



Title

Subtitle

SubSubtitle

SS 2018 - WS 2018/2019

Authors

Contents

1	Begriffe	1
1.1	IT-Sicherheit: Ziele	1
1.2	IT-Sicherheit: optionale Ziele	1
2	Authentifizierung	4
2.1	Voraussetzung für Authentifizierung	4
2.2	Realisierung der Authentifizierung	4
2.2.1	Multi-Faktor-Authentifizierung	5
2.3	Ergebnis der Authentifizierung	5
2.4	Anforderungen an Authentifizierung	6
2.5	Authentifizierung mit Passwort	6
2.6	Angriffe auf Passwortauthentisierung	6
2.7	Biometrische Authentifizierung	7
2.7.1	False-Reject vs. False-Accept	7
2.8	Authentifizierung über Netzwerke	8
2.9	Einmalpasswörter	9
2.10	Challenge-Response-Authentifizierung	10
2.11	GSM-Authentifizierung	10
2.12	Private-Key-Authentifizierung	10
2.13	x.509 Authentifizierung	12
2.14	FIDO-Alliance: U2F (Universal Second Factor)	12
2.15	Single Sign On	13
2.16	Probleme	13
2.17	Mapping der Authentifizierungsinformation	14
2.18	Zentraler Authentifizierungsserver	14
2.19	Beispiel: Kerberos	15
2.19.1	Haupteigenschaften von Kerberos	15
2.19.2	Grundbausteine von Kerberos	15

Contents

2.19.3	Kerberos Verfahren	16
2.19.4	Datenelemente von Kerberos	16
2.19.5	Ablauf der Kerberos-Authentifizierung	17
2.19.6	Zugriff auf den Server	18
2.19.7	Interdomain-Authentifizierung	19
2.19.8	Stärken und Schwächen von Kerberos	19
2.20	Shibboleth Authentifizierung	20
2.21	OAuth	20
2.21.1	Rollen in OAuth	21
2.21.2	Ablauf von OAuth	21
2.21.3	OAuth-Flow	22
2.22	Diffie-Hellman	22
2.22.1	Problem: Kommunikation über unsicheren Kanal	23
2.22.2	Lösung: Diffie-Hellman	24
2.22.3	Diffie-Hellman: Schlüsselaustausch	24
2.22.4	Klassischer Diffie-Hellman	25
2.22.5	LogJam-Angriff	26
2.22.6	Elliptic Curves	27
3	Web Service Security	29
3.1	SSO über Trust Domänen hinweg	30
3.2	Federated Identity: Account Linking	31
3.3	XML - Signaturen	32
3.3.1	XML-Signatur - Enveloped Signature	32
3.3.2	XML-Signatur - Enveloping Signature	33
3.3.3	XML-Signatur - Detached Signature	33
3.3.4	XML-Signatur - Detached Signature2	34
3.4	XML-Encryption	34
3.4.1	Klartextnachricht	34
3.4.2	encrypted Element	35
3.4.3	encrypted content	35
3.4.4	KeyInfo in EncryptedData	36
3.5	Verschlüsseln und Signieren	36
3.6	SAML-assertions	36
4	Angriffe und Schwachstellen	37

1 Begriffe

1.1 IT-Sicherheit: Ziele

- **Vertraulichkeit:** Zugriff auf autorisierte Personen begrenzt.
- **Integrität:** Informationen nur von autorisierten Personen veränderbar.
- **Verfügbarkeit:** Autorisierte Personen können auf die Informationen zugreifen, wenn benötigt.

1.2 IT-Sicherheit: optionale Ziele

- **Auditierbarkeit:** Sicherheitsrelevante Eigenschaften einsehbar und überprüfbar.
- **Non-Repudation:** Aktionen am System nicht abstreitbar.
- **Accountability:** Änderungen am System immer einer Person zuzuordnen.
- **Privacy:** Personenbezogene Daten werden geschützt.
- **Authentizität:** Informationen einem bestimmten Sender zuzuordnen.
- **Deniability:** Inhalte oder Beiteilung einer Kommunikation im Nachhinein nicht nachweisbar.

1.2 IT-Sicherheit: optionale Ziele

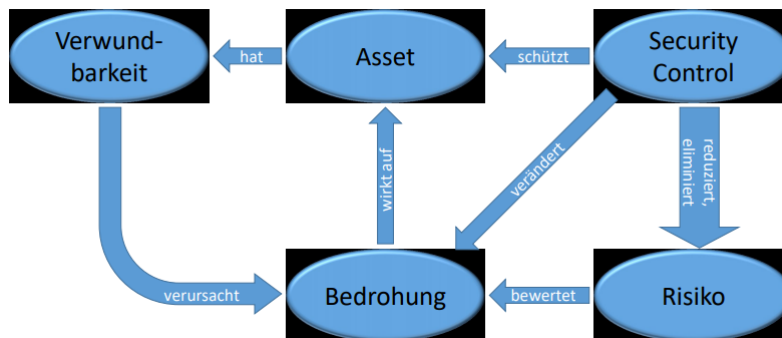


Figure 1.1

Asset

- Ressource, Prozess, Produkt oder System
- besitzt **Wert** für die Organisation
- muss **geschützt** werden

Bedrohung

- kann **unerwünschten** Effekt haben (schädlich)
- **Ursachen in der Umwelt** (Überschwemmung, Feuer)
- **Von Menschen verursacht** (Fehler oder Vorstatz)

Verwundbarkeit fehlender, oder schwacher Schutz eines Assets. Verursacht, dass Bedrohung:

- auftritt
- mit höherer Wahrscheinlichkeit oder Häufigkeit auftritt
- einen höheren Schaden verursacht

Risiko Kombination aus Wahrscheinlichkeit und Auswirkungen einer Bedrohung.

$$R_{Thread} = P_{Thread} \times D_{Thread}$$

$$P_{Thread} = \text{Eintrittswahrscheinlichkeit}$$

1.2 IT-Sicherheit: optionale Ziele

$$D_{Thread} = \text{Schaden}$$

Risikobewertung qualitativ oder quantitativ.

Security Control

- **Deterrent Control:** Verringert P_{Thread}
- **Preventative Control:** Eliminierung des Risikos durch entfernen der Schwachstelle.
- **Detective Control:** Erkennung der Bedrohung und Auslösung von:
- **Corrective Control:** Verringerung von D_{Thread}

2 Authentifizierung

- **Authentifizierung:** Nachweis der *Identität* eines Subjektes gegenüber einem anderen
- **Verifikation:** Identität wird bestätigt.
- **Identifikation:** Finden und Identifizieren eines Subjektes anhand von Referenzdaten (Fingerabdruck, Bild, ...)

2.1 Voraussetzung für Authentifizierung

Identifikationsmittel

- muss **eindeutig** sein
- kann **öffentlich** bekannt sein

Beweismittel

- meist unter Verschluss
- Beispiele: Passwort, private key, Fingerabdruck, preshared key, Iris, Chipkarte

2.2 Realisierung der Authentifizierung

Wissen

- Passwort
- PSK, SSH-Private-Key
- **Einfach anzugreifen**

2.3 Ergebnis der Authentifizierung

- **Einfach zu ändern**

Besitz

- SmartCard
- Schlüssel
- **Kann entzogen werden!**

⇒ **Nicht (einfach) zu kopieren!**

Eigenschaften

- Biometrisches Merkmal einer Person (Iris, Fingerabdruck)
- False acceptance/reject

⇒ **nicht revozierbar!**

2.2.1 Multi-Faktor-Authentifizierung

Kombination von **min. 2** verschiedenen Beweismitteln **unterschiedlicher** Kategorien.

⇒ **Kompromittierung eines Faktors reicht nicht aus!**

2.3 Ergebnis der Authentifizierung

Das Subjekt erhält **Authentifizierungsbeweis**:

- Session Cookie (Webbrowser)
- Session Key (TLS)
- Shell (Linux)

⇒ Identität wird auf **rechnerinternes** Objekt abgebildet.

⇒ **Schutz** des Authentifizierungsbeweises ist notwendig!

2.4 Anforderungen an Authentifizierung

Allgemeine Anforderungen

- Schutz des Authentifizierungsbeweises
- Schutz des Beweismittels
- Ergonomisch
- Einfach zu administrieren

Anforderungen bei Netzwerkauthentifizierung

- Keine sensiblen Daten über das Netz, im Klartext!
- Verhindern von Replay-Attacken!
- Verhindern von Man-in-the-Middle Angriffen!

2.5 Authentifizierung mit Passwort

- idR. wird das Passwort im Rechner als **Hash** hinterlegt. **Salted Hash:** Passwort wird um Salt ergänzt, gehashed und mit dem hinterlegtem Wert verglichen.
- Salt:
 - **Zufällige, pro Eintrag individuelle** Zeichensequenz
 - Verhindert, dass identische Passwörter mit identischen Hashes abgespeichert werden
 - Erschwert Wörterbuch- und Rainbowtable-Angriffe

2.6 Angriffe auf Passwortauthentisierung

Angriffe

- Wörterbuchattacken
- Wörterbuchattacken auf Hashes

2.7 Biometrische Authentifizierung

- Rainbow-Tables

Gegenmaßnahmen

- Langsame Hash-Algorithmen verwenden
- Salt
- Schutz der Hashes vor Auslesen
- Automatisches Sperren der Authentifizierung nach definierter Anzahl von Fehlversuchen
- Multifaktor-Authentifizierung

2.7 Biometrische Authentifizierung

- verwendet **individuelle Körpermerkmale** (Fingerabdruck, Irismuster, etc)
- Wesentliches Merkmal: biometrische Eigenschaften sind **immer leicht unterschiedlich**

Ablauf:

- Einlernphase: mehrere Datenproben werden entnommen, daraus **Referenzdaten** erstellt
- Authentifizierung: neue Datenprobe wird genommen und mit Referenzdaten verglichen.
Genügend Ähnlichkeit \Rightarrow authentifiziert.

