Informationssicherheit:

Sicherheitsprotokolle (Theorie)
Sicherheitsprotokoll: TLS (SSL)

Informationssicherheit:

Sicherheitsprotokolle (Theorie)

Sicherheitsprotokoll: TLS (SSL)

Sicherheitsprotokolle

- Wir haben bislang verschiedene kryptographische Grundbausteine kennen gelernt wie z.B.:
  - Symmetrische und asymmetrische Verschlüsselung
  - Hash, MACs, Digitale Signaturen
  - Zufallswerte, Nonces, Zeitstempel

- Sicherheitsprotokolle (***security protocols, cryptographic protocols***)
  - Basieren auf diesen Grundbausteinen
  - Verschiedene Arten von Protokollen:
    - Protokolle zur Authentisierung (z.B. Challenge Response-Protokolle, Kerberos, Needham-Schroeder)
    - Schlüsselvereinbarungsprotokolle (z.B. IKE, Diffie-Hellman)
    - Zusammengesetzt zu E-Commerce-Protokollen (SSL/TLS), E-Government (OSCI – in Bremen entwickelt)

Grundlegende Notation (1)

- Ein Protokoll besteht aus einer Menge von Regeln, die den Zustand des Nachrichtenaustauschs zwischen zwei oder mehreren Kommunikationsteilnehmern beschreiben:
  - Kommunikationsteilnehmer: Benutzer, Prozesse, Rechner, ...
  - auch “Principals” genannt (s. zweite Vorlesung)
- Protokollschritte

  n. A → B : M
  - “**A** sendet **M** an **B** gemäß dem n-ten Protokollschritt.”
    A, B Principals, M Nachricht
  - Nachrichten können strukturiert sein: $M = M_1, \ldots, M_n$

Grundlegende Notation (2)

- Ein **Protokoll** ist eine Folge von Protokollschritten:

$$
\begin{aligned}
1.\ & A_1 \rightarrow B_1 : \ M_1 \\
2.\ & A_2 \rightarrow B_2 : \ M_2 \\
& \vdots
\end{aligned}
$$

- Protokolldurchläufe
- Berücksichtigung einer feindlichen Umgebung: Fehler, Verlust von Vertraulichkeit/Datenintegrität
  - Angreifer beherrscht Kommunikationskanal, kann aber nicht die zugrundeliegenden kryptographischen Verfahren brechen (Dolev-Yao-Modell, 1981)

Challenge-Response-Protokolle (1)

- Einfaches Protokoll: Passwort
  1. A → B : K_{AB}
  Problem: u.a. Mitschneiden der Passwörter

- Verbesserung: Challenge-Response-Protokoll
  Ziel: B authentisiert sich gegenüber A
  1. A → B : N
  A schickt B eine Herausforderung (***challenge***) **N** (nonce)
  2. B → A : $\{N\}_{K_{AB}}$
  B beantwortet die Challenge mit $\{N\}_{K_{AB}}$ (***response***)

Challenge-Response-Protokolle (2)

- Mutual Challenge-Response:
  1. A → B : N_A

  2. B → A : $\{N_A, N_B\}_{K_{AB}}$

  3. A → B : N_B

  Bemerkung: Dieses einfache Protokoll ist angreifbar (s. folgende Folien).
- Anwendung von Challenge-Response-Protokollen:
  - GSM/GPRS/UMTS
  - PPP (Authentisierungsprotokoll CHAP)
  - Kerberos

Angriff auf das Mutual Challenge-Response-Protokoll

- Idee:
  - Angreifer B‘ fängt alle Nachrichten zwischen A und B ab und gibt sich als B aus (vgl. Dolev-Yao-Modell eines Angreifers)
  - Angreifer B‘ startet nun parallel zum Authentisierungsvorgang zwischen A und B eine eigene Session
  - Angreifer B‘ bringt A dazu, die eigene Nonce zu verschlüsseln

Die einzelnen Schritte des Angriffs (1)

1. $A \rightarrow B' : N_A$

1'. $B' \rightarrow A : N_A$

Angreifer spielt die Nachricht einfach zurück (***reflection attack***).

2'. $A \rightarrow B' : \{N_A, N'_A\}_{K_{AB}}$

A meint, dass B einen neuen Durchgang des Protokolls gestartet hat, und antwortet.

Die einzelnen Schritte des Angriffs (2)

2. B‘ → A : $\{N_A, N_B = N'_A\}_{K_{AB}}$

B‘ spielt auch Nachricht 2‘. zurück.

3. A → B‘ : N_B

A meint, es handele sich nun um die Antwort von B aus dem ersten Protokolldurchgang, und antwortet.

B‘ authentisiert sich also gegenüber A als B, obwohl B‘ den Schlüssel K_{AB} nicht kennt!

Einschub: Reflection-Angriff

- Typisches Angriffsmuster:

  Reflection-Angriff:
  - Der Angreifer spielt einfach Nachrichten ***zurück***.
  - Systeme zur Freund-Feind-Erkennung (***identify friend-or-foe systems***, ***IFF***) waren angreifbar durch Reflection-Angriffe
  - Reflection-Angriffe sind häufig verbunden mit der Vermischung verschiedener Protokollläufe

Lösung des Problems

- Lösungsmöglichkeiten:
  1. Für beide Richtungen verschiedene Schlüssel verwenden (K_{AB} bzw. K_{BA}).
  2. Den Namen des Absenders in Schritt 2 mit angeben, also:
     $\mathbf{B \rightarrow A : \{B, N_A, N_B\}_{K_{AB}}}$

- Wichtige Designregeln für Protokolle
  - Nicht denselben Schlüssel für verschiedene Zwecke verwenden
  - Soviel explizite Informationen wie möglich verwenden (hier den Namen explizit erwähnen)

Das Denning-Sacco-Protokoll

- D.E. Denning und G.M. Sacco, 1982
- Schlüsselaustauschprotokoll mittels Public-Key-Verfahren und Zertifikaten
- Ziel:
  - Alice besitzt nach Durchführung der Protokollschritte einen gemeinsamen symmetrischen Schlüssel K_{AB} mit Bob.
  - Dieser Schlüssel soll frisch, also keine Wiedereinspielung (***replay***) sein.
- Voraussetzung: Es existiert eine vertrauenswürdige Certification Authority S;
  - Zertifikate CA und CB für Alice und Bob
  - Diese Zertifikate binden den öffentlichen Schlüssel für die Verschlüsselung und den Verifikationsschlüssel für digitale Signaturen an den Namen des jeweiligen Besitzers.

