TLS, IPSEC UND WIREGUARD ALS BEISPIELE FÜR SICHERHEITSPROTOKOLLE

Allgemeines

- Kombination von:
  - Schlüsselaustausch
    - Asymmetrisch Kryptographie
      - RSA
      - Diffie-Hellman
      - Public Key Infrastructure (PKI): Zertifikate
    - Symmetrisch
      - Pre-Shared Secret

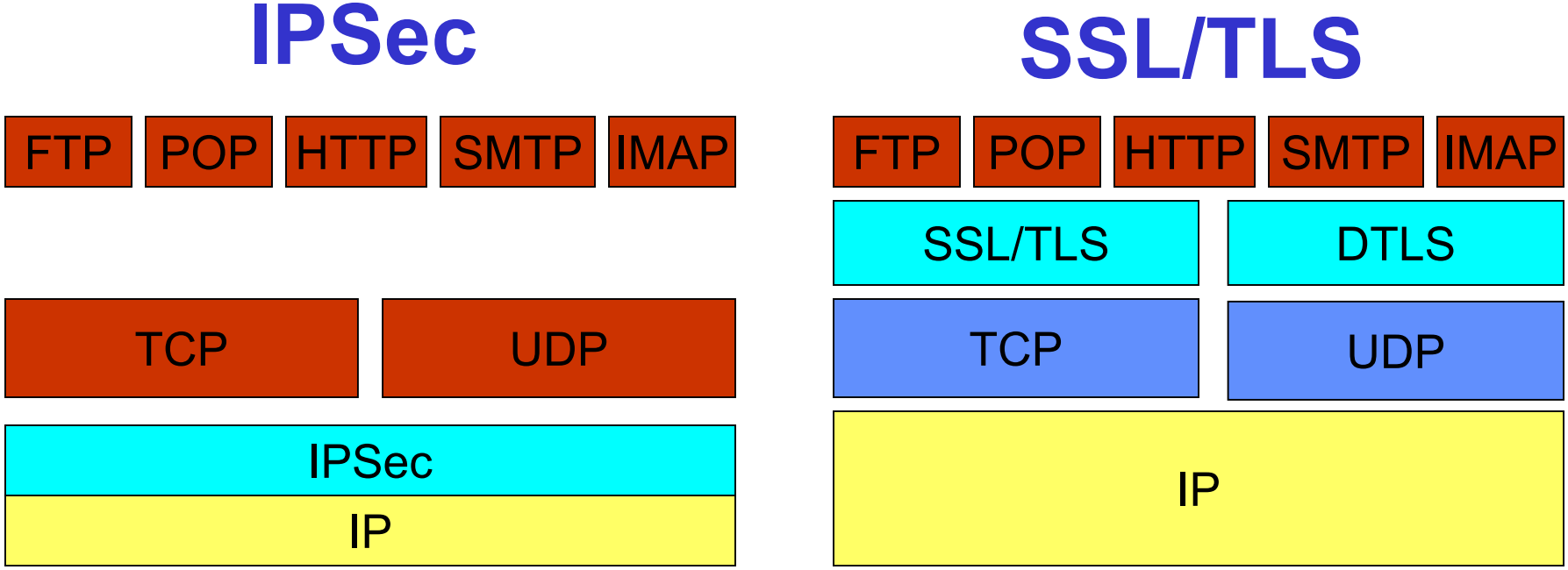
Herausforderung

  - Nutzdatenabsicherung
    - Vertraulichkeit: symmetrische Verschlüsselung
    - Integrität: Message Authentication Code (MAC)
    - kombiniert: Authenticated Encryption

Kleine Auswahl bekannter Sicherheitsprotokolle

- **X.509 Zertifikate / PKIX**
  - Standardisierte, häufig verwendete Datenstruktur zur Bindung von kryptographischen Schlüsseln an eine Person
  - Eventuell gekoppelt mit zusätzlichen Attributen
  - Hierarchische Baum-Struktur des “Vertrauens”
- **SSL / TLS / DTLS**
  - Secure Socket Layer / Transport Layer Security / Datagram TLS
  - symmetrisch verschlüsselte und integere Verbindung oberhalb TCP/UDP
  - Tunnelung von Anwendungsprotokollen wie HTTP, FTP SMTP, IMAP etc.
  - Identifizierung von Server oder Client mittels X.509 Zertifikaten
  - Aushandelbare Algorithmen
- **SSH**
  - gedacht für sicheren Remote-Zugriff
  - Verschiedene Authentifizierungsmöglichkeiten z.B. X.509 Zertifikate, Paßwort
  - aber z. B. auch Port-Forwarding möglich --> sicherer Tunnel
- **IPSec (VPN)**
  - Sicherheitsarchitektur oberhalb von IP
  - sicherer Tunnel für IP-Verkehr --> IP-Pakete werden verschlüsselt und dann als Payload verschickt
  - völlig transparent für Anwendungsprogramme
  - gesicherte Anbindung von Rechnern an Netz oder Verbindung von Netzen

