

IT-Sicherheit



Kapitel 2: Verschlüsselung (Teil 2)

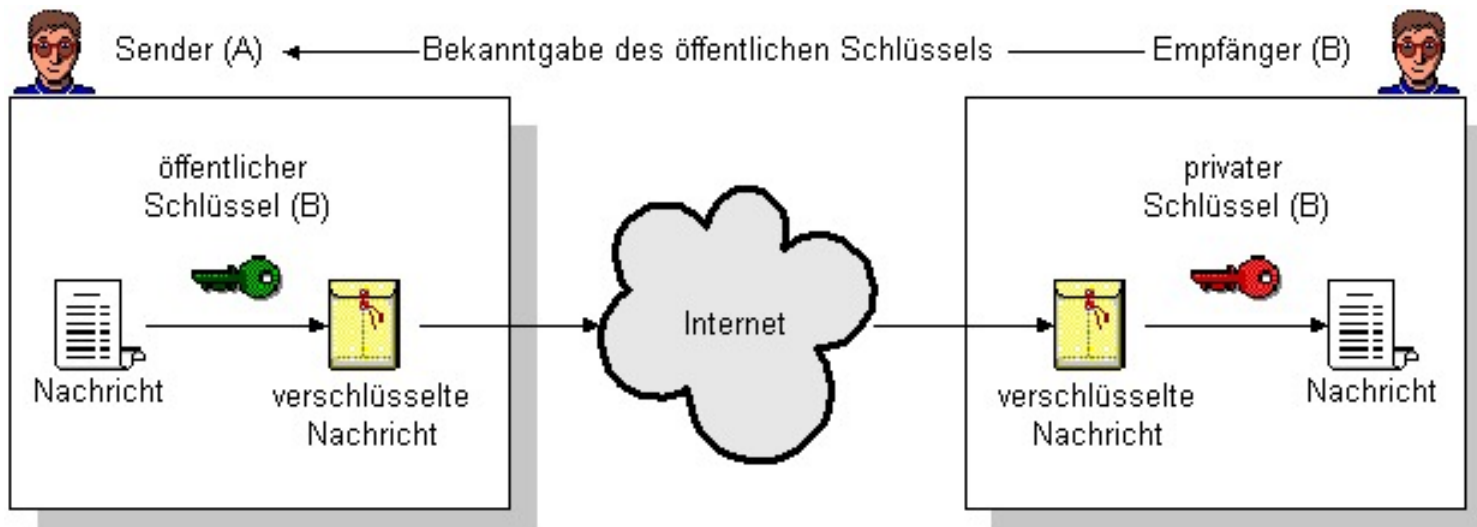
- ▶ Asymmetrische Verschlüsselung
- ▶ Praktische Aspekte bei der Verschlüsselung
- ▶ Zufallszahlen
- ▶ Schlüsselmanagement





Asymmetrische Verschlüsselung

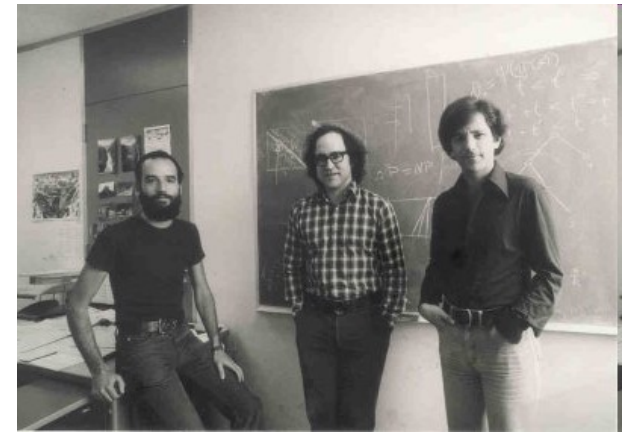
- ▶ Basiert auf Verwendung von Schlüsselpaaren (privater und öffentlicher Schlüssel)
- ▶ Es gibt weniger Verfahren, da sie schwieriger zu konstruieren sind (basieren auf Problemen der algorithmischen Zahlentheorie)
- ▶ Nachteil: Hoher Rechenaufwand
- ▶ Vorteil: Schlüsselaustausch kein Problem
- ▶ Beispiele: RSA, ECIES (Elliptic Curve), DLIES (Discrete Logarithm)





RSA (Rivest-Shamir-Adleman)

- ▶ Ist ein asymmetrischer Algorithmus
- ▶ Er ist einfach zu verstehen und implementieren, populär, lizenzfrei
- ▶ Um den geheimen Schlüssel aus dem öffentlichen zu finden müssen die Primfaktoren von n gefunden werden.
- ▶ Die Faktorisierung großer Zahlen ist sehr schwierig.
- ▶ Es sind aktuell Schlüssellängen von 2000 oder 3000 erforderlich
- ▶ Die Implementierungen sind viel langsamer als AES





RSA Algorithmus

1. wähle 2 große ($> 2^{512}$) Primzahlen p und q ,
2. Berechne RSA-Modul $n=p*q$
3. wähle $e < n$ mit $\text{ggT}(e, (p-1)(q-1))=1$
4. wähle d , so dass $ed \bmod (p-1)(q-1) = 1$
5. public key ist (e, n) , private key ist (d, n)

Verschlüsseln: $c = m^e \bmod n$

Entschlüsseln: $m = c^d \bmod n$

▶ Elliptic Curve Cryptography

▶ Bsp. Für eine Kurvengleichung:
 $y^2 = x^3 - 4x^2 + 10$

▶ Vorteile von ECC:

- ▶ kürzere Schlüssellängen wie RSA
- ▶ Dadurch schneller
- ▶ Sicher auch bei Quanten Computer

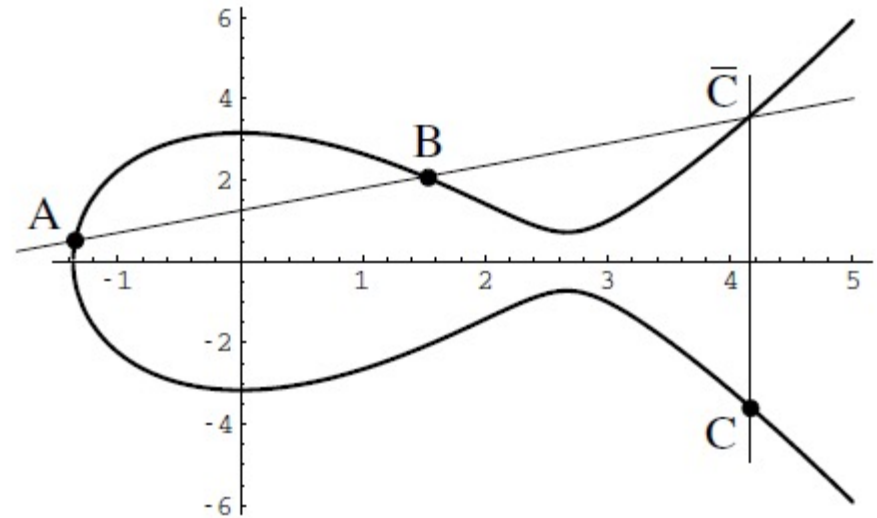
▶ Vorgehen: mehrere Durchgänge mit Geraden durch zwei Punkte und Spiegelung des dritten Schnittpunkts

▶ Public Key: Anfang und Endpunkt, Private Key: Anzahl der Durchgänge

▶ Video zur Veranschaulichung



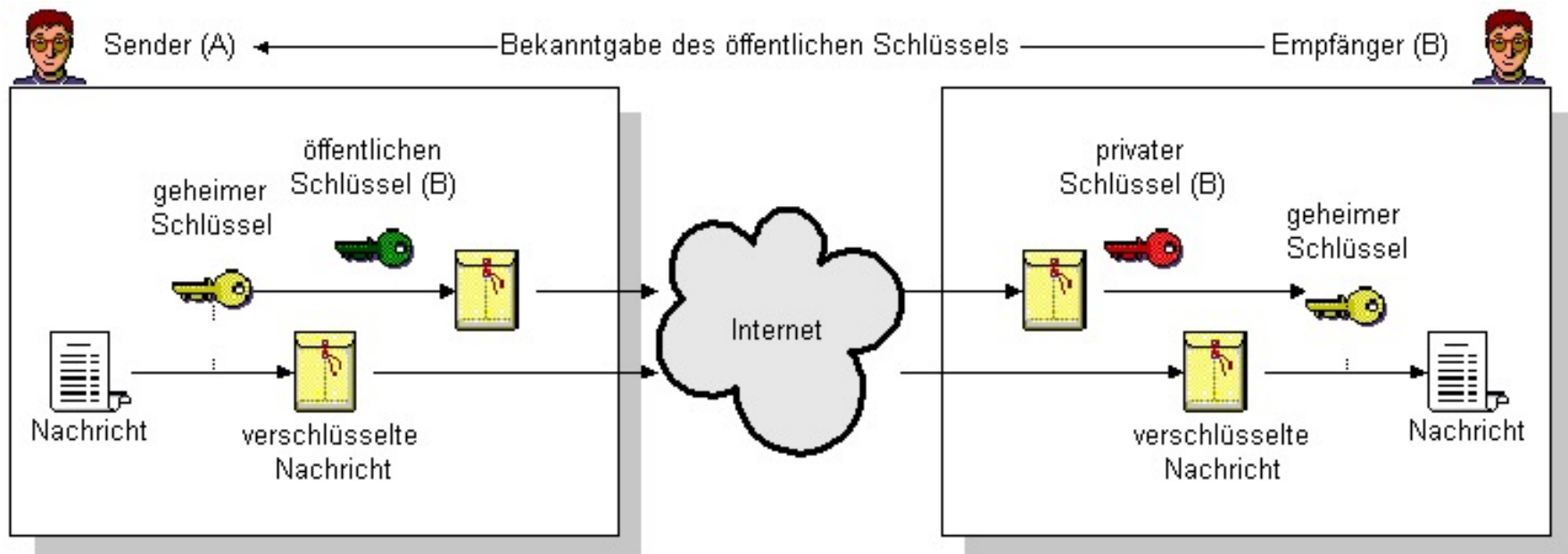
<https://www.youtube.com/watch?v=dCvB-mhkT0w>