Die einzelnen Schritte des Denning-Sacco-Protokolls

Bezeichnungen: $\mathbf{K_X^{-1}}$ bezeichne den Signaturschlüssel von X,

$\mathbf{K_X}$ den Public-Key von X.

1. A → S : A, B

2. S → A : CA, CB

A erhält die entsprechenden Zertifikate von S.

3. A → B : CA, CB, $\mathbf{\{T_A, K_{AB}, \{T_A, K_{AB}\} K_A^{-1} \}_{K_B}}$

- Vertraulichkeit durch Verschlüsselung
- Authentisierung durch Zertifikat und digitale Signatur
- Replay-Angriffe werden durch Zeitstempel verhindert.

Der Angriff

- Bob kann sich gegenüber anderen Kommunikationspartnern als Alice ausgeben:
  1. Bob besorgt sich ein Zertifikat für Charlie,
  2. entschlüsselt die Nachricht 3.,
  3. schneidet die Signatur aus und
  4. erzeugt eine Nachricht an Charlie:

  $\mathbf{B \rightarrow C : CA, CC, \{T_A, K_{AB}, \{T_A, K_{AB}\}\ K_A^{-1}\ \}_{K_C}}$

- Charlie meint wegen der digitalen Signatur von Alice, dass die Nachricht von Alice stammt.

- Der Fehler wurde erst 1994 von Abadi aufgedeckt!

Problembehebung

- Lösung:

  Einfügen des Namens von Bob in den signierten Teil der Nachricht, also:

  $\{T_A, B, K_{AB}\}\ K_A^{-1}$

- Regel: soviel explizite Informationen wie möglich verwenden (hier den Namen explizit erwähnen)

Zusammenfassung

- Sicherheitsmechanismen werden oft über die Protokolle und nicht über unzureichende kryptographische Algorithmen gebrochen.
- Scheinbar einfachste Protokolle können subtile Sicherheitslücken besitzen.
- Dieses Problem ist immer noch aktuell, da viele neue bzw. spezielle Protokolle entworfen werden:
  - Nicht nur Authentisierungs- und Schlüsselaustausch-Protokolle, sondern auch:
    - E-Purse-Protokolle (z.B. bei der Geldkarte), E-Commerce-Protokolle
    - OSCI-Protokoll (Ist dieses Protokoll überhaupt schon einmal von Sicherheitsexperten analysiert worden?)
- Lösungsmöglichkeiten:
  - Sich an Designregeln halten (s. auch vorherige Folien)
  - Einsatz formaler Methoden

Protokollanalyse mit Formalen Methoden

- Verschiedene Ansätze wie z.B.:
  - Modellprüfer (***model checker***):
    - Protokolle werden als Finite State Machines (FSM) wie z.B. Kripke-Strukturen modelliert.
    - Untersuchung aller Ausführungspfade in der FSM
    - Aber: State Explosion Problem
    - Werkzeuge für die Verifikation von Security-Protokollen:
      Casper-Werkzeug von Gavin Lowe, CAPSL, SPIN
  - Theorembeweisen
    - Induktive Methode von Paulson
    - Spezielle Security-Logiken: BAN-Logik (Burrows, Abadi, Needham)
- Kein Allheilmittel, oft unzulässige Abstraktion
- Lars Knudsen: ***If it‘s provably secure, it probably isn‘t!***

Literatur zu Sicherheitsprotokollen

- Ross Anderson, Roger Needham: *Programming Satan's Computer*. In J. v. Leeuwen (ed.) Computer Science Today, LNCS 1000, Springer, 1995

- M. Abadi, R. Needham: *Prudent Engineering Practice for Cryptographic Protocols*, IEEE Transactions on Software Engineering, Vol.23, No.3, März 1997 (auch DEC SRC-125, Juni 1994)

- P.Ryan, S.Schneider: *Security Protocols*. Addison-Wesley, 2001

Informationssicherheit:

Sicherheitsprotokolle (Theorie)

Sicherheitsprotokoll: TLS (SSL)

Transport Layer Security: SSL/TLS

- Entwickelt von Netscape für „E-Commerce“ im Web
  - Online-Shops: Schutz von persönlichen Daten, Kreditkartennummern etc.
  - „HTTPS“ = HTTP über TLS (RFC 2818)
- Secure Sockets Layer (SSL)
  - SSL 2.0 war unsicher
  - letzte Version unter dem Namen SSL 3.0 (in 2014 auch hier Lücken gefunden)
- Transport Layer Security (TLS, RFC 2246+3546/4346+4366/5246 ➜ 8446): leicht abgewandelter Nachfolger von SSL (= „SSL 3.1“)
  - Allgemein verwendbar, nicht nur HTTPS (z.B. POP3, IMAP, SMTP)
- Literatur: E. Rescorla, *SSL and TLS – Designing and Building Secure Systems*, Addison-Wesley, 2001.