Layering: IPSec vs. SSL/TLS



Eine kurze Geschichte von SSL/TSL

- ursprünglich für gesicherte Web-Kommunikation entwickelt:
  - 1994: SSL 1.0 von Netscape Communications (für Mosaic)
  - wenig später: SSL 2.0 für Netscape Navigator
  - 1996: SSL 3.0
    - behebt Sicherheitsprobleme von SSL 2.0
    - nachzulesen in RFC 6101
  - 1999: TLS 1.0 standardisiert als IETF RFC 2246
    - im Wesentlichen SSL 3.0
  - 2006: TLS 1.1 in RFC 4346
    - behebt Sicherheitsproblem von TLS 1.0 im Zusammenhang mit Betriebsart CBC
  - 2008: TLS 1.2 in RFC 5246
    - Verbesserungen bzgl. verwendbarer Hash-Funktionen
  - 2018: TLS 1.3 in RFC 8446
    - starke Protokollveränderungen gegenüber TLS 1.2
      - Ziel: Verbesserung der Sicherheit
    - unsichere kryptographische Algorithmen werden nicht mehr unterstützt
- mittlerweile in sehr vielen Protokollen verwendet, die sichere Ende-zu-Ende Kommunikation erfordern

TLS: Prinzipieller Ablauf

1. Aushandlung von Sicherheitsparametern (Handshake):
   - kryptographische Algorithmen
   - symmetrische Schlüssel
   - Authentifizierung von Server und Client (optional)

2. Aktivierung & Überprüfung der Sicherheit
   - Aktivierung durch Change Cipher Spec Nachricht
   - Überprüfung auf korrekte (gemeinsame) Sicherheitsparameter mit Finished Nachricht

