

Vorlesung Netzsicherheit

Kapitel 8 – TLS

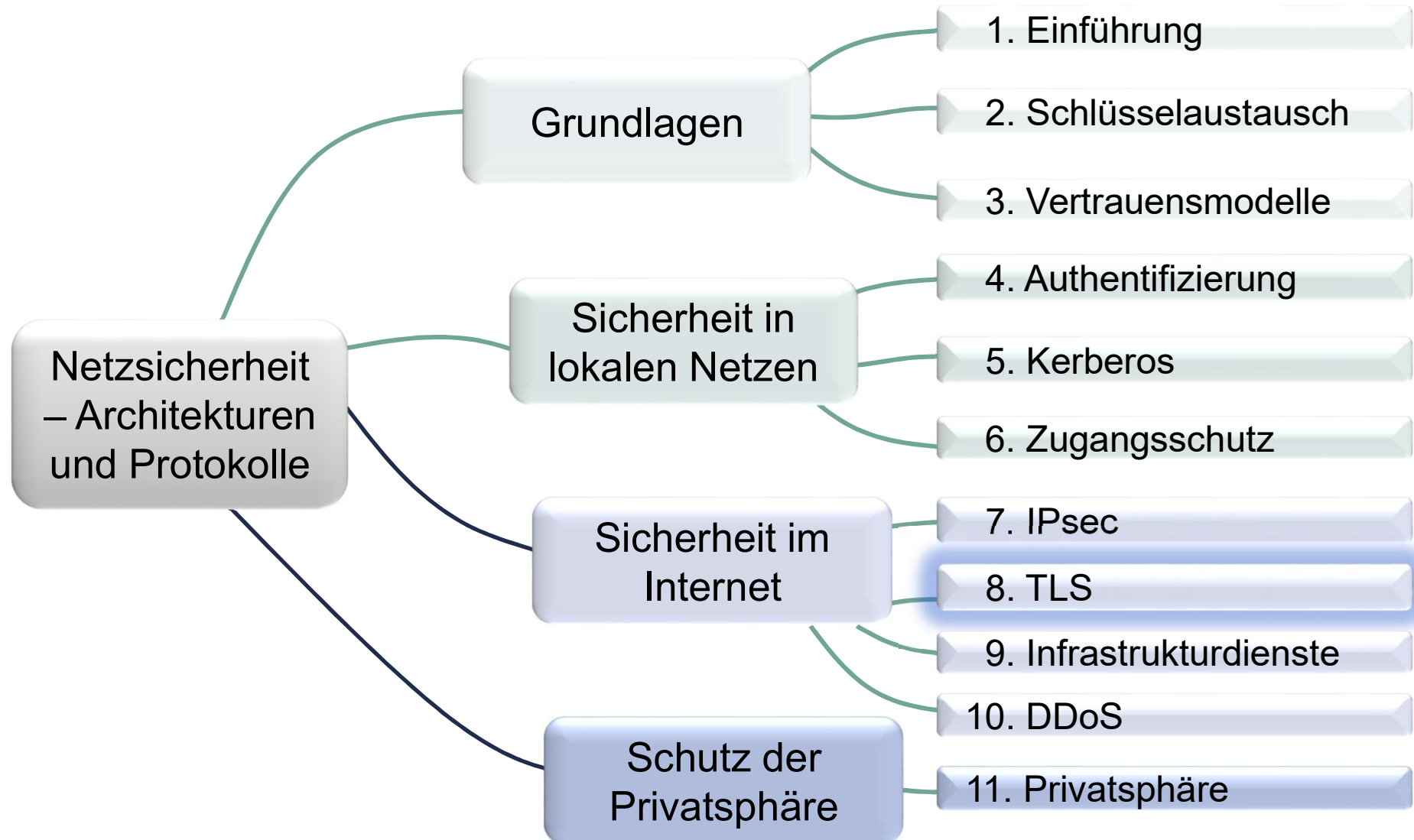
PD Dr. Ingmar Baumgart, PD Dr. Roland Bless, Matthias Flittner, Prof. Dr. Martina Zitterbart
baumgart@fzi.de, [bless, flittner, zitterbart]@kit.edu

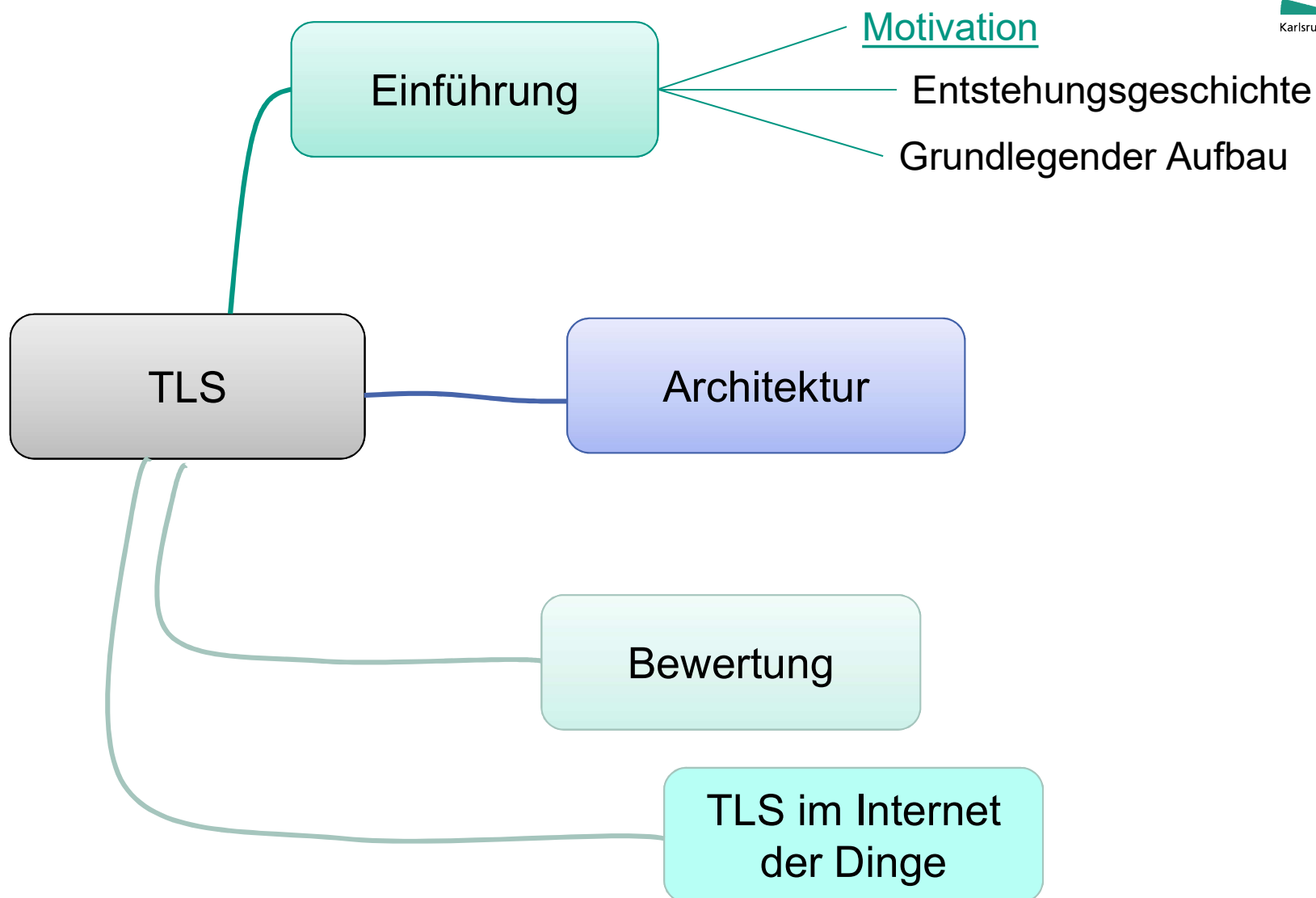
Institut für Telematik, Prof. Zitterbart



© Peter Baumung

Inhalte der Vorlesung





Datentransport im Internet

■ Standardmäßig eingesetzte Protokolle

- **UDP** (User Datagram Protocol)
 - Bietet unzuverlässigen Dienst
- **TCP** (Transmission Control Protocol)
 - Bietet zuverlässigen Dienst

Keine Maßnahmen
zur Unterstützung von
Sicherheit

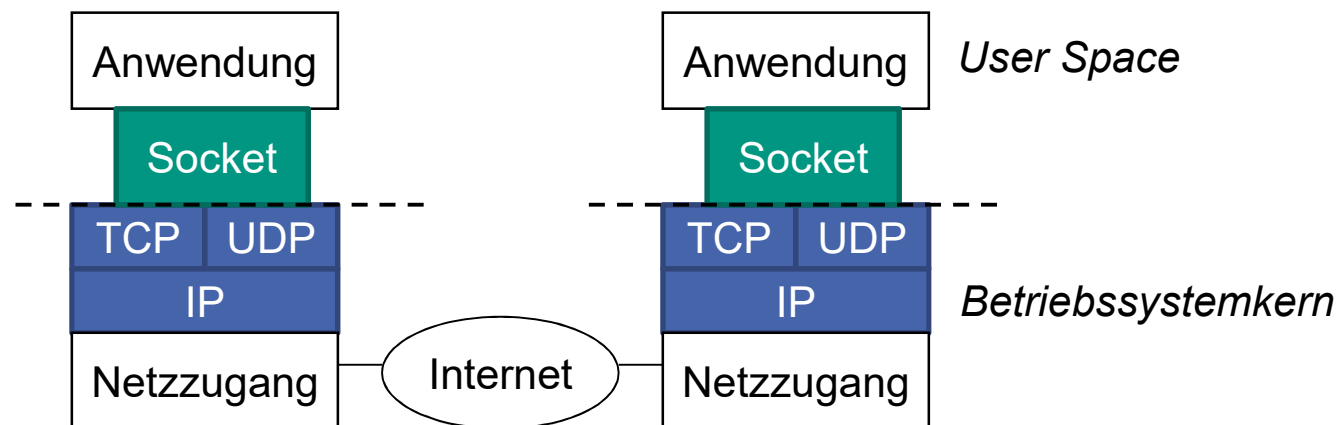


■ Dienstnehmer

- Anwendungen

■ Typische Implementierung des Dienstzugangspunktes

- Socket-Interface (API: Application Programming Interface)



Nicht gewährleistete Schutzziele

■ Authentizität und Integrität der Dateneinheiten

- Dateneinheit tatsächlich von angegebenem Sender gesendet?
- Ist Inhalt unverfälscht?
- Ist Zieladresse tatsächlich das ursprüngliche Ziel?

■ Vertraulichkeit

- Wurde Dateneinheit vom Angreifer gelesen?

■ Authentifizierung der Kommunikationspartner

- Wem sende ich gerade die Dateneinheiten?

■ Schutz vor Wiedereinspielen

- Aktuelle Dateneinheit oder Wiedereinspielung von Angreifer?

TLS: The Transport Layer Security Protocol

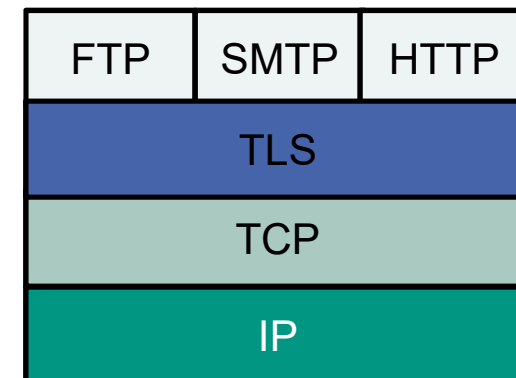
■ Ziel

- Vertrauliche und integere Kommunikation zwischen Anwendungen
 - Online-Banking, Online-Shopping, ...
- Authentifizierung der Kommunikationspartner

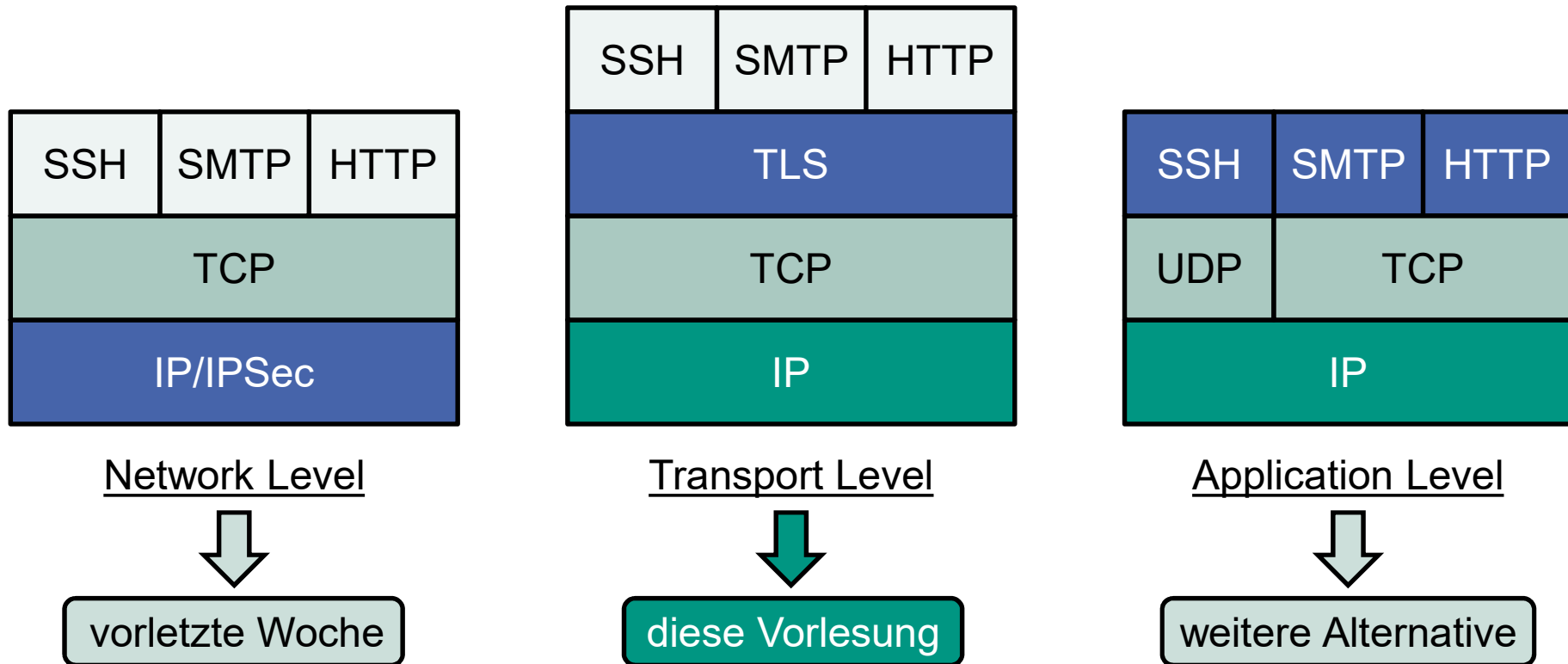


■ Entwickelt für Client-Server Architekturen

■ Einordnung in den Protokollstapel



Aber wo Sicherheit am Besten gewährleisten?



■ Vorteile oder Nachteile?