DTLS: RFC 6347

SSL/TLS: grundlegende Idee

- Hauptziele:
  - **Vertraulichkeit** (Secrecy) zwischen dem Kunden (Web-Browser) und einem Web-Server
  - **Authentisierung** (des Web-Servers)
  - **Integrität** der übertragenen Daten
- Authentisierung des Clients laut Spezifikation auch möglich, wird aber selten genutzt (Warum?)
- SSL/TLS nutzt hybrides Verschlüsselungsverfahren:
  - Schlüsselvereinbarung mit Public-Key-Verfahren
    - Öffentlicher Schlüssel des Web-Servers wird dem Web-Browser per X.509-Zertifikat bekannt gemacht
  - Anschließend: symmetrisches Verfahren zum Datenaustausch

SSL/TLS: Teilprotokolle

- SSL/TLS Teilprotokolle:
  - Handshake-Protokoll (Schlüsselvereinbarung, Aushandlung der Parameter für die Verschlüsselung, MAC-Bildung, ...)
  - Record-Protokoll (verschlüsselte Daten, Bildung von MACs)
  - Alert-Protokoll (aufgetretene Fehler während des Protokolllaufes)

- Wir betrachten das Handshake-Protokoll in zwei Varianten
  1. Auf RSA basierender Schlüsselausstausch
  2. Diffie-Hellman-Schlüsselvereinbarung

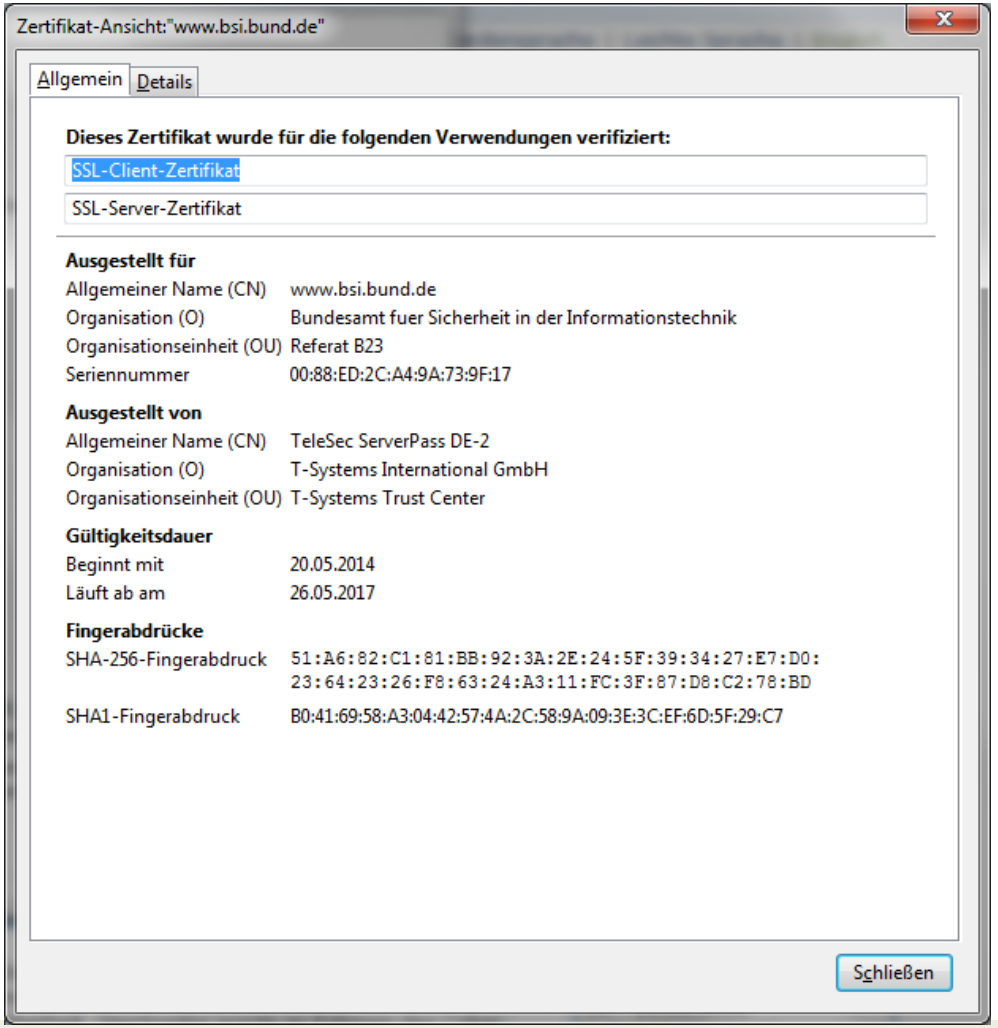
SSL/TLS-Handshake mit RSA: Vorausgesetzte Notationen & Bezeichnungen

- Kommunikationspartner C (***client***) and S (***server***)
- CS: Serverzertifikat
  $CS = Cert_{CA}(S) = \{S, K_S, T, L\}_{K_{CA}^{-1}}$ (Beispiel hierfür: nächste Folie)
- K_S öffentlicher Schlüssel des Servers
- K_{CA}^{-1} privater Schlüssel der CA

- **Vorbemerkung**: Es wird im Folgenden eine vereinfachte Fassung des Handshake-Protokolls vorgestellt
- Session-ID lassen wir weg, ebenso Details zu Zufallszahlen **$Random_X$**