3. sichere Übertragung der Anwendungsdaten

4. Beendigung der gesicherten Kommunikation

TLS Protokollstapel

TLS Handshake Protocol

TLS Change Cipher Spec. Protocol

TLS Alert Protocol

TLS Application Data Protocol

TLS Record Protocol

Content type

Version

Length

Payload

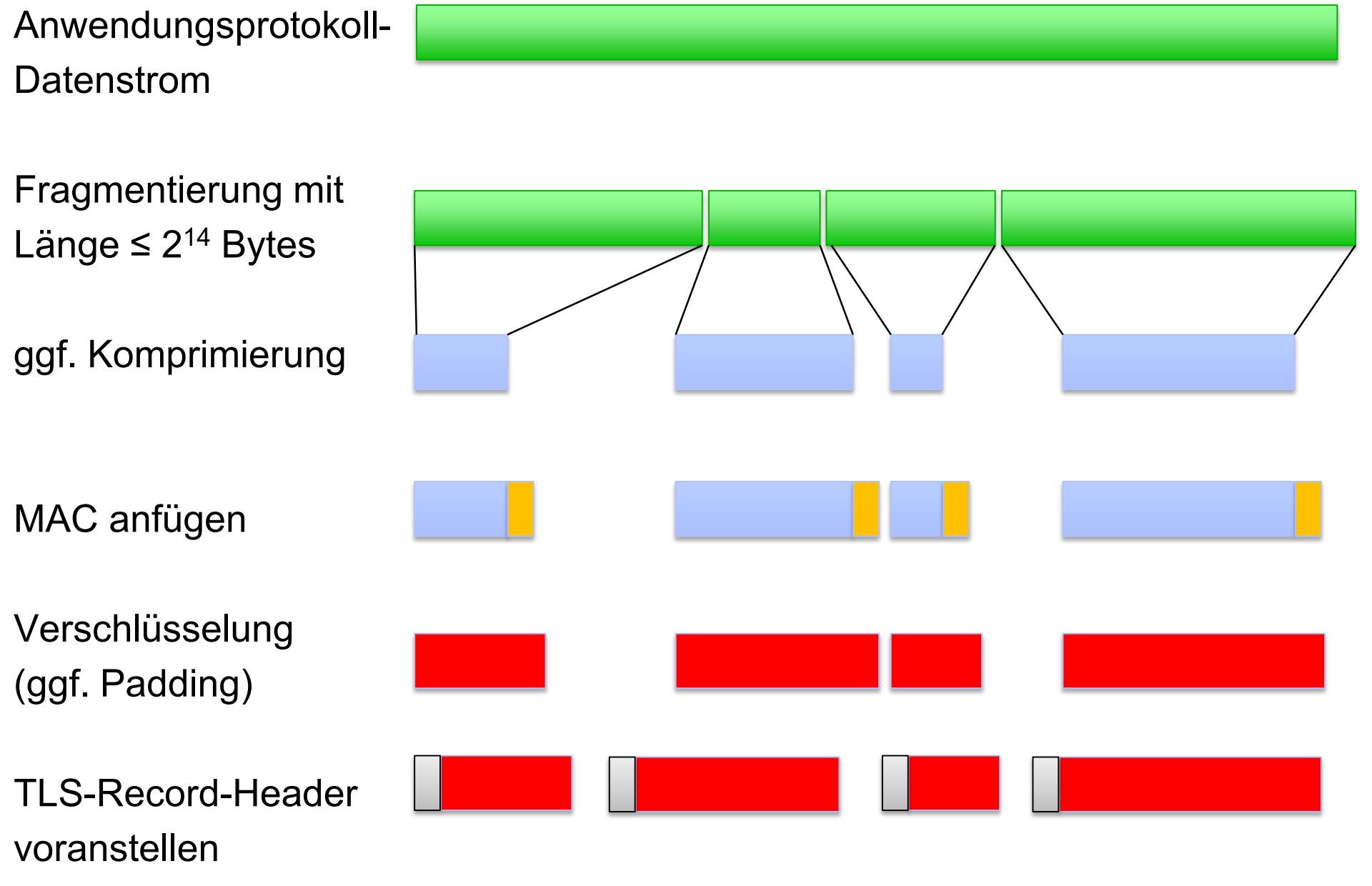
MAC

Padding

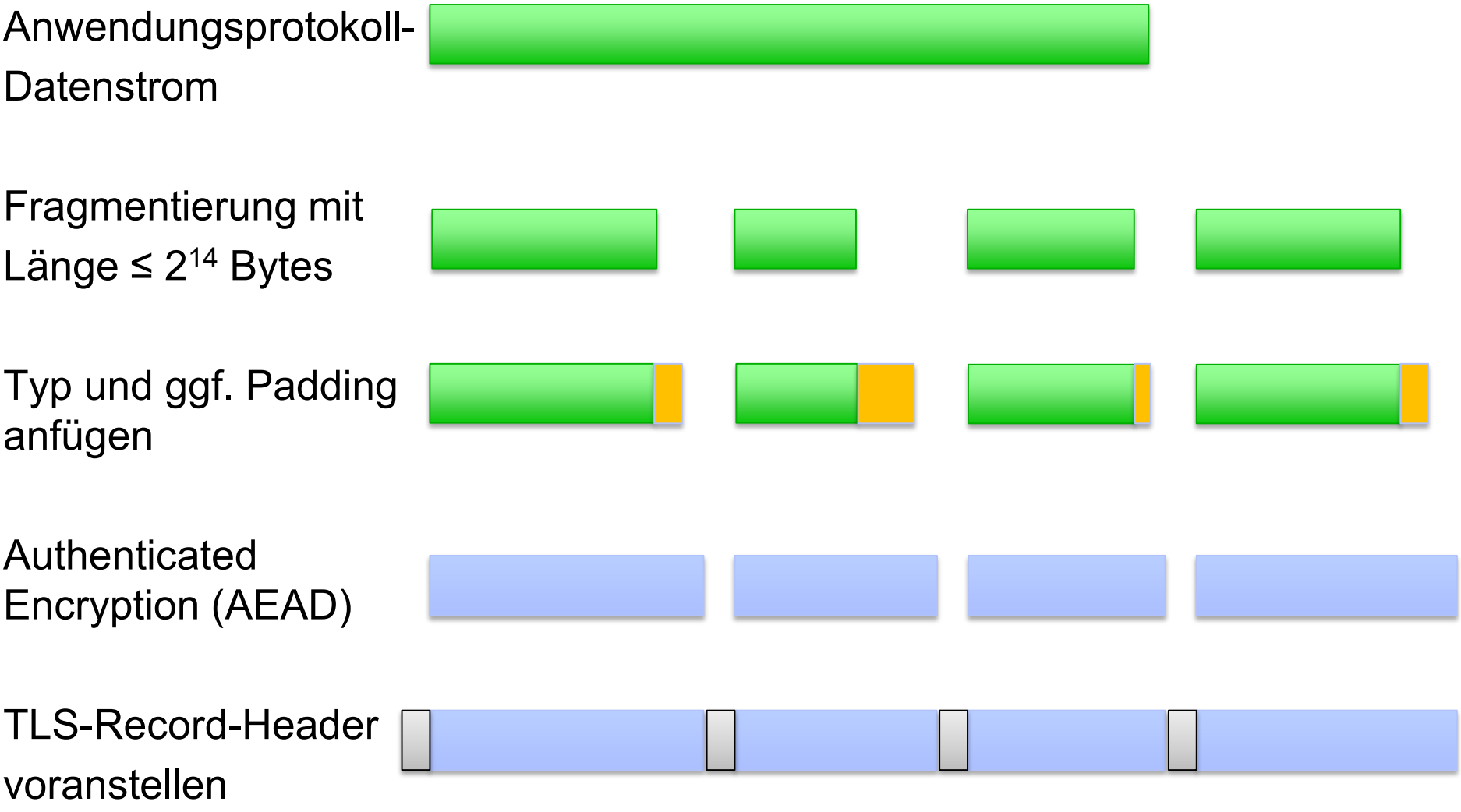
Ziel: gesicherte Ende-zu-Ende Kommunikation

