

Kryptografie mit geheimen Schlüsseln

Kryptologie IL (Vorlesungsteil)

Andreas Unterweger

Studiengang ITS
FH Salzburg

Sommersemester 2026

- Traditionelle Benennung von Akteuren
 - Alice
 - Bob
 - Eve
- Begriffe
 - Klartext (weiter: Chosen-Plaintext-Angriff)
 - Chiffretext (weiter: Chosen-Ciphertext-Angriff)
 - Schlüssel
 - Verschlüsselung(-sfunktion)
 - Entschlüsselung(-sfunktion)
- Arten von Chiffren
 - Symmetrische vs. asymmetrische Kryptografie
 - Strom- vs. Blockchiffren

Überblick über Kryptografie mit geheimen Schlüsseln

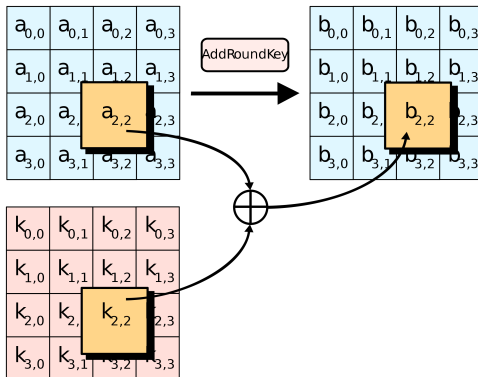
- Symmetrische Kryptografie: Schlüssel wird geheim gehalten
- Perfekte Sicherheit (Vernam) benötigt Schlüssel so lang wie die Nachricht
- Unpraktikabel (kürzere Schlüssel erwünscht)
- Praktische Lösung: Ansätze, die nicht perfekt sicher sind, sondern nur rechnerisch sicher (schwächere Sicherheitsgarantie)
- Beispiel für rechnerische Sicherheit: Chiffre kann mit dem schnellsten verfügbaren Computer in 200 Jahren mit einer Wahrscheinlichkeit von $< 10^{-30}$ gebrochen werden
- Rechnerisch sichere Verfahren basieren auf (Schwierigkeits-)Annahmen
- Schwerpunktthemen:
 - Beispiel einer symmetrischen Chiffre: Advanced Encryption Standard
 - Schlüsseltausch
 - Kryptografische Hashes (für Integrität)

- Der Advanced Encryption Standard (AES)
 - Standardisiert als Gewinner eines Wettbewerbs (2001)
 - Weit verbreitet und unterstützt
 - Blockchiffre (128 Bit) mit einem Substitutions-Permutations-Netzwerk
 - Unterstützt 128-, 192- oder 256-Bit-Schlüssel
 - Bekannte Angriffe auf vereinfachte Variationen von AES, aber kein besserer Weg als erschöpfende/Brute-Force-Schlüsselsuche bekannt, um volles AES zu brechen
- Zur Erinnerung: $2^{128} \approx 10^{38}$ mögliche Schlüssel sind **viel**
 - Optimistische 10^{10} Entschlüsselungen/s auf einem Desktop-Computer:
 $10^{28} \text{ s} \approx 10^{20} \text{ a!}$
 - Optimistische 10^{20} Entschlüsselungen/s auf einem Computer-Cluster:
 $10^{18} \text{ s} \approx 10^{10} \text{ a!}$

- Überblick über Verschlüsselung aus der Vogelperspektive:
 - $4 \cdot 4$ -Byte-Status-Array, initial gleich der(/dem) Eingabe(-block)
 - Mehrere Runden, die das Status-Array verändern
 - Anzahl der Runden hängt von der Schlüssellänge ab (z.B. 14 bei 256 Bit)
 - Jede Runde: 4 Phasen (Substitutions-Permutations-Netzwerk)
 - Schlüssel beeinflusst Status-Array in jeder Runde (ohne Details)
 - Letzte Runde hat etwas andere Phasen (ohne Details)
- Entschlüsselung kehrt Verschlüsselungsschritte um (vereinfacht)
- Nachrichten, die kein Vielfaches der Blockgröße lang sind, müssen aufgefüllt werden (ohne Details)

Der AES-Algorithmus II

- Phase 1: AddRoundKey
 - 128-Bit-Rundenschlüssel aus Schlüssel ableiten
 - Status-Array mit Rundenschlüssel XOR-verknüpfen