Serverzertifikat

- Bindet den Public Key des Servers an den DNS-Namen des Servers

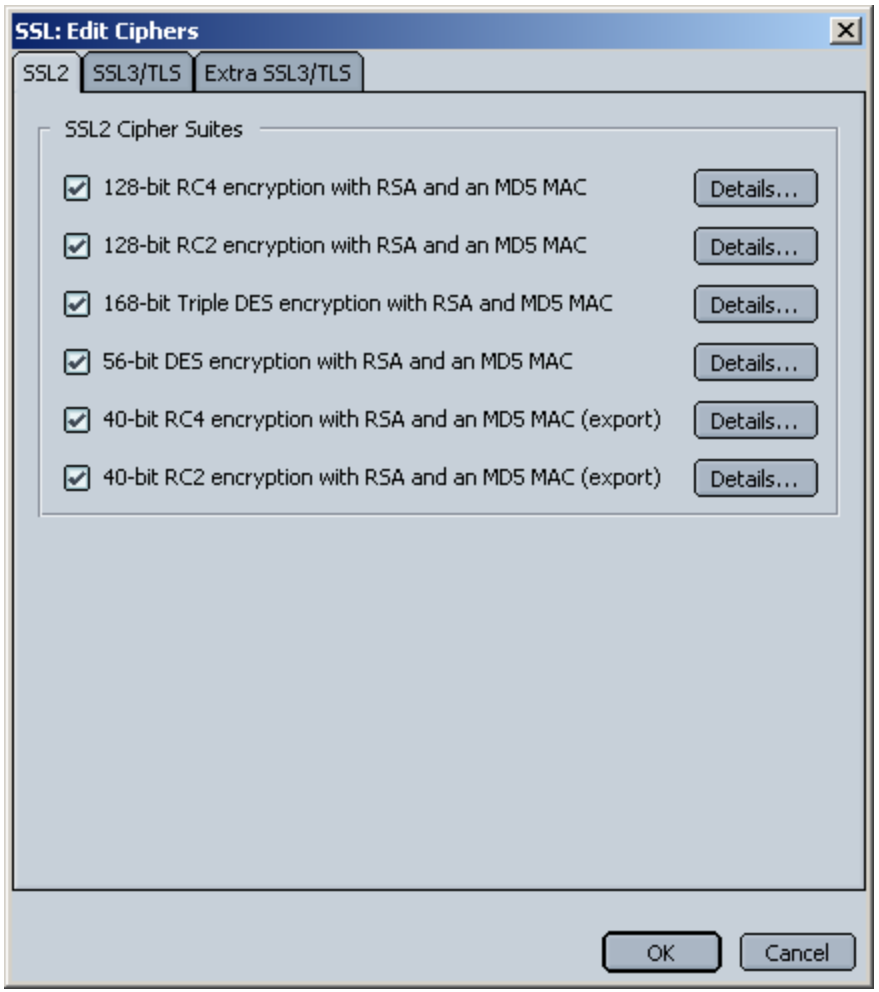


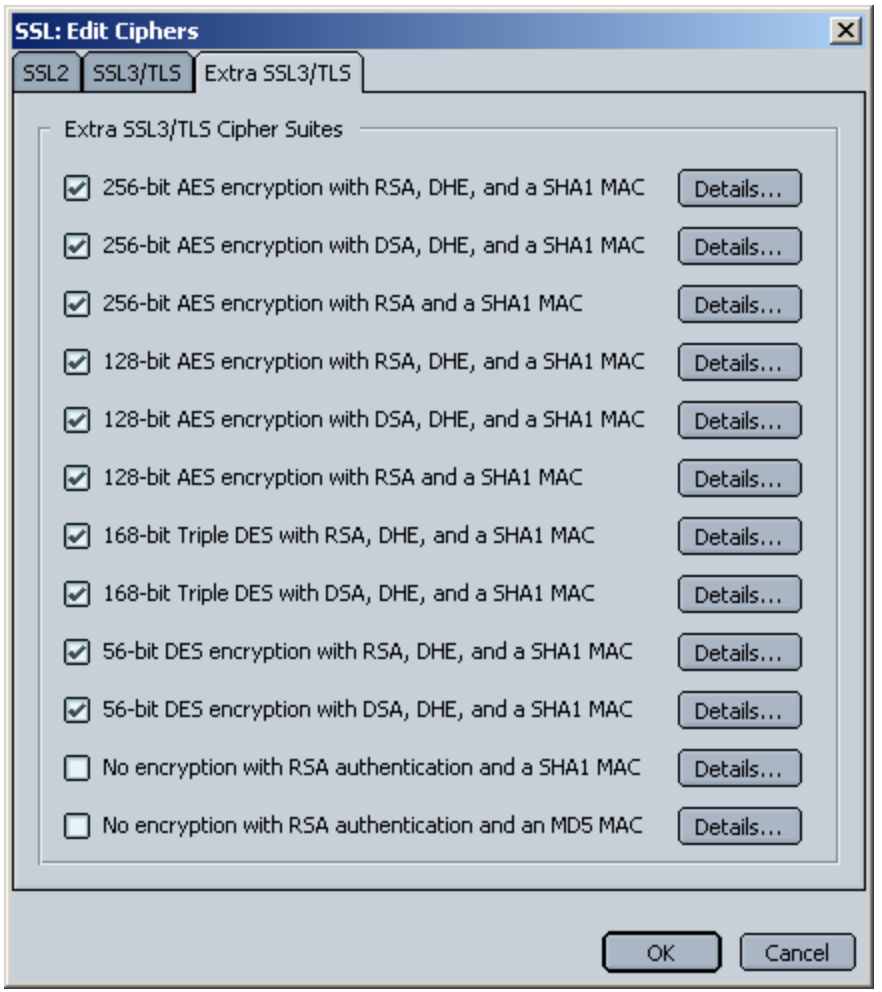
SSL/TLS: Handshake mit RSA (1)

1. C → S: Random_C, Supported_Ciphers ClientHello

- **Supported_Ciphers** sind die Cipher-Suites, die der Client unterstützt
  - können im Mozilla-Browser ausgewählt werden (vgl. folgende Folie)
  - Bsp.: TLS_RSA_WITH_AES_128_CBC_SHA256
- **Random_C** wird später bei der Schlüsselerzeugung benötigt; verhindert Replay-Angriffe (Wiedereinspielen von alten Nachrichten)

Kryptographische Suiten





Crypto Agility: Ciphersuites

- Cipher **suite** [| swit |]
  Set gut zusammenpassender kryptographischer Algorithmen
- Cipher **suit** [| sut |]:

Crypto Agility: Ciphersuites

- Cipher **suite** [| swit |]
  Set gut zusammenpassender kryptographischer Algorithmen
- Cipher **suit** [| sut |]:

SSL/TLS: Handshake mit RSA (2)

2. S → C: $Random_S$, CS, Chosen_Cipher

ServerHello
Certificate

- **Chosen_Cipher**:
  - Auswahl aus **Supported_Ciphers**
  - Auswahl des vom Client präferierten Verfahrens aus der Liste (aber nicht vorgeschrieben)
- **CS** (Server-Zertifikat): Zur vertrauenswürdigen Verteilung des öffentlichen Server-Schlüssels K_S (RSA-Key)
- **$Random_S$**: wie **$Random_C$** zusätzlicher Schutz vor Replay-Angriffen

SSL/TLS: Handshake mit RSA (3)

3. C → S: $\{K_0\}_{K_S}$ (vgl. hybride Verschlüsselung)

K_0 – Premaster Secret: 48 Bytes lang

- Berechnung eines Master Secrets K_1 aus K_0
  (durch beide Kommunikationspartner unabhängig voneinander):
  K_1 = PRF(K_0,"master_secret", Random$_c$+ Random$_s$),
  „+" bedeutet hier Konkatenation
  PRF – Pseudo Random Function zur Erzeugung von diversen Schlüsseln
- **Random$_c$** und **Random$_s$** verhindern Replay-Angriffe (s.o.)
- **S** und **C** können dann aus dem Master Secret **K_1**
  das eigentliche Schlüsselmaterial berechnen
  - Wieder unter Verwendung von PRF (nächste Folie)

Einschub: PRF (1)

- In TLS, in SSL 3.0 ähnliche Funktion
- Basiert auf HMAC unter Verwendung von verschiedenen Hashfunktionen
  - Nun keine Festlegung mehr auf SHA-1 und MD5 (unsicher), Verwendung vom sicheren SHA-256 möglich
  - Verkettung von Hashfunktionen
- Berechnet verschiedenes Schlüsselmaterial

  Berechnung eines Schlüssels pro Kommunikationsrichtung:
  - für die symmetrische Verschlüsselung ($\mathbf{K_{cs}}$, $\mathbf{K_{sc}}$),
  - für MACs
  - für IVs (z.B. für CBC-Mode)
- PRF liefert diese sechs Schlüssel in einem Schlüsselblock zurück
- PRF(secret, label, seed)

  Beispielaufrufe:

  keyblock=PRF(K_1,"key_expansion", $Random_S$ + $Random_c$)

  K_1=PRF(K_0,"master_secret", $Random_C$ + $Random_S$)

Einschub: PRF (2)

P_hash(secret, seed) = HMAC_hash(secret, A(1) + seed) +
HMAC_hash(secret, A(2) + seed) +
HMAC_hash(secret, A(3) + seed) + ...

PRF(secret, label, seed) = P_<hash>(secret, label + seed)

Beispiel:

keyblock =
PRF(K_1,“key_expansion“, Random) = P_SHA-256(K_1,“key_expansion“ + Random)

SSL/TLS mit RSA: Handshake (4)

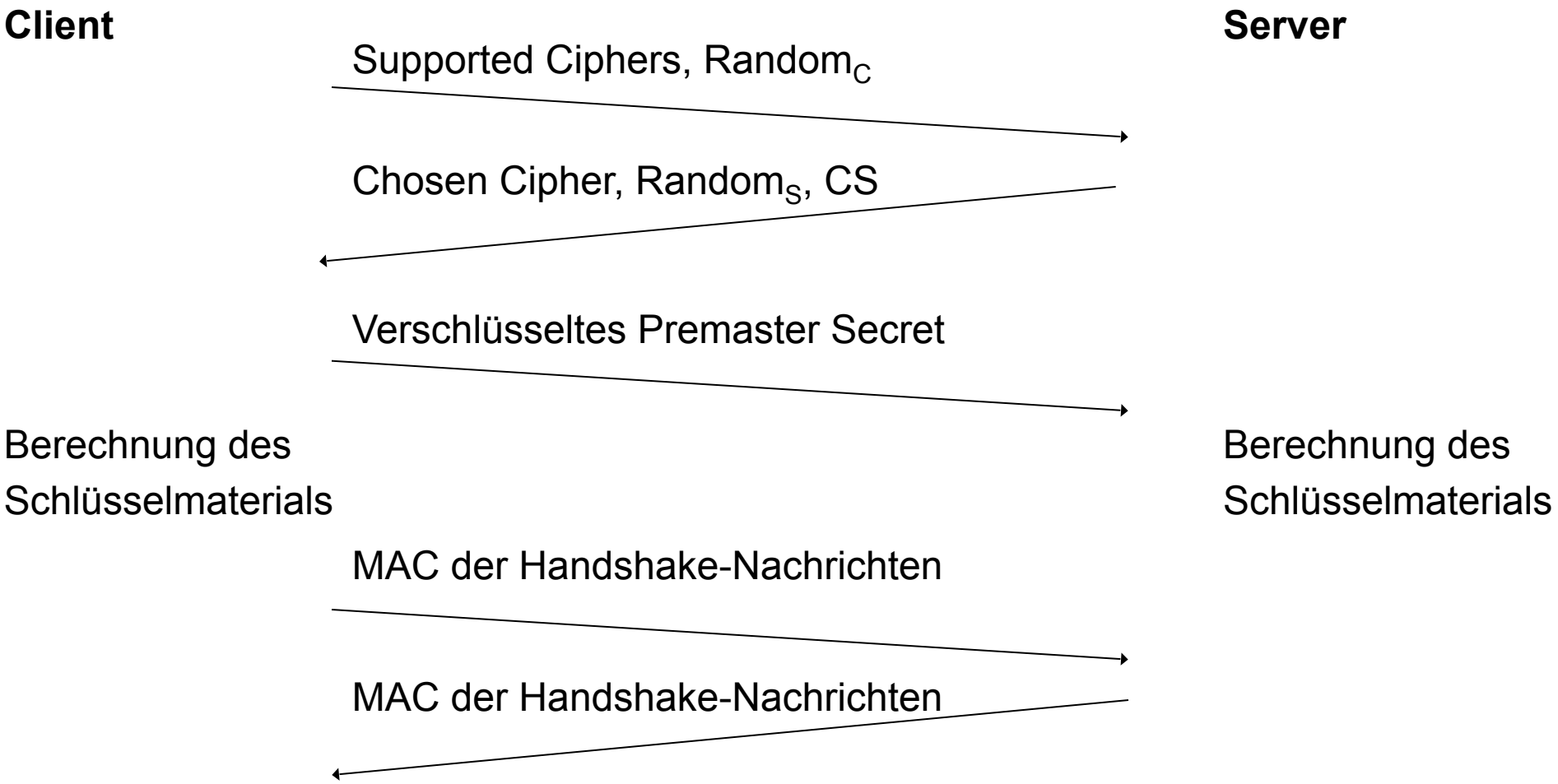
4. C→ S: {finished, MAC(K_1, alle bisher gesendeten Nachrichten)}$_{K_{CS}}$