- **Vertraulichkeit: symmetrische Verschlüsselung**
- **Integrität: MAC**

TLS 1.2: Sicherung des Anwendungsprotokolls



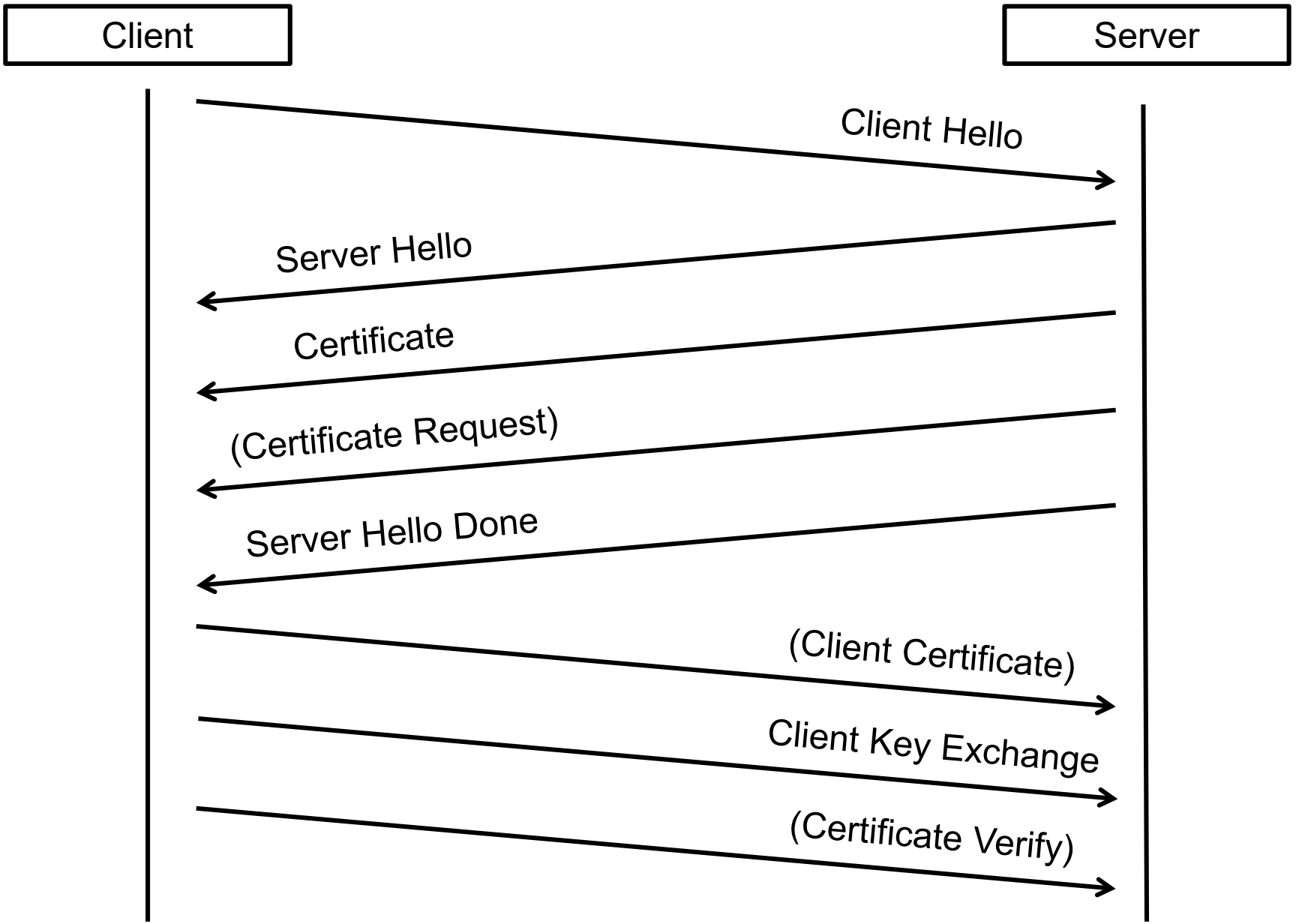
TLS 1.3: Sicherung des Anwendungsprotokolls