Quelle: Angewandte Kryptographie, W.Ertl, Hanser

Hybride Verschlüsselung

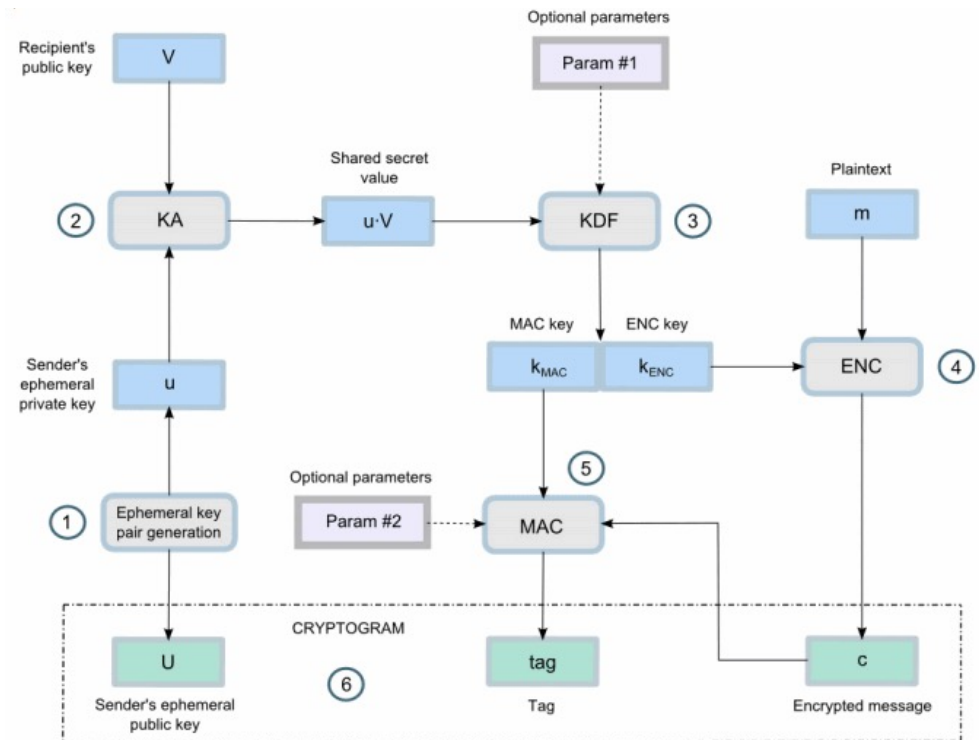
- ▶ Kopplung von symmetrischer und asymmetrischer Verschlüsselung
- ▶ Die Nachricht wird zuerst symmetrisch verschlüsselt (Sitzungsschlüssel) und dann der Sitzungsschlüssel asymmetrisch verschlüsselt
- ▶ Vorteil: Hohe Geschwindigkeit bei Verschlüsselung und sicherer Schlüsseltausch





ECIES Elliptic Curve Integrated Encryption Scheme

- ▶ Hybrides Verschlüsselungsverfahren das ECC asymmetrische Kryptographie mit symmetrischen Cipher und MAC kombiniert
- ▶ ECIES ist nicht ein konkreter Algorithmus sondern ein Framework
- ▶ Man kann verschiedene Algorithmen verwenden (AES-CTR, AES-GCM, ChaCha20-Poly1305, ...)



KA Key Agreement (e.g. ECDH)
KDF Key Derivation Function (e.g. cryptographic hash)
ENC Encryption

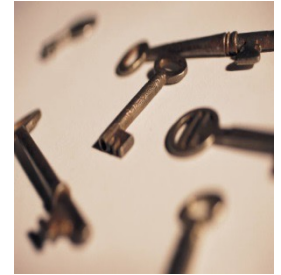
<https://cryptobook.nakov.com/asymmetric-key-ciphers/ecies-public-key-encryption>

- ▶ Weitere Details stehen unter

<https://medium.com/asecuritysite-when-bob-met-alice/go-public-and-symmetric-key-the-best-of-both-worlds-ecies-180f71eebf59>

https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile&v=12

▶ **Praktische Aspekte bei Verschlüsselung**



- ▶ Wie baue ich Kryptographie in meine Anwendungen ein?
- ▶ In welchem Format speichere ich verschlüsselte Daten ab?
- ▶ Welche Angriffe gibt es auf Verschlüsselung?
- ▶ Welche Schlüssellängen und Verfahren sind aktuell erforderlich?
- ▶ Wie erzeuge ich geeignete Schlüssel?
- ▶ Wie bekomme ich kryptographisch sichere Zufallszahlen?
- ▶ Was muss ich beim Schlüsselmanagement beachten?
- ▶ Wie tausche ich Schlüssel sicher aus?



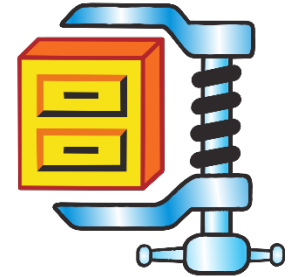
Wie baue ich Kryptographie in meine Anwendungen ein?

- ▶ Niemals „**Security by obscurity**“ machen:
 - ▶ Security by obscurity: System ist eine Art Black Box, Teile vom Verfahren oder Algorithmus müssen geheim gehalten werden.
- ▶ Sondern **Open Design** anwenden:
 - ▶ “The security of a mechanism should not depend on the secrecy of its design or implementation.”
OWASP Secure Design Principles <https://github.com/OWASP/DevGuide/blob/master/02-Design/01-Principles%20of%20Security%20Engineering.md>
- ▶ **Kerkhoffs-Prinzip**:
 - ▶ Die Sicherheit eines kryptographischen Verfahren darf alleine von dem verwendeten Schlüssel abhängen
- ▶ Keine eigene Kryptographie verwenden !!!
 - ▶ Sondern Einsatz von Crypto-Bibliotheken



Was muss ich noch beachten?