- **finished** bestätigt das Ende des Handshake-Vorganges des Clients

5. S → C: {finished, MAC(K_1, alle bisher gesendeten Nachrichten)}$_{K_{SC}}$

- **finished** bestätigt das Ende des Handshake-Vorganges des Servers
- Warum Verschlüsselung der Nachrichten 4. und 5.?
- Erst **nach** dieser Nachricht können verschlüsselte und durch einen MAC gesicherte Daten gesendet werden wie z.B. Kreditkartennummern (Record-Protokoll)

SSL/TLS-Handshake: Überblick



Ergänzung:SSL/TLS-Handshake mit DH

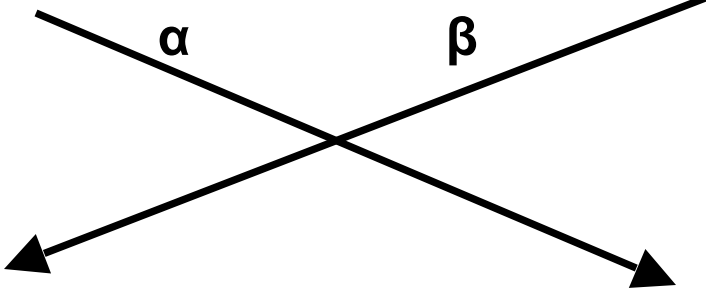
- Schlüsselvereinbarung nach Diffie und Hellman
- Verwendung von DSA (ECDSA) zum digitalen Signieren
- Ansonsten:
  ähnliche Voraussetzungen wie bei der RSA-Variante

Wdhlg.: Schlüsselvereinbarung nach Diffie und Hellman

Alice | Bob

Wählt zufällig eine Zahl **a**.
Berechnet $\alpha := g^a \bmod p$.

Wählt zufällig eine Zahl **b**.
Berechnet $\beta := g^b \bmod p$.



Berechnet $\beta^a \bmod p$

Berechnet $\alpha^b \bmod p$

$$K_{AB} = \beta^a \bmod p = (g^b)^a \bmod p = g^{ba} \bmod p$$
$$= g^{ab} \bmod p = (g^a)^b \bmod p = \alpha^b \bmod p$$

SSL/TLS:Handshake mit DH (1)

1. C → S: $Random_C$, Supported_Ciphers ClientHello

- Wie bei der RSA-Variante

2. S → C: $Random_S$, CS,

Chosen_Cipher, $\{p,g, \alpha\}_{K^{-1}_S}$ ServerHello
Certificate
ServerKeyExchange

- Ähnlich wie bei RSA-Variante
- **Aber: CS** enthält jetzt den Public Key von S zur Signatur-Verifikation (DSA)
- Mit DSA digital signierter DH-Public Key (vgl. letzte Folie) :

$\{p,g, \alpha\}_{K^{-1}_S}$

SSL/TLS:Handshake mit DH (2)

3. C → S: β ClientKeyExchange

$\beta^a \bmod p = \alpha^b \bmod p = K_0$ Premaster Secret;

Aus **K_0** kann das Master Secret berechnet werden (wie im RSA-Fall):

- **K_1 =PRF(K_0, "master_secret", $Random_c$+ $Random_s$)**
- **...**

SSL/TLS: Handshake mit DH (3)

4. C→ S: {finished, MAC(K_1, alle bisher gesendeten Nachrichten)}$_{K_{CS}}$

5. S → C: {finished, MAC(K_1, alle bisher gesendeten Nachrichten)}$_{K_{SC}}$

- Erst **nach** dieser Nachricht können verschlüsselte und durch einen MAC gesicherte Daten gesendet werden wie z.B. Kreditkartennummern (Record-Protokoll)

SSL/TLS: Record-Protokoll

- Nach erfolgreichem Handshake können die Daten verschlüsselt und durch einen MAC gesichert gesendet werden
  - z.B. Kreditkartennummern
  - Schutzziele: Vertraulichkeit (Secrecy), Datenintegrität
- Aufspaltung der Nachricht in Blöcke
- Unverschlüsselte Header-Informationen pro Block
- Bei Blockchiffren evtl. Padding erforderlich
- Verschlüsselung und Sicherung durch MAC pro Block

$$\mathbf{C \rightarrow S: \{data, MAC(K_{CS\_MAC}, data)\}_{K_{CS}}}$$

SSL/TLS: Alert-Protokoll

- Ziel: den Kommunikationspartner über Ausnahmebedingungen informieren, z.B.:
  - Handshake failure
  - Close notify
  - Unkown CA
  - Certificate expired
  - Certificate unknown
  - No certificate
- Unterscheidung zwischen Warn- und Fehlermeldungen
  - Fehlermeldungen führen immer zum Verbindungsabbruch