TLS Handshake: Ziele

- Aushandlung der zu verwendenden kryptographischen Algorithmen:
  - Client schickt Liste aller unterstützten Algorithmen
  - Server wählt aus und informiert den Client
- Gegenseitige Authentifizierung (optional)
  - für sichere Kommunikation notwendig
    - andernfalls: Man-In-The-Middle-Angriffe leicht möglich
  - üblich: Server authentifiziert sich durch Übermittlung von X.509-Zertifikat
    - ggf. zusätzlich: nur (vor-)definierte Zertifikate zulässig (*certificate pinning*)
  - möglich: Server fordert Client-Zertifikat an
  - alternativ: Verwendung von Pre-shared-Keys
- Schlüsselaustausch

TLS 1.2 Handshake: Ablauf



TLS Handshake: Nachrichten

- Client Hello, Server Hello:
  - Aushandlung der Sicherheitsmechanismen
  - Austausch von Zufallszahlen (nonce)
    - Challenge-Response Authentifizierung
    - Freshness (Verhinderung Replay-Angriffe)
- Certificate:
  - Authentifizierung
- Client Key Exchange:
  - Übermittlung des Pre-Master-Secret an den Server (verschlüsselt)
  - Berechnung des Master-Secret aus Pre-Master-Secret & Zufallszahlen
    - Ableitung der kryptographischen Schlüssel mit Hilfe von Key Derivation Function (KDF)

Pseudo Random Functions (PRF)

- ***Pseudo Random Functions (PRF):***
  - $F\colon K \times X \longrightarrow Y$
  - „domain“ X
  - „range“ K
  - „efficient“ algorithm to evaluate $F(k, x)$
- if $k \in K$ is fixed: F “looks” like a random function
- without knowing k
  - one cannot calculate $F(x)$
  - one cannot calculate $F^{-1}(y)$
  - given pairs of (x_i, y_i) it is hard to calculate k
- symmetric encryption is PRF

Key Derivation Function (KDF)

- naming of PRF for special use case: key derivation

- TLS 1.2:
  - MasterSecret = KDF(PreMasterSecret, “master secret” | ClientHello.nonce | ServerHello.nonce)
  - SessionKeys = KDF(MasterSecret, “key expansion” | ClientHello.nonce | ServerHello.nonce)

KDF in TLS 1.3 (aus RFC 8446)

```
          0
          |
          v
PSK ->  HKDF-Extract = Early Secret
          |
          +-----> Derive-Secret(., "ext binder" | "res binder", "")
          |                     = binder_key
          |
          +-----> Derive-Secret(., "c e traffic", ClientHello)
          |                     = client_early_traffic_secret
          |
          +-----> Derive-Secret(., "e exp master", ClientHello)
          |                     = early_exporter_master_secret
          v
    Derive-Secret(., "derived", "")
          |
          v
(EC)DHE -> HKDF-Extract = Handshake Secret
          |
          +-----> Derive-Secret(., "c hs traffic",
          |                     ClientHello...ServerHello)
          |                     = client_handshake_traffic_secret
          |
          +-----> Derive-Secret(., "s hs traffic",
          |                     ClientHello...ServerHello)
          |                     = server_handshake_traffic_secret
          v
    Derive-Secret(., "derived", "")
          |
          v
0 -> HKDF-Extract = Master Secret
          |
          +-----> Derive-Secret(., "c ap traffic",
          |                     ClientHello...server Finished)
          |                     = client_application_traffic_secret_0
          |
          +-----> Derive-Secret(., "s ap traffic",
          |                     ClientHello...server Finished)
          |                     = server_application_traffic_secret_0
          |
          +-----> Derive-Secret(., "exp master",
          |                     ClientHello...server Finished)
          |                     = exporter_master_secret
          |
          +-----> Derive-Secret(., "res master",
                                ClientHello...client Finished)
                                = resumption_master_secret
```

