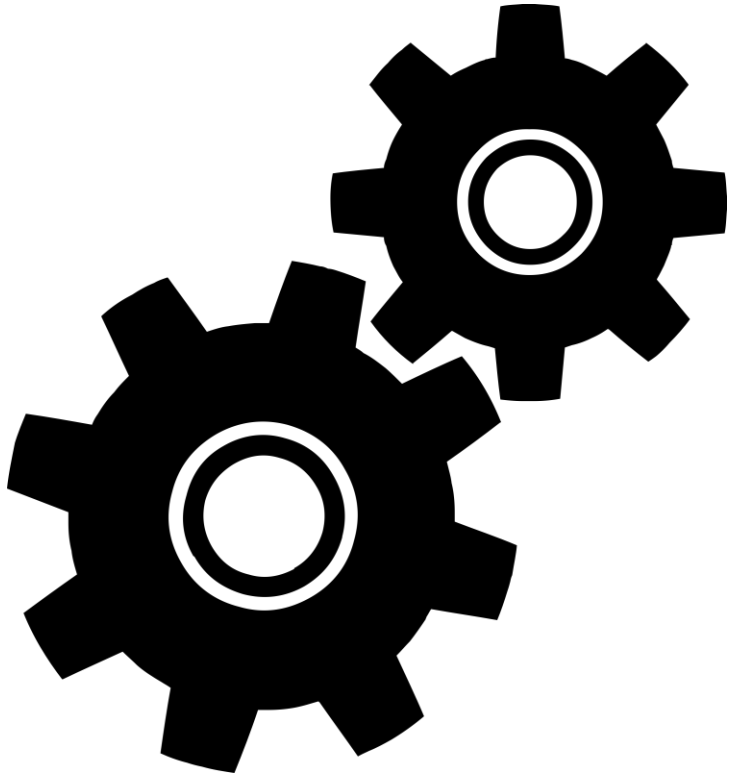

ClientHello
ServerHello
Certificate
ServerKeyExchange
ServerHelloDone
ClientKeyExchange
Finished
Finished
Application data
Application data


Finished
Finished
Application data
Application data


Wireshark · Folge TCP Stream (tcp.stream eq 1) · wireshark_4CAA16...
17 Client Pakete,24 Server Pakete,17 Runden.
Gesamte Verbindung (15 kB)
Daten anzeigen und speichern als
ASCII
Stream
1
Suchen:
Nächstes suchen
Diesen Stream filtern
Drucken
Save as...
Zurück
Schließen
Hilfe


ClientHello
ServerHello
Certificate
ServerKeyExchange
ServerHelloDone
ClientKeyExchange
[ChangeCipherSpec]
Finished
[ChangeCipherSpec]
Finished
Application data
Application data

Pause (danach Hands on)