- ▶ Kompression – Verschlüsselung – Fehlererkennung
 - ▶ Die richtige Reihenfolge ist wichtig
 - ▶ **Kompression** ist sinnvoll weil
 - ▶ Verschlüsselung wird schneller, falls Kompression schnell ist
 - ▶ Die Kryptoanalyse (beruht auf Ausnutzung von Redundanz im Klartext) wird erschwert
 - ▶ Eine zusätzliche Signatur oder **Prüfsumme** auf verschlüsselte Daten ist sinnvoll dadurch erkennt man Manipulationen am Ciphertext
- ▶ Bei Einsatz von Kryptographie in Softwareprodukten sind **Exportbeschränkungen** und **Patente** zu beachten
 - ▶ Kryptographie fällt unter das Waffengesetz
 - ▶ z.B. Bureau of Industrie and Security USA (<https://www.bis.doc.gov/>) reguliert Ausfuhr und Einfuhr von Technologien die Kryptographie verwenden



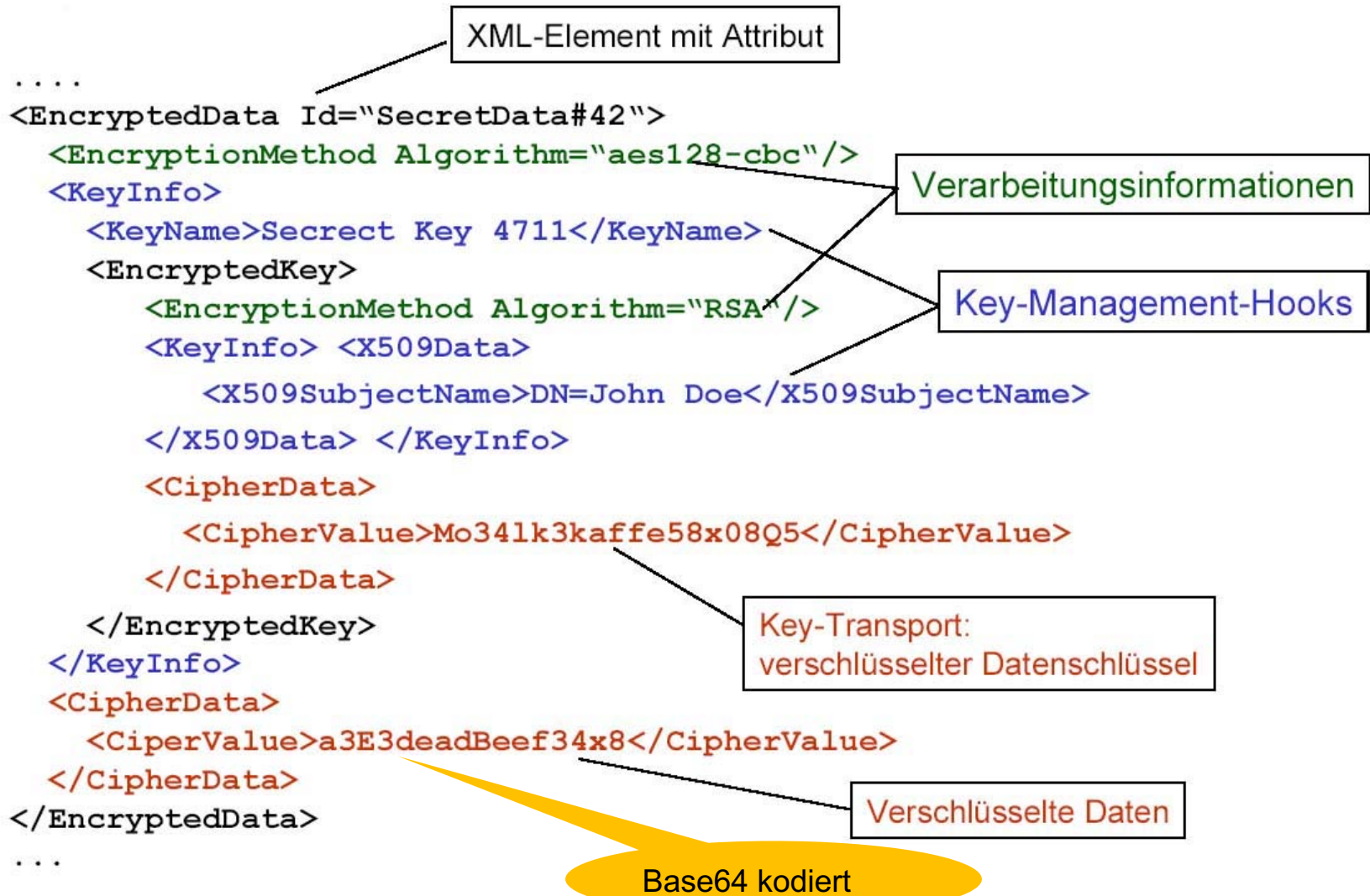


Ein Format zum Speichern von Verschlüsselung: XML Encryption Standard

- ▶ W3C Recommendation (2013) <https://www.w3.org/TR/xmlenc-core/>
 - ▶ Beschreibt die XML-Syntax wie verschlüsselte Dokumente in XML abgespeichert werden.
- ▶ Es gibt folgende Verschlüsselungsmöglichkeiten
 - ▶ Verschlüsselung des gesamten XML-Dokuments
 - ▶ Verschlüsselung eines einzelnen Elements und seiner Unterelemente
 - ▶ Verschlüsselung des Inhalts eines XML-Elements
 - ▶ Verschlüsselung für mehrere Empfänger
- ▶ Es gibt folgende Elemente zum Aufbau der XML Struktur
 - ▶ **EncryptedData**: der umhüllende Tag für die Verschlüsselung
 - ▶ **EncryptionMethod**: Verschlüsselungsmethode
 - ▶ **CipherData**: stellt verschlüsselte Daten direkt oder indirekt per Referenz zur Verfügung (**CipherValue**, **CipherReference**)
- ▶ Vorteil von XML-Encryption: wird von vielen Bibliotheken unterstützt