TLS Handshake: Nachrichten

- Client Hello, Server Hello:
  - Aushandlung der Sicherheitsmechanismen
  - Austausch von Zufallszahlen (nonce)
    - Challenge-Response Authentifizierung
    - Freshness (Verhinderung Replay-Angriffe)
- Certificate:
  - Authentifizierung
- Client Key Exchange:
  - Übermittlung des Pre-Master-Secret an den Server (verschlüsselt)
  - Berechnung des Master-Secret aus Pre-Master-Secret & Zufallszahlen
    - Ableitung der kryptographischen Schlüssel mit Hilfe von Key Derivation Function (KDF)
- Abschluß: Change Cipher Spec & Finished
  - „Aktivierung“ der Sicherheitsmechanismen
  - Authentikation der Handshake-Nachrichten (mittels MAC)

TLS Anmerkungen

- TLS ist erweiterbar mit:
  - neuen symmetrischen Verschlüsselungsverfahren
  - neuen Schlüsselaustauschverfahren
  - neuen MAC-Verfahren
- Erweiterungen meist in eigenen RFCs spezifiziert
- HTTP Strict Transport Security (HSTS)
  - Server fordert Client auf, ausschließlich TLS für einen bestimmten Zeitraum zu verwenden
  - Verhinderung von Downgrade-Angriffen: https → http

Woher Zertifikate?

- kommerzielle Anbieter
- freie Alternative: Let‘s Encrypt (letsencrypt.org)
  - kostenfreie Zertifikate für Domain-Inhaber
  - Zertifikatsausstellung mittels ACME-Protokoll
    - Besitz einer Domain wird durch Ausliefern von (zufälligen, vorgegebenen) Daten nachgewiesen
- Probleme:
  - kompromittierte Schlüssel
  - nicht zuverlässige Zertifizierung
  - nicht vertrauenswürdige Zertifizierungsstellen

TLS Risiken & Probleme

- technische Risiken:
  - kryptographische Schwächen im Protokoll / den verwendeten Algorithmen
  - Implementierungsfehler
  - (Serverseitige) Konfigurationsfehler
    - unsichere Algorithmen (RC4)

➔ vergleichsweise leicht zu beheben

- menschliche Risiken / Schwächen
  - fehlendes bewußtes Benutzen von TLS
  - Überprüfung der Authentizität

➔ schwer zu lösen

- zusätzliche (organisatorische) Risiken:
  - Kompromittierung der Zertifikate / geheimen Schlüssel

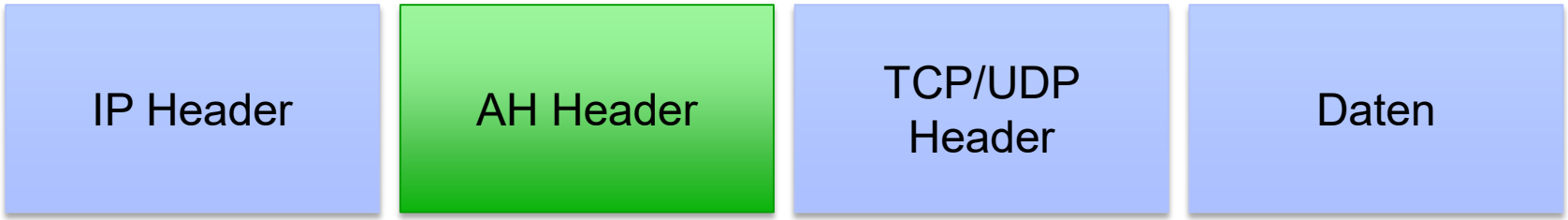
  ➔ erhebliches Risiko

IPSec

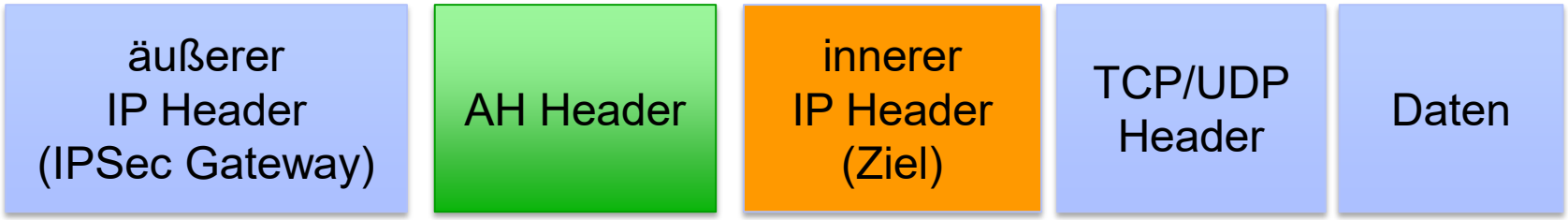
- Ziel:
  - Absicherung von IP
- Ergebnis
  - äußert flexibles, aber sehr komplexes Protokoll
  - verschiedene Transport-Modi:
    - Tunnel-Modus: Netz-zu-Netz Koppelung
    - Transport-Modus: Host-zu-Host Koppelung
  - verschiedene Sicherheits-Modi:
    - nur Authentizität: Authentication Header (AH)
    - Authentizität und Vertraulichkeit: Encapsulating Security Payload (ESP)
  - verschiedene Möglichkeiten des Schlüsselaustausches
    - manuell
    - mittels eigener Protokollsuite: Internet-Key-Exchange-Protokoll (IKE)