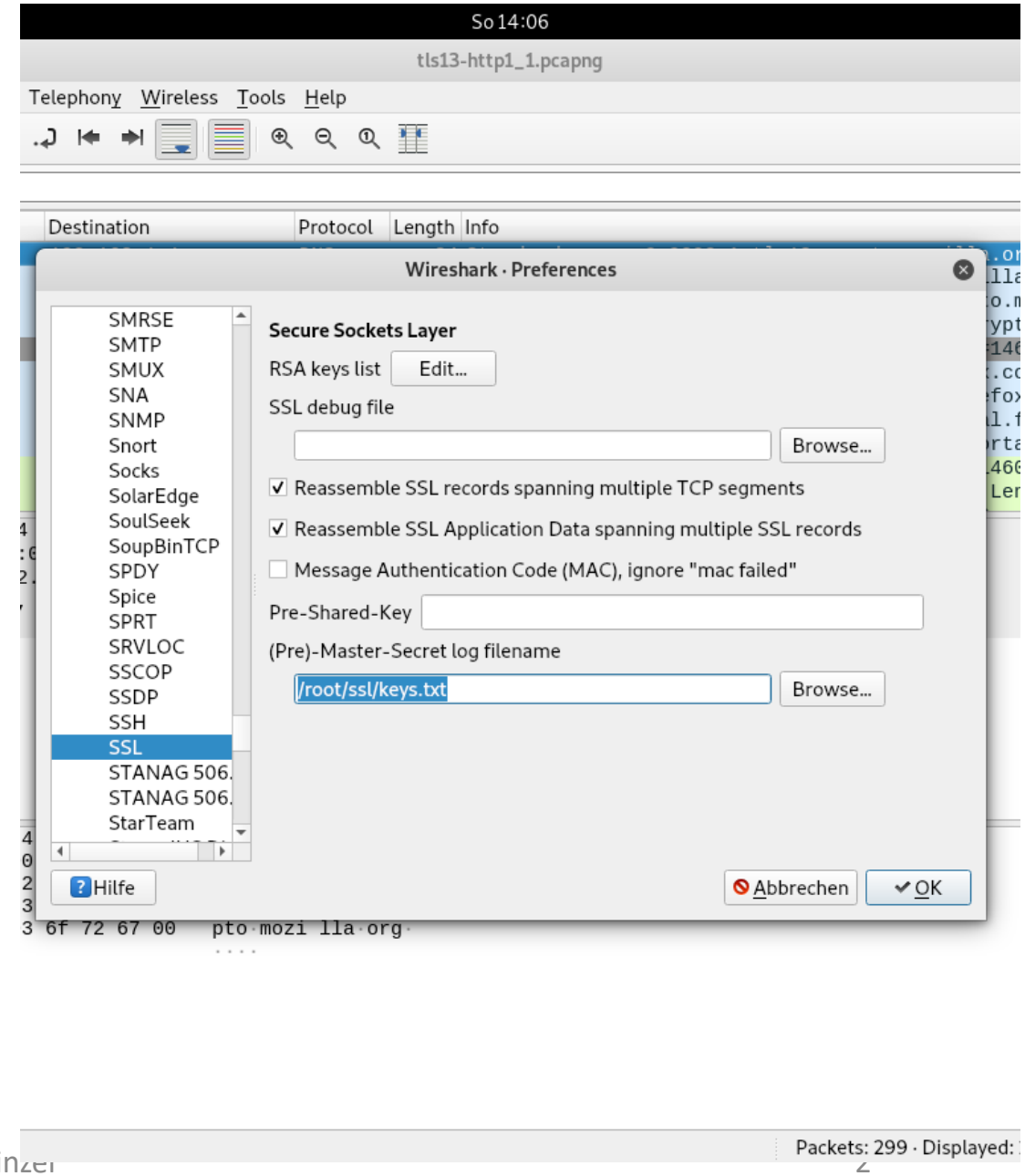
TLS Workshop

German OWASP Day