RFC 8446: TLS 1.3

- Ziele:
  - Höhere Sicherheit durch heftiges Aufräumen
  - Privacy-Ziele angehen
  - Performance steigern (0-RTT)
- Ciphersuites entbündeln
  - Ciphersuite beschreibt nur noch Record Protocol
    - Nur noch AEAD
  - Key Exchange (Diffie-Hellman) separat aushandeln
    - Nur noch PFS
  - Authentication (RSA, ECDSA) separat aushandeln
  - (Pre-Shared-Key separat aushandeln)

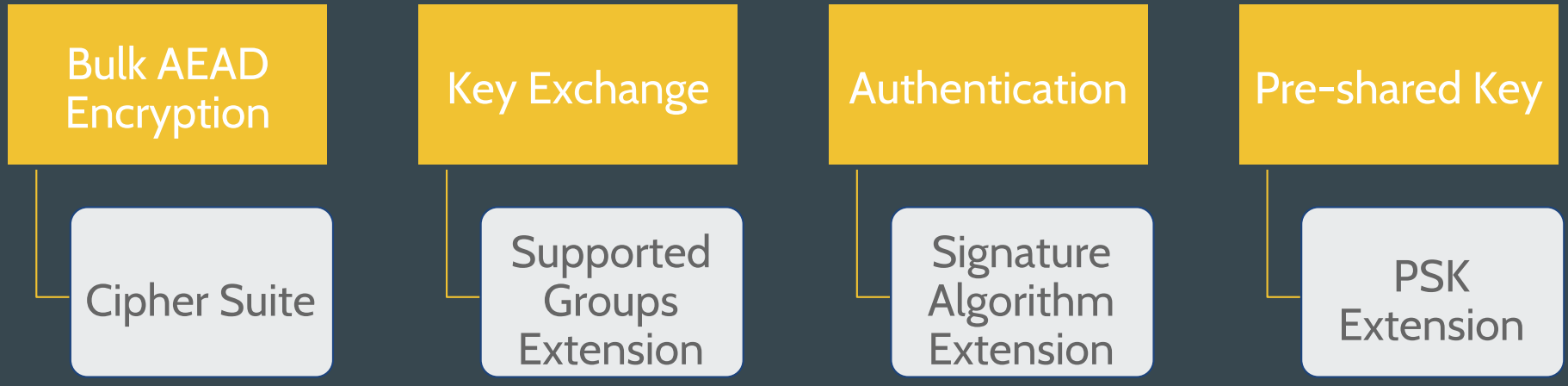
What were the design goals?

PRIVATE

How do you specify ciphers?

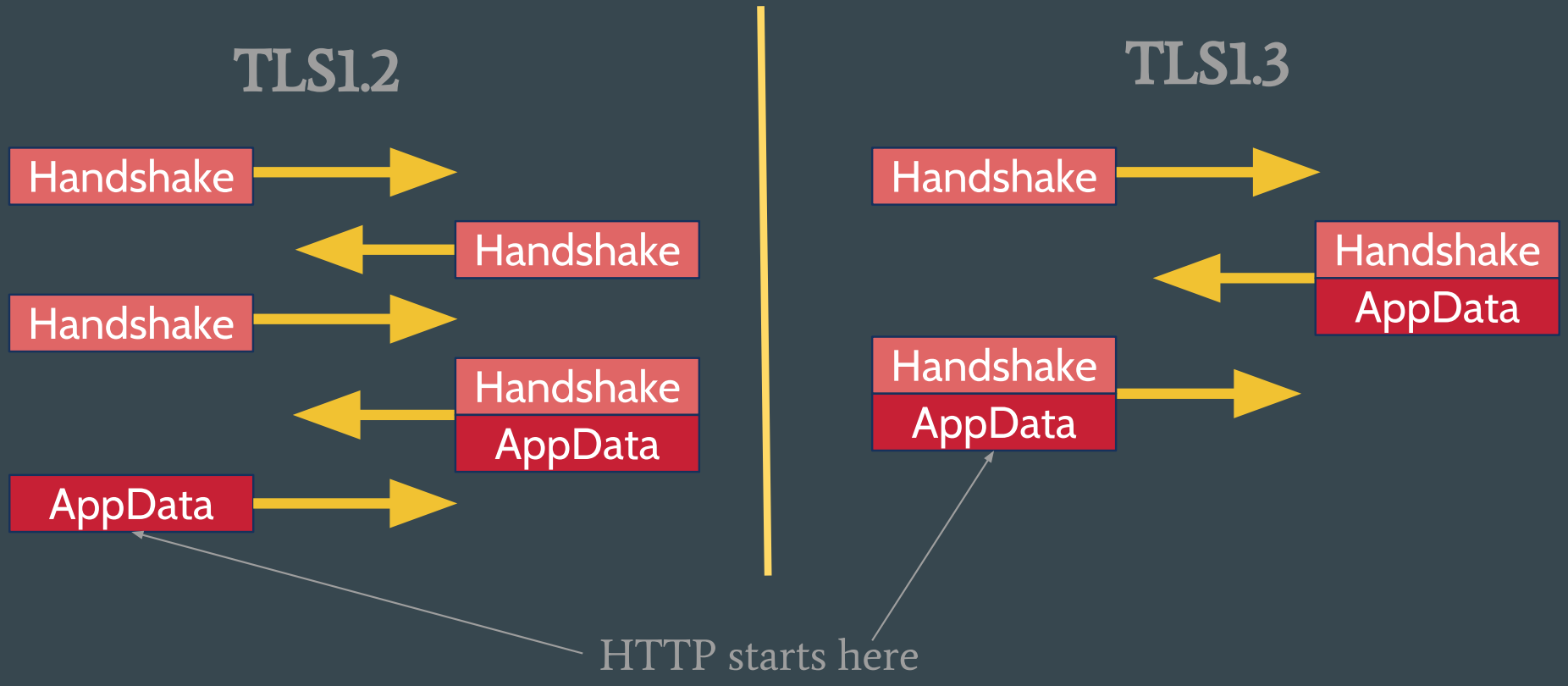
OLD: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