2.7.1 False-Reject vs. False-Accept

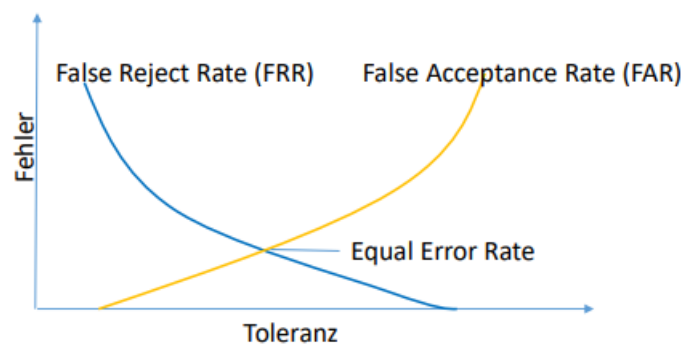


Figure 2.1

2.8 Authentifizierung über Netzwerke

False Reject Rate (FRR)

- Anteil der fälschlicherweise fehlgeschlagenen Authentifizierungsversuche
- Hohe FRR verringert Akzeptanz des Systems

False Acceptance Rate (FAR)

- Anteil der erfolgreichen Authentifizierungsversuche, bei denen fälschlicherweise eine andere Person akzeptiert wird
- Hohe FAR verringert die Sicherheit deutlich!

Equal Error Rate

- Gibt die Toleranzschwelle an, bei der genauso viele Personen fälschlicherweise abgelehnt wie fälschlicherweise akzeptiert werden ($FAR = FRR$)
- ist ein **Gütekriterium** für das biometrische Verfahren
- ist **nicht die Toleranzschwelle** mit der das System betrieben wird (normalerweise gilt: $FAR < FRR$)

2.8 Authentifizierung über Netzwerke

zu Verhindern:

- Abhören
- Replay
- Man-in-the-Middle
- Übernehmen der authentifizierten Verbindung
- Ausfall des Authentifizierungssystems

Erreicht wird dies durch..

- Authentifizierung per Passwort (aber Verschlüsselte Verbindung)
- Zeitabhängige Passwörter / Einmalpasswörter

2.9 Einmalpasswörter

- Challenge-Response Authentifizierung
- Zertifikats- bzw. Public/Private-Key-Authentifizierung

2.9 Einmalpasswörter

Passwörter werden durch **Zähler** oder basierend **Zeitstempel** generiert. Client und Server teilen gemeinsamen Schlüssel K_A (individuell pro Client). Der Schlüsselwert wird verwendet um:

- aus einem Zählerwert C und K_A das Einmalpasswort zu bestimmen.
- aus dem aktuellen Zeitstempel T und K_A das Einmalpasswort zu bestimmen.

Server und Client berechnen unabhängig voneinander das Einmalpasswort. Die Authentifizierung ist erfolgreich, wenn das Einmalpasswort übereinstimmt.

Vorteile:

- Abgefangene Passwörter sind wertlos
- gemeinsamer Schlüssel K_A kann clientseitig auf auslese-sicherer Hardware hinterlegt werden.
- Server erkennt Replay-Attacken

Nachteile:

- Gemeinsamer Schlüssel K_A muss sicher auf Client und Server hinterlegt werden
- Ist K_A bekannt können Einmalpasswörter vorrausberechnet werden.
- Server hat Liste aller gemeinsamer Schlüssel
- Anfällig gegenüber Man-in-the-Middle.
 - Separater Schutz gegen MiM notwendig!
 - Typisch: Public-Private-Key-Auth des Server mittels SSL-Zertifikaten

2.10 Challenge-Response-Authentifizierung

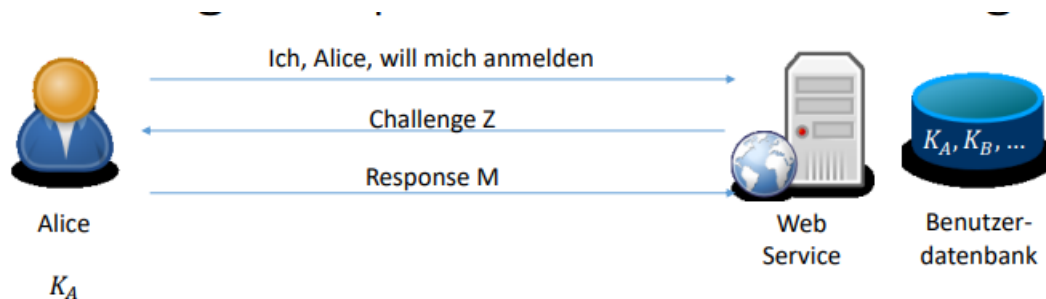


Figure 2.2

1. Alice schickt ihren Namen an den Server
2. Server antwortet mit einer Nonce Z ('Number used once') an Alice
3. Alice berechnet und schickt $M = Hash(K_A, Z)$ an den Server
4. Server berechnet $M' = Hash(K_A, Z)$, falls $M' = M$ dann ist Alice authentifiziert.
5. Alice und Server erzeugen gemeinsamen Session Key $K_S = KDF(K_A, Z)$
6. Weitere Kommunikation wird mit K_S verschlüsselt. K_S ist der Authentifizierungsbeweis.

2.11 GSM-Authentifizierung

Authentifizierung für eine hohe Anzahl von Teilnehmern. Zweck:

- Missbräuchliche Verwendung des Mobilfunknetzes verhindern
- Abrechnung von Mobilfunknutzung

Implementierung: Challenge-Response-Auth: SIM-Karte enthält K_A , der nicht direkt ausgelesen werden kann.

2.12 Private-Key-Authentifizierung

Authentifizierung durch Nachweis des Besitz des Privaten Schlüssels.

2.12 Private-Key-Authentifizierung

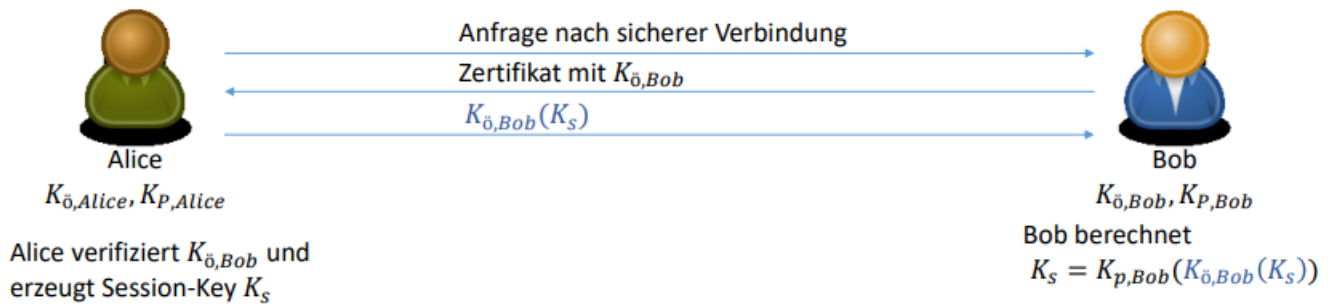


Figure 2.3

Probleme:

- Session-Key K_S wird über Netzwerk übertragen, und mit $K_{pub, Bob}$ verschlüsselt
- Wird K_{pri} bekannt, können **aufgezeichnete Sessions im Nachhinein entschlüsselt werden**
- Jede Schwachstelle, die K_{pri} freigibt ermöglicht Entschlüsselung aller vergangenen Sessions!
- Erneuerung des SSL-Zertifikats **tauscht Schlüssel idR nicht aus**

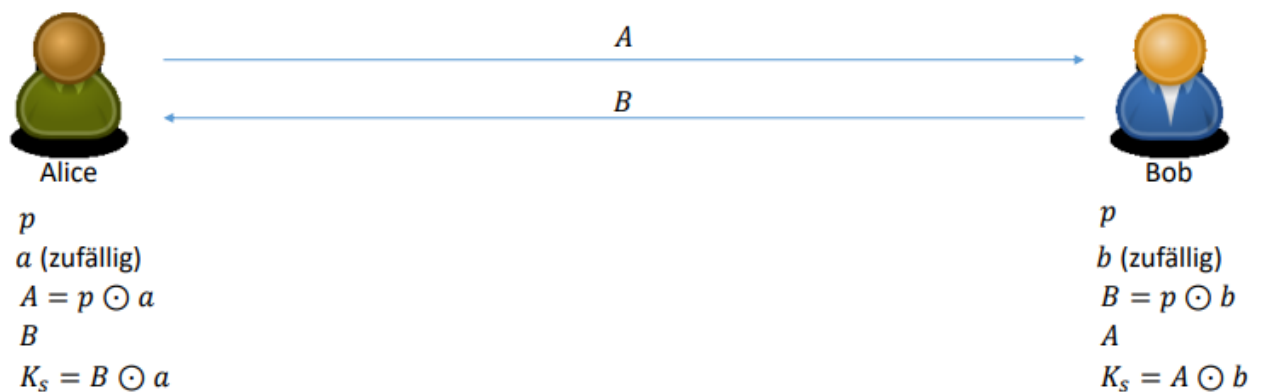


Figure 2.4: Diffie-Hellman Schlüsselaustausch

Abhilfe: Diffie-Hellman-Schlüsselaustausch

- K_S wird via DH-Algorithmus erzeugt.