Vergleich: IPsec vs. TLS

■ IPsec



- Mächtiges, flexibles Verfahren
- Administrator kann in Policy eingreifen
- Aggregation mehrerer Verbindungen möglich
- Schützt transparent alle Schicht-4-Protokolle



- Hohe Komplexität

■ TLS



- Geringere Komplexität
- Anwendungsnah
- Nutzt Kontext von TCP
- Firewall Traversal oft gegeben



- Aggregation Verbindungen unterschied. Systeme nicht möglich
- TLS schützt nur Anwendungsdaten, nicht Schicht 3 und 4
- Administrator kann nicht in Policy eingreifen, von Anwendung definiert

TLS - Rückgrat des „sicheren“ Internets

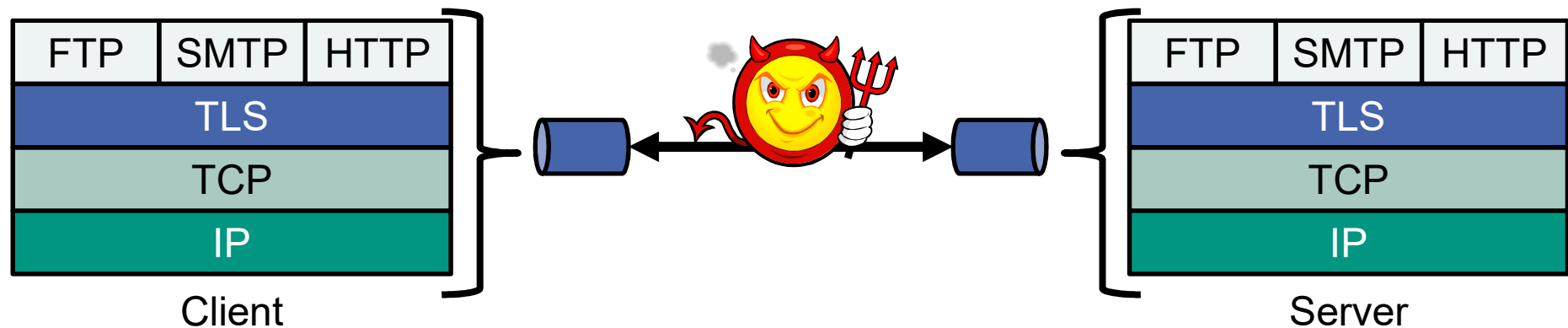
■ Häufigstes eingesetztes Sicherheitsprotokoll im Internet



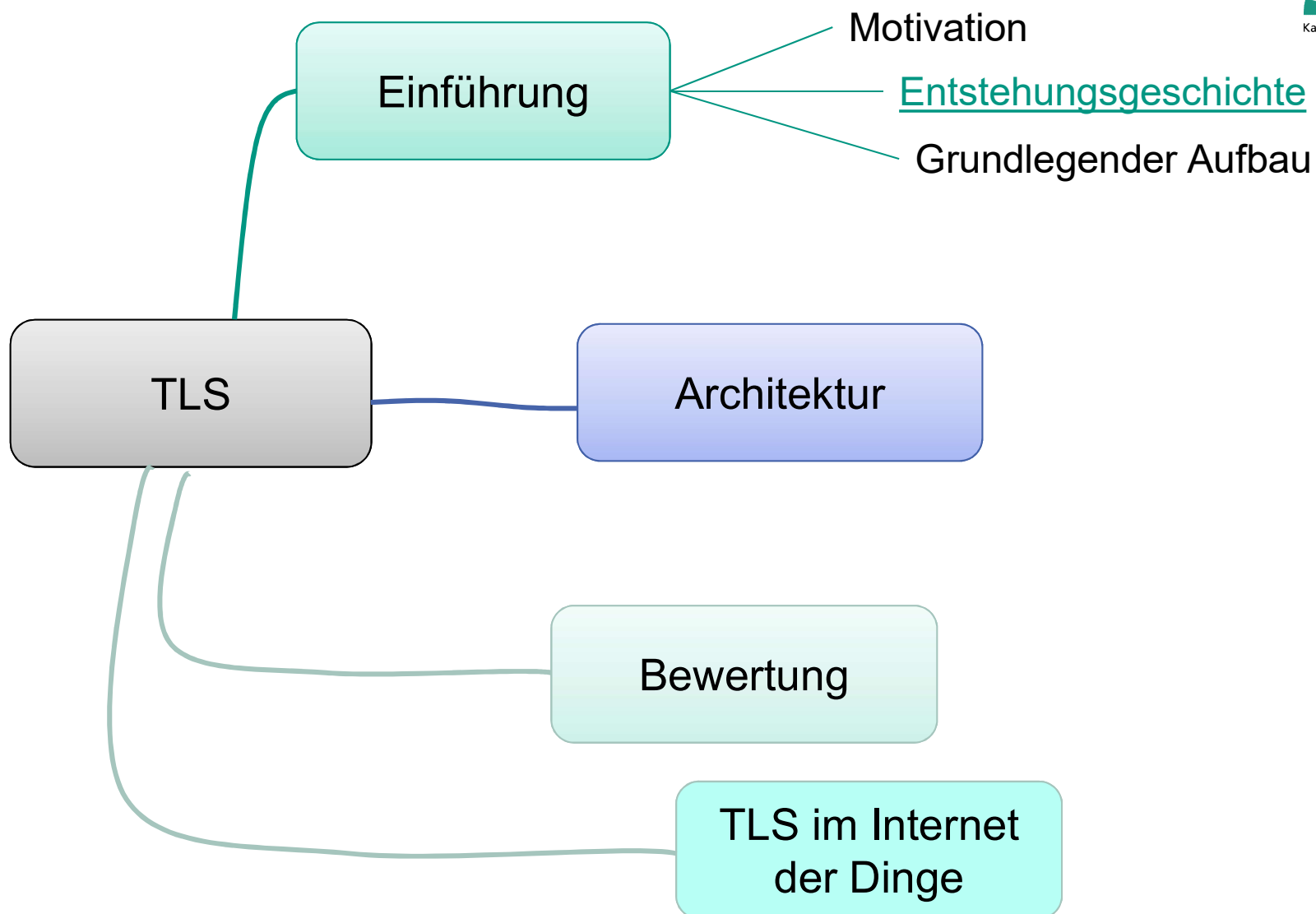
- TLS für Web (HTTP) → **HTTPS**
- TLS für Mail-Versand (SMTP) → **SMTPS**
- TLS für Mail-Abruf (IMAP) → **IMAPS**
- TLS für Mail-Abruf (POP3) → **POP3S**
- TLS für VoIP (SIP) → **SIPS**
- TLS für Dateitransfer (FTP) → **FTPS**
- TLS für Instant-Messaging (XMPP) → **XMPPS**
- TLS für Chat (IRC) → **IRCS**
- ...

Angreifermodell

■ Dolev-Yao (siehe Kapitel 1)



- Wie kann ein Angreifer Zugriff erlangen?
 - Angriff auf physische Infrastruktur (Medium)
 - Angriff auf Routing/Weiterleitung
 - Angriff auf DNS

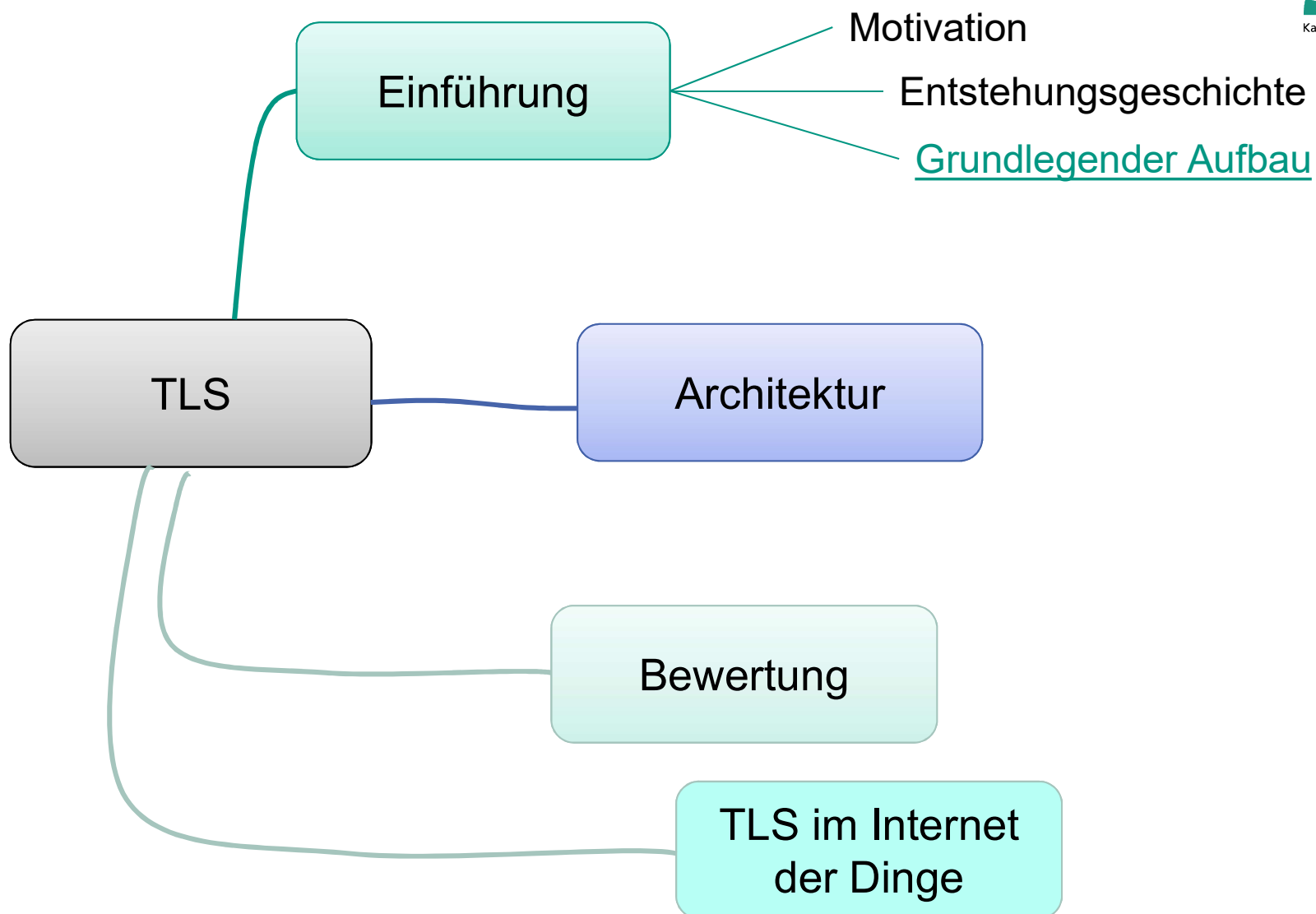


Geschichte von SSL bzw. TLS

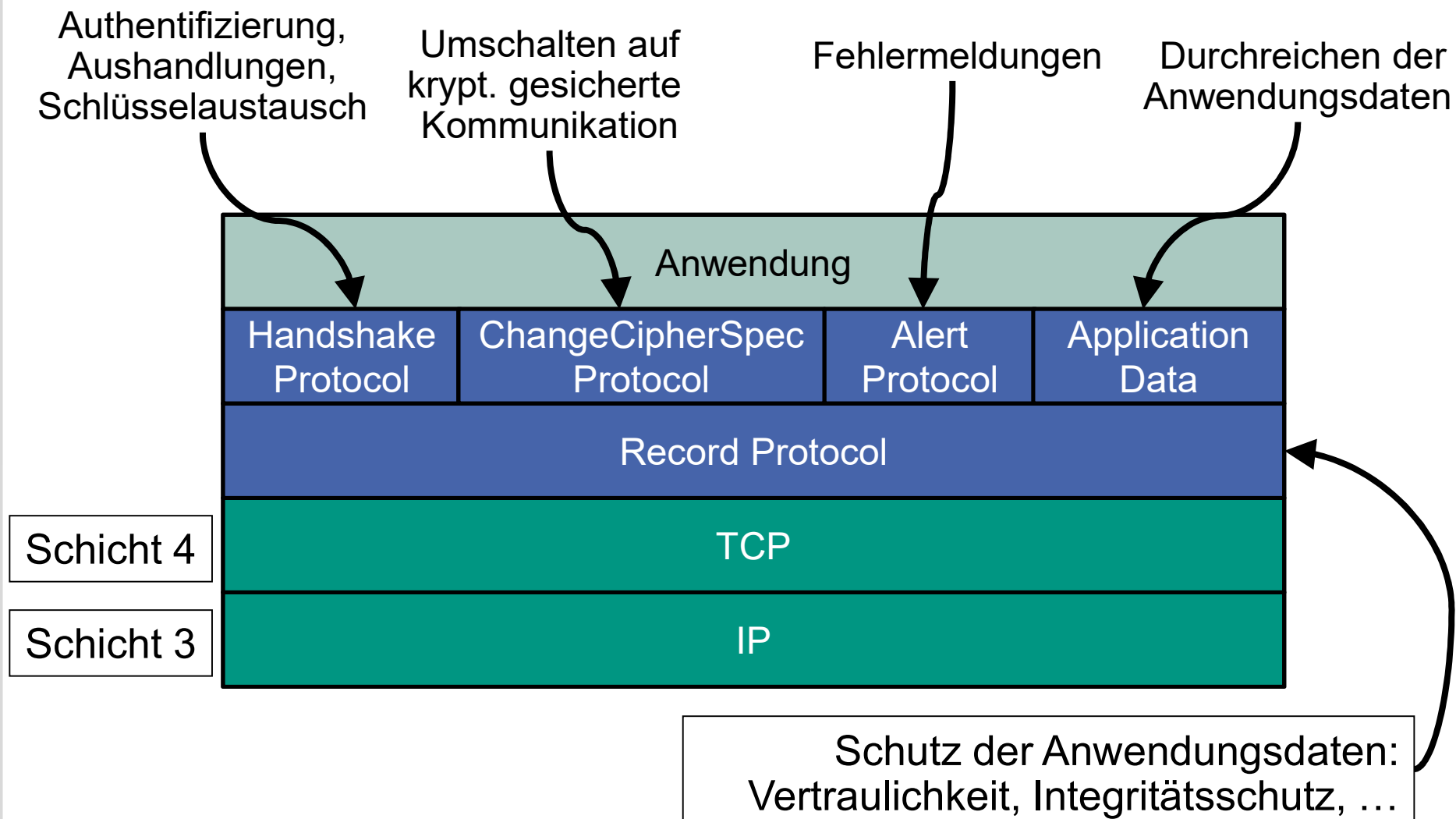
- 1994
 - Netscape: HTTP muss gesichert werden
 - Entwicklung des Secure Socket Layer Protokolls (SSL)
 - SSL v1: Closed Source ohne externes Review, Schwachstellen
 - SSL v2: Diskussion mit externen Spezialisten, erstes Produkt
- 1995
 - Erstes Treffen der IETF
 - SSL v3: vollständige Neuentwicklung des Protokolls
- 1999
 - TLS v1.0: Entwicklung von Transport Layer Security (TLS) in der IETF auf Basis von SSL v3
- 2006
 - TLS v1.1 RFC 4346
- 2008
 - TLS v1.2 RFC 5246

Geschichte von SSL bzw. TLS

- 2010
 - Erweiterung RFC 5746
 - Sicherung der **Session-Wiederaufnahme**
 - Erweiterung RFC 5878
 - Erweiterungen für Autorisierung in TLS Session, z.B. über Autorisationszertifikate
- 2011
 - Erweiterung RFC 6176
 - Entfernung der Rückwärtskompatibilität zu SSL 2.0/3.0
- 2012
 - **Datagram Transport Layer Security (DTLS)** Version 1.2 RFC 6347
 - TLS über unzuverlässige Transportprotokolle wie z.B. UDP
 - Erweiterung RFC 6520
 - Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension

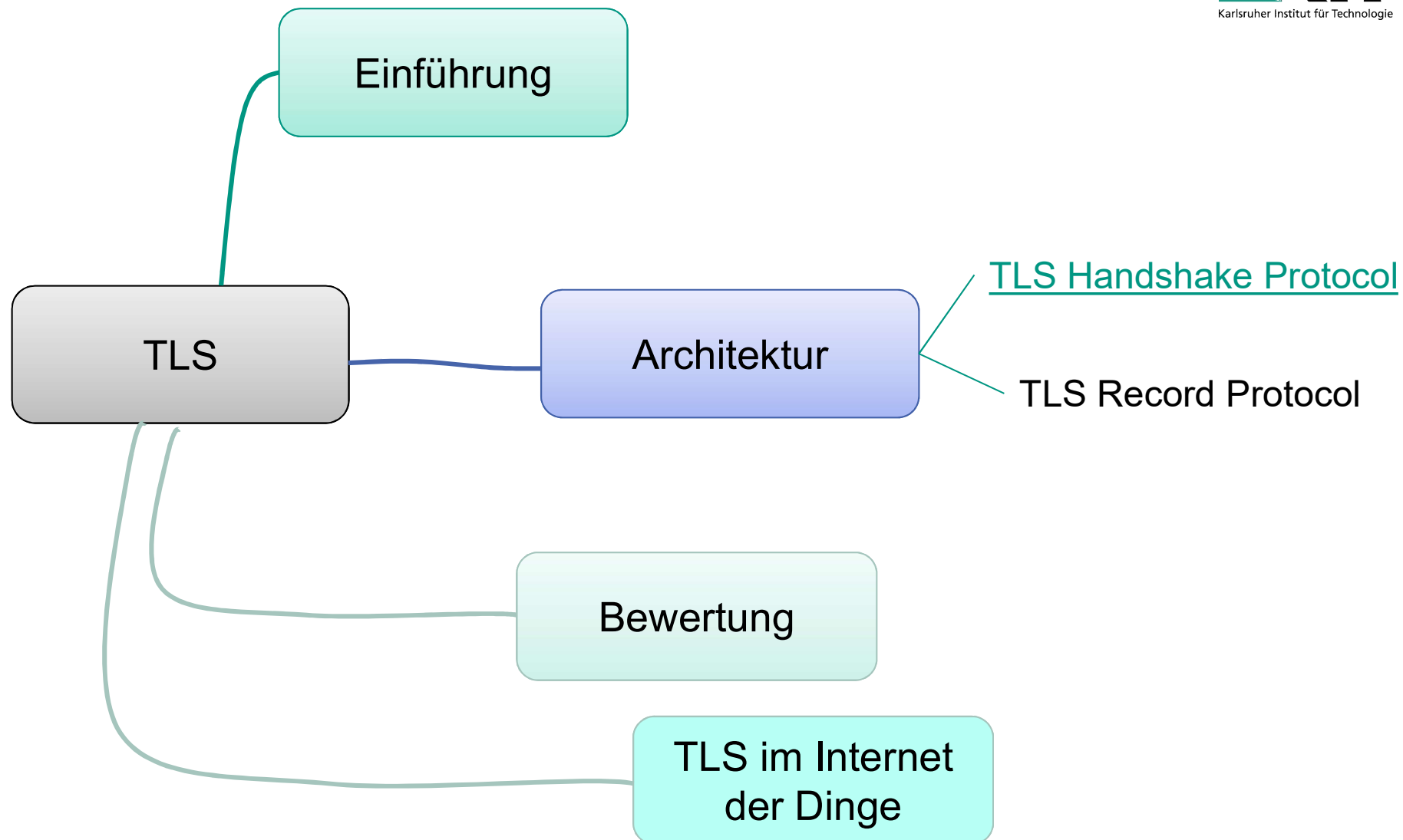


Grundlegende TLS-Protokollstruktur



Grundlegende Komponenten von TLS

- TLS ist unabhängig von der Anwendung!
- Anwendungsprotokolle können transparent auf TLS aufgesetzt werden
- TLS schreibt **nicht** vor, wie Anwendungen „Sicherheit“ mit TLS hinzufügen können
 - Anwendung bestimmt eingesetzte kryptografische Verfahren
 - Anwendung muss bewerten, ob beispielsweise gegenseitige Authentifizierung erforderlich ist



TLS Handshake Protocol

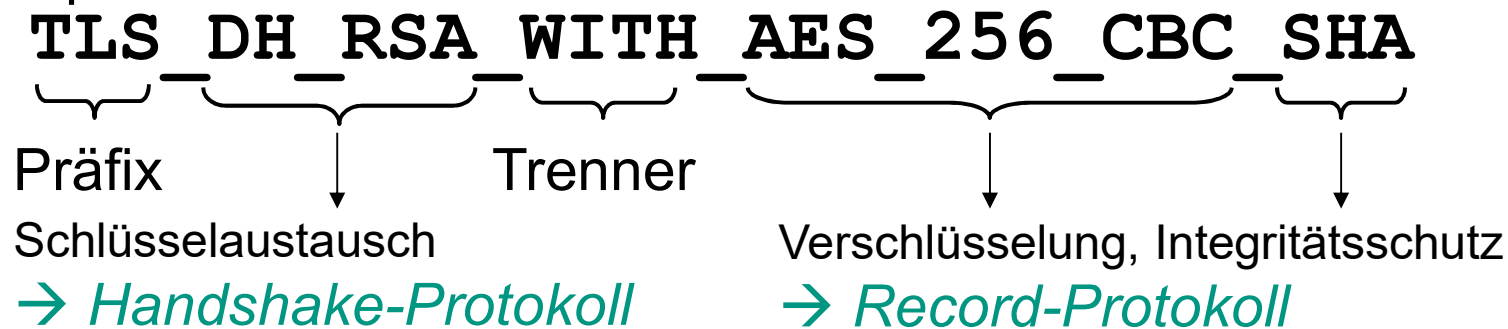
- TLS Handshake Protocol bietet
 - Aushandlung der **verwendeten** kryptografischen Algorithmen
 - Sowohl zur Verschlüsselung als auch Integritätssicherung
 - **Authentifizierung** der Kommunikationspartner (Client und/oder Server)
 - Aushandlung des **Schlüsselmaterials**
 - Sitzungswiederaufnahme und Rekeying werden unterstützt
 - Arbeitet oberhalb des TLS Record Protocol
- Anwendungen sollten sich **nicht** darauf verlassen, dass TLS immer stärkste mögliche kryptografischen Algorithmen auswählt
 - Anwendung muss **vor** Etablierung der Verbindung wissen, was minimale Sicherheitsanforderungen sind
 - ... und nie Informationen über eine schlechter gesicherte Verbindung senden
 - Kryptografischen Algorithmen definiert über TLS Cipher Suites
 - Ausgehandelte Cipher Suite wird Anwendung mitgeteilt

Einschub: TLS Cipher-Suites (I)

- Definition von Cipher-Suites
 - Vordefinierte Kombinationen von Sicherungsalgorithmen
 - Präfix: TLS_
 - Zwei Sektionen
 - Algorithmen für Schlüsselaustausch
 - Algorithmen für Datenaustausch
 - getrennt durch Symbol: _WITH_
 - Kennt eine Implementierung eine Cipher Suite nicht, so wird diese ignoriert
- Sonderfall: TLS_NULL_WITH_NULL_NULL
 - Kein Schutz
 - Zustand bei Initialisierung (→ kein Schlüsselmaterial vorhanden)

Einschub: TLS Cipher-Suites (II)

■ Beispiel



■ Erklärung

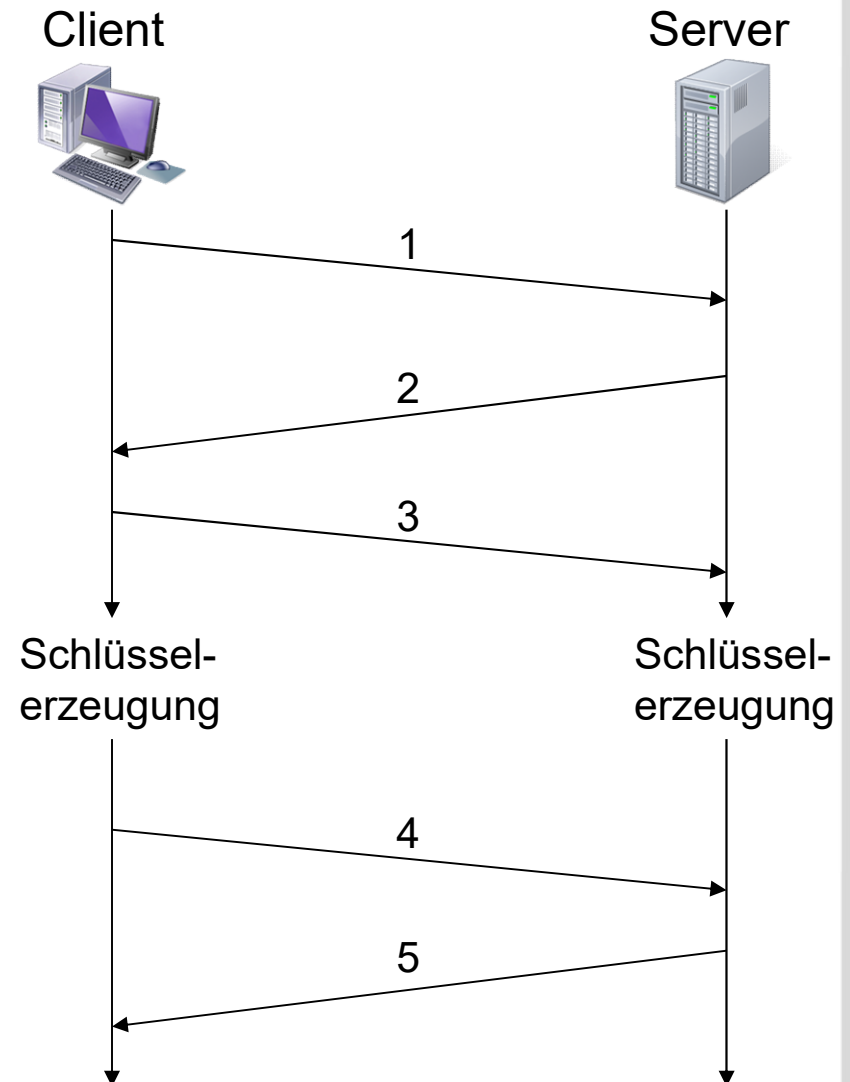
- **TLS**: Präfix zum Erkennen des Protokolls
- **DH**: Schlüsselaustausch durch Diffie-Hellman
- **RSA**: Authentifizierung mit Zertifikat durch RSA
- **WITH**: Trenner für die Struktur
- **AES_256_CBC**: Verschlüsselung der Daten mit AES 256 CBC
- **SHA**: Integritätsschutz mittels HMAC-SHA1

Einschub: Teil der Cipher-Suites aus RFC 5246

Cipher Suite	Key Ex.	Cipher	Mac
TLS_NULL_WITH_NULL_NULL	NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	RSA	NULL	MD5
TLS_RSA_WITH_NULL_SHA	RSA	NULL	SHA
TLS_RSA_WITH_NULL_SHA256	RSA	NULL	SHA256
TLS_RSA_WITH_RC4_128_MD5	RSA	RC4_128	MD5
TLS_RSA_WITH_RC4_128_SHA	RSA	RC4_128	SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES_EDE_CBC	SHA
TLS_RSA_WITH_AES_128_CBC_SHA	RSA	AES_128_CBC	SHA
TLS_RSA_WITH_AES_256_CBC_SHA	RSA	AES_256_CBC	SHA
TLS_RSA_WITH_AES_128_CBC_SHA256	RSA	AES_128_CBC	SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256	RSA	AES_256_CBC	SHA256
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES_EDE_CBC	SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES_EDE_CBC	SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES_EDE_CBC	SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES_EDE_CBC	SHA
TLS_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4_128	MD5
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES_EDE_CBC	SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA	DH_DSS	AES_128_CBC	SHA
TLS_DH_RSA_WITH_AES_128_CBC_SHA	DH_RSA	AES_128_CBC	SHA
[...]			

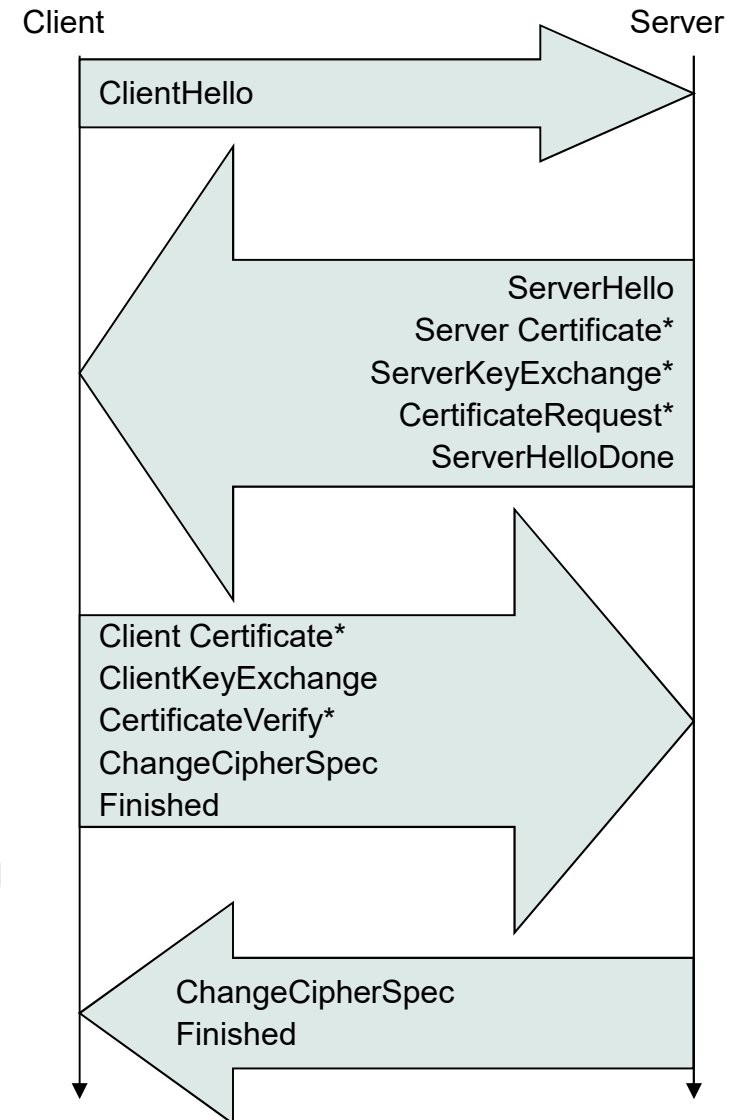
Vereinfachter Ablauf im Überblick

- Ausgangslage
 - Anwendung des Client will mit TLS gesicherte Daten versenden
 - TLS Verbindung soll neu etabliert werden
- Ablauf vereinfacht
 - 1 Auswahl der kryptografischen Verfahren
 - 2,3 Schlüsselaustausch und Authentifizierung
 - Erzeugung des Schlüsselmaterials
 - 4,5 Abschluss und Überprüfung mittels MAC