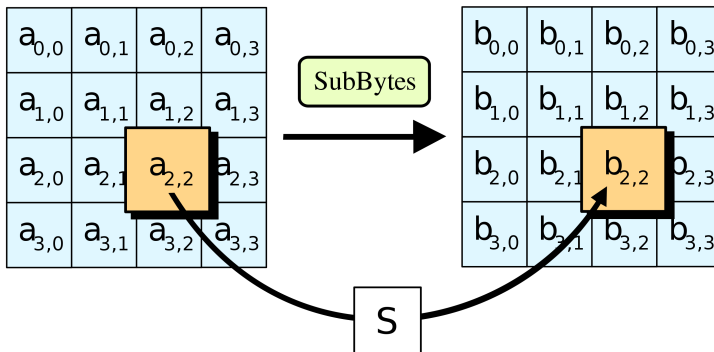
Quelle: Matt Crypto: AddRoundKey operation for AES.

<https://commons.wikimedia.org/wiki/File:AES-AddRoundKey.svg#/media/File:AES-AddRoundKey.svg> (Zugriff am 23.8.2022), 2006.

Der AES-Algorithmus III

• Phase 2: SubBytes

- Bytes basierend auf einer Nachschlagetabelle ersetzen
- Nachschlagetabelle ist fix für alle Bytes und Runden



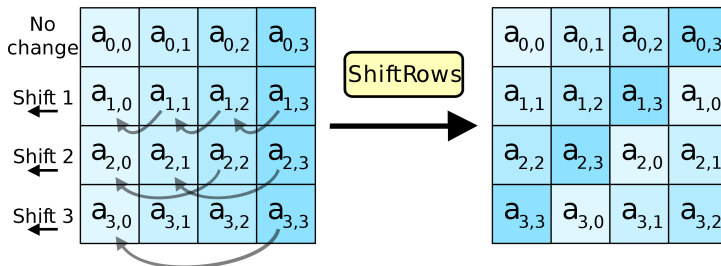
Quelle: Matt Crypto: SubBytes operation for AES.

<https://commons.wikimedia.org/wiki/File:AES-SubBytes.svg#/media/File:AES-SubBytes.svg> (Zugriff am 23.8.2022), 2006.

Der AES-Algorithmus IV

• Phase 3: ShiftRows

- Zyklisches Verschieben von Bytes in Zeilen des Status-Arrays
- Verschiedener Versatz in jeder Zeile



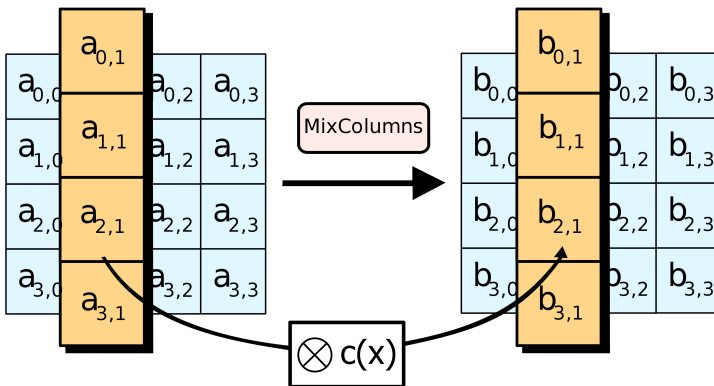
Quelle: Matt Crypto: ShiftRows operation for AES.

<https://commons.wikimedia.org/wiki/File:AES-ShiftRows.svg#/media/File:AES-ShiftRows.svg> (Zugriff am 23.8.2022), 2006.

Der AES-Algorithmus V

- Phase 4: MixColumns

- Spaltenweise Transformation (ohne Details)
- Erreicht Diffusion gemeinsam mit Phase 3

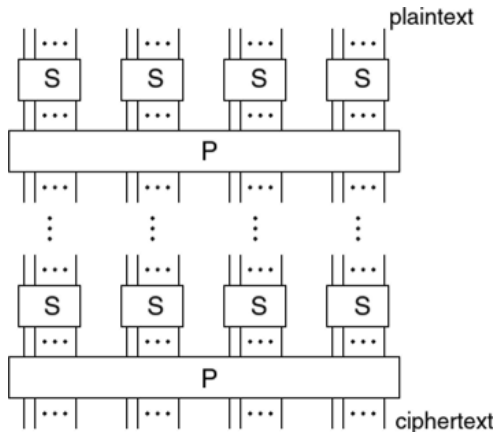


Quelle: Matt Crypto: MixColumns operation for AES.