2.13 x.509 Authentifizierung

- Anwendung bei SSL in Cipher-Suites: DHE (Diffie-Hellman Ephemeral) oder ECDHE (Elliptic Curve, DHE)
- **Perfect-Forward-Secrecy:** Kein nachträgliches Entschlüsseln von Sessions durch Bekanntwerden von K_{priv}

2.13 x.509 Authentifizierung

Authentifizierung mittels Zertifikaten:

- Alice \Rightarrow Bob: $Sign[K_{priv,A}, K_{pub,B}(K_S, Alice, Z, T)]$
- $K_{priv,A}$ private Key von Alice
- $K_{pub,B}$ public Key von Bob
- K_S der Session-Key
- Z eine Nonce (Replay Schutz)
- T ein Zeitstempel (Replay Schutz)

1. Alice ist authentifiziert, da Bob die **Signatur** von Alice prüft.
2. Durch **Angabe der Identität** in den signierten Daten wird verhindert, dass sich ein MiM einschaltet und die Signatur ersetzt
3. Die Kombination von Z und T dient als **Replay Schutz**: Der Server merkt sich für eine bestimmte Zeit alle Nonces und lehnt Auth ab, wenn eine Noce wiederverwednet wird.

2.14 FIDO-Alliance: U2F (Universal Second Factor)

Standard von 2014 für zert-basierte Auth. via USB oder NFC-Token.

Initialisierung:

- Token erzeugt Schlüsselpaar für jede Site. Privater Schlüssel verlässt das Token nicht.
- Key-Handle identifiziert das Schlüsselpaar und muss die Information encodieren, für welche Website das Schlüsselpaar ist

2.15 Single Sign On

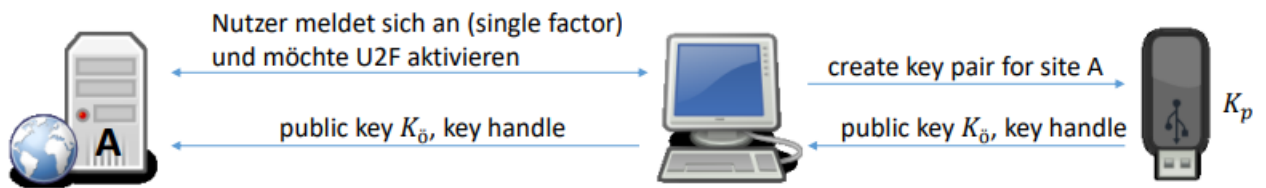


Figure 2.5

Authentifizierung:

- Basierend auf dem Nutzernamen und der optionalen Auth mit Passwort oder PIN liefert der Server keyhandle und Noce
- Token Signiert Noce mit dem zum KeyHandle passenden K_{priv} .

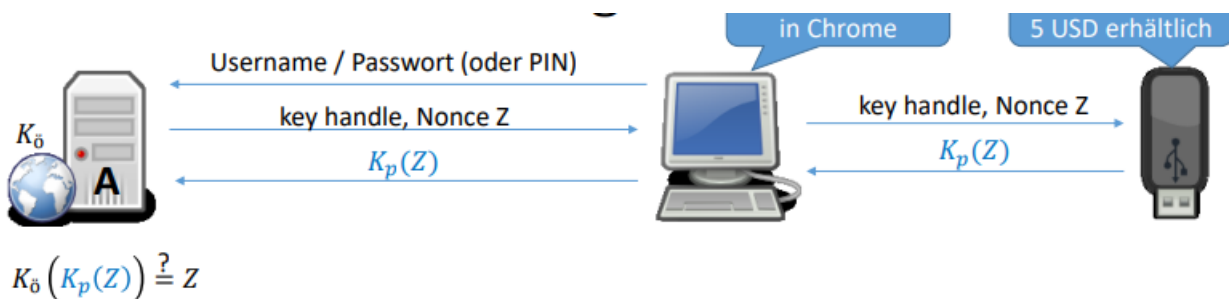


Figure 2.6

2.15 Single Sign On

If someone steals my Laptop while I am logged inn, they can do everything but install drivers...

2.16 Probleme

- einfache Passwörter
- wiederverwendete Passwörter
- unsicher gespeicherte Passwörter

2.17 Mapping der Authentifizierungsinformation

- Software/Dienst hält alle Auth-Infos
- Nutzer entsperrt diese Daten mit **Masterpassword**
- **Vorteile:**
 - starke Servicepasswörter (generiert)
 - nur **ein Passwort** zu merken (deshalb meistens besseres PW!)
- **Nachteile:**
 - Ausfall des Mappingdienst sperrt sämtliche Zugriffe
 - Sicherheit hängt an einem Passwort

2.18 Zentraler Authentifizierungsserver

- Nutzer authentifiziert sich bei zentralen Server (AS) und erhält ein Token.
- Token \Rightarrow Auth bei Zielsevern

Zu lösendes Problem:

- Token muss an die ID gebunden sein.
- Token darf auf dem Netzwerk nicht geklaut werden
- Server müssen verifizieren können, dass das Token vom korrektem Eigentümer verwendet wird.

\Rightarrow Komplexe Sicherheitsprotokolle.

Vorteile:

- nicht clientgebunden
- zentrale Administration
- Hohe Sicherheit bei starker Anfangsauthentisierung

Nachteile:

- Jeder Server muss das Protokoll können \Rightarrow aufwendige Migration
- AS ist single point of failure

2.19 Beispiel: Kerberos

- SSO-System (Single Sign On) mit symmetrischen Schlüsseln
- eingesetzt in Windows, Linux, NFS, SAP, Oracle ...

2.19.1 Haupteigenschaften von Kerberos

- Zentrale Komponente hat Kenntnis aller Schlüssel (alle permanenten Schlüssel der Principal und Session-Keys)
- Datenelemente sind Ticket und Authenticator
- Authentifizierung erfolgt in drei Schritten
 1. Ausstellung des Ticket Granting Ticket (TGT)
 2. Ausstellung des Zielserverticket
 3. Kommunikation mit dem Zielserv

2.19.2 Grundbausteine von Kerberos

- Key Distribution Center (KDC):
 - Authentifizierung (AS)
 - Ticket Granting Service (TGS)
 - Authentifizierung, Tokenerstellung, Schutz der Tokens
- Registry (sichere DB):
 - enthält Namen und Schlüssel aller Benutzer

2.19.3 Keberos Verfahren

- Kerberos verwendet **nur symmetrische Kryptographie**
- Server müssen um Kerberos-Komponente erweitert werden
- Clients müssen um Kerberos-Komponente erweitert werden

2.19.4 Datenelemente von Kerberos

- **Schlüssel:**
 - **für Personen:** Schlüssel werden aus Passwort abgeleitet.
 - **für Server:** starke Schlüssel werden zufällig generiert und mit Betriebssystemmitteln geschützt.
- **Tickets:**
 - erlauben Nutzer (Prinzipal) die Authentifizierung an System oder Dienst
 - Tickets transportieren Session-Keys
- **Authenticator:**
 - Binden Tickets an den Eigentümer
 - bieten Replay-Schutz

Inhalt eines Kerberos-Tickets

- Zielsevername
- verschlüsselt mit Schlüssel des Zielsevers, ausgestellt durch TGS (Ticket Granting Service):
 - Clientname
 - Session-Key
 - Gültigkeitsdauer

Inhalt eines Kerberos-Authenticators

verschlüsselt mit Session-Key aus dem zugehörigem Ticket:

- Clientname
- Zeitstempel als Replay-Schutz
- Hashwert für mitgelieferte Daten

Eigenschaften:

- Kann mehrmals verwendet werden
- Authenticator wird jedes mal vom Client neu erzeugt

2.19.5 Ablauf der Kerberos-Authentifizierung

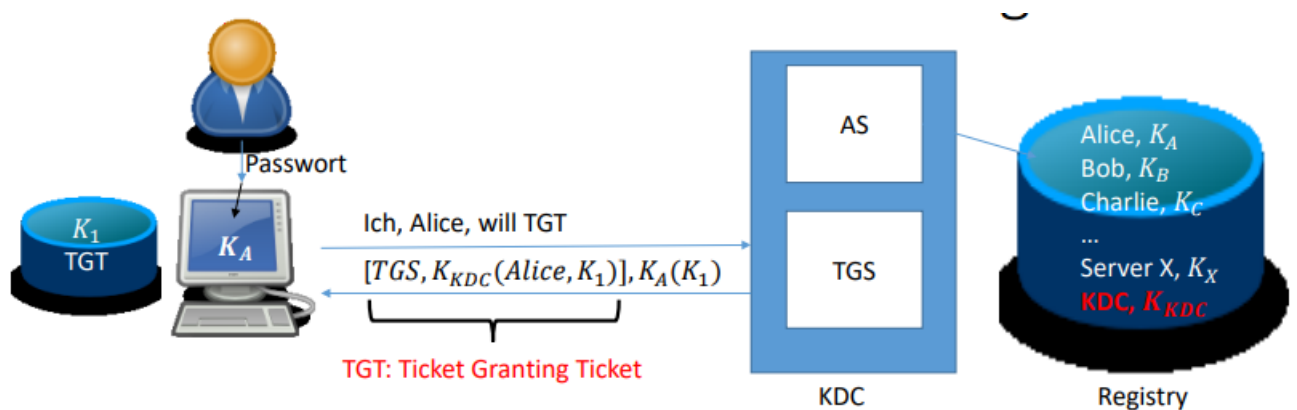


Figure 2.7

1. Alice fragt TGT an
2. AS liest Alices Schlüssel aus der Registry aus
3. KDC erzeugt ein Ticket für Alice zur Nutzung des TGS. \Rightarrow TGT (**T**icket **G**ranting **T**icket)
4. Alice gibt ihr Passwort ein, daraus wird K_A berechnet
5. Mit K_A entschlüsselt Alice den Session-Key K_1
6. Alice speichert K_1 und das TGT lokal