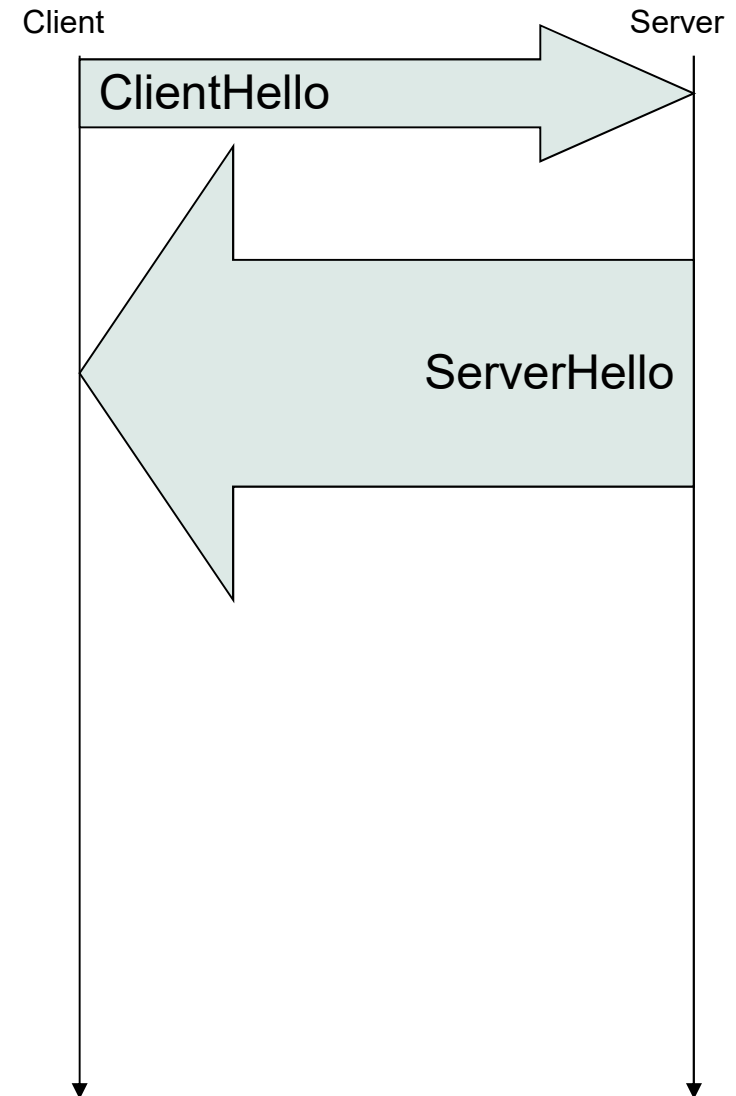
TLS-Schlüsselaustausch im Detail

- Ablauf und entsprechende Nachrichten
 - Auswahl der kryptografischen Verfahren
 - ClientHello
 - ServerHello
 - Schlüsselaustausch und Authentifizierung
 - Server Certificate
 - Client Certificate
 - ClientKeyExchange
 - ServerKeyExchange
 - CertificateVerify
 - Abschluß und Überprüfung
 - ChangeCipherSpec
 - Finished
- Mit * gekennzeichneten Nachrichten abhängig von eingesetzter CipherSuite
 - Maximal also 13 Nachrichten und 2 RTTs notwendig



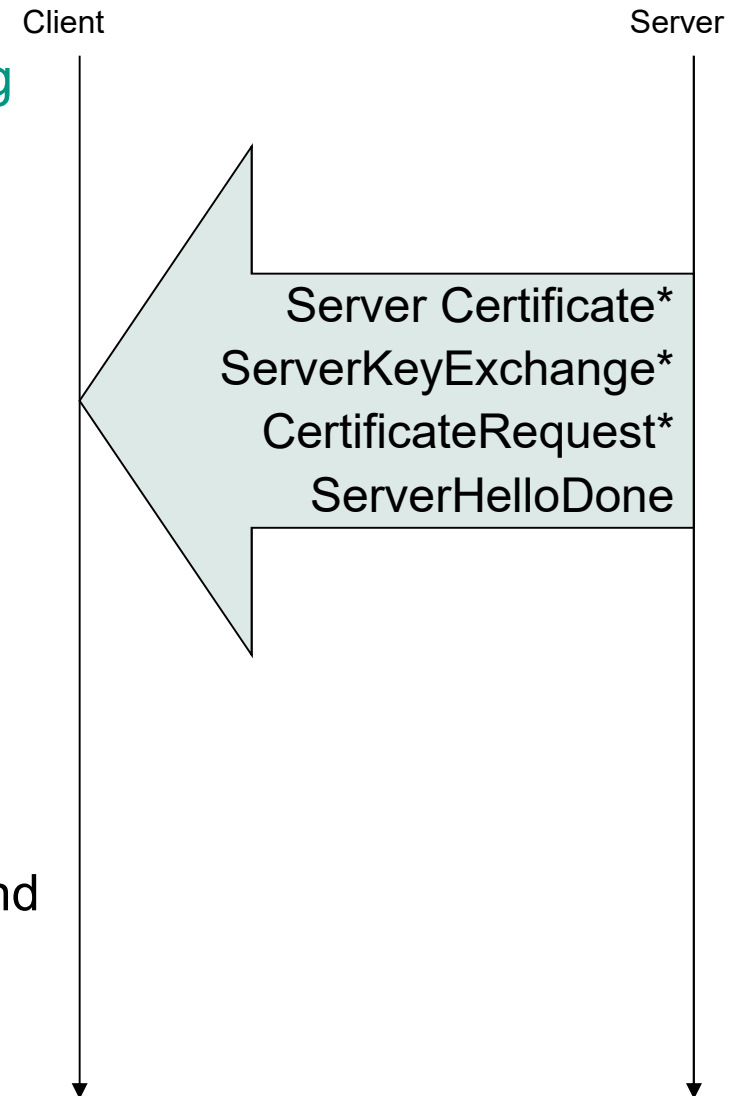
TLS-Schlüsselaustausch im Detail

- Auswahl der kryptografischen Verfahren
- ClientHello enthält
 - Liste an unterstützten Cipher Suites
 - 32 Byte Nonce des Clients
 - 28 Byte Zufallszahl
 - 4 Byte aktuelle Zeit des Client
 - Sitzungsnummer
 - Falls Sitzungswiederaufnahme
- ServerHello enthält
 - Ausgewählte CipherSuite
 - Nonce Server
 - Gewählte Sitzungsnummer



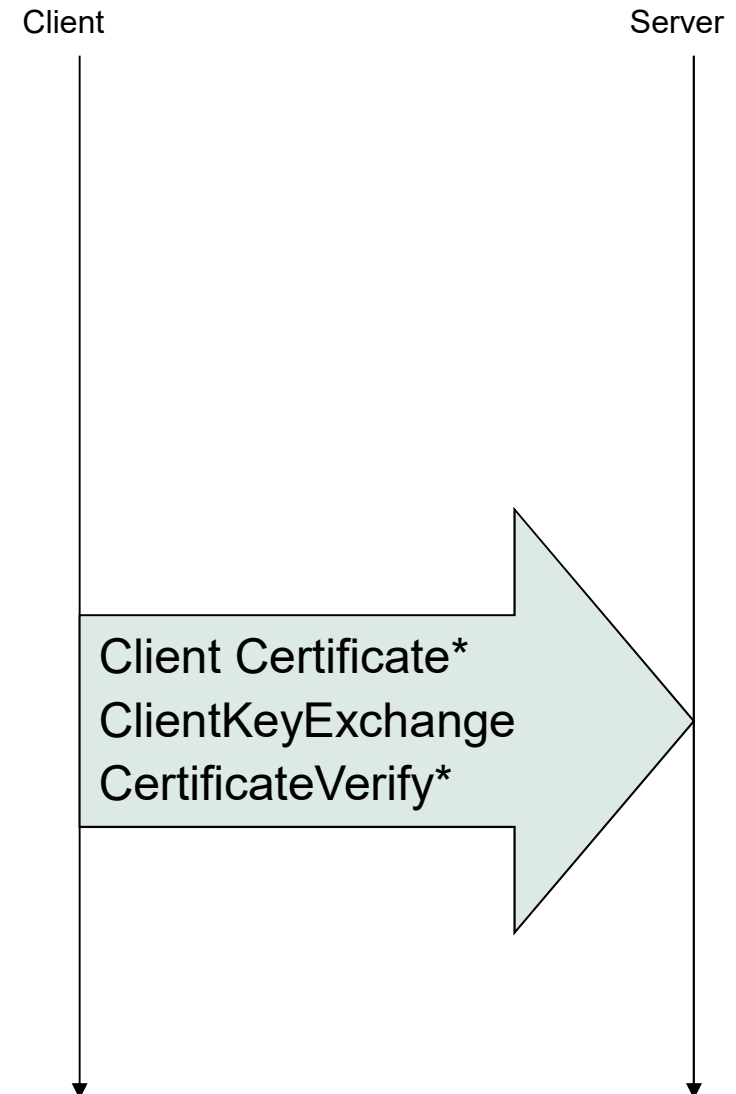
TLS-Schlüsselaustausch im Detail

- Schlüsselaustausch und Authentifizierung
- Server Certificate
 - X509 Zertifikat passend zu gewählter CipherSuite
- ServerKeyExchange
 - Gesendet falls Zertifikat nicht alle Parameter für Schlüsselberechnung enthält
- CertificateRequest
 - Anforderung des Client Zertifikats falls gewünscht
- ServerHelloDone
 - Abschluss des Schlüsselaustauschs und der Authentifizierung des Servers
 - Warten auf Antwort des Client



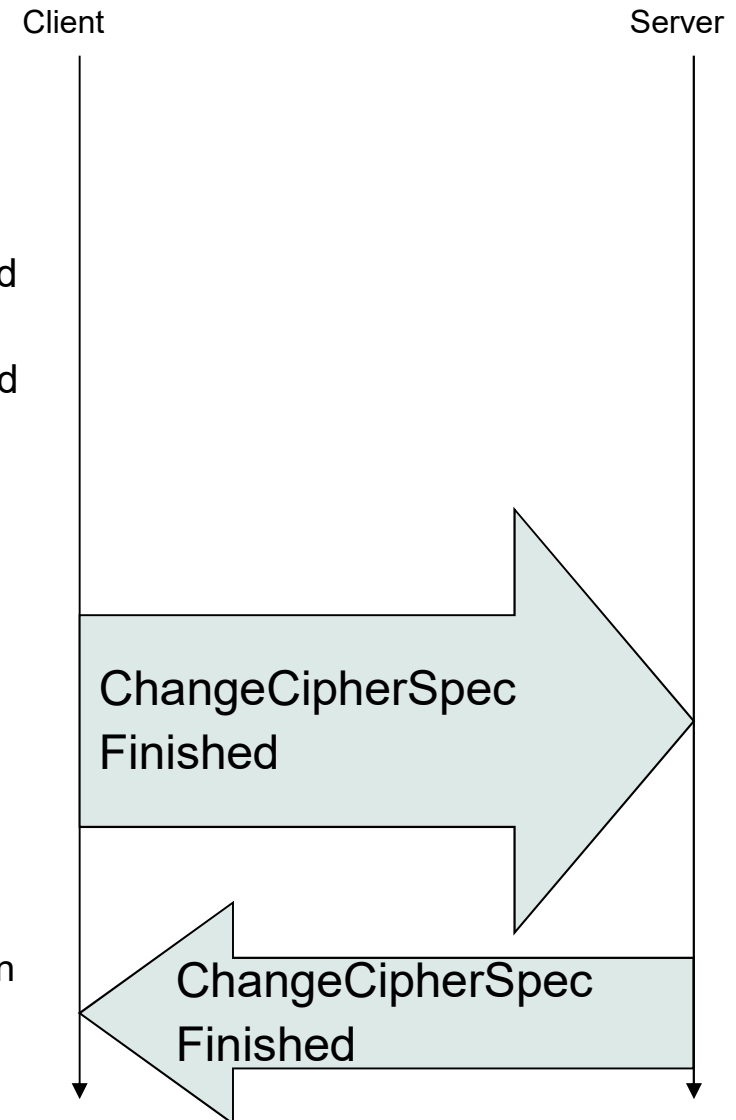
TLS-Schlüsselaustausch im Detail

- Client überprüft nach Empfang von ServerHelloDone
 - Zertifikat des Servers
 - Ausgewählte Parameter
- Client Certificate
 - X509 Zertifikat passend zu gewählter CipherSuite
 - Nur gesendet falls CertificateRequest empfangen wurde
- ClientKeyExchange
 - Enthält entweder PreMaster-Secret oder benötigte Information um dieses zu berechnen
- CertificateVerify
 - Explizite Verifikation des Client Zertifikats
 - ...also Besitz des privaten Schlüssels



TLS-Schlüsselaustausch im Detail

- Abschluss des Schlüsselaustauschs
- ChangeCipherSpec
 - Anzeige, dass ab nun gesichert kommuniziert wird
 - TLS Record Layer nutzt ab diesem Zeitpunkt die ausgehandelten kryptografischen Algorithmen und Schlüssel
- Finished
 - Alle Nachrichten wurden übertragen
 - MAC über Hashwert aller ausgetauschten Nachrichten
 - Muss jeweils von Client und Server überprüft werden
 - → Schutz vor u.a. Downgrade-Angriff
 - Ziel des Angreifers: Client und Server einigen sich auf ein schwache kryptografische Verfahren
 - Angreifer löscht dazu Cipher-Suites aus ClientHello



What could possibly go wrong?



■ Protocol (downgrade) attack

- Aufbau starker TLS-Versionen verhindern z.B. mit **RST-Flag**
 - → Client versucht neuen Verbindungsaufbau mit schwächerer Version
- Nutzung von TLS verhindern z.B. mit **SSLStrip**
 - → Client versucht sich ohne TLS zu verbinden

■ Gegenmaßnahmen?

- Fallback Signaling Cipher Suite Value (SCSV)  [RFC7507]
- Für HTTP z.B. HTTP Strict Transport Security (HSTS)  [RFC6797]
- Verbindung ohne TLS verbieten



What could possibly go wrong?



■ Cipher attack

- RC4 „knacken“
 - → veraltetes Verschlüsselungsverfahren (2011)
- BEAST (Browser Exploit Against SSL/TLS)
 - → AES-CBC Anfälligkeit; veralteter Modus (2011)
- FREAK (Factoring RSA Export Keys)
 - → RSA512 „knacken“; veraltete Parameter (1990);
Standard sollte ≥ 3072 sein

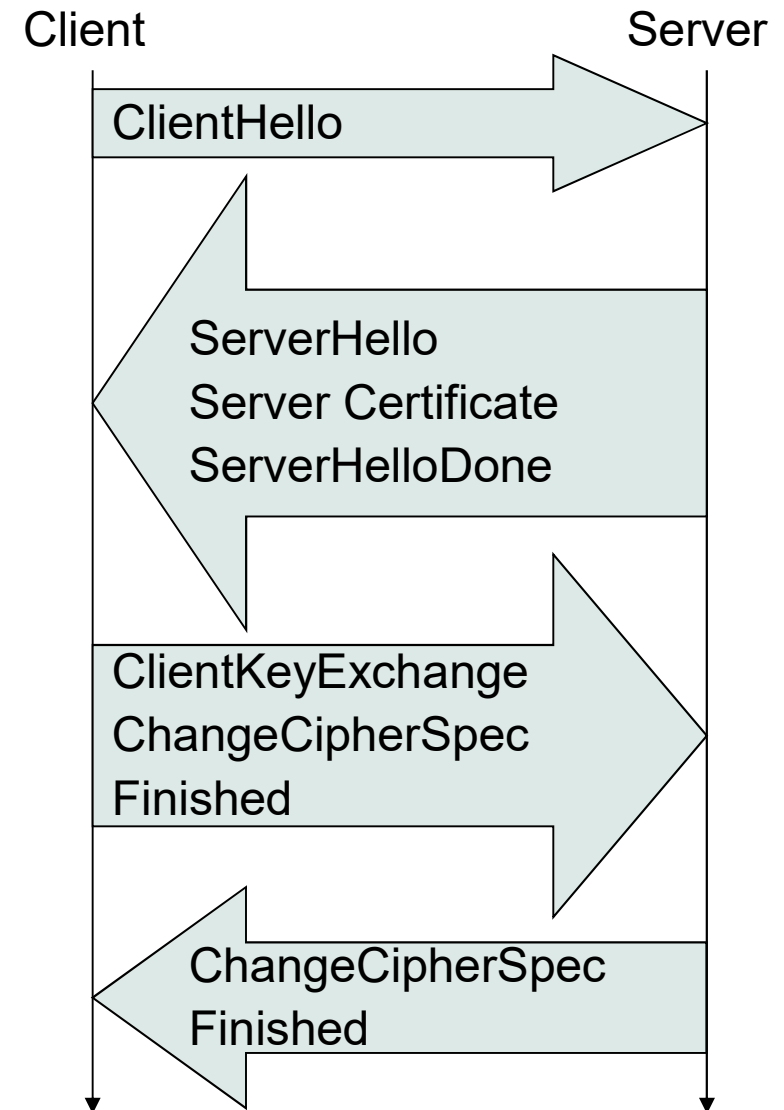
■ Gegenmaßnahmen?