NEW: a la carte

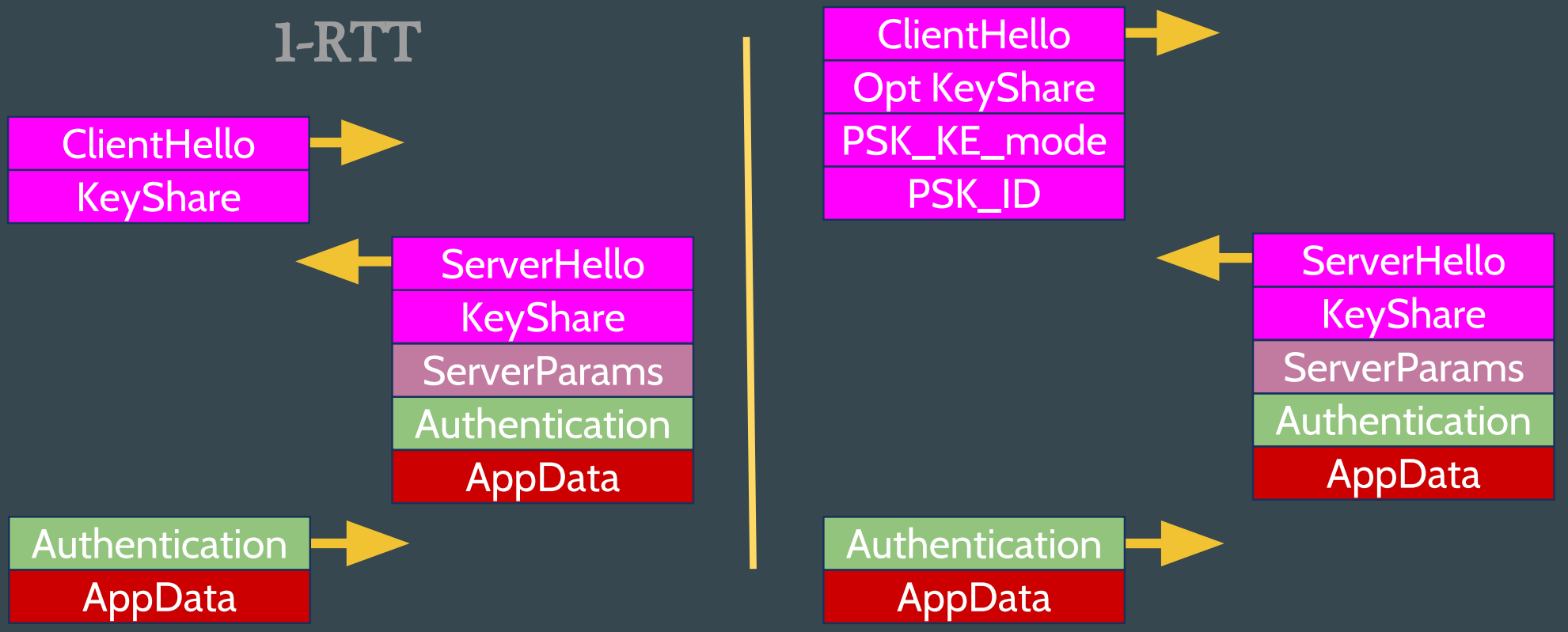


IANA Registry will include Recommended column

Come again - it's faster?

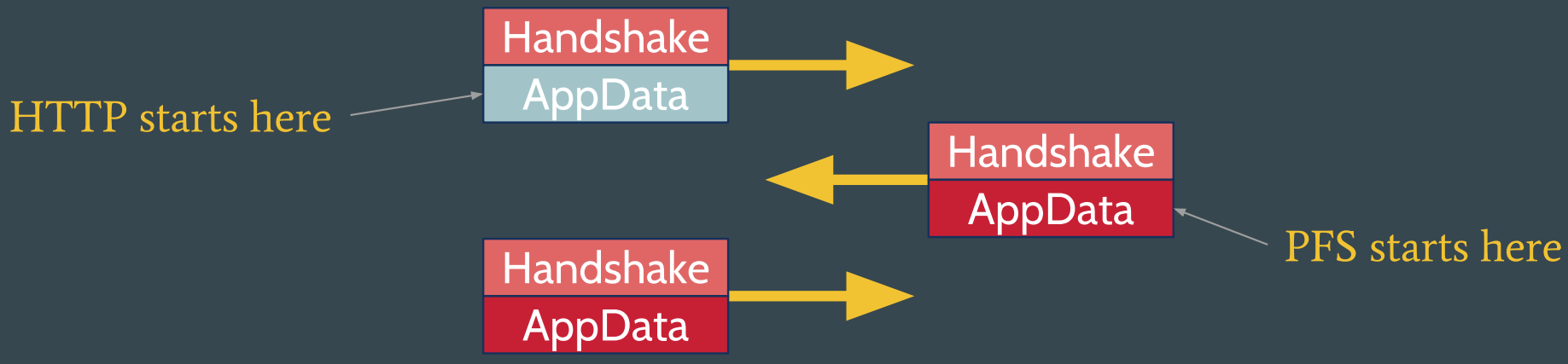


What are the normal modes?



Is that *all* you got?

TLS1.3 0-RTT Data



WARNING: 0-RTT Data is replayable and not PFS!