2.19 Beispiel: Kerberos

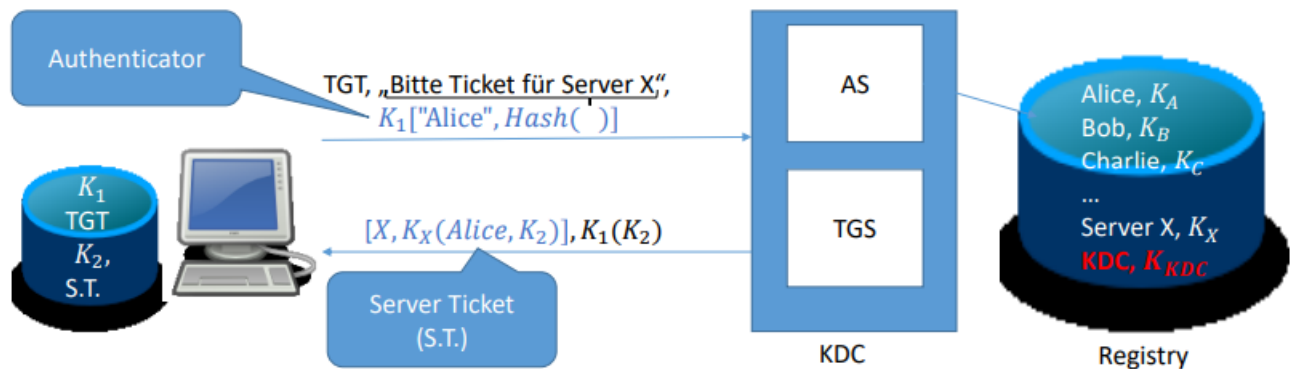


Figure 2.8

1. Alice schickt TGT mit Anfrage nach Server-Ticket an TGS.
2. TGS kann TGT entschlüsseln, liest daraus den Session-Key K_1 und prüft damit den Authenticator \Rightarrow Anfrage kann nur von Alice kommen
3. TGS schickt Server Ticket und zusätzlich den Session-Key K_2 verschlüsselt für Alice
4. Alice speichert Server Ticket und den Session Key K_2

2.19.6 Zugriff auf den Server



Figure 2.9

- Alice schickt Server Ticket und Authenticator an Server
- Server kann mit K_X das Serverticket entschlüsseln und mit dem darin enthaltenen K_2 Authenticator von Alice überprüfen \Rightarrow Nachricht M ist authentisch von Alice! K_2 ist Authentifizierungsbeweis von Alice für X

2.19.7 Interdomain-Authentifizierung

Wenn sich zwei Key-Distribution-Center KDC_1 und KDC_2 gegenseitig als Server eingetragen haben dann:

- kann sich ein Nutzer von KDC_1 ein TGT für KDC_2 besorgen
- mit diesem TGT kann sich der Nutzer aus der Domäne von KDC_1 dann via KDC_2 Server Tickets für Server der Domäne von KDC_2
- Interdomain-Auth kann auch einseitig erfolgen (KDC_1 stellt TGTs für KDC_2 aus, aber nicht umgekehrt)

2.19.8 Stärken und Schwächen von Kerberos

Stärken

- Protokoll ist gut analysiert (da alt)
- clientunabhängiges SSO bei allen Teilnehmern einer Domäne
- flexibles und erweiterungsfähiges Protokoll

Schwächen

- Bei menschlichen Nutzern kann die Auth-anfrage für Passwortattacken genutzt werden (Key wird aus Passwort abgeleitet)
- KDC hat keine Zustandsinformationen (Logoff nur durch Ablauf des Tickets)
- KDC muss absolut vertrauenswürdig sein (ansonsten Golden Ticket möglich!)
- Client muss sicher sein
- keine Rechteverwaltung

2.20 Shibboleth Authentifizierung

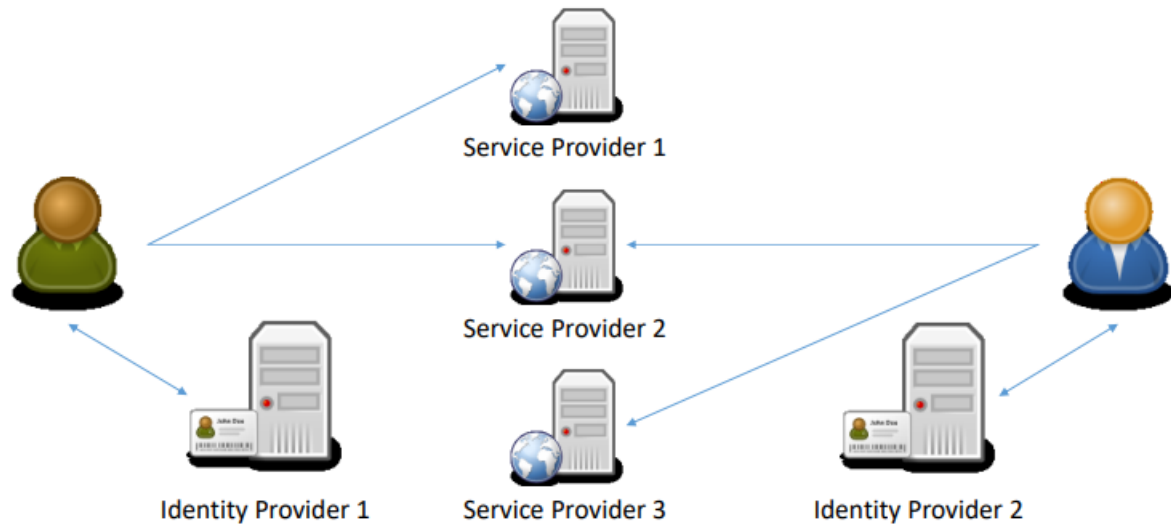


Figure 2.10

Es gibt **Identity Provider (IdP)** und **Service Provider (SP)**. Ablauf des Single-Sign-ONS (SSO):

1. Client möchte Zugriff auf die Website
2. Wenn dort nicht authentifiziert \Rightarrow Umleitung auf den IdP (auch mehrere IdPs möglich)
3. Bei IdP wird Auth durchgeführt
4. Mit Bestätigung der Auth (Assertion) wird erneut der Service-Provider kontaktiert. \Rightarrow bei weiteren SPs ist kein erneutes Login möglich
5. IdP kann auch servicespezifische Attribute an den SP weiterreichen.

Existenz von Sessions wird durch entsprechende Cookies im Browser bestätigt.

2.21 OAuth

- eigentlich ein **Authorisierungs**-Protokoll, kann aber zur **Authentifizierung** eingesetzt werden (?)

2.21 OAuth

- Webanwendung will Zugriff auf die Ressourcen einer anderen Webanwendung (zB. Facebook-Pic auf Stackexchange)
- mit OAuth: Stackexchange kann mit Zustimmung des Facebook-Nutzers eine genau definierte Teilmenge der Nutzerrechte erhalten

2.21.1 Rollen in OAuth

- **Resource Owner (auch End-User)**: entscheidet über Art und Umfang der Zugriffsberechtigungen
- **Resource Server**: im Besitz der Ressourcen. Er gewährt Zugriff, wenn **Access Token** präsentiert wird.
- **Client**: Webanwendung, die auf geschützte Ressource zugreifen will.
- **Auth-Server**: Server **authentifiziert** Client und übergibt im daraufhin das entsprechende Access-Token.

2.21.2 Ablauf von OAuth

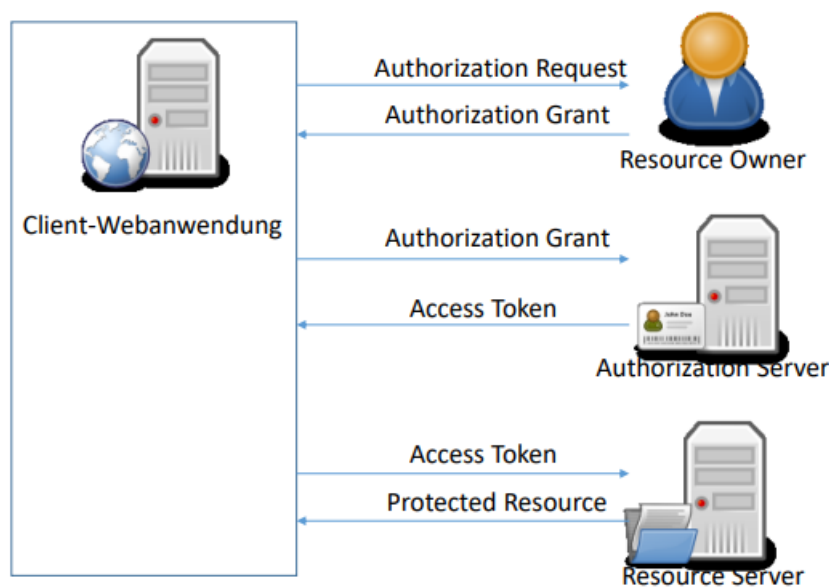


Figure 2.11

- Alle Verbindungen über **https**

2.22 Diffie-Hellman

- Der Authorization-Request kann indirekt über den Authorization-Server erfolgen

2.21.3 OAuth-Flow

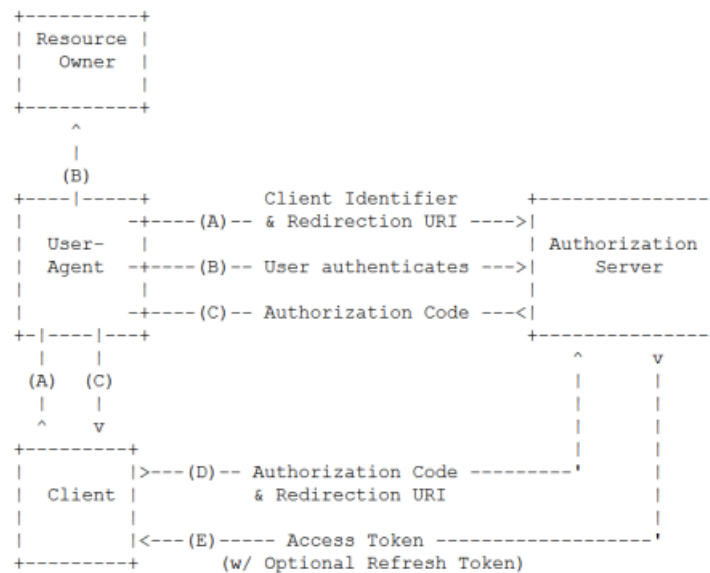


Figure 2.12

- empfohlene Abweichung vom ursprünglichem Flow für den Fall, dass es sich bei dem Client ebenfalls um eine Webanwendung handelt.
- Client und Authorization-Server müssen die Möglichkeit haben, im Browser einen **redirect** anzustoßen.