- Regelmäßig Verfahren, Modus und Parameter der Kryptoverfahren aktualisieren
- Unsichere Verfahren, Modi, Parameter nicht aushandeln



Beispiel: TLS-Schlüsselaustausch ohne DH

- **ClientHello**
 - Liste angebotener Cipher-Suites und eine Nonce
- **ServerHello**
 - Gewählte Cipher-Suites und eine Nonce
 - → z.B. TLS_RSA_WITH_...
- **Certificate**
 - RSA Server-Zertifikat
- **ServerHelloDone**
 - Anzeige, dass alle Nachrichten übertragen wurden
- **ClientKeyExchange**
 - RSA-verschlüsseltes, zufällig gewähltes PreMaster-Secret generiert von Client
- **ChangeCipherSpec**
 - Anzeige, dass alle weiteren Nachrichten gesichert werden
- **Finished**
 - Alle Nachrichten wurden übertragen, MAC über alle Nachrichten

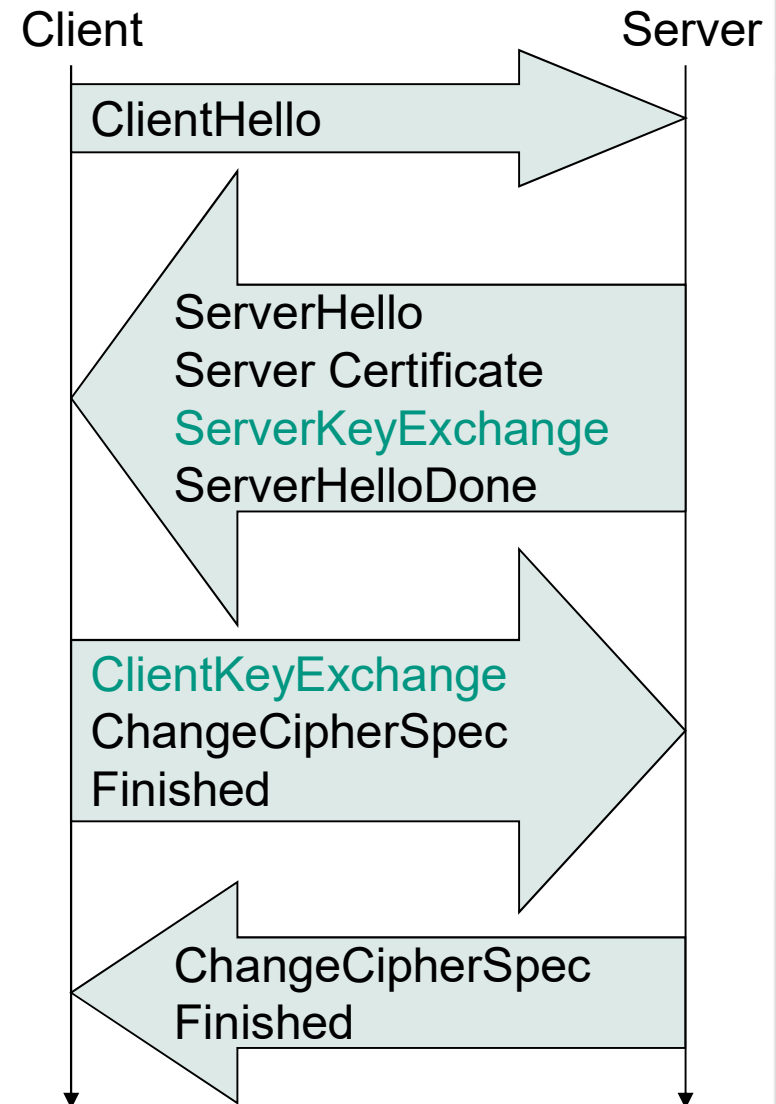


Beispiel: TLS-Schlüsselaustausch mit DH

- Schlüsselaustausch mit Diffie-Hellmann
 - TLS_DHE_RSA_WITH_...
 - **ServerKeyExchange**: DH-Wert des Servers
 - **ClientKeyExchange**: DH-Wert des Client
 - DH-Wert des Servers mittels RSA signiert

- Pre-Master-Secret durch DH erstellt
 - → Client und Server generieren Master-Secret selbst aus Pre-Master-Secret und Nonce

- TLS unterstützt verschiedene DH-Varianten
 - Fester DH-Wert im Zertifikat (**DH**),
 - ephemeral für PFS (**DHE**, ähnl. IKE),
 - anonym (**DH_anon**, ohne Authentifizierung)



What could possibly go wrong?



■ Logjam Angriff

- 2015: Angriff auf Diffie-Hellman-Schlüsselaustausch bei TLS

■ Protocol attack

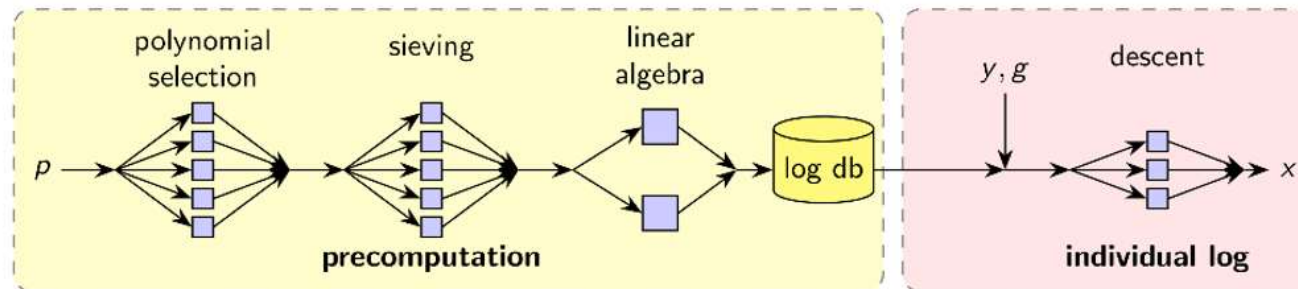
- **Downgrade:** erzwingen von Verwendung **512 Bit Primzahl für DH** mit DH-EXPORT (vergleichbar mit FREAK)

■ Cipher attack

- Viele Webserver verwenden selbe Primzahl für DH
- **Vorbereitung pro Primzahl und Generator möglich (512 Bit in 1–2 Wochen)**
- Danach geringer Aufwand (= wenige Minuten) um individuelle TLS-Verbindung anzugreifen
- Vermutung: NSA könnte auch am häufigsten verwendete 1024-Bit Primzahl vorberechnet haben
→ **Angriff auf 18% der Top-1-Million-Webseiten möglich**

Logjam Detail (I)

Goal: Given $g^x \equiv y \pmod{p}$, compute x .



	Sieving	Linear Algebra	Descent
RSA-512	0.5 core-years	0.33 core-years	
DH-512	2.5 core-years	7.7 core-years	10 core-mins

Precomputation can be done once and reused for many individual logs!

■ Gegenmaßnahmen?

- Verwendung individueller Parameter
- Verwendung hinreichend langer Schlüssel



Logjam Detail (II)



	Sieving core-years	Linear Algebra core-years	Descent core-time
RSA-512	0.5	0.33	
DH-512	2.5	7.7	10 mins
RSA-768	800	100	
DH-768	8,000	28,500	2 days
RSA-1024	1,000,000	120,000	
DH-1024	10,000,000	35,000,000	30 days

- ▶ Special-purpose hardware $\rightarrow \approx 80\times$ speedup
- ▶ $\approx \$100$ Ms machine precomputes for one 1024-bit p every year
- ▶ Then, individual logs can be computed in close to real time

Beispiel: Client-Authentifizierung

■ Client-Authentifizierung

■ CertificateRequest

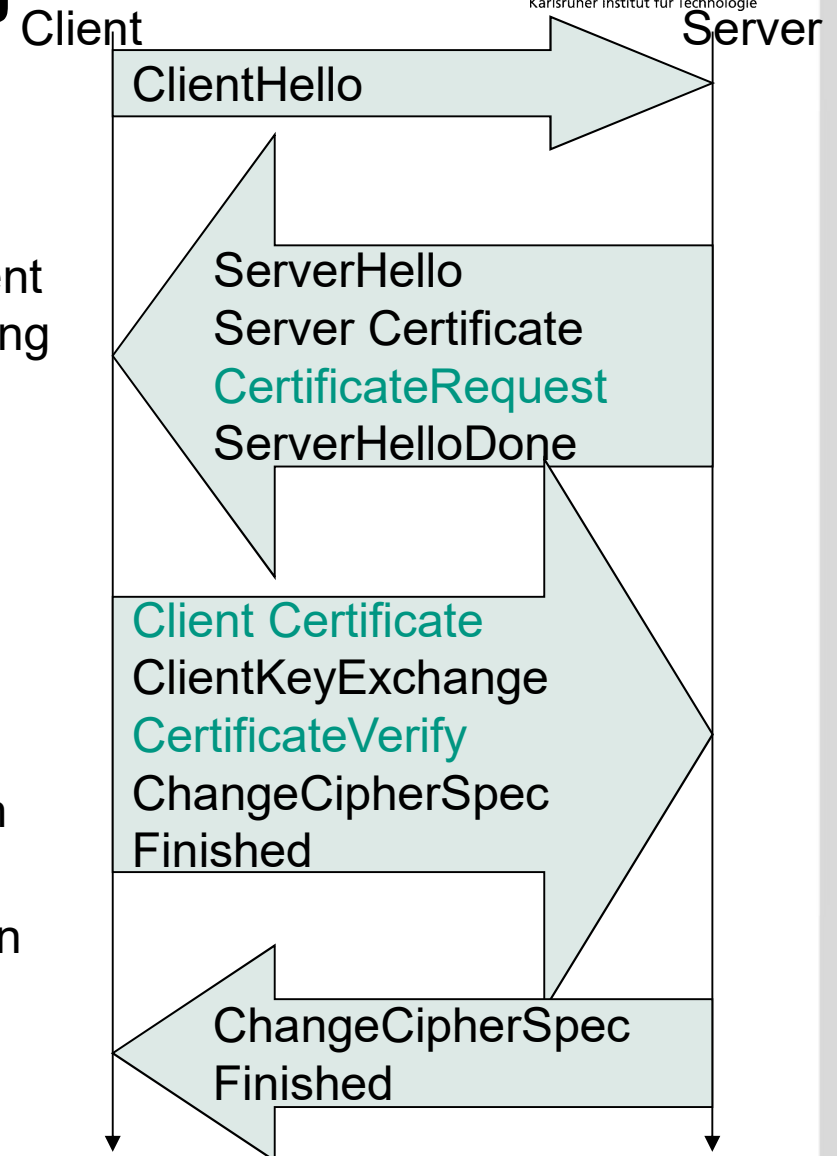
- Anforderung vom Server, dass Client ein X.509-Zertifikat zur Identifizierung verwenden muss

■ Client Certificate

- Enthält das Zertifikat
- Analog zur Zertifikat-Nachricht des Servers

■ CertificateVerify

- Signierter Hash über die bisherigen Nachrichten
- Nachweis, dass der Client auch den dazugehörigen privaten Schlüssel besitzt



TLS Sitzungswiederaufnahme

- Ziel: Vermeidung rechenintensiver Schlüsselaushandlung

- Erster Verbindungsaufbau wie bisher

- Server wählt und sendet Sitzungs-ID in ServerHello
- Speichert Master-Secret mit Sitzungs-ID

- Wiederaufnahme einer Verbindung

- Client verwendet Sitzungs-ID in ClientHello
- Server zeigt Einverständnis durch Übertragung der gleichen Sitzungs-ID in ServerHello
- Überspringen der restlichen Aushandlung

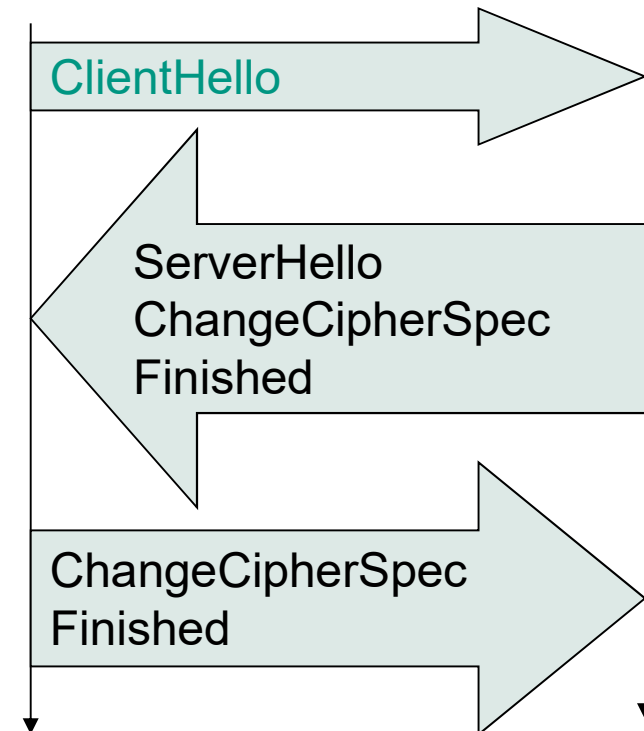
- Diskussion

- Wann benötigt man das Verfahren?
 - Server hält Zustand, andere Möglichkeit?
- Server lagert Zustand durch Cookie aus (RFC 5077)

Client

Server

Zweiter Verbindungsaufbau:



TLS-Rekeying / Renegotiation

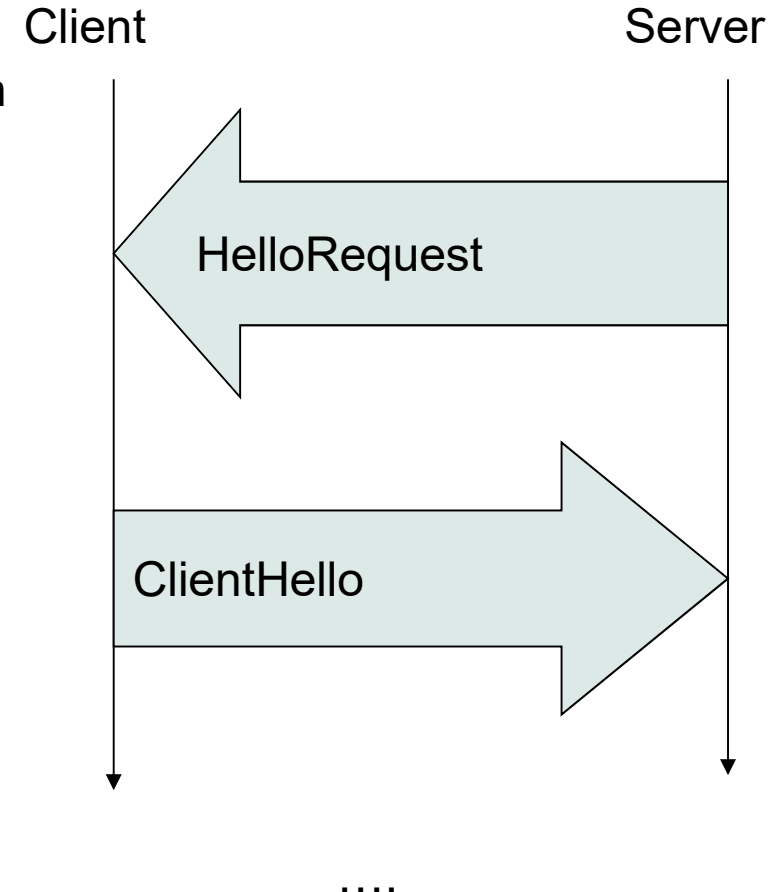
- Ziel: **Neuaushandlung des Schlüsselmaterials**

- Jederzeit von Client oder Server aus möglich

- Aushandlung unter Schutz der bestehenden Verbindung
- Neuaushandlung des Premaster-Secrets

Ablauf

- Triggern des Rekeying
 - Server sendet eine HelloRequest-Nachricht
→ Anforderung an Client Rekeying zu beginnen
 - Client sendet eine ClientHello-Nachricht
- Neuer Handshake wird durchgeführt
- Neuaushandlung vom Client nicht erwünscht
 - Ignorieren der Nachricht oder
 - Alert-Nachricht



What could possibly go wrong?