Beispiel für XML-Encryption





Ein Format zum Speichern von Verschlüsselung: PKCS#7 Enveloped Data

- ▶ Wird z.B. verwendet bei SMIME
- ▶ Kann mit Signaturen kombiniert werden: *Signed and Enveloped Data*
- ▶ Syntax von Enveloped Data

Weitere Details lernen wir
im Kapitel Digitale
Signaturen kennen

```
EnvelopedData ::= SEQUENCE {  
    version                Version,  
    recipientInfos          RecipientInfos,  
    encryptedContentInfo    EncryptedContentInfo  
}  
RecipientInfos ::= SET OF RecipientInfo  
EncryptedContentInfo ::= SEQUENCE {  
    contentType            ContentType,  
    contentEncryptionAlgorithm ContentEncryptionAlgorithmIdentifier,  
    encryptedContent [0] IMPLICIT EncryptedContent OPTIONAL  
}  
EncryptedContent ::= OCTET STRING
```

▶ Angriffe auf Verschlüsselung, Kryptoanalyse



- ▶ Kryptoanalyse: Mathematisch analytisch
 - ▶ Known Ciphertext, Known Plaintext, Chosen Plaintext, Chosen Ciphertext
- ▶ Brute Force
- ▶ Social Engineering
- ▶ Schwachstellen im Gesamtsystem suchen, um an Schlüssel zu kommen
- ▶ Seitenkanal-Angriff
 - ▶ Angriff auf ein kryptographisches System, der die Ergebnisse von physikalischen Messungen am System (zum Beispiel Energieverbrauch, elektromagnetische Abstrahlung, Zeitverbrauch einer Operation) ausnutzt, um Einblick in sensible Daten zu erhalten.
- ▶ Fault Attack
 - ▶ Angriff auf ein kryptographisches System, in dem der Angreifer eine fehlerhafte Ausführung einer kryptographischen Operation nutzt beziehungsweise aktiv hervorruft.
- ▶ Man-in-the-Middle-Angriff



Welche Schlüssellängen und Verfahren sind aktuell erforderlich?

▶ Schlüssellänge – Geltungsdauer

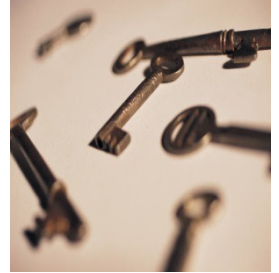
▶ Symmetrische Verfahren:

- ▶ Angriffe über Brute-Force
- ▶ Länge mindestens 256

▶ Asymmetrische Verfahren:

- ▶ Angriffe über mathematische Verfahren
- ▶ Länge mindestens 2000 (RSA), 200 (ECC)

- ▶ Kryptosysteme müssen so gebaut werden, das Wechsel der Verfahren und Schlüssellängen möglich ist



▶ Diese Informationen müssen regelmäßig überprüft werden

▶ Eine wichtige Quelle ist das BSI

Quelle für aktuell empfohlene Verfahren und Schlüssellängen:

https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile&v=12



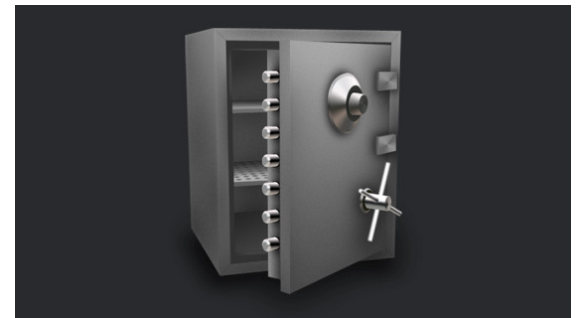
Schlüsselmanagement



- ▶ Kryptographische Verfahren sind nur so sicher wie die Schlüssel geschützt werden
- ▶ Ein sauber implementiertes Schlüsselmanagement ist dafür erforderlich
- ▶ Wichtige Funktionen des Schlüsselmanagements
 - ▶ Erzeugung von Schlüssel
 - ▶ Speicherung der Schlüssel
 - ▶ Sichere Übertragung – Austausch von Schlüssel
 - ▶ Zurückziehen ungültiger oder kompromittierter Schlüssel
 - ▶ Sicherungskopien von Schlüssel
 - ▶ Vernichtung von Schlüssel



Schlüsselmanagement: Speicherung



- ▶ Wo speichere ich Schlüssel, Credentials, Tokens?
 - ▶ Im Source Code -> **niemals!!**
 - ▶ In Konfigurationsdateien, Umgebungsvariablen: entweder strenge Zugriffskontrolle oder verschlüsseln
 - ▶ Passwort geschützt (**PBE**: Password Based Encryption)
 - ▶ Bsp. Java-Keystore
 - ▶ Externe Keystores
 - ▶ **Vaults/Secret Manager** als Service oder in einem **HSM** (Hardware Security Module), Zugriff über API mit Authentifizierung und Autorisierung und mit audit logs
Bsp: Microsoft Azure Key Vault, AWS Secrets Manager
 - ▶ **Sealed Secrets** in git auf Basis asymmetrischer Crypto
<https://learnk8s.io/kubernetes-secrets-in-git>
- ▶ Chipkarte und PIN
- ▶ Schlüsselaufteilung (Secret Splitting, Secret Sharing: n:m-Prinzip)