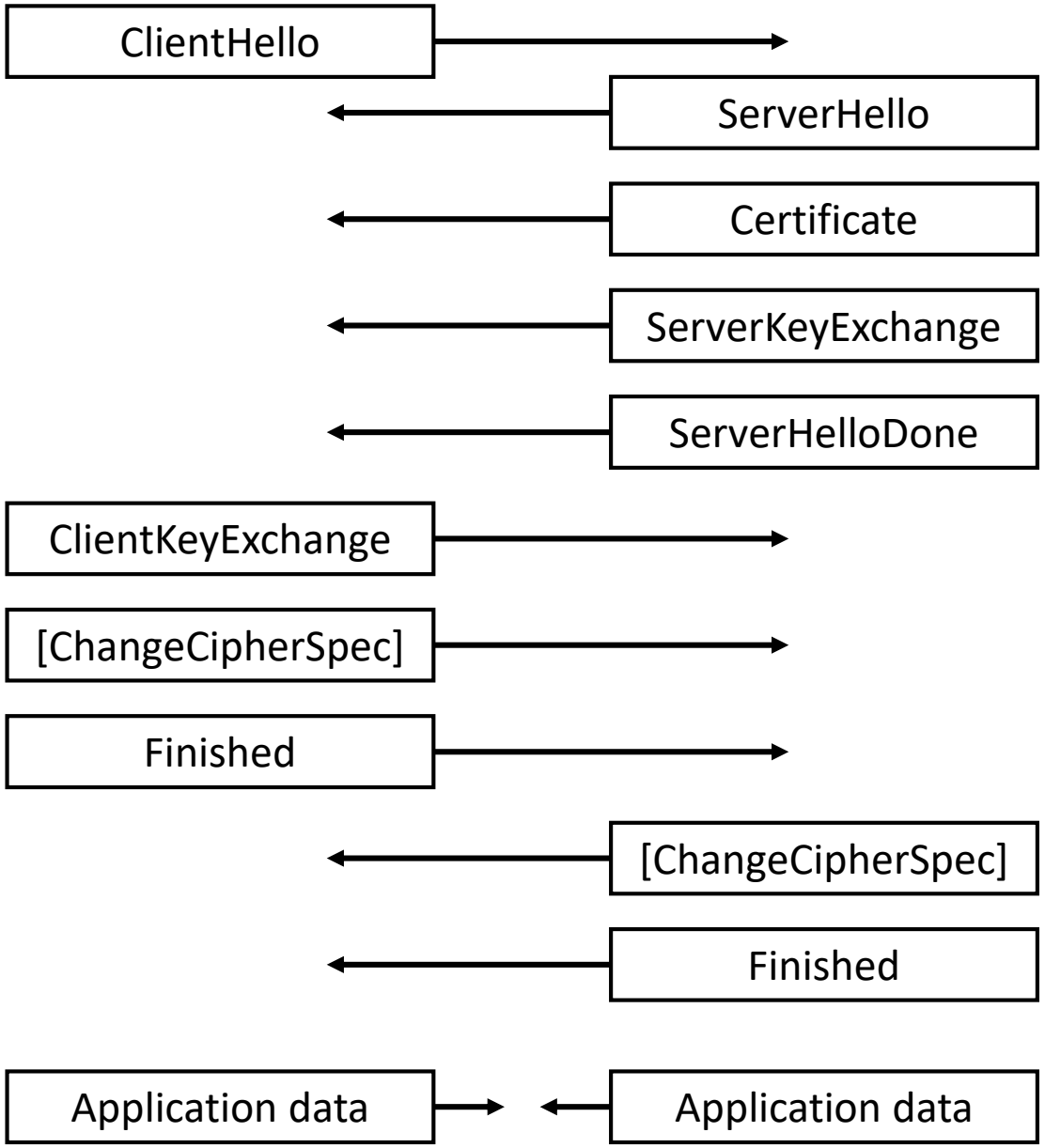
Sebastian Schinzel, Damian Poddebniak, Achim Hoffmann