<https://commons.wikimedia.org/wiki/File:AES-MixColumns.svg#/media/File:AES-MixColumns.svg> (Zugriff am 23.8.2022), 2006.

Einschub: Substitutions-Permutations-Netzwerke I

- Mehrere Runden umkehrbarer Substitutionen und Permutationen
- Setzt das Konfusions-Diffusions-Paradigma um

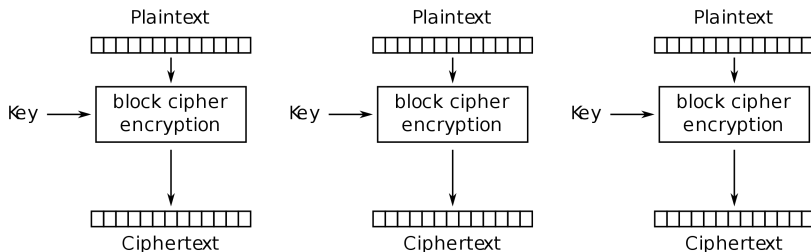


Quelle: Ebrary.net: DES. https://ebrary.net/134519/computer_science/ (Zugriff am 23.8.2022), 2022.

- Konfusion-Diffusions-Paradigma
 - Konfusion: Permutiere alle Teile/Bytes eines Blocks separat
 - Diffusion: Ordne Bits um, um Änderungen in alle Teile der Ausgabe zu verbreiten
 - Wiederholte Verwendung von Konfusion und Diffusion ergibt insgesamt eine zufällig aussehende Permutation (ohne Details)
 - Lawineneffekt: Kleine Änderungen in der Eingabe führen zu großen Änderungen an der Ausgabe (ein einzelnes Eingabebit sollte alle Ausgabebits beeinflussen)
 - Bei einem geänderten Bit in der Eingabe wird erwartet, dass es 50% der Ausgabebits umdreht
- Sicherheit hängt von Wahl der Substitutionen und Permutationen sowie der Anzahl der Runden ab
- Netzwerke/Blockchiffren selbst sind **nicht** sicher gegen Chosen-Plaintext-Angriffe (Details in Kapitel 3 von Katz (2021))

Betriebsmodi: ECB

- ECB: Electronic-Code-Book-Modus
- Selber Klartext liefert selben Chiffretext bei selbem Schlüssel → identische Klartextblöcke wiederholen sich im Chiffretext → **nicht** sicher!



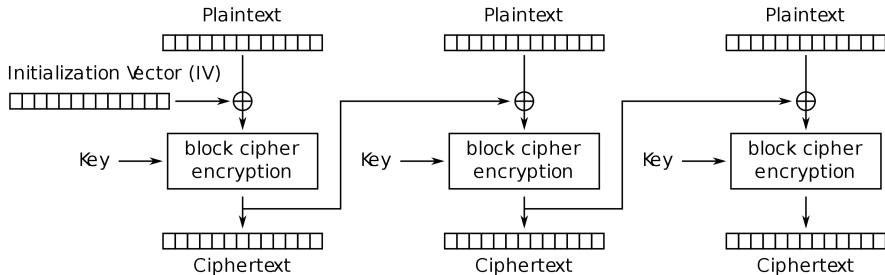
Electronic Codebook (ECB) mode encryption

Quelle: WhiteTimberwolf: Encryption using the Electronic Code Block (ECB) mode.

https://commons.wikimedia.org/wiki/File:ECB_encryption.svg#/media/Datei:ECB_encryption.svg (Zugriff am 23.8.2022), 2013.

Betriebsmodi: CBC I

- CBC: Cipher-Block-Chaining-Modus
- Chiffretexte von vergangenen Blöcken beeinflussen Chiffretext von aktuellem Block
- Initialisierungsvektor (IV) muss für ersten Block gewählt werden