Schlüssel auf **n** Instanzen (Mitwisser) aufteilen, **m** Instanzen sind erforderlich um Schlüssel zu rekonstruieren



Schlüsselmanagement: Erzeugen, Übertragen, Sperren

- ▶ Erzeugung von Schlüssel
 - ▶ Es sind gute Zufallszahlen erforderlich
 - ▶ Die sollten an einem sicheren Ort erzeugt werden (z.B. Chipkarte, HSM)
- ▶ Sichere Übertragung – Austausch von Schlüssel
 - ▶ Meist auf Basis asymmetrischer Kryptographie (DH, ECDH, RSA und PKI)
- ▶ Bei Kompromittierung eines Schlüssel muss er ausgetauscht werden
 - ▶ Schlüssel als gesperrt markiert (z.B. Öffentlich Sperren)
 - ▶ Schlüssel austauschen
 - ▶ Umverschlüsselung aller betroffenen Daten
 - ▶ Die Aktionen **Sperren-Austauschen-Umverschlüsseln** sind ebenfalls notwendig wenn ein Schlüssel nicht mehr sicher ist (z.B. Verfahren unsicher, Länge des Schlüssels zu kurz)

▶ Schlüsselmanagement: Sichern, Vernichten



- ▶ Sicherungskopien von Schlüssel sind meistens notwendig
 - ▶ Für Ausnahmesituationen (Todesfall, Kündigung, Urlaub, ...) und Verlust
 - ▶ Key **Recovery**: Hinterlegung für Schlüsselerlust
 - ▶ Key **Escrow**: Hinterlegung für Strafverfolgung
 - ▶ Secret Sharing (z.B. Shamir Secret Sharing)
 - ▶ Master Key
 - ▶ ACHTUNG: keine Sicherungskopie von privaten Signaturschlüssel machen
- ▶ Vernichtung von Schlüssel
 - ▶ Dateien und Festplatte → Speicherbereiche überschreiben, Cache und Swap-Bereiche löschen
 - ▶ Hardware muss eventuell zerstört werden

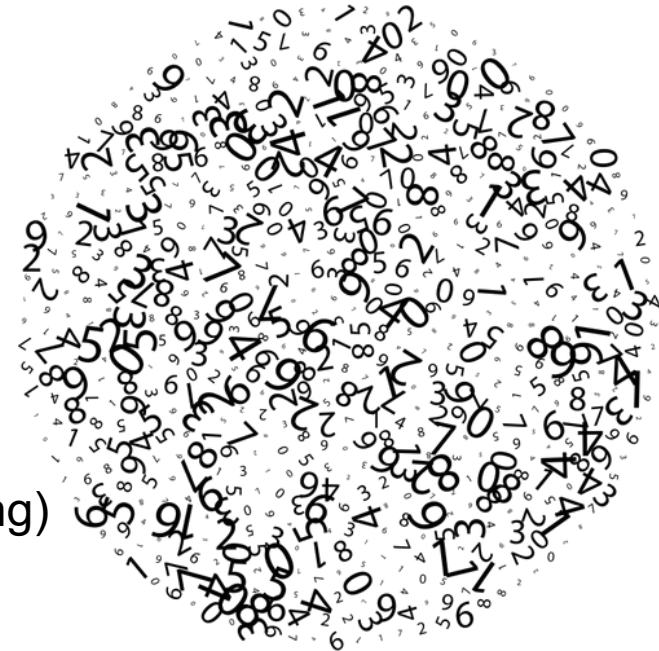
Sonst können
Signaturen
gefälscht werden

▶ Randomness: Wie zufällig ist der Zufall?

- ▶ Ausgangssituation
 - ▶ Die Qualität der Verschlüsselung hängt von Qualität der Schlüssel ab
 - ▶ Die Entropie ist ein Maß für den Informationsgehalt einer Nachricht (maximale Entropie bei Gleichverteilung)
 - ▶ Viele weitere Algorithmen brauchen Zufallszahlen (z.B. salt bei Hash)

- ▶ Problem:
 - ▶ Computer sind deterministische Maschinen, Zufall ist schwer zu bekommen

- ▶ Lösung:
 - ▶ Möglichst viele Entropiequellen einsammeln, die man schwer vorhersagen kann (Interrupts, Userinput, Rauschen der Soundkarte, etc.)