Wireshark

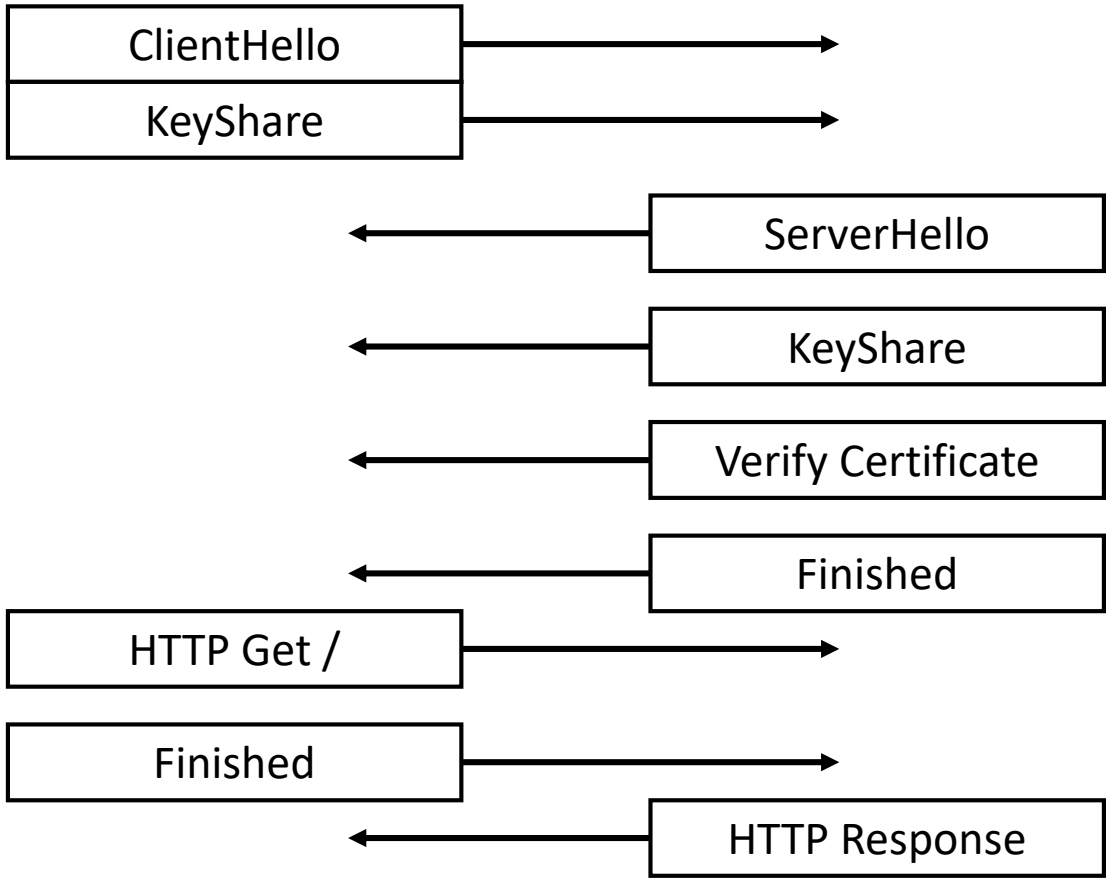
- Entschlüsseln von TLS über Wireshark:
- Firefox starten und vorher setzen: `SSLKEYLOGFILE=/root/ssl/keys.txt`
- Dann Key log file im Wireshark setzen
- Domains:
  - https://tls13.crypto.mozilla.org/
  - https://cloudflare.com/