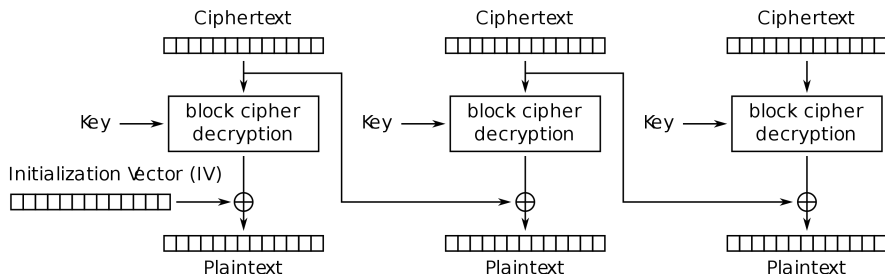
Cipher Block Chaining (CBC) mode encryption

Quelle: WhiteTimberwolf: Encryption using the Cipher Block Chaining (CBC) mode.

https://commons.wikimedia.org/wiki/File:CBC_encryption.svg#/media/File:CBC_encryption.svg (Zugriff am 23.8.2022), 2013.

Betriebsmodi: CBC II

- IV sollte zufällig sein und nicht wiederverwendet werden
- IV muss mit Chiffretext zur Entschlüsselung übertragen werden
- Ver- und Entschlüsselung können nicht parallelisiert werden



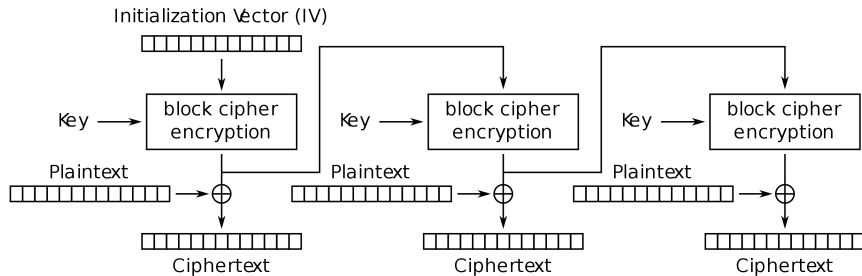
Cipher Block Chaining (CBC) mode decryption

Quelle: WhiteTimberwolf: Decryption using the Cipher Block Chaining (CBC) mode.

https://commons.wikimedia.org/wiki/File:CBC_decryption.svg#/media/File:CBC_decryption.svg (Zugriff am 23.8.2022), 2013.

Betriebsmodi: OFB

- OFB: Output-Feedback-Modus
 - Nur IV wird wiederholt in Blockchiffre gespeist, nicht der Klartext
- Klartext wird mit verschlüsseltem Bytestrom XOR-verknüpft



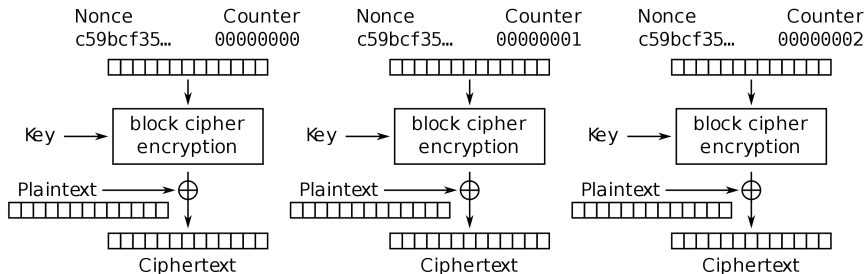
Output Feedback (OFB) mode encryption

Quelle: WhiteTimberwolf: Encryption using the Output Feedback (OFB) mode.

https://commons.wikimedia.org/wiki/File:OFB_encryption.svg#/media/File:OFB_encryption.svg (Zugriff am 23.8.2022), 2013.

Betriebsmodi: CTR

- CTR: Counter-Modus (verschiedene Variationen)
- Wie OFB, aber mit sich schrittweise erhöhendem Zähler (beginnt bei Zufallszahl)
- Parallelisierbar (wie OFB) und sicher gegen Chosen-Plaintext-Angriff



Counter (CTR) mode encryption

Quelle: WhiteTimberwolf: Encryption using the Counter (CTR) mode.

https://commons.wikimedia.org/wiki/File:CTR_encryption_2.svg#/media/File:CTR_encryption_2.svg (Zugriff am 23.8.2022), 2013.