2.22 Diffie-Hellman

- erlaubt zwei Kommunikationspartnern das Generieren von einem geheimen Schlüssel über eine unsichere Verbindung
- Für MiN-Angriffe sind zusätzlich Zertifikate und Signaturen nötig
- Elliptic Curves lösen Primzahlenkörper als Konstrukt ab, um geeignete Einwegfunktionen zu berechnen.

2.22.1 Problem: Kommunikation über unsicheren Kanal

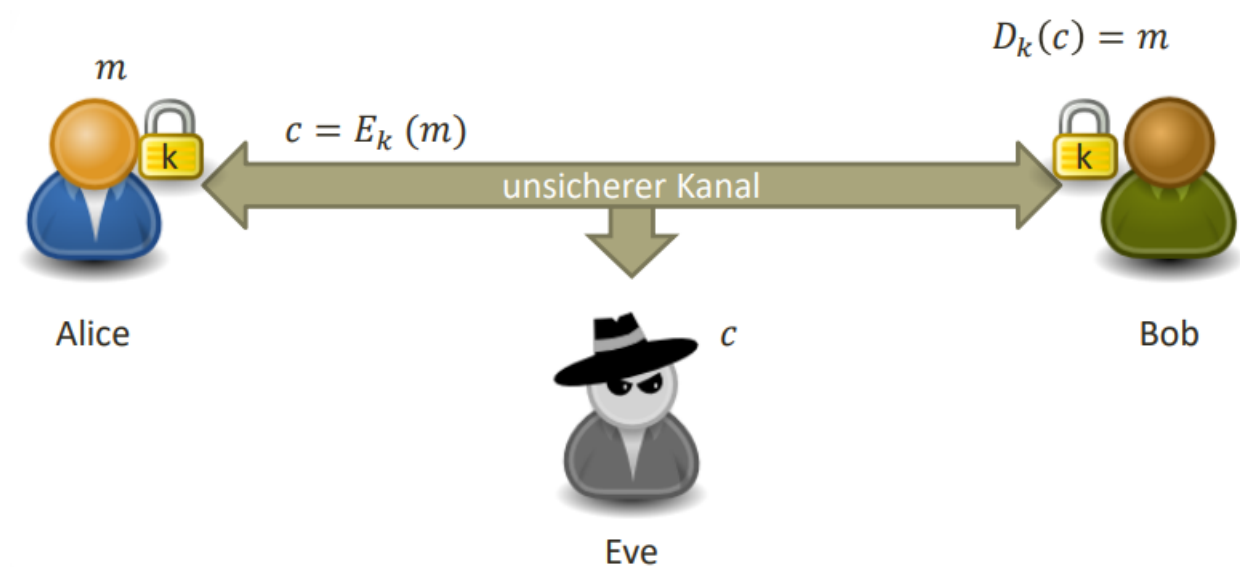


Figure 2.13

Probleme:

- Wie können A und B sicherstellen, dass nur sie k kennen?
- Wenn k über einen sicheren Kanal ausgetauscht werden muss, warum nicht dann gleich die ganze Nachricht

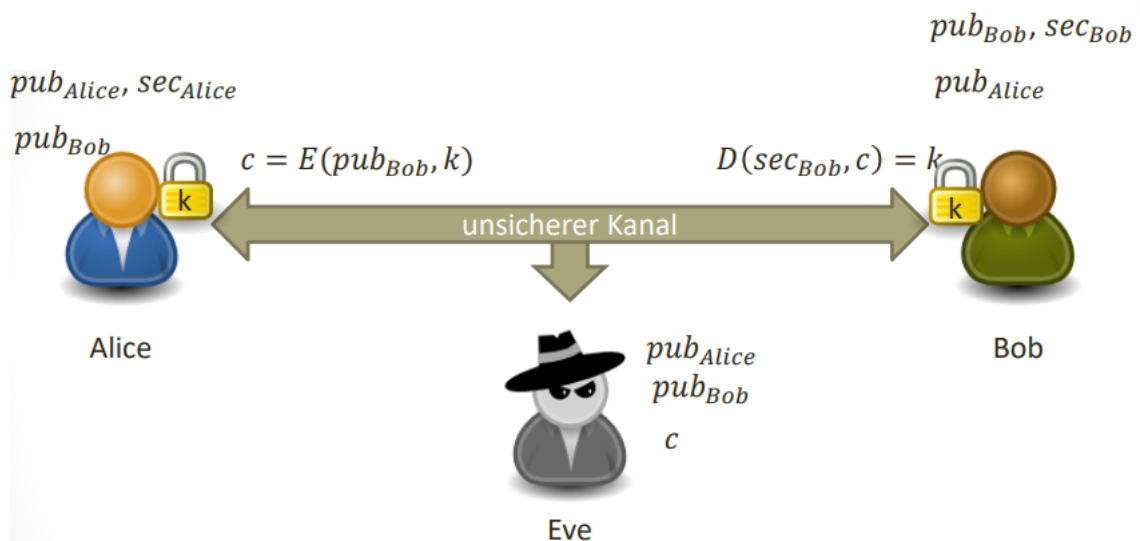


Figure 2.14

Probleme:

- Wie kann sichergestellt werden, dass der Public Key zum angeblichem Owner gehört?
⇒ **Digitale Signaturen**
- **Kompromittierte Schlüssel:** Sekret Key hat lange Lebensdauer und wird oft wiederverwendet. Auch im Nachhinein kann aus c bei Bekanntwerden des private Keys k berechnet werden.

2.22.2 Lösung: Diffie-Hellman

Idee: Finde und benutze Rechenoperationen \odot , die:

- kommutativ sind: $A \odot B \odot C = A \odot C \odot B$
- einfach durchzuführen, aber schwer umzukehren:
 - **einfach:** $A \odot B \rightarrow X$
 - **schwierig:** $\odot^{-1}(A, X) \rightarrow B$

2.22.3 Diffie-Hellman: Schlüsselaustausch

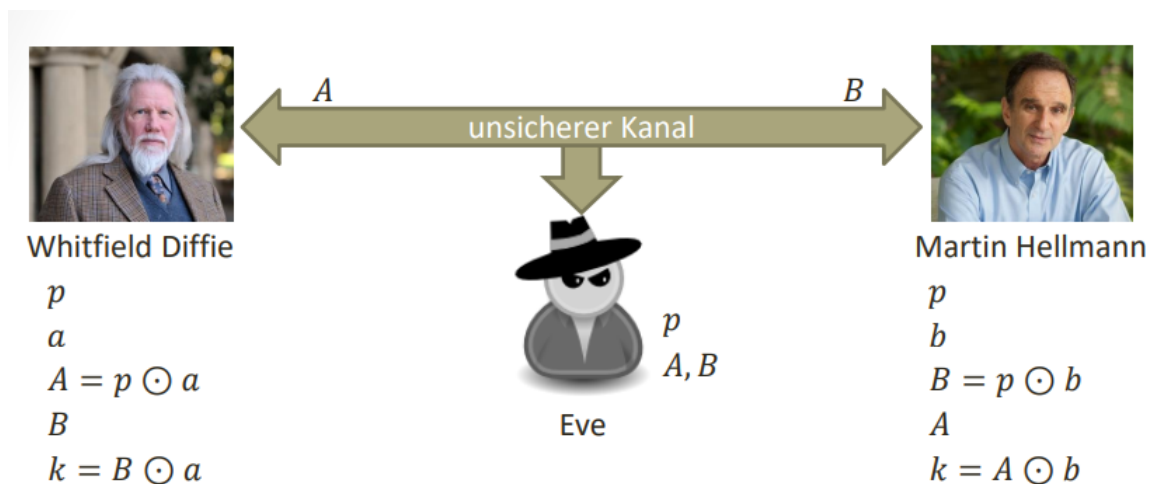


Figure 2.15

Beide Kommunikationspartner:

- einigen sich auf Operanden p von \odot (public)

2.22 Diffie-Hellman

- wählen den anderen Operanden a, b (privat)
- führen \odot durch und übertragen das Ergebnis A, B
- berechnen den gemeinsamen geheimen Schlüssel:
 $k = p \odot a \odot b = p \odot b \odot a = A \odot b = B \odot a$:

2.22.4 Klassischer Diffie-Hellman

Mathematisches Konstrukt: Primzahlenkörper

- $x \odot y = g^{x \times y} \bmod p$
- Potenzieren ist einfach
- Logarithmieren ist schwierig

Ablauf:

1. **Public Informationen** austauschen:

Primzahl p

Generator g , wobei gilt $g < p$

2. Alice wählt a , berechnet $A = g^a \bmod p$ und sendet A
3. Bob wählt b , berechnet $B = g^b \bmod p$ und sendet B
4. beide berechnen
 $k = g^{a \times b} \bmod p = A^b \bmod p = B^a \bmod p$

Sicherheit:

- Hängt stark von der **Länge der Primzahl** ab
- bis 512bit \Rightarrow bereits gebrochen
- bis 768bit \Rightarrow mit moderatem Aufwand zu brechen
- bis 1024bit \Rightarrow vermutlich mit staatlicher Unterstützung zu brechen

Number Field Sieve:

2.22 Diffie-Hellman

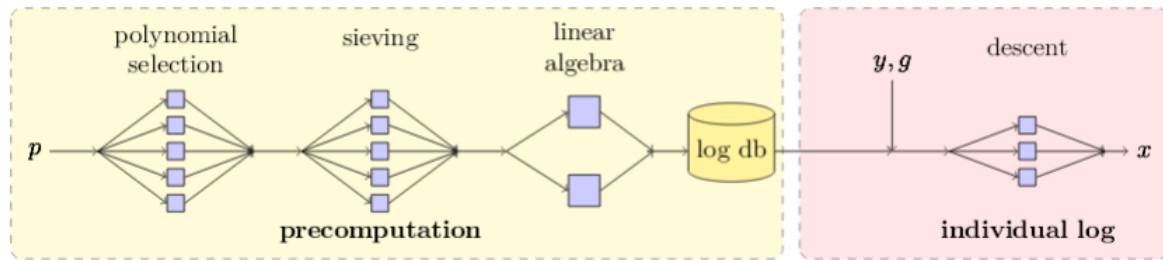


Figure 2.16

2.22.5 LogJam-Angriff

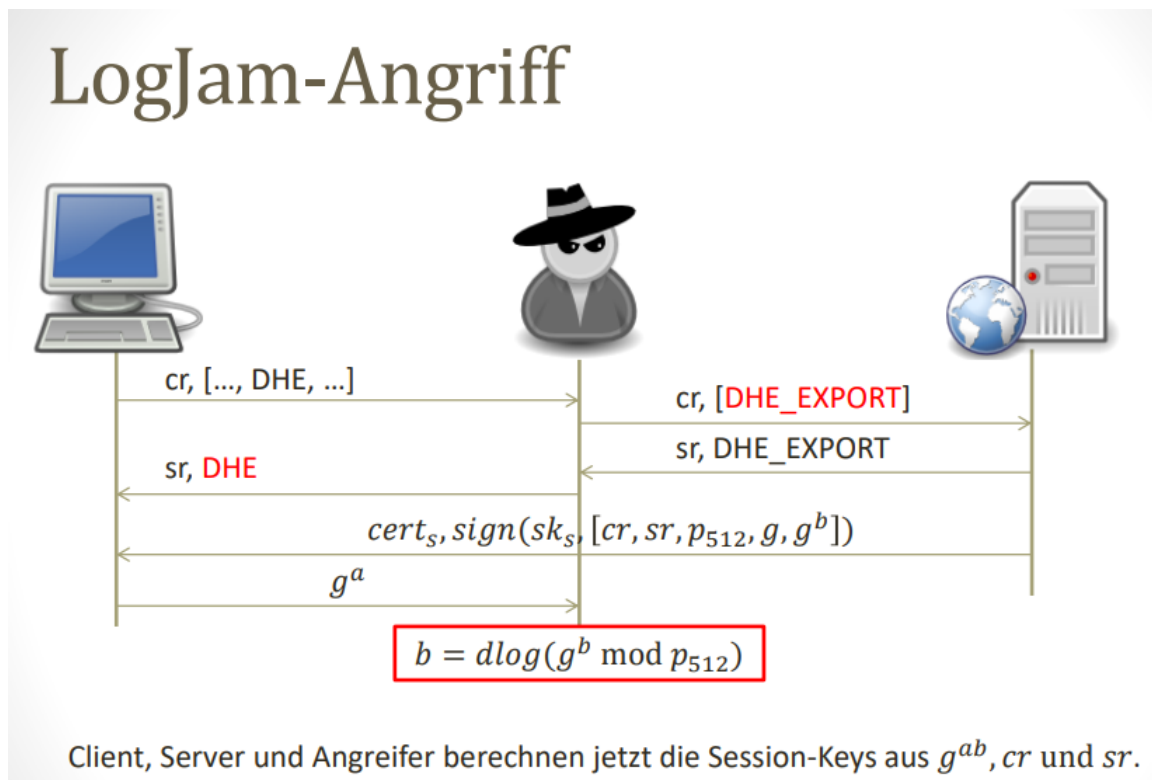


Figure 2.17

⇒ Downgrade-Angriffe: Angreifer zwingt die Teilnehmer, den Schlüsselaustausch mit einer 512bit Primzahl zu vollziehen. In diesem Zahlenraum können die Logarithmen bereits vorberechnet werden.

2.22.6 Elliptic Curves

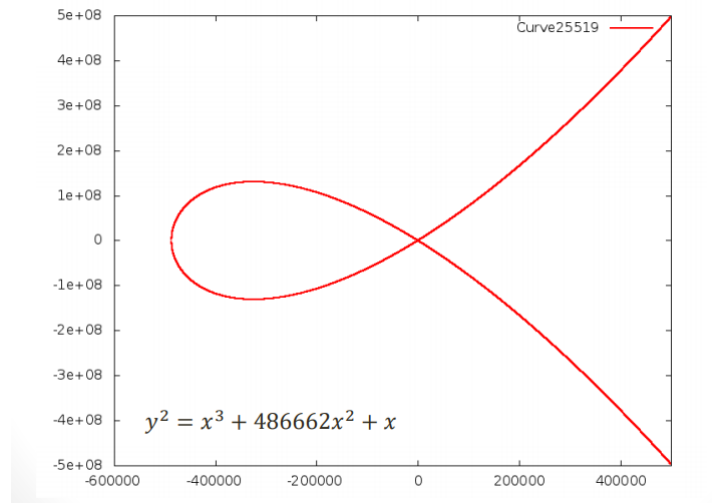


Figure 2.18

- alternative Menge, auf der \odot durchgeführt wird
- \odot : Multiplikation von Punkten auf einer Elliptic-Curve mit einem Integer
- **Public Informationen:**
 - Definition der Kurve
 - Startpunkt G auf dieser Kurve
 - errechneterpunkt $Q = nG$
- **Private Information:** die Zahl n

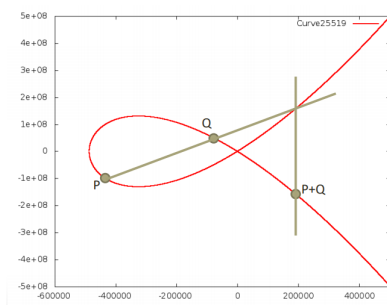


Figure 2.19: Addition

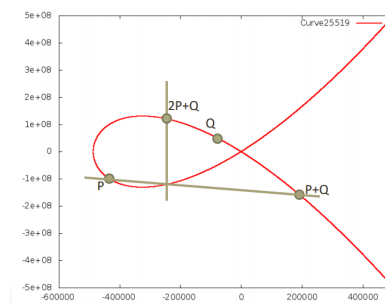


Figure 2.20: Addition

2.22 Diffie-Hellman

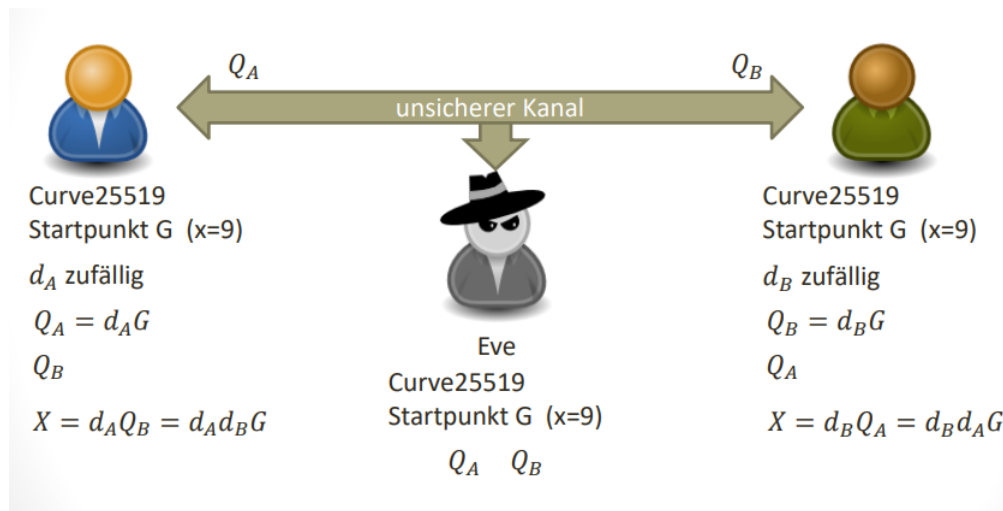


Figure 2.21: Ablauf

3 Web Service Security

- **Frühe Architektur: Die Insel:** Daten und Applikationen bleiben lokal und werden dort geschützt.
- **Später: Burgarchitektur:** Applikationen bleiben lokal, 'Burgen' sind vernetzt, Daten müssen auch auf dem Transport geschützt werden
- **Heute: Cloudarchitektur:** Daten und Applikationen können überall sein. Daten müssen unabhängig vom Ort geschützt werden.