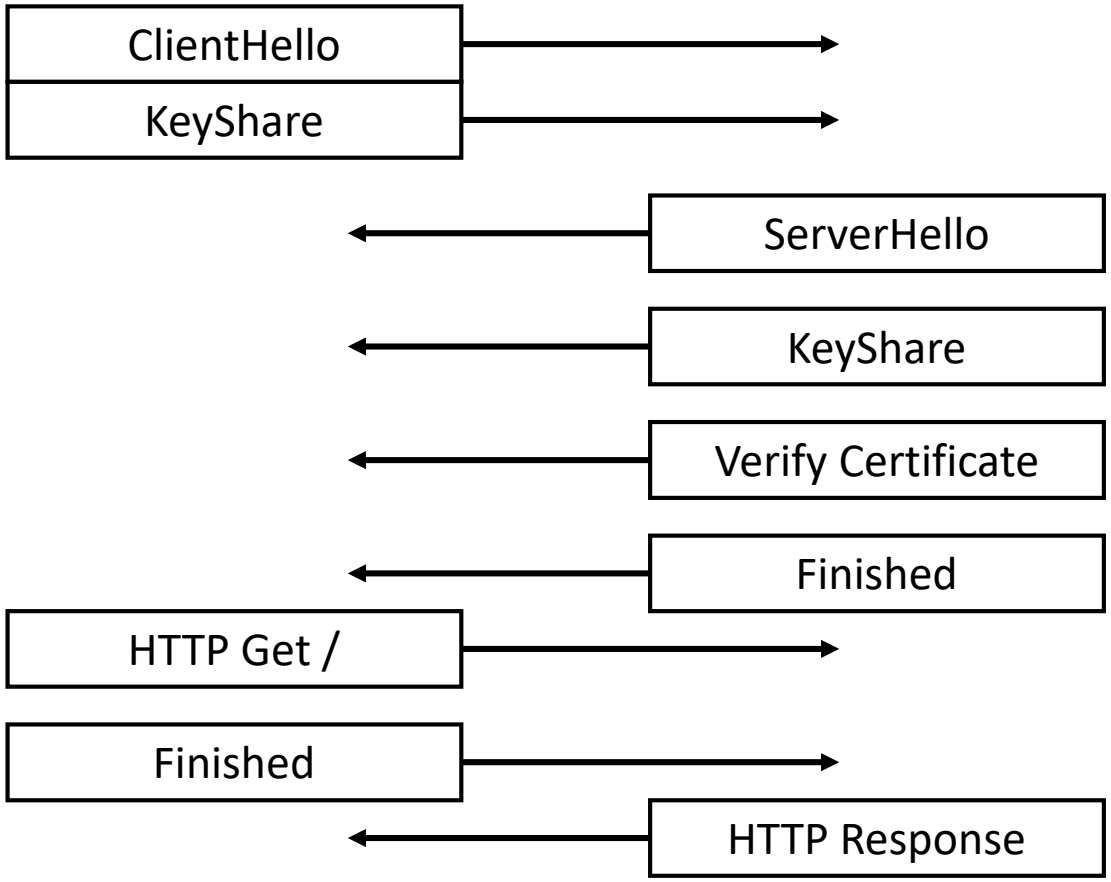


TLSv1.0 - TLSv1.2

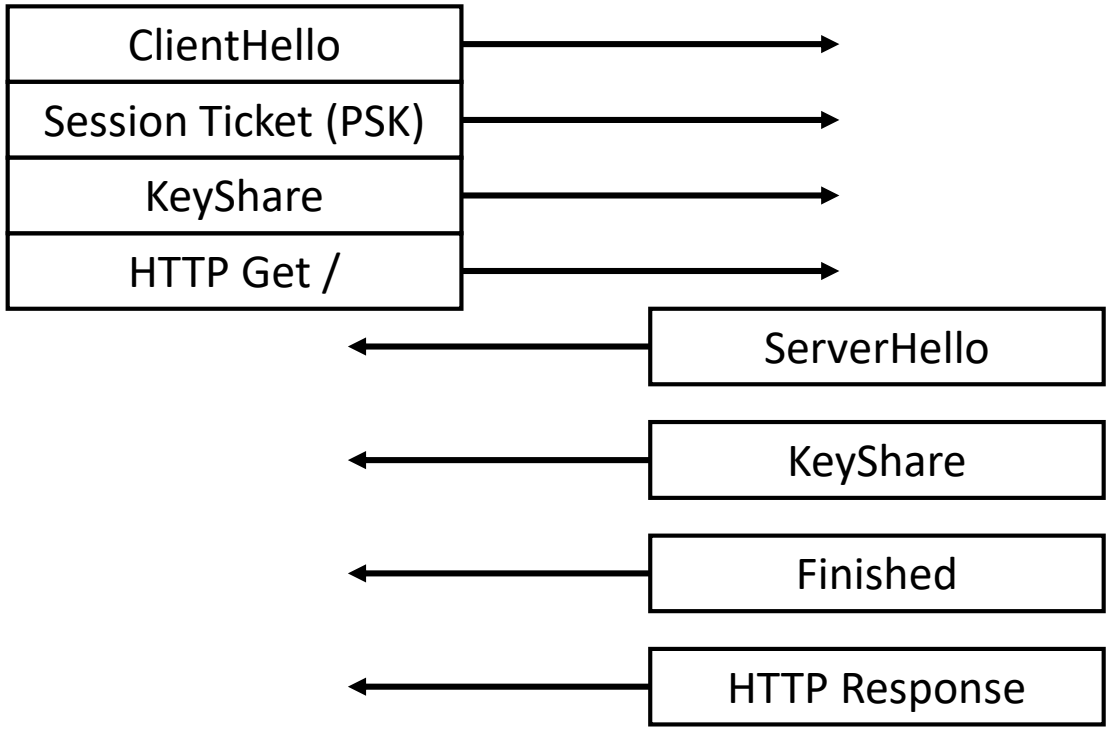


TLSv1.3





TLS 1.3



TLS 1.3 0-RTT

Wireshark Trace

OWASP
German Chapter

?