73735	45963	78134	63873
02965	58303	90708	20025
98859	23851	27965	62394
33666	62570	64775	78428
81666	26440	20422	05720

15838	47174	76866	14330
89793	34378	08730	56522
78155	22466	81978	57323
16381	66207	11698	99314
75002	80827	53867	37797

99982	27601	62686	44711
84543	87442	50033	14021
77757	54043	46176	42391
80871	32792	87989	72248
30500	28220	12444	71840



Kryptographische Zufallszahlen



- ▶ Eigenschaften eines guten Zufallszahlengenerators
 - ▶ Er erzeugt gleichmäßig verteilte Zahlen
 - ▶ Die Zahlen lassen sich nicht vorhersagen
 - ▶ Er hat einen langen und vollständigen Zyklus
- ▶ Problem: SW-implementierte PRNG's (Pseudo Random Number Generator) sind meist vorhersehbar
- ▶ Lösungen:
 - ▶ Echte ZFZG basierend auf HW (z.B. elektronisches Rauschen) sind teuer und aufwendig
 - ▶ PRNG mischen verschiedene Zufallsquellen (Zeit, interne CPU Zähler, Tastatureingaben, Mausbewegung etc.) als Startwert und verwenden eine Fortschaltfunktion zur Berechnung der Zufallszahl
 - ▶ Kryptographische Bibliotheken bieten PRNG an:
 - .Net: `System.Security.Cryptography.RNGCryptoServiceProvider`,
 - JCE: `java.security.SecureRandom`



Populäre Fehler

- ▶ /dev/random verwenden (GNU/Linux)
 - ▶ Blockt, wenn Entropie “leer” ist
 - ▶ Stattdessen: /dev/urandom oder getrandom(2) syscall
<https://www.2uo.de/myths-about-urandom>
- ▶ Mathematische Random Funktion von Bibliotheken verwenden
 - ▶ Schnell, aber unsicher (vorhersagbar)
 - ▶ Stattdessen bei Crypto: **IMMER** SecureRandom hernehmen!
- ▶ SecureRandom.getInstanceStrong() verwenden
<https://tersesystems.com/blog/2015/12/17/the-right-way-to-use-securerandom>
 - ▶ Liest von /dev/random
 - ▶ new SecureRandom() langt vollkommen



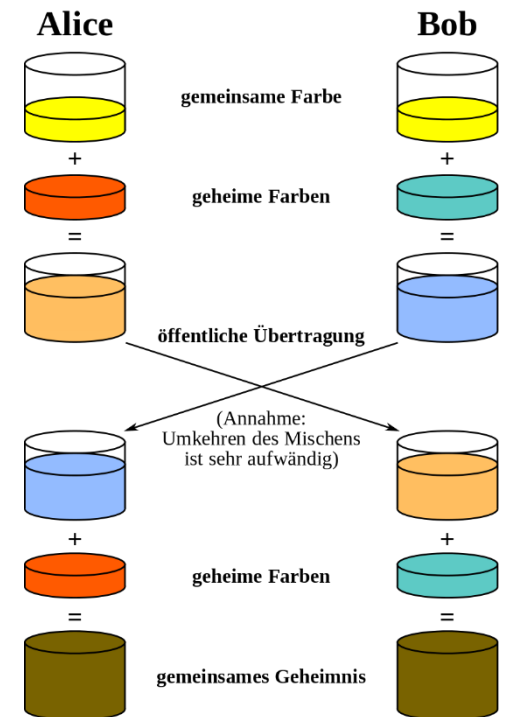
Beispiel

```
private void random() {  
    SecureRandom random = new SecureRandom();  
  
    byte[] data = new byte[256 / 8];  
    random.nextBytes(data);  
    System.out.println(bytesToHex(data));  
  
    // Do not create a new SecureRandom, reuse it  
    byte[] moreData = new byte[256 / 8];  
    random.nextBytes(moreData);  
    System.out.println(bytesToHex(moreData));  
}
```

▶ Diffie-Hellman Verfahren

- ▶ DH ist ein Public Key Algorithmus für die Verteilung von Schlüsseln
- ▶ DH ist nicht zum Verschlüsseln geeignet
- ▶ Jeder Kommunikationspartner berechnet sich den gemeinsamen, geheimen Sitzungsschlüssel
- ▶ Dadurch muss der Schlüssel nicht über das Netzwerk transportiert werden
- ▶ Basiert auf dem mathematischen Problem des diskreten Logarithmus
- ▶ Empfohlene Schlüssellänge 2000 – 3000 Bit

Grundidee



<https://de.wikipedia.org/wiki/Diffie-Hellman-Schl%C3%BCsselaustausch>



Diffie-Hellman Verfahren

▶ Protokollablauf

- ▶ Alice und Bob einigen sich auf eine große Primzahl n und eine Zahl g
 - ▶ Alice wählt eine große Zufallszahl x und sendet $X = g^x \bmod n$ an Bob
 - ▶ Bob wählt eine große Zufallszahl y und sendet $Y = g^y \bmod n$ an Alice
 - ▶ Alice berechnet $k = Y^x \bmod n$
 - ▶ Bob berechnet $k' = X^y \bmod n$
 - ▶ Es gilt $k' = k$
-
- ▶ Public key: n, g, X, Y
 - ▶ Private key: x, y, k
 - ▶ Schlüssellänge: Länge von n in Bit



Elliptic Curve Diffie-Hellman ECDH

- ▶ Die Sicherheit beruht auf der Schwierigkeit des Diffie-Hellman Problems in elliptischen Kurven
 - ▶ Schlüssellängen viel kürzer als bei klassischen DH

p Primzahl
a,b Kurvenparameter
P Basispunkt auf Kurve
q Ordnung von P
i Kofaktor

▶ Protokollablauf

- ▶ Wähle kryptographisch starke EC-Systemparameter (p, a, b, P, q, i) , Schlüssellänge q sollte mindesten 250 Bit sein
- ▶ Tausche Systemparameter und alle folgenden Schritte authentisch aus
- ▶ A wählt gleichverteilt einen Zufallswert $x \in \{1, \dots, q - 1\}$ und sendet $Q_A = x \cdot P$ an B.
- ▶ B wählt gleichverteilt einen Zufallswert $y \in \{1, \dots, q - 1\}$ und sendet $Q_B = y \cdot P$ an A.
- ▶ A berechnet $x \cdot Q_B = xy \cdot P$
- ▶ B berechnet $y \cdot Q_A = xy \cdot P$