IPSec: Authentication Header

Transport-Modus



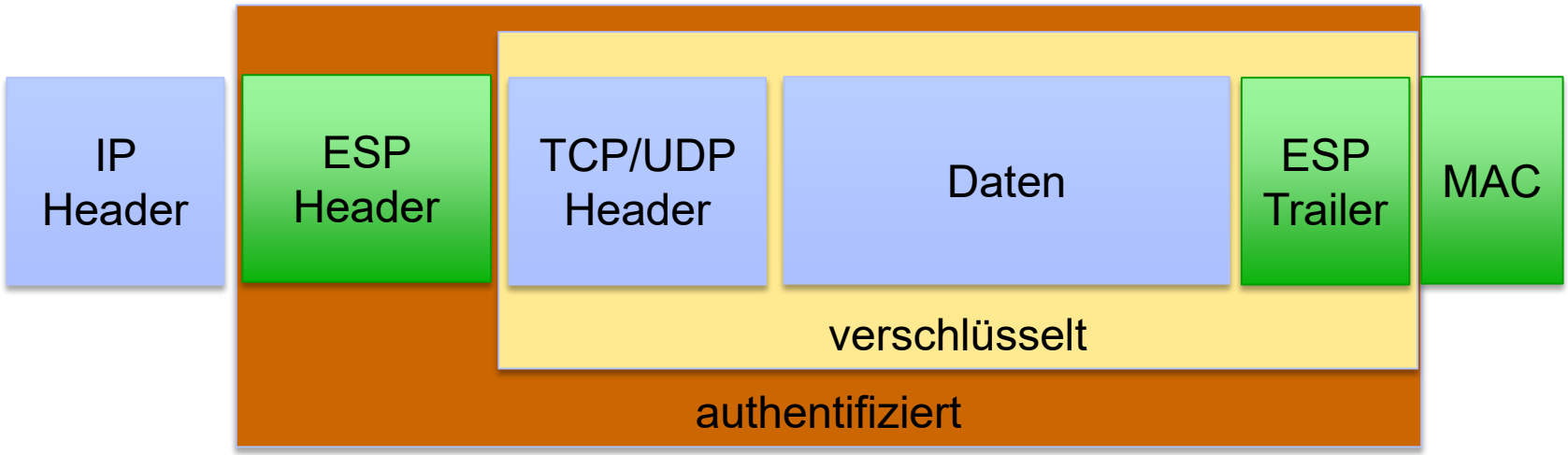
Tunnel-Modus



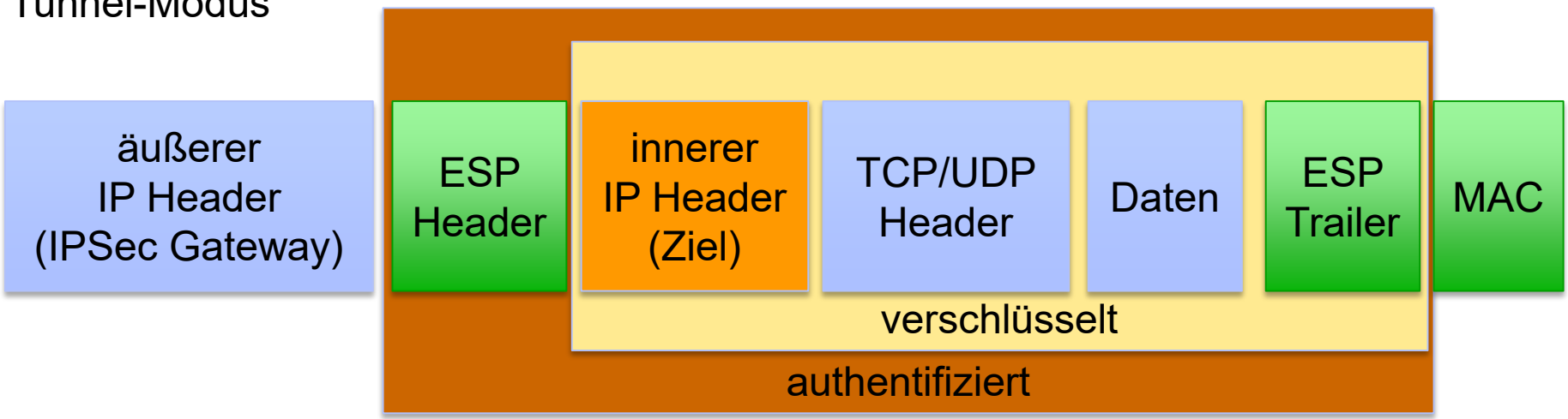
- komplettes Paket ist authentifiziert
  - Problem: NAT