OWASP
German Chapter

!

sslyze.py vessl jcertchecker ssl-cert-check THCSSLCheck.exe
ssldiagnos.exe testssl.sh ssltest heartbeat.py
SSLSmart
SSLCertScanner.exe ssl-check-heartbleed.pl
o-saft.pl beast.pl SSLDigger sslcat TLSSLed v1.3.sh TestSSLServer.jar
sslsplit
sslstrip ssl-cipher-check.jar
ssltap
sslthing.sh
SSL Analyzer nmap openssl ssldump
sslsniff
ssl-cipher-check.pl cvt chksslkey sslscan sslsqueeze
stunnel
sslmap.py ssl-renegotiation.sh
ssltest.pl manyssl.pl cnark.pl SSLAudit.pl ssl-beast.pl
tls-check.pl

OWASP
German Chapter

Tools
Functionality

- Mainly command-line (CLI)
- Some with GUI (mainly Windows)
- Tools online (15+)
- Tools, checking vulnerabilities and bugs (11)
- Tools, checking supported ciphers (22)
- Tools, checking certificate, PKI certificate chain (10)

OWASP
German Chapter

Tools
Online

- https://www.ssllabs.com/
- https://sslguru.com/ssl-tools/check-ssl-certificate.html
- http://certlogik.com/ssl-checker/
- http://www.sslshopper.com/ssl-checker.html
- https://www.howsmyssl.com/
- https://sslcheck.globalsign.com
- https://confirm.globessl.com/ssl-checker.html
- https://filippo.io/Heartbleed/
- http://possible.lv/tools/hb/
- https://ssl-tools.net/heartbleed-test
- https://www.cloudflarechallenge.com
- http://ccsbug.exposed/

Take care: some check certificate and/or certificate chain only.

OWASP
German Chapter

Tools
CLI

- **Cipher Checks (22)**
  - openssl *nmap cnark.pl manyssl.pl* o-saft.pl ssl-cipher-check.pl athena-ssl-cipher-check_v062.jar sslthing.sh SSLAudit.pl ssl-cipher-check.pl ssldiagnose.exe sslmap.py sslscan ssltest.pl ssltlstest.exe sslyze.py testssl.sh tls-check.pl tls-scan TestSSLServer.jar THCSSLCheck.exe tlsenum tls-check.pl TLSSLed_v1.3.sh
- **Vulnerability Checks (11)**
  - beast.pl ccs-injection.sh OSSL_CCS_InjectTest.py robot-detect ssl-renegotiation.sh ssltest_heartbeat.py ssl-check-heartbleed.pl ssl-heartbleed.sh sslscan (1.11) testssl.sh thc-ssl-dos o-saft.pl
- **Certificate Checks (10)**
  - certtool certutil chksslkey cvt jcertchecker keytool SSLCertScanner.exe ssl-cert-check xca (o-saft.pl) testssl.sh

OWASP
German Chapter

Tools
Not in Scope

- `cnark.pl` and `manyssl` and `sslcaudit`
  - because they require sophisticated setup (installation pre requests like perl modules)
- `certstealer`
  - seems not to be working as promised
- `certtool`, `certutil`, `xca`, `keytool`
  - these are tools to manage certificates

OWASP
German Chapter

Tools
Installation: Why

- Before you do any work, you should know which tool version, in particular OpenSSL, you'll be using.*
- Do not trust any pre-installed or pre-configured tool, always check version and configuration.

  Various operating systems often modify the OpenSSL code, usually to fix known issues. However, the name of the project and the version number generally stay the same, and there is no indication that the code is actually a fork of the original project that will behave differently.*
- Systems known for silently changing OpenSSL's code or configuration: *SuSE*, *Ubuntu*, *Apple's* debian

* Text mainly stolen from *OpenSSL Cookbook, Ivan Ristiç*

OWASP
German Chapter

Installation
openssl

- using installed `/usr/bin/openssl`
  - `openssl version`
  - `openssl ciphers -V`
  - `openssl ciphers -V |wc`
  - `openssl ciphers -V ALL:aNULL:eNULL |wc`
  - `openssl ciphers -V ALL:COMPLEMENTOFALL |wc`

  (Standard: 90 - 100 ciphers)
- install `openssl-chacha`, `openssl` 1.0.2d, `sslscan`, `stunnel`
- using installed `openssl-chacha` …

  (> 200 ciphers)

OWASP German Chapter

Tools
Problem: usage ...