▶ Video zur Veranschaulichung

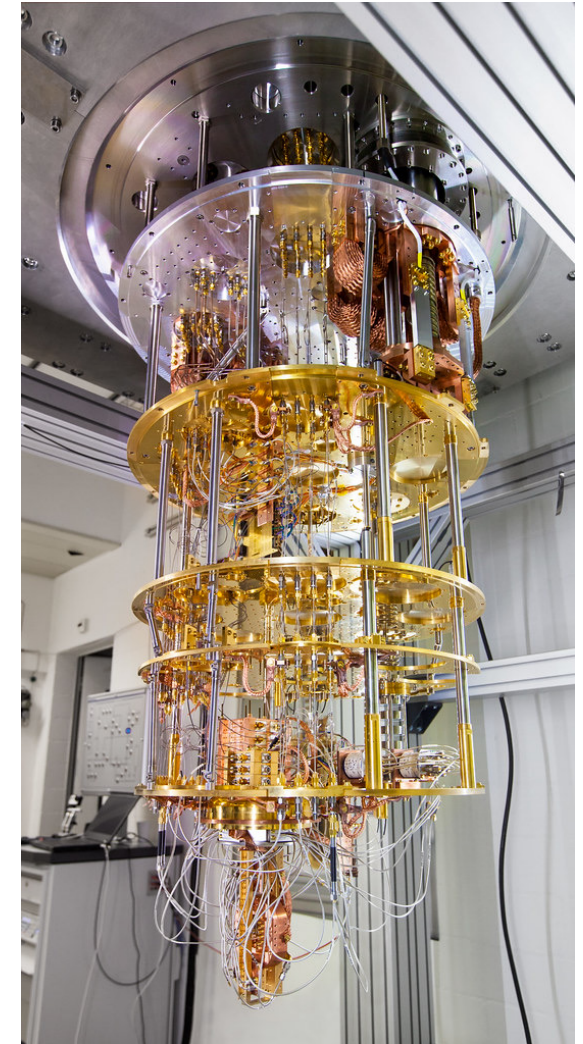


<https://www.youtube.com/watch?v=yDXiDOJgxmg>



Quantencomputer gefährden die Kryptographie



- ▶ Quantencomputer gefährden asymmetrische Verfahren
- ▶ Symmetrische Verfahren sind nicht so stark betroffen (Schlüssellänge erhöhen)
- ▶ Quantencomputer sind bisher nur Theorie
 - ▶ Aber sie werden auf jeden Fall kommen
 - ▶ Geheimdienste/Organisationen können heute Daten sammeln und in Zukunft entschlüsseln
 - ▶ Der Austausch der heutigen Kryptographie und Infrastruktur ist zeitintensiv



https://en.wikipedia.org/wiki/File:Quantum_Computer_Zurich.jpg



Quantensichere Kryptographie

- ▶ Deswegen gibt es heute schon Ansätze für quantensichere Kryptographie
 - ▶ **PQC Post Quantum Cryptography**: mathematische Probleme die durch QC nicht lösbar sind (z.B. FrodoKEM, Classic McEliece)
 - ▶ **QKD Quantum Key Distribution**: sichere Schlüsselverteilung (z.B. NewHope key exchange)
- ▶ Wettbewerb der NIST
 - ▶ Das NIST sucht quantum-resistente public-key Algorithmen
<https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions>
- ▶ Videos zu Quanten Kryptographie
 - ▶ Was bedeutet Post Quantum Cryptography
 <https://www.youtube.com/watch?v=zw1KHLOOIA8>
 - ▶ Mathematische Erklärung wie Quanten Computer aktuelle Algorithmen knacken können
 <https://www.youtube.com/watch?v=lvTqbM5Dq4Q&t=25s>



Fazit

- ▶ Learning: Crypto is hard
 - ▶ Gilt nicht nur für die Algorithmen, auch für deren Verwendung
 - ▶ Gilt auch für das Kapitel zu Hash, MAC, Signaturen
- ▶ Nehmen sie die richtige Library
 - ▶ Tink von Google <https://github.com/google/tink>
 - ▶ NaCl (<https://nacl.cr.yp.to/>) bzw. Sodium (<https://libsodium.gitbook.io/doc/>)
 - ▶ Lazysodium für Java
<https://github.com/terl/lazysodium-java>
- ▶ Literatur
 - ▶ <https://cryptobook.nakov.com/>
 - ▶ <https://www.garykessler.net/library/crypto.html>



Zusammenfassung Verschlüsselung



- ▶ Symmetrische Verschlüsselung ist schnell.
- ▶ Hybride Verschlüsselung ermöglicht schnelle Verschlüsselung mit sicheren Schlüsselaustausch.
- ▶ Der beste Verschlüsselungsalgorithmus nützt nichts bei einem schlechten Schlüssel-Management.
- ▶ Es ist wichtig, dass man die bekannten Verfahren richtig einsetzt und auch auf die Details achtet (Zufallszahlen, Betriebsmodi, Initialisierungsvektor, Prüfsummen, ...)