■ Protocol attack

- Renegotiation Angriff - Angriff auf SSL 3.0 (2009)
- Zuerst TLS-Verbindung zwischen Angreifer und Server aufbauen
- Dann Renegotiation veranlassen und zwischen Client und Server vermitteln
- TLS-Verbindungsaufbau des Clients findet im TLS-Kontext des Angreifers statt

■ Gegenmaßnahmen?

- Renegotiation deaktivieren
- Der Hello-Message Inhalte der letzten Finish-Message hinzufügen

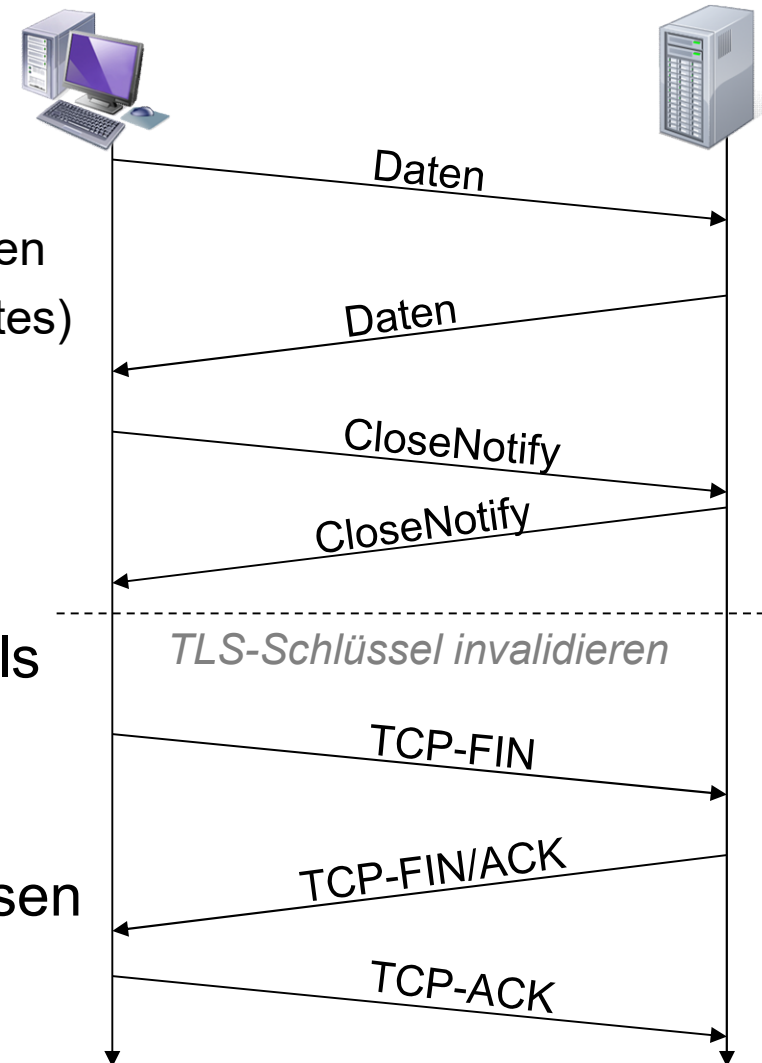


TLS-Verbindungsabbau

- Expliziter Verbindungsabbau notwendig
 - Sonst: **Truncation Angriff** möglich
 - Client will TLS-Verbindung schließen
 - Angreifer schleust (unverschlüsseltes) TCP-FIN Packet ein
 - Auslogversuch des Client kommt daher nie bei Server an
- Lösung: Nutzung des Alert Protokolls
 - **CloseNotify Nachricht**
- Beide Kommunikationspartner müssen CloseNotify verschicken

Client

Server



TLS-Alert Protokoll

■ Behandlung von Fehlerzuständen

■ Zwei Arten von Fehlern

- **Fataler Fehler:** sofortiger Verbindungsabbruch
- **Hinweis:** CloseNotify oder NoRenegotiation
 - später mehr zu CloseNotify

■ Fehler des Zertifikats

- Signatur-Algorithmus nicht unterstützt
- Signatur falsch
- Zertifikat zurückgezogen
- CA nicht vertrauenswürdig

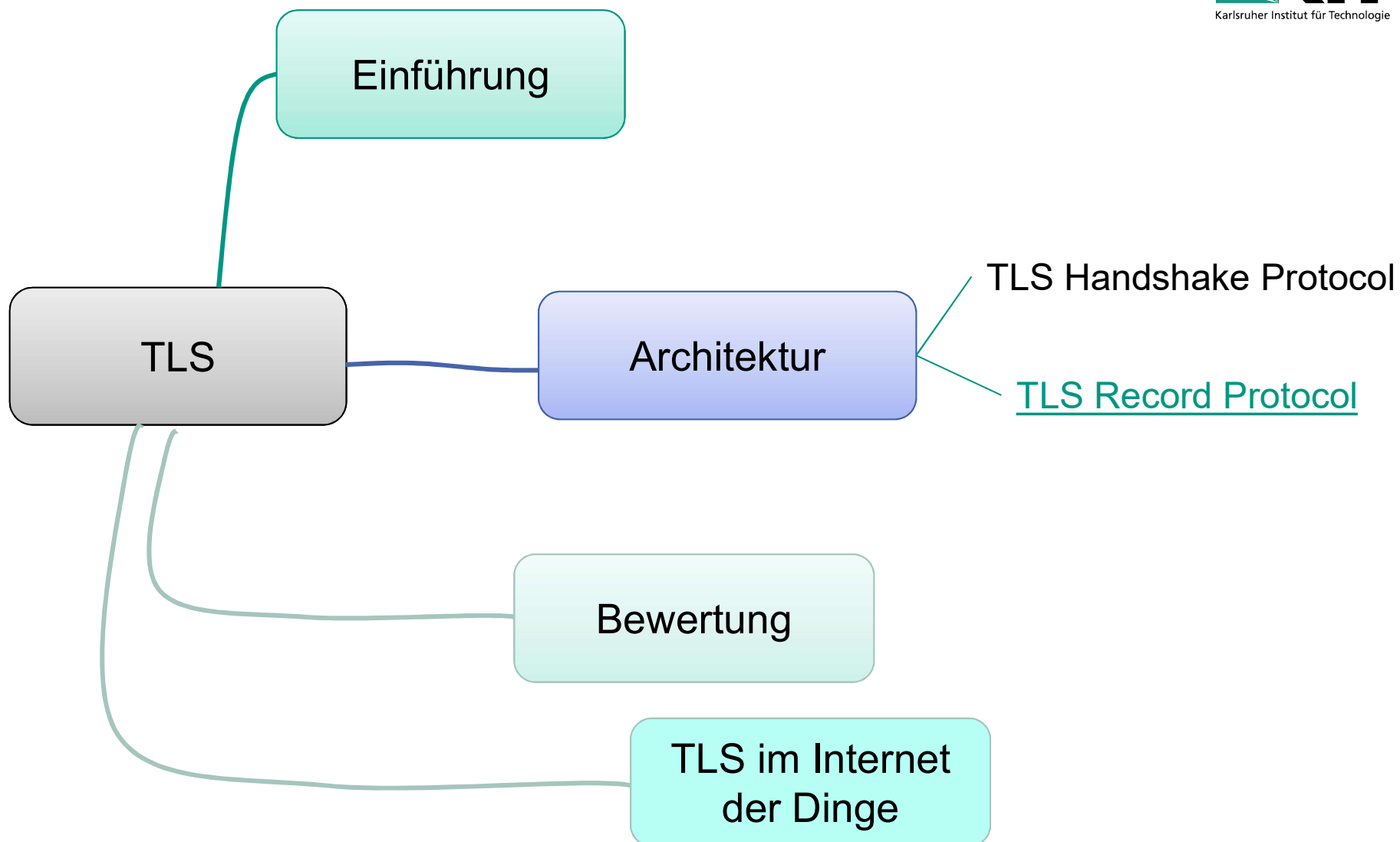
■ Fehler beim Datenempfang

- MAC des Schlüsselaustauschs falsch
- entschlüsseltes Paket nicht authentisch
- überlanges Paket

```
enum { warning(1), fatal(2), (255) } AlertLevel;
```

```
enum {
    close_notify(0),
    unexpected_message(10),
    bad_record_mac(20),
    decryption_failed(21),
    record_overflow(22),
    decompression_failure(30),
    handshake_failure(40),
    bad_certificate(42),
    unsupported_certificate(43),
    certificate_revoked(44),
    certificate_expired(45),
    certificate_unknown(46),
    illegal_parameter(47),
    unknown_ca(48),
    access_denied(49),
    decode_error(50),
    decrypt_error(51),
    export_restriction(60),
    protocol_version(70),
    insufficient_security(71),
    internal_error(80),
    user_canceled(90),
    no_renegotiation(100),
    (255) } AlertDescription;
```

```
struct {
    AlertLevel level;
    AlertDescription description; } Alert;
```



TLS Record Protocol

■ Aufgaben des TLS Record Protocol

- Verwaltung der TLS-Sitzung
- Fragmentierung (und Komprimierung) von Anwendungsdaten
- Kryptografische Verarbeitung

■ Record Protocol erfüllt zwei Schutzziele

- Vertraulichkeit und Integritätssicherung



■ Kryptografische Verarbeitung der zu übertragenden Daten

- Symmetrische Kryptografie zum Schutz der Vertraulichkeit
- Integritätssicherung mittels (H)MAC
- Schlüsselmaterial muss durch anderes Protokoll (TLS Handshake Protocol) ausgehandelt werden
- Record Protocol nutzbar für Protokolle der höheren Schichten
 - z.B. TLS Handshake Protocol, TLS Alert Protocol, ...

TLS-Sitzung vs. Verbindung

- **Sitzung** spezifiziert den Kontext des Record Protocols
 - TLS bietet Dienste aus Schicht 5 des ISO/OSI Modells

- Sitzung vs. Verbindung
 - Sitzung speichert Zustandsinformationen länger als bsp. TCP-Verbindung
 - Sitzung kann mehrere Verbindungen zusammenfassen
 - Eine Verbindung gehört genau zu einer Sitzung
 - Beispiel: Zustandsloses HTTP greift mit mehreren TCP-Verbindungen auf Webseite zu
 - → TLS-Sitzung umfasst alle Verbindungen
 - Verwaltung kann effizienter erfolgen als für Einzelverbindungen

- Sitzungsverwaltung Aufgabe des Record Layers

TLS-Sitzungszustand

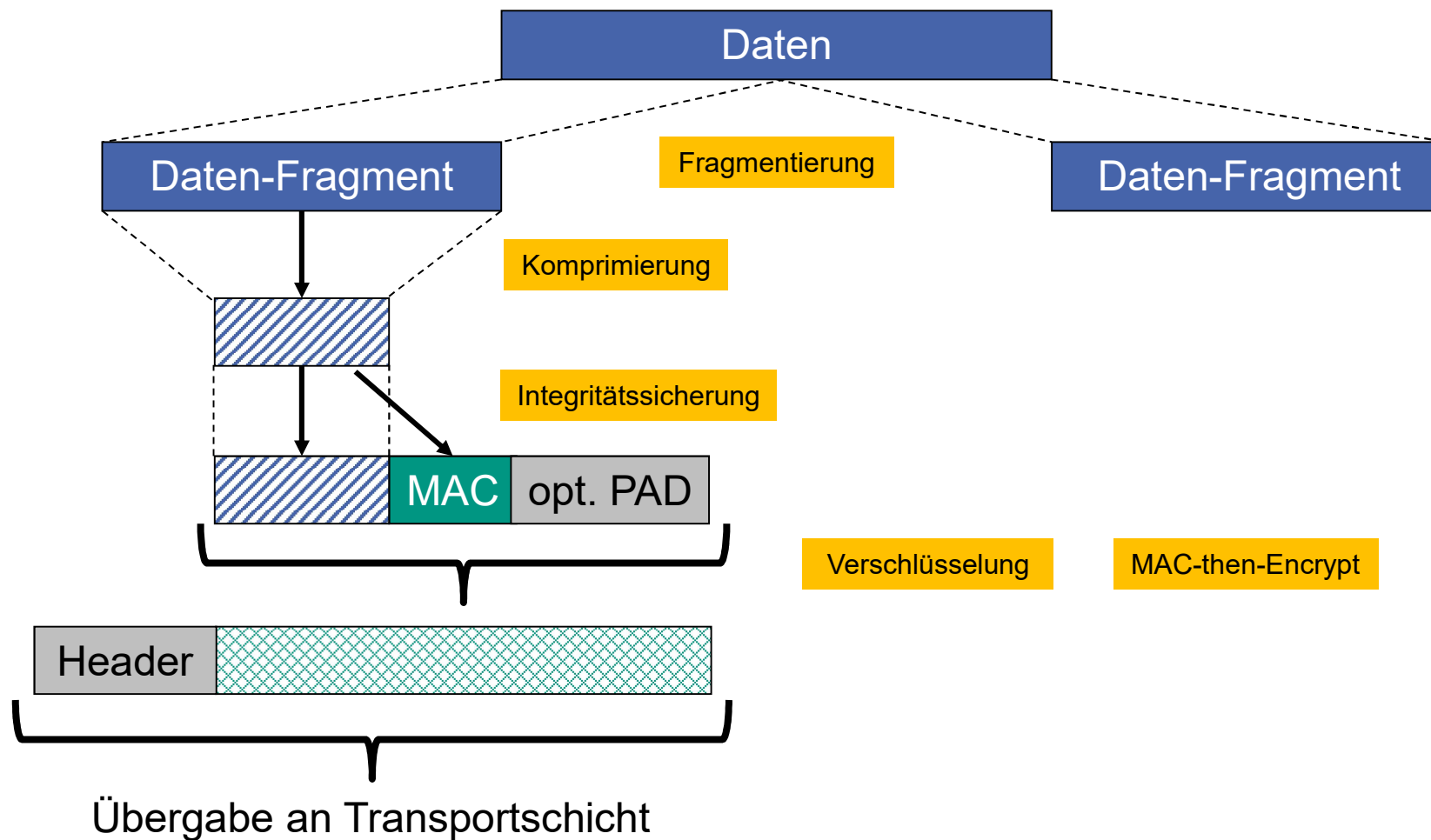
- Sitzung ist charakterisiert durch
 - **Identifikator**
 - Vom Server während Handshake generiert, siehe Sitzungswiederaufnahme
 - **Zertifikat**
 - X509 Zertifikat des Kommunikationspartners
 - **Komprimierungsverfahren**
 - Default: NULL
 - **CipherSpec**
 - Verwendete CipherSuite inklusive kryptografischer Attribute
 - **Master Secret**
 - Zwischen Client und Server berechnetes Schlüsselmaterial