- **Die IT in der Wirtschaft**
 - IT ist Dienstleister zur Durchführung verteilter Geschäftsprozesse
 - Weltweit verteilte IT-Ressourcen werden je nach Bedarf genutzt.
 - Immer höhere Firmenwerte liegen in den IT-Services (Banking, Shops, Web 2.0....)
- **IT-Architektur**
 - Service Oriented Architecture
 - Geschäftsprozesse werden durch verteilte Services unterstützt, diese müssen koordiniert werden (orchestriert)
 - Insel- und Burgarchitekturen werden abgelöst durch Cloudlösungen.
 - Absicherung von Objekten
 - Nicht mehr auf Rechnerebene, sondern auf Dateiebene
 - Zugriffsschutz muss über Firmengrenzen hinweg gewährleistet werden.
 - Rechtemanagement muss für verteilte Systeme implementiert werden.
- **Angriffe auf die IT**
 - Neue Bedrohungsdimensionen
 - Speicherung von Informationen und Datenverarbeitung in der Cloud
 - Vernetzung praktisch aller Devices (Mobile Devices, Haushaltsgeräte)
 - Netzwerkgrenzen werden unscharf (Firewall, DMZ helfen nicht weiter)
 - Erforderliches technisches Wissen ist universell verfügbar
 - Komplexität der rechtlichen Situation (Internationalität) garantiert Quasi-Straffreiheit
- **Technologische Trends**
 - Message Level Security statt Transport Level Security (End to End statt Hop to Hop)
 - Integrated Rights Management / Data Leakage Prevention: Rechte werden an content gekoppelt, unabhängig vom Aufenthaltsort
 - Identity Federation: Kopplung von Identitäten über Firmengrenzen hinweg.

Figure 3.1

Folgen dieser Entwicklungen:

- **Unscharfe Netzwerkgrenzen: klassischer Perimeterschutz** versagt zunehmend
- Umgehung von Sicherheitsmechanismen zunehmend leichter für Nutzer (Dropbox statt firmeneigener Dateiablage, etc.)

3.1 SSO über Trust Domänen hinweg

⇒ Sicherheit auf Kommunikationswegen muss ergänzt werden um Mechanismen, welche Daten überall schützen.

⇒ Produktivitätseinschränkende Sicherheitsmechanismen laufen ins Leere!