IPSec: Encapsulating Security Payload

Transport-Modus



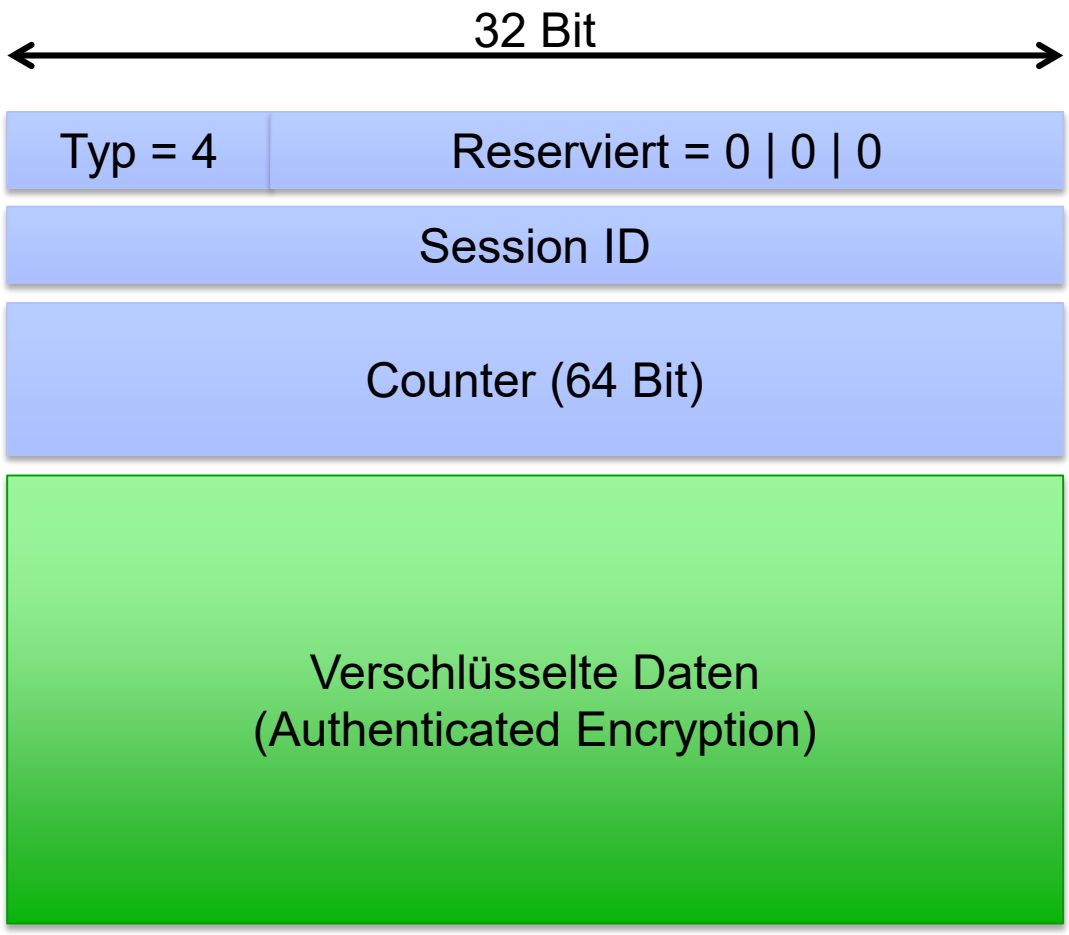
Tunnel-Modus



Wireguard

- VPN Protokoll
- Ziel:
  - Einfach Einrichtung
  - Konzentration auf das Wesentliche
  - Unterstützung ausschließlich aktueller kryptographischer Algorithmen
- Aktuell noch „experimentell“
- Implementierung von Linux (Kernel), Windows, etc. vorhanden
- Transportprotokoll: UDP
  - Problematisch bezüglich mancher Firewall-Einstellungen
- Austausch symmetrischer Schlüssel mit Hilfe von Diffie-Hellman-Schlüsselvereinbarung

Wireguard: Datenpakete



- UDP → Paketverluste, Re-Ordering:
  - alle notwendigen Informationen in jedem Paket