How to specify hostname and port

FQDN – hostname or IP
PORT – mandatory
port – optional

Cipher Checker	target	target:port	URL
openssl		FQDN:PORT	
nmap		FQDN -p port	
cnark.pl	-h FQDN	-h FQDN -p port	
cipherscan	FQDN	FQDN:PORT	
manyssl.pl		-s IP -p port	
SSLAudit.pl	FQDN	FQDN PORT	
sslmap.py		--host FQDN --port port	
sslscan	FQDN	FQDN:port	
ssltest.pl		FQDN PORT	
sslyze.py	FQDN	FQDN:port	
ssldiagnose.exe	-t FQDN	-t FQDN -p port	
ssl-cipher-check.pl	FQDN	FQDN PORT	
testssl.sh	FQDN	FQDN:port	https://
TestSSLServer.jar	FQDN	FQDN PORT	
THCSSLCheck.exe		FQDN PORT	
tls-scan	--host=FQDN	--host=FQDN --port=port	
o-saft.pl *	FQDN	FQDN:PORT	https://

* o-saft.pl supports all other variants too

OWASP German Chapter

Tools
can, cannot, ...

Amount of ciphers to be checked

Amount for TLSv1.2 inaccurate as some tools use absolute numbers others additional to TLSv1.1

Cipher Checker	SSLv2	SSLv3	TLSv1	TLSv1.2
nmap	?	?	?	?
cnark.pl	6	48	48	?
cipherscan **	0	106?	120?	64
manyssl.pl	?	?	?	?
SSLAudit.pl	8	27	37	38
sslmap.py	12	229	229	229
sslmap.py --fuzz	16M	65536	65536	65536
sslscan	3	28	28	0
ssltest.pl	6	25	8	0
sslyze.py	6	-	12	?
ssldiagnose.exe	7	70	79	?
ssl-cipher-check.pl	9	133	133	33
testssl.sh	12	26	197	33?
TestSSLServer.jar	3	?	9	?
THCSSLCheck.exe	8	27	27	0
tls-scan	7	26	82	121
o-saft.pl +cipher *	13	201	201	201
o-saft.pl +cipherall	63	65536	65536	65536

* with patched Peter Mosman's openssl-chacha ** with its own openssl

OWASP

Tools
Do you know?

Do you know?	Tools
List of ciphers hardcoded	cnark.pl, SSLAudit.pl, sslmap.py, tls-scan
Need to adapt path for openssl	ssl-cipher-check.pl, sslthing.sh
Needs to be started in installation directory	SSLAudit.pl, ssldiagnos.exe, THCSSLCheck.exe, chksslkey
Requires file with ciphers	SSLAudit.pl athena-ssl-cipher-check.jar
Fails for name-based virtual hosts	SSLAudit.pl
Buggy code (prints syntax errors)	sslmap.py (--fuzz), tls-check.pl sslthing.sh, chksslkey,
Output only with -v	sslthing.sh
Change code for SSLv2, SSLv3	sslthing.sh
Need to disable SSLv2 (sometimes)	SSLAudit.pl, ssltest.pl
Support of TLSv1.2	sslmap.py, sslyze.py, o-saft.pl

OWASP

Tools
Pros & Cons

Tool	Comment
openssl	Cryptic usage
nmap	Cryptic usage, needs plugin
cnark.pl	Needs special perl modules
manyssl.pl	Needs special perl modules
SSLAudit.pl	Buggy, sometimes
sslmap.py	Buggy but: SIP, STARTTLS, fuzzing ciphers
sslmap.py	Fast, confusing output (personal opinion!)
sslscan	Fast! (different versions in the wild: 1.8 and 1.11)
ssltest.pl	Conformance checks, output?
sslyze.py	Lot of options, multiple targets
ssldiagnose.exe	Supports SIP, STARTTLS

OWASP

Tools
Pros & Cons

Tool	Comment
ssl-cipher-check.pl	?
testssl.sh	Rich feature set, very nice output, human reading
TestSSLServer.jar	Fast, short output
THCSSLCheck.exe	?
TLSSLed_v1.3.sh	Confusing output (personal opinion!)
o-saft.pl	*biased*
sslcaudit	Proxy to check behaviour of VPN clients
jcertchecker	Works with port 443 only
cipherscan	Comes with its own openssl executable
tls-check.pl	Buggy and strange usage, needs special perl modules