3.1 SSO über Trust Domänen hinweg

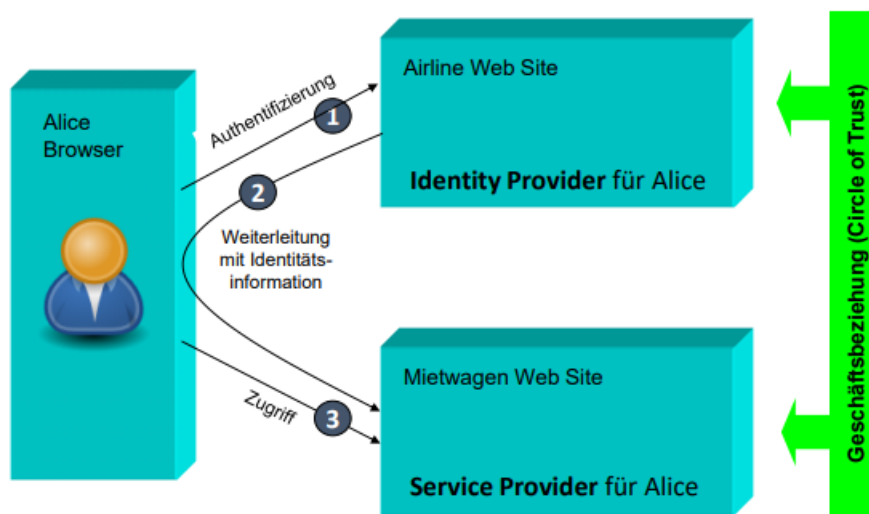


Figure 3.2

3.2 Federated Identity: Account Linking

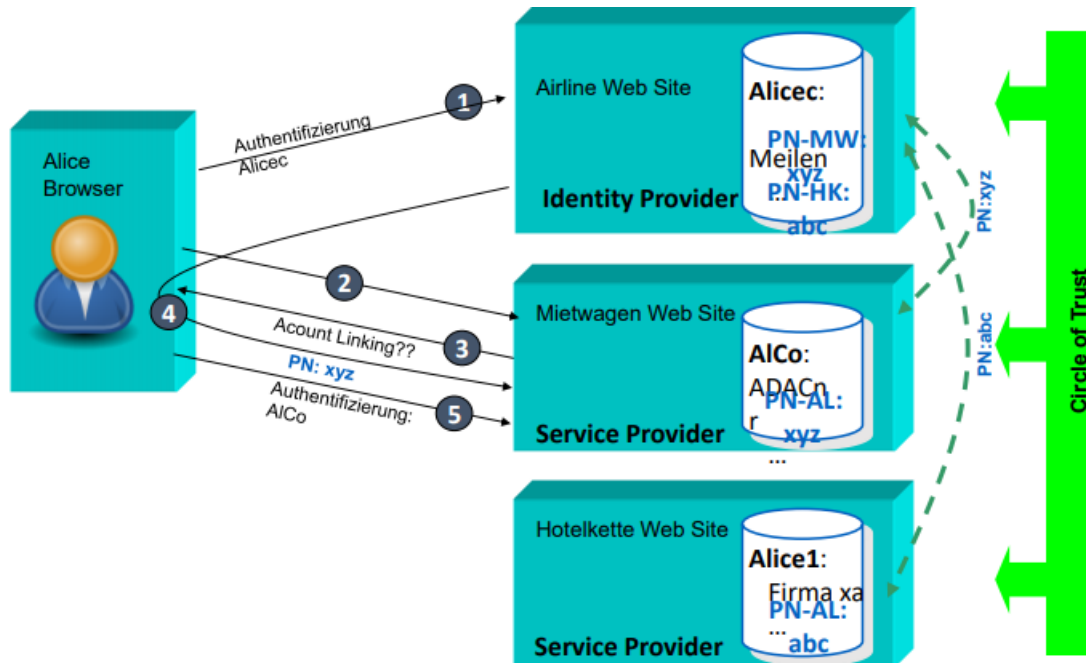


Figure 3.3: Account Linking

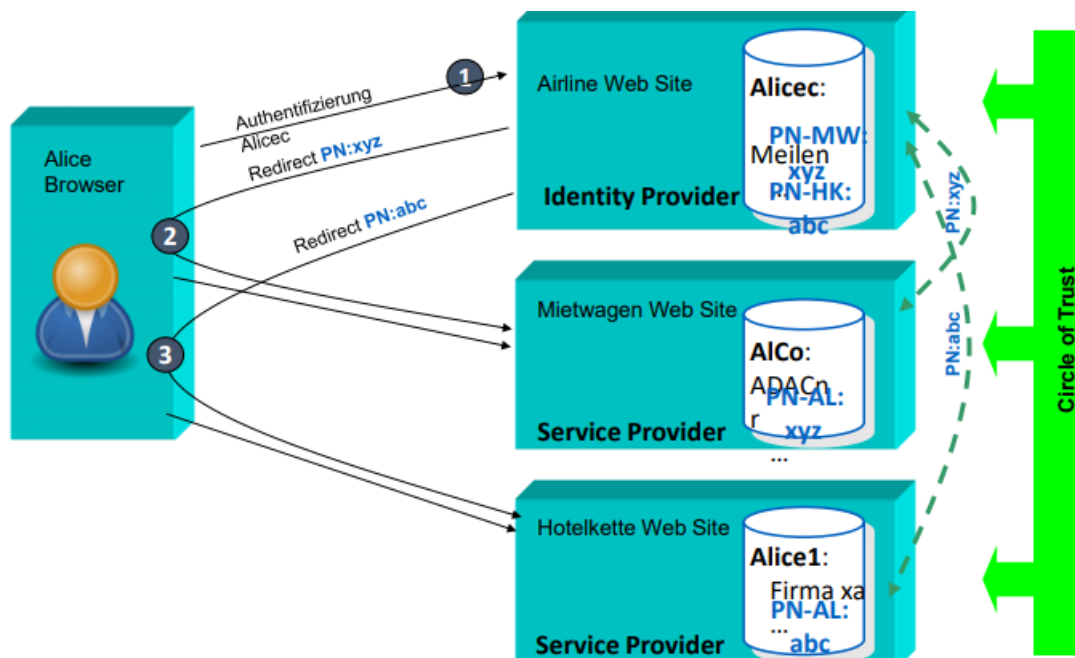


Figure 3.4: nach dem Linken

3.3 XML - Signaturen

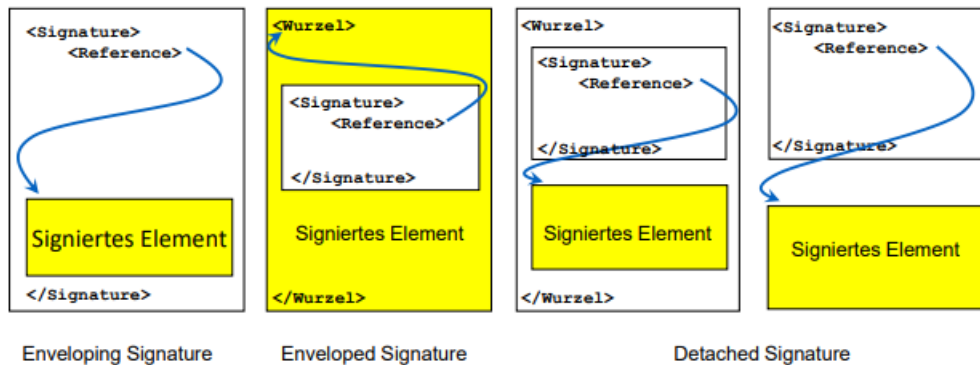


Figure 3.5

3.3.1 XML-Signatur - Enveloped Signature

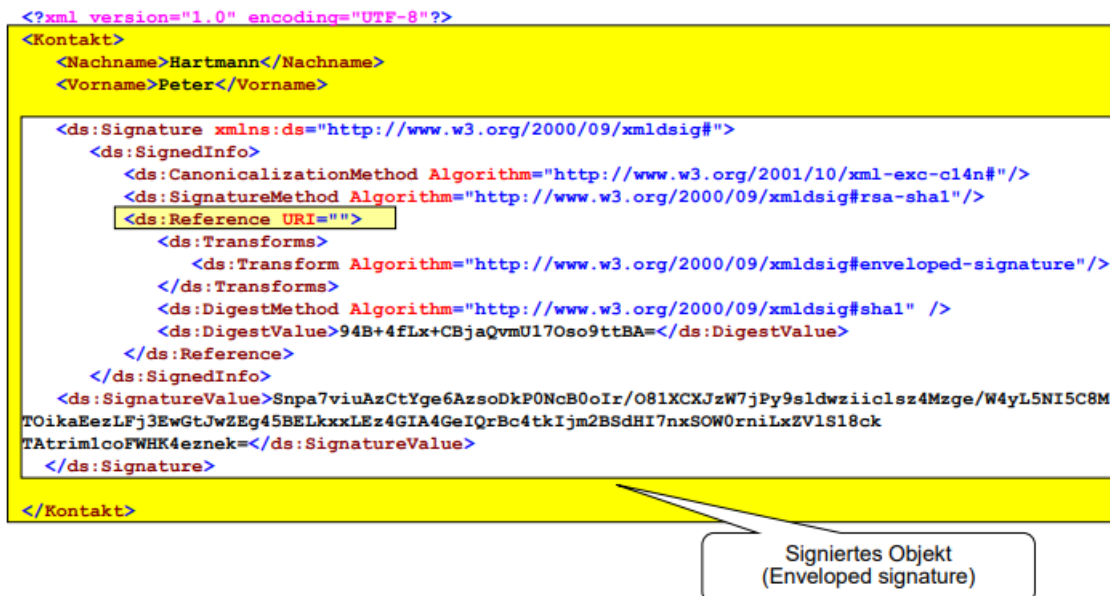


Figure 3.6

3.3 XML - Signaturen

3.3.2 XML-Signatur - Enveloping Signature


```
<?xml version="1.0" encoding="UTF-8"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <ds:Reference URI="#DerKontakt">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <ds:DigestValue>r0fBufNHbk8t/Xi3CwSi5hssVrQ=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>

  <ds:SignatureValue>gQvztpFujbhIVPU/Hh5QwOFFygGhMOZYoxfIx/1yQv3aC0dbUE+17k8IH96QRqBX+UccJ2
1FxpQRk3XuhRj+2Gfcj7qRamtgE9OQfSClNkd9xZp/wKvU6fdxA+LYmOQDHhfcGh3++9Zruh1ZtmYIwwfj
JWG/NsQ/B1w1HYnI0Js=</ds:SignatureValue>

  <ds:Object Id="DerKontakt">
    <Kontakt>
      <Nachname>Hartmann</Nachname>
      <Vorname>Peter</Vorname>
    </Kontakt>
  </ds:Object>

</ds:Signature>
```



The diagram illustrates the Enveloping Signature structure. A yellow box contains the XML code for the `<ds:Object Id="DerKontakt">` block, which includes a `<Kontakt>` element with `<Nachname>Hartmann</Nachname>` and `<Vorname>Peter</Vorname>`. A callout bubble points to this box with the text "Signiertes Objekt (Enveloping signature)".

Figure 3.7

3.3.3 XML-Signatur - Detached Signature

```
<?xml version="1.0" encoding="UTF-8"?>
<SignierterKontakt>

  <Kontakt Id="DerKontakt">
    <Nachname>Hartmann</Nachname>
    <Vorname>Peter</Vorname>
  </Kontakt>

  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
      <ds:Reference URI="#DerKontakt">
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <ds:DigestValue>r0fBufNHbk8t/Xi3CwSi5hssVrQ=</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>gQvztpFujbhIVPU/Hh5QwOFFygGhMOZYoxfIx/1yQv3aC0dbUE+17k8IH96QRJ21
k3XuhRj+2Gfcj7qRamtgE9OQfSClNkd9xZp/wKvU6fdxA+LYmOQDHhfcGh3++9Zruh1ZtmYIwwfj
JWG/NsQ/B1w1HamtgE9OQfSYnI0Js=</ds:SignatureValue>
  </ds:Signature>
</SignierterKontakt>
```



The diagram illustrates the Detached Signature structure. A yellow box contains the XML code for the `<Kontakt Id="DerKontakt">` block, which includes a `<Kontakt>` element with `<Nachname>Hartmann</Nachname>` and `<Vorname>Peter</Vorname>`. A callout bubble points to this box with the text "Signiertes Objekt (Detached signature)".

Figure 3.8

3.3.4 XML-Signatur - Detached Signature2

```
<?xml version="1.0" encoding="UTF-8"?>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc14n#" />
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <ds:Reference URI="https://if-portal.haw-landshut.de/if/logo.jpg">
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <ds:DigestValue>r0fBuFNHbk8t/Xi3CwSi5hssVrQ=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>gQvztpPujbhIVPU/Hh5QwOFFygGhMOZYoxfIx/1yQv3aC0dbUE+17k8IH96QRJ21
k3XuhRj+2Gfcj7qRamtgE9OQfSClNkd9xZp/wKvU6fdxA+LYmOQDhHfcGh3++9Zruh1ZtmYIwwfj
JWG/NsQ/B1w1HamtgE9OQfSYnI0Js=</ds:SignatureValue>
</ds:Signature>
```



Figure 3.9

3.4 XML-Encryption

3.4.1 Klartextnachricht

XML-Encryption: Klartextnachricht

```
<PaymentInfo xmlns="http://badbank.org/paymentv2">
  <Name>Joe Soap</Name>

  <CreditCard Limit="5,000" Currency="USD">
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Lehman Brothers</Issuer>
    <Expiration>04/10</Expiration>
  </CreditCard>

</PaymentInfo>
```

Figure 3.10

3.4 XML-Encryption

3.4.2 encrypted Element

```
<PaymentInfo xmlns="http://badbank.org/paymentv2">
  <Name>Joe Soap</Name>

  <EncryptedData xmlns="http://www.w3.org/2001/04/xmenc#"
    Type="http://www.w3.org/2001/04/xmenc#Element">
    <EncryptionMethod
      Algorithm="http://www.w3.org/2009/xmenc11#aes256-gcm/"
    <CipherData>
      <CipherValue>A23B4...5C56</CipherValue>
    </CipherData>
    </EncryptedData>
  </PaymentInfo>
```



Figure 3.11

3.4.3 encrypted content

```
<PaymentInfo xmlns="http://badbank.org/paymentv2">
  <Name>Joe Soap</Name>

  <CreditCard Limit="5,000" Currency="USD">
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmenc#"
      Type="http://www.w3.org/2001/04/xmenc#Content">
      <EncryptionMethod
        Algorithm="http://www.w3.org/2009/xmenc11#aes256-gcm/"
      <CipherData>
        <CipherValue>A23B...45C56</CipherValue>
      </CipherData>
      </EncryptedData>
    </CreditCard>
  </PaymentInfo>
```



Figure 3.12

3.4.4 KeyInfo in EncryptedData

```
<PaymentInfo xmlns="http://badbank.org/paymentv2">
  <EncryptedData Id="ED" xmlns="http://www.w3.org/2001/04/xmenc#">
    <EncryptionMethod
      Algorithm=http://www.w3.org/2001/04/xmenc#aes128-cbc/>

    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      <ds:KeyName>session_key_24fe43cd</ds:KeyName>
      <ds:RetrievalMethod URI="#HierIstDerKey"
        Type="http://www.w3.org/2001/04/xmenc#EncryptedKey" />
    </ds:KeyInfo>

    <CipherData>
      <CipherValue>DEADBEEF</CipherValue>
    </CipherData>
  </EncryptedData>
```

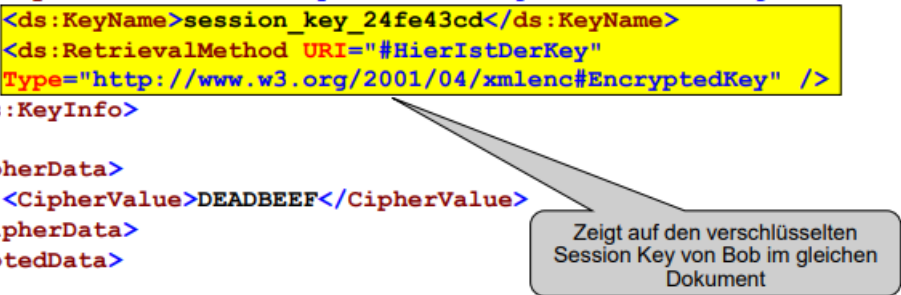


Figure 3.13

3.5 Verschlüsseln und Signieren

Encrypt-Sign

- Trudy kann abhören und die Signatur austauschen
- **Abhilfe:** Vor dem Verschlüsseln in den Klartext den Absendernamen schreiben.

Sign-Encrypt

- Der vorgesehene Empfänger kann die Nachricht um-verschlüsseln und weiterleiten
- der falsche Empfänger denkt die Nachricht ist für ihn
- **Abhilfe:** Alice kann vor dem Signieren den Klartext in den Empfängernamen schreiben

3.6 SAML-assertions

4 Angriffe und Schwachstellen

Fokus in diesem Kapitel liegt auf der Obersten Schicht von Webdiensten, der Applikation selbst. **Unterliegende Komponenten dürfen trotzdem nie vernachlässigt werden!**

OWASP Open Web Application Security Project: Gemeinnützige Organisation mit dem Ziel sicherer Webanwendungen.

4.1 Injection-Schwachstellen

- entstehen, wenn Eingaben eines Clients **ungeprüft an einen Interpreter** weitergeleitet werden
- einfach auszunutzen, weit verbreitet, verursachen hohen Schaden
- **Vermeidung:**
 - Sichere APIs
 - Escaping, um Metazeichen der jeweiligen Syntax zu 'entschärfen'
 - White-Lists bei Eingaben