TLS-Verbindungszustand

- Verbindung ist charakterisiert durch
 - Client- und Serverzufallszahlen
 - Austausch siehe Handshake, Nutzung um u.a. Master Secret zu berechnen
 - Client- bzw. Server-Schlüssel für MAC
 - Client- bzw. Server-Schlüssel für Verschlüsselung
 - Initialisierungsvektor
 - Sequenznummer
 - Jeder Kommunikationspartner verwaltet Sequenznummern für gesendete und empfangene Pakete
 - Erhöhen der Sequenznummer nach jedem TLS-Record
 - Maximale Größe von $2^{64} - 1$
 - Bei Überlauf Neuaushandlung der Verbindung notwendig
 - Empfang der ChangeCipherSpec Nachricht setzt Sequenznummer auf 0

TLS Record Layer – Beispiel

- Sender will **Daten** TLS gesichert übertragen



TLS Record Layer – Ablauf

■ Grundlegender Ablauf als Sender

- Anwendung übergibt zu übertragene Daten beliebiger Länge
- Fragmentierung der Daten sofern notwendig
- Komprimierung der Daten
- Integritätssicherung mittels MAC
- Verschlüsselung
- Übergabe an Transportschicht

■ Grundlegender Ablauf als Empfänger

- Empfang der Datenpakete aus Transportschicht
- Entschlüsselung
- Verifizierung des MAC
- Dekomprimierung
- Defragmentierung
- Übergabe an Anwendung

TLS Record Layer – Fragmentierung

■ Fragmentierung an Anwendungsschnittstelle

- Anwendung $\leftrightarrow$ TLS
- ...nicht mit TCP-Segmentierung verwechseln!

■ Record Layer erhält Datenblöcke beliebiger Länge von Anwendung

- Datenblock wird in TLS-Fragmente von maximal 2^{14} Byte zerteilt
 - Optional: Datenblöcke kleinerer Länge können in TLS-Fragment zusammengefasst werden
- Anwendung kann auch Datenblöcke der Größe 0 Byte versenden
 - Ggf. sinnvoll zur Erschwerung von Verkehrsanalyse

TLS Record Layer – Komprimierung

- Optional führt TLS **Komprimierung der Anwendungsdaten** durch
 - Default Algorithmus: NULL
 - Aushandlung des genutzten Komprimierungsverfahren im Handshake
 - Komprimierung muss verlustfrei sein
 - Komprimierung darf Daten nicht um mehr als 1024 Byte verlängern

- Warum Komprimierung?
 - Geringerer Kommunikationsaufwand
 - Verringerung der Redundanz der zu verschlüsselnden Daten

What could possibly go wrong?



■ Protocol attack + crypto attack

- CRIME (Compression Ratio Info-leak Made Easy)
- TIME (Timing Info-leak Made Easy)
- BREACH (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext)

■ Komprimierung führt zu Sicherheitslücken

- Chosen Plaintext Attack
- Angreifer „testet“ welcher bekannte Klartext komprimiert werden kann (Länge) == zu suchendes Geheimnis (Cookie)

■ Gegenmaßnahmen?

- Komprimierung verbieten
- Cross-site request forgery (CSRF) protection



TLS Record Layer – Integritätssicherung

- TLS nutzt zur **Integritätssicherung** einen MAC
 - Nutzbare Hashfunktionen laut RFC 5246 MD5, SHA1, SHA224, SHA256, SHA384 und SHA512
- Eingabewerte für den MAC bei Verschlüsselung mit CBC Block Cipher oder Stromchiffren
 - Client- bzw. Server-Schlüssel für MAC
 - Sequenznummer
 - Fragmentierte und komprimierte Dateneinheit
 - Plus entsprechende Headerinhalte
 - Der MAC wird berechnet als
 - $\text{MAC}(\text{MAC_write_key}, \text{seq_num} + \text{TLSCompressed.type} + \text{TLSCompressed.version} + \text{TLSCompressed.length} + \text{TLSCompressed.fragment})$
- **Kombinierte Verschlüsselungsverfahren** wie CCM bzw. GCM benötigen keine zusätzliche Integritätssicherung
 - Wird bei Verschlüsselung gleichzeitig durchgeführt: **Authenticated Encryption**

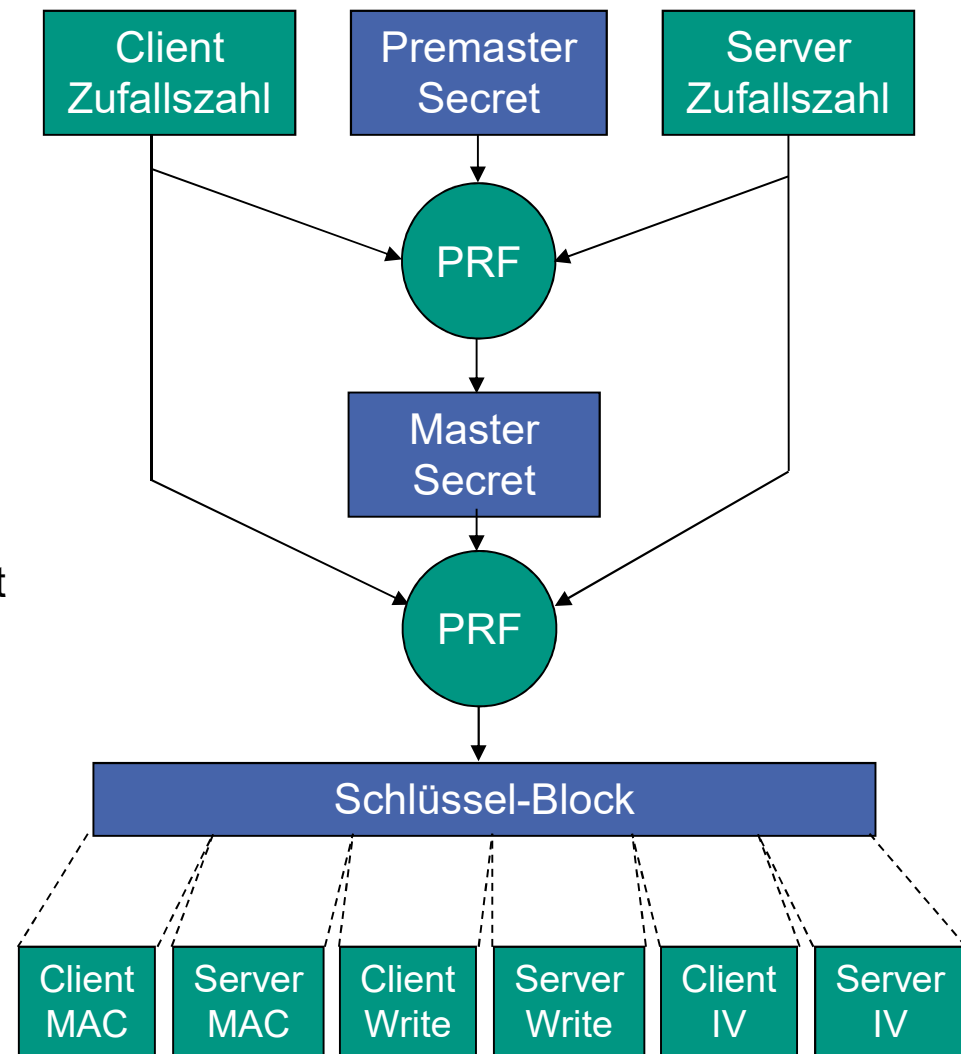
TLS Record Layer – Verschlüsselung

- Verschlüsselung mit unterschiedlichen **symmetrischen** Verfahren möglich
 - Stromchiffren
 - Blockchiffren
 - Kombinierte Verfahren

- Eingesetztes Verfahren abhängig von ausgehandelter CipherSuite
 - Ebenso die Schlüssellänge

TLS Record Layer – Berechnung des Schlüsselmaterials

- Aus Handshake
 - Premaster Secret
 - Generierung des Master Secrets innerhalb des Record Protokolls
- Erzeugung Schlüsselmateriale
 - Verwendung einer **Pseudo-Zufalls-Funktion (PRF)** in der Schlüsselerzeugungsfunktion
 - Premaster Secret ist geheim, beide Zufallszahlen sind öffentlich bekannt
- $\text{MasterSecret} = \text{PRF}(\text{PreMasterSecret}, \text{"master secret"}, \text{ClientHello.rand} \parallel \text{ServerHello.rand})$;
- Länge des Key Block ist Summe der Schlüssellängen
 - Zerschneiden des Key-Blocks in Schlüssel



Premaster Secret vs. Master Secret

- Warum nicht direkt Premaster Secret verwenden?
 - Master Secret muss ausreichende Länge haben
 - Damit genug Schlüsselmateriale generiert werden kann
 - Client Zufallszahl möglicherweise nicht sicher
 - Server will auch beitragen, und andersherum
- Wiederholungsangriffe
 - Angreifer könnte ohne die Zufallszahlen das verschlüsselte Premaster Secret wieder einspielen

What could possibly go wrong?



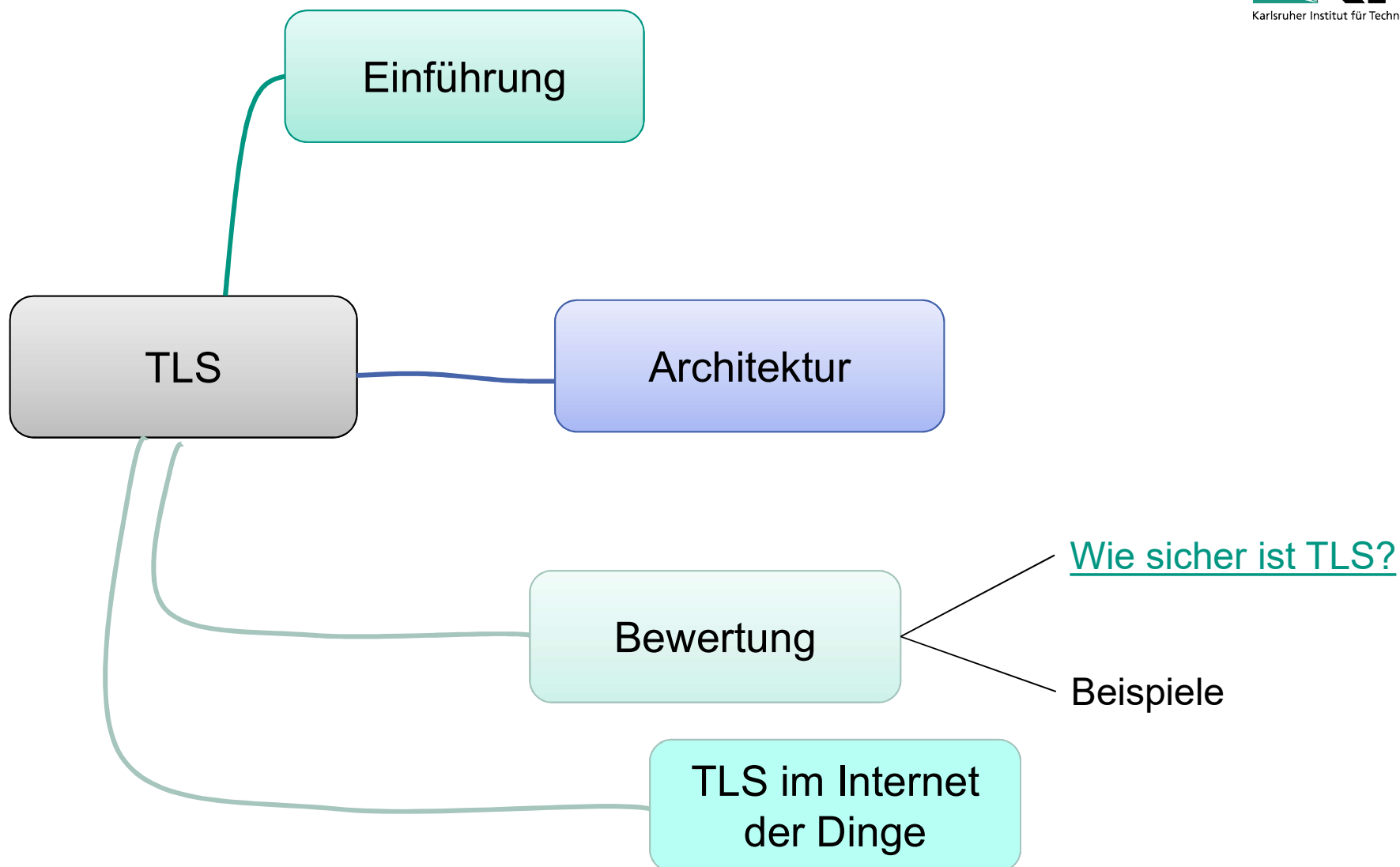
■ Crypto attack

- Lucky Thirteen attack
- Poodle
- DROWN: SSLv2 -> Oracle TLS
- **Padding Oracle**
 - multi-session timing attack
 - Ausnutzung unterschiedlicher Alerts (Encrypt-Error; MAC-Error)

■ Gegenmaßnahmen?

- Encrypt-then-MAC
- Authenticated encryption (z.B. AES-GCM)





Wie sicher ist TLS?

- Unterschiedliche Betrachtungsweisen
 - Welche (bekannten) Angriffe auf TLS gibt es?
 - Welche Programmierfehler führten zu Angriffen auf TLS?
 - Welche Protokollversion unterstützt der Server/Browser?
 - Welche Cipher-Suites sind verfügbar bzw. werden ausgehandelt?

- Wie kann ein Nutzer das erkennen?
 - <https://www.trustworthyinternet.org/ssl-pulse/>
 - Untersuchung der SSL bzw. TLS-Kompatibilität bzw. Konfiguration der TOP 1 Mio. Webseiten im Internet
 - Zusätzlich gezielte Abfrage beliebiger Webseiten möglich

Welche Protokollversionen sind verbreitet?

- Historisch bedingt existieren unterschiedliche Versionen von SSL bzw. TLS
 - Neuste Version gemeinhin sicherer als ältere Versionen
 - Unterschiedliche Angriffe auf SSL bzw. TLS haben unter anderem zu neuen TLS Versionen geführt
- Verbreitung der unterschiedlichen TLS-Versionen Stand 6/2017
 - <https://www.trustworthyinternet.org/ssl-pulse/>

unsicher
bedingt
sicher

relativ
sicher

Protokollversion	Anteil der Webseiten mit Unterstützung + Änderungen zum Vorjahr	
SSL 2.0	4,3%	-43,4%
SSL 3.0	15,1%	-34,1%
TLS 1.0	93,4%	-4,0%
TLS 1.1	84,0%	+10,7%
TLS 1.2	87,3%	+11,5%

What could possibly go wrong?