OWASP
German Chapter

Tools
nmap

- Dependencies
  tls.lua → sslcert.lua → xmpp.lua → sasl.lua
  unicode.lua → unittest.lua
- if installed in `/usr/share/nmap/scripts`
  - `nmap --script sslv2 -p 443 localhost`
  - `nmap --script ssl-cert -p 443 localhost`
  - `nmap --script ssl-ccs-injection -p 443 localhost`
  - `nmap --script ssl-enum-ciphers -p 443 localhost`
  - `nmap --script tls-nextprotoneg -p 443 localhost`
  - `nmap --script ssl-heartbleed -p 443 localhost`
  - `nmap --script ssl-known-key -p 443 localhost`

→ see cmd-n-nmap

O-Saft
Agenda

O-Saft:

- Motivation and features
- Checks and ciphers
- Print certificate informations
- Print SSL connection informations
- Check SSL connection, vulnerabilities, ..
- Automation, postprocessing, docker
- GUI, help – help yourself

OWASP
German Chapter

O-Saft
Motivation

Advantages:

- Platform independent
- Independent from SSL libraries (+cipherall)
- Repeatable results
- Automatable (script, batch, CGI)
- Usable in Intranet (without Internet access)
- Docker installation (including adapted openssl)
- No Root CA and no client-side certificate necessary
- Open Source

OWASP
German Chapter

O-Saft
More Features

Additional Features:

- Detect/check any cipher (up to 65536)
- STARTTLS (IMAP, LDAP, POP3, RDP, SMTP, XMPP, ...)
- Detect server-side cipher priority
- Flexible use of TLS extensions
- Check known vulnerabilities (BEAST, BREAST, CRIME, CSS, DROWN, FREAK, Heartbleed, POODLE, Renegotiation, Sweet32, Sloth, TIME, ROBOT, ...)

OWASP
German Chapter

O-Saft
Dependencies

Does not rely on any underlaying SSL library for

- `+protocols`
- `+heartbleed`
- `+cipherall` checks even when client certificate required

Relys on underlaying SSL library for

- most checks in `+check`
- most information (`+info` and others)

OWASP
German Chapter

O-Saft
check

Output

- Label text followed by *yes* if considered good
- Label text followed by "description" if considered bad

Following checks are based on given information (not really checked)

- CRIME, BEAST, BREACH, FREAK, POODLE, Lucky13, PFS, FIPS, PCI, TR-02102
- Supported EDH, ADH, RC4
- Renegotiation, Resumption

OWASP
German Chapter

O-Saft
Usage

Don't think about:

- *target* and *port* parameters – supports any syntax from other tools including full URL
- Positional parameters – supports any sequence
- Option names – supports w/o _ or – or . in its name

Just check what's needed:

- Ciphers, or certificate information, or a single value

Format your output

- Using postprocessors

OWASP
German Chapter

O-Saft
„Trust“

Certificate:

- Validity (dates, Root CA)
- Wildcards
- Extended Validation (DV, OV, EV)
- Extensions (CRL, OCSP, SRP, STS, TLS session, ..)
- Compliance (BSI TR-02102-2 , FIPS, ISM, PCI, ...)
- „Bugs“ (invalid characters, length restrictions, ...)

OWASP
German Chapter

In Practice:

- Arbitrary order of options: host, port, any other
- Supports syntax of (most) other tools
- Supports full URI
- Variants of options: `no-DNS, nodns, no_DNS`
- Most options as „on“ and „off“ variant
- Formated results
- Check single value: `+cn, +wildcard, ...`
- To be used as CGI (not recomended)

OWASP
German Chapter

O-Saft
Example

In Practice:

- --no-sslv2 --noSSLv3
- --header --enabled
- --no-http --no-sni --no-dns
- --http --sni
- --trace --trace-key
- --cipherrange=rfc
- --ssl-use-reneg
- --sni-name=