■ Implementation attack

- Apple iPhone „goto fail“ (2014)
 - Simpler Tippfehler „eines“ Programmierers beim SSL Authentication Check
- Heartbleed
 - Fehlerhafte Implementierung in *optionaler* TLS-Heartbeat-Funktion von OpenSSL
 - Erlaubt remotes auslesen von Arbeitsspeicher == auslesen des SK

```
static OSStatus  
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,  
                                uint8_t *signature, UInt16 signatureLen)  
{  
    OSStatus      err;  
    ...  
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)  
        goto fail;  
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)  
        goto fail;  
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)  
        goto fail;  
    ...  
fail:  
    SSLFreeBuffer(&signedHashes);  
    SSLFreeBuffer(&hashCtx);  
    return err;  
}
```

■ Gegenmaßnahmen?

- Qualitätssicherung / Code Review
- Externes Code Audit



RFC7457 – Summarizing Known Attacks



■ Zusammenfassung möglicher Schwachstellen von TLS

■ Protocol attack

- z.B. Downgrade, SSLStrip, CRIME, ...

■ Crypto attack

- z.B. Padding Oracle, logjam, FREAK, ...

■ Implementation attack

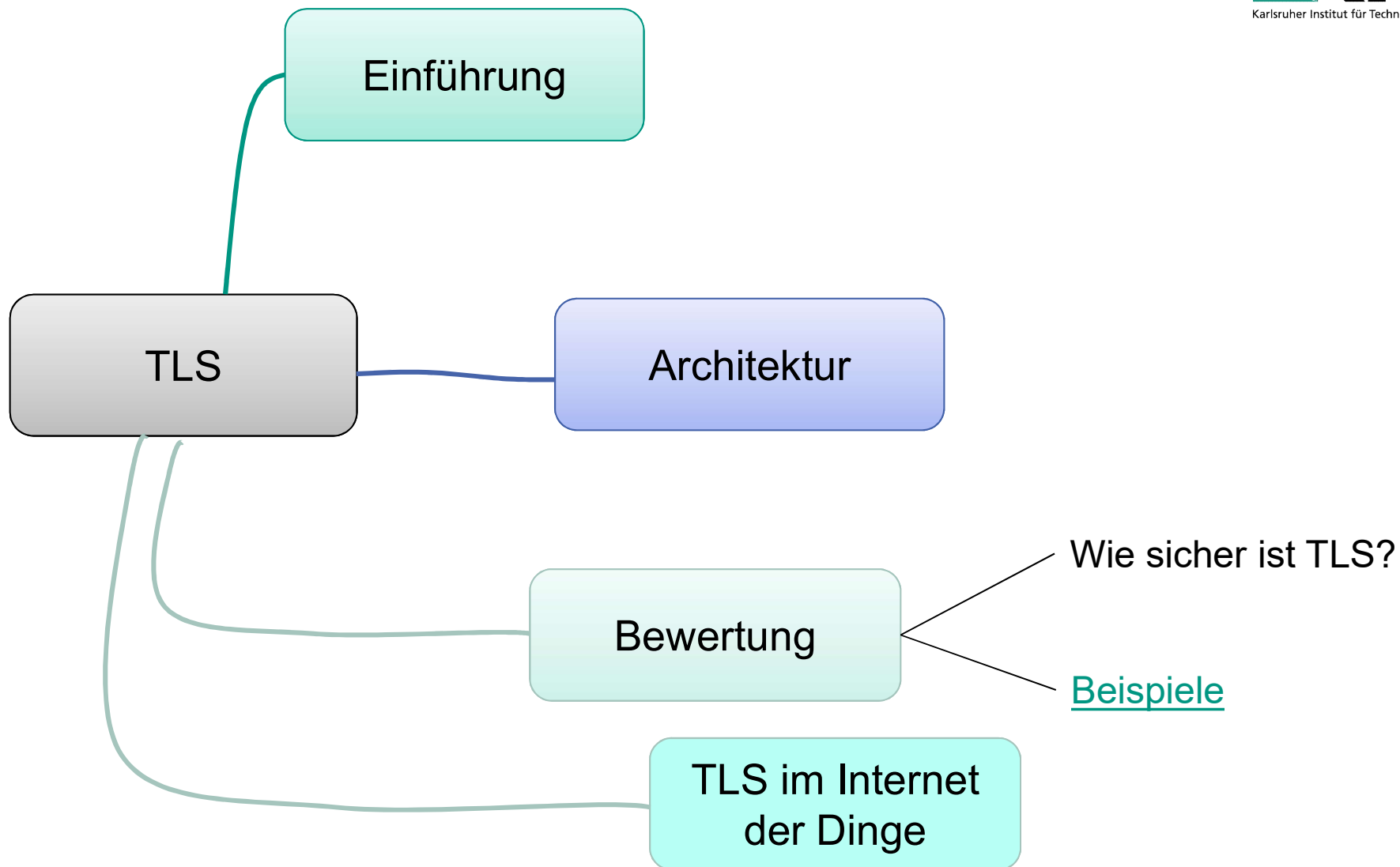
- z.B. Heartbleed, goto fail, ...

■ Fehlt da noch was?

■ Endsystem / Nutzer!

■ Zertifikate





Beispiel

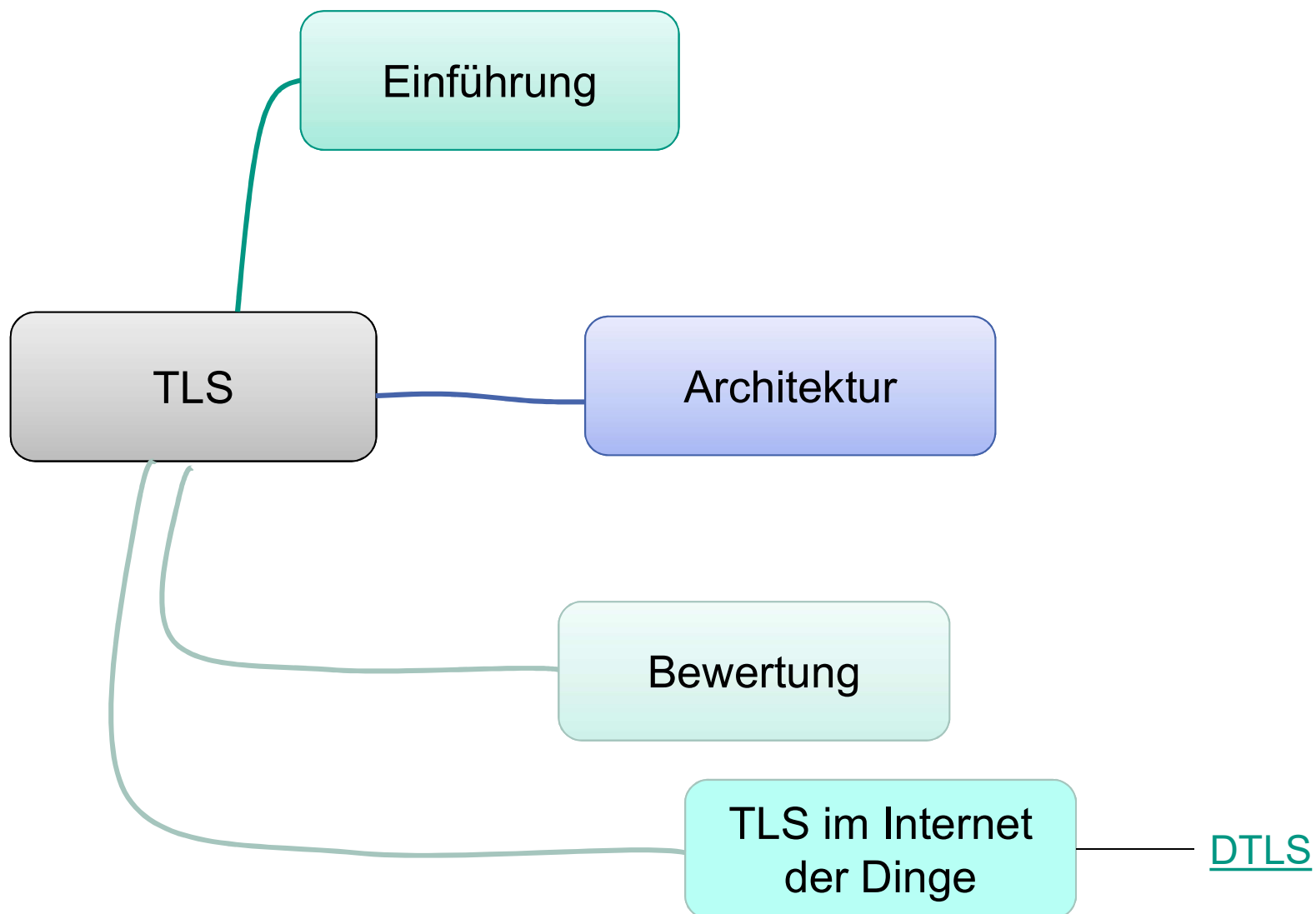
- Überprüfung des Campus Managements einer deutschen Hochschule
 - Note **F** (A beste Note)
 - “is vulnerable to the DROWN attack”
 - “supports SSL 2, which is obsolete and insecure”
 - “supports insecure Diffie-Hellman (DH) (logjam)”
 - “is vulnerable to the POODLE attack”
 - “is vulnerable to MITM attacks because it supports insecure renegotiation”
 - “supports 512-bit export suites and might be vulnerable to the FREAK attack”
 - “does not mitigate the CRIME attack”
 - “certificate has a weak signature and expires after 2015”
 - “supports only older protocols, but not the current best TLS 1.2”
 - “accepts RC4 cipher, but only with older protocol versions”
 - “there is no support for secure renegotiation”
 - “does not support Forward Secrecy with the reference browsers”

Beispiel campus.studium.kit.edu

- Überprüfung des Campus Management des KITs führt zu
 - Note **A+** (A beste Note)
 - “HTTP Strict Transport Security (HSTS) with long duration deployed on this server”

Unterstützte Protokolle	
TLS 1.2	Yes
TLS 1.1	Yes
TLS 1.0	Yes
SSL 3	No
SSL 2	No

Angebotene Cipher Suites mit Schlüssellänge für Verschlüsselung (suites in server-preferred order;)	
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	256
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	128
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	128
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	128
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	128
TLS_DHE_RSA_WITH_CAMELLIA_256_CBC	256



TLS im Internet der Dinge

- TLS entworfen oberhalb eines zuverlässigen Transportprotokolls
- Anwendungen verwenden immer häufiger UDP
 - VoIP
 - Constrained Application Protocol (CoAP)
 - Kein zuverlässiges Transportprotokoll
- Wo genau benötigt TLS ein zuverlässiges Transportprotokoll?
 - Handshake Protocol
 - Zuverlässigkeit des Handshake muss garantiert sein
 - Paketverluste führen sonst zu neuem Handshake!
 - Record Protocol
 - Kryptografische Verarbeitung der Pakete nicht unabhängig voneinander möglich, z.B. Integritätsschutz über Sequenznummer, implizit in Paket enthalten
 - Erkennung von Duplikaten

TLS + UDP = DTLS

■ RFC 6347: Datagram Transport Layer Security Version 1.2

- Ziel: Zu TLS äquivalente Sicherheit auch für unzuverlässige Transportprotokolle bereitstellen

■ Umsetzung durch

- Einführen von eigenen Sequenznummern im Record Protocol
 - Schutz vor Wiedereinspielungsangriffen
 - Sliding Window zur Duplikaterkennung
- Keine Verwendung von Stromchiffren im Record Layer
 - Abhängigkeit zwischen aufeinanderfolgenden Paketen vermeiden
- Modifizierung des Handshake Protocol
 - Umgang mit verlorenen Paketen → Einführung von Retransmission-Timern

→ Mehr dazu in der Vorlesung **Internet of Everything** (im Wintersemester)

Ausblick: Laufende Standardisierung TLS 1.3

■ Draft: Transport Layer Security (TLS) Protocol Version 1.3

- <https://tswg.github.io/tls13-spec/> (aktuell: draft-ietf-tls-tls13-21)
- Finale Verabschiedung des Standards in den nächsten Monaten

■ Erhöhung der Sicherheit durch Einschränkung der Verfahren

- Schwache Verschlüsselungsverfahren wurden entfernt: Es werden nur noch moderne **AEAD-Verfahren** (z.B. AES-GCM) unterstützt
 - Auch neues Format für Cipher Suites definiert
- Schlüsselaustausch per **RSA** oder **(statischem) DH** wurde **entfernt**
→ nur noch Verfahren, die PFS bieten, werden unterstützt
- **Komprimierung** wurde **entfernt**
- Altes Verfahren zur Versionsaushandlung entfernt
- **ChangeCipherSpec-Nachricht** entfernt
- Sitzungswiederaufnahme vereinfacht

Ausblick: Laufende Standardisierung TLS 1.3

■ Hinzufügen von neuen sicheren Verfahren

- Handshake-Nachrichten nach ServerHello sind **verschlüsselt**
→ Schutz der Privatsphäre
- Moderne Kryptoverfahren (u.a. **Curve25519** und **Ed25519**) hinzugefügt
- Neue Funktion (HKDF) zur Berechnung der einzelnen Schlüssel

■ Verbesserung der Performanz

- Neuer Handshake erlaubt bereits nach **1 RTT** das Senden von Nutzdaten
- **0-RTT Modus** zur schnellen Sitzungswiederaufnahme unter bestimmten Bedingungen → Kontroverse Diskussionen bzgl. Sicherheit von 0-RTT

Zusammenfassung TLS

■ Transport Layer Security: TLS

- TLS erzeugt einen sicheren Kanal
- Ende-zu-Ende, pro TCP Socket von Anwendung implementiert
- Standardprotokoll zur sicheren Kommunikation im Internet (z.B. *HTTPS*)

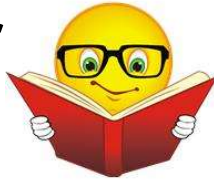
■ Vorgehensweise

- Schlüsselaustausch und Authentifizierung mittel **Handshake Protocol**
- Schutz der zu übertragenden Daten im **Record Protocol**
 - Authentizität und Integrität mittels kryptografischer Hashfunktionen
 - Vertraulichkeit mittels symmetrischer Verschlüsselung

■ Weiteres

- TLS immer wieder Ziel von Angriffen
 - Unterscheide Angriffe auf Protokoll vs. Angriff auf Implementierung
- TLS nur für zuverlässige Transportprotokolle geeignet
 - Sonst DTLS
- TLS 1.3 bringt wesentliche Neuerungen

Literatur



- [RFC5246] Dierks, E. Rescorla; The Transport Layer Security (TLS) Protocol Version 1.2, RFC 5246, August 2008, aktueller TLS Standard
- [RFC6347] E. Rescorla, N. Modadugu; Datagram Transport Layer Security Version 1.2, RFC 6347, Januar 2012, aktueller DTLS Standard
- [RFC6797] J. Hodges, C. Hackson, A. Barth; HTTP Strict Transport Security (HSTS), RFC 6797, November 2012
- [RFC7505] B. Moeller, A. Langley; TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks, RFC 7505, April 2015
- [Rescorla] E. Rescorla; SSL and TLS – Designing and Building Secure Systems, Addison-Wesley, 2001
- [Kau02] C. Kaufman; Network Security: Private Communication in a Public World, 2nd Edition, Prentice Hall, 2002
- [Pulse] <https://www.trustworthyinternet.org/ssl-pulse/>