- Welche Schlüssellänge ist ausreichend?
 - Hängt davon ab, bis wann der Chiffretext sicher sein soll
 - Hängt davon ab, wen Sie fragen (professionelle Empfehlungen)
 - Vergleich auf <https://www.keylength.com/en/compare/>
- AES und die präsentierten Modi bieten keine
 - Integrität/Authentifizierung: Eve kann Bits des Chiffretexts ändern, ohne dass Bob es bemerkt → GCM (Galois/Counter-Modus, ohne Details) oder zusätzliche Integritätsüberprüfungen → kryptografische Hashes
 - Sicherheit gegen Chosen-Ciphertext-Angriffe ohne zusätzliche Maßnahmen
 - Sicherheit, wenn sie unsachgemäß verwendet werden, z.B. wenn IVs wiederverwendet werden

- Kryptografie mit geheimen Schlüsseln erfordert einen vorab geteilten Schlüssel
- Wie teilen Alice und Bob einen geheimen Schlüssel?
 - Physisch (Treffen, Postversand etc.)
 - Schlüsselverteilungszentren (erfordert Vertrauen)
 - ... (andere Lösungen, die für flüchtige Kommunikation unpraktikabel sind)
- Bei mehreren kommunizierenden Parteien
 - Jedes Paar kommunizierender Parteien benötigt seinen eigenen Schlüssel
 - Quadratische Komplexität (unpraktikabel für Speicherung und Verwaltung)
- Alternative: Diffie-Hellman-Schlüsseltausch

Einschub: Modulare Arithmetik I

- Für alle $a, b, N \in \mathbb{N} \setminus \{0, 1\}$, $a \equiv b \pmod{N}$, wenn $a \bmod N = b \bmod N$
- a und b sind kongruent modulo N , wenn ihre Reste nach Division durch N gleich sind, z.B. $15 \equiv 3 \pmod{12}$; $123 \equiv 35 \equiv 2 \pmod{11}$



Quelle: Time Clock Experts.com: Pyramid 13Änalog STD 12/24-Hr Clock Battery Operated (For 915MHz).
<https://www.timeclockexperts.com/Pyramid-13-915MHz-9A13D-Battery-Operated-p/s9a3acgbxb.htm> (Zugriff am 25.8.2022), 2006.

- Addieren in einem Modul ist wie auf einer Uhr zu addieren –
Beispiele: $11 + 3 \equiv 14 \equiv 2 \pmod{12}$; $123 + 35 \equiv 158 \equiv 4 \pmod{11}$
- Die meisten gewöhnlichen Rechenregeln gelten auch in modularer Arithmetik:
 - Addition: Wenn $x \equiv x' \pmod{N}$ und $y \equiv y' \pmod{N}$, dann gilt $x + y \equiv x' + y' \pmod{N}$
 - Subtraktion (analog)
 - Multiplikation: Wenn $x \equiv x' \pmod{N}$ und $y \equiv y' \pmod{N}$, dann gilt $x \cdot y \equiv x' \cdot y' \pmod{N}$
 - Division funktioniert im Allgemeinen **nicht**
- Umkehrbarkeit (nicht immer möglich):
 - Definiere a^{-1} , sodass $a \cdot a^{-1} \equiv 1 \pmod{N}$
 - Beispiel: $a = 3$, $a^{-1} = 4$, $N = 11$; Gegenbeispiel: $a = 3$, $N = 12$
 - Erfordert, dass $\gcd(a, N) = 1$ (Details in Katz (2021))

Einschub: Multiplikative Gruppen I

- Eine multiplikative Gruppe ist eine Menge S ,
 - die multiplikativ abgeschlossen ist (Multiplizieren liefert wieder ein Element der Menge)
 - die ein neutrales Element e hat, sodass $\forall s \in S : e \cdot s = s$
 - in der jedes Element ein inverses hat (umkehrbar ist)
- Gegenbeispiele:
 - $\{2\}$ erfüllt keines der Kriterien
 - $\mathbb{R}$: Null ist nicht umkehrbar
 - $\{1, j, -j\}$ ist nicht abgeschlossen (z.B. $j \cdot j = -1$)
- Einfaches Beispiel: $\mathbb{R} \setminus \{0\}$ ist eine multiplikative Gruppe
 - $\mathbb{R} \setminus \{0\}$ ist abgeschlossen
 - Das neutrale Element ist 1
 - Jedes Element hat ein inverses: $e^{-1} = \frac{1}{e}$
- Multiplikationen können auch in einem Modul durchgeführt werden

Einschub: Multiplikative Gruppen II

Multiplikationstabelle der Gruppen $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$ modulo 5 (links)
und $\{3, 6, 9, 12\}$ modulo 5 (rechts):

×	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

×	3	6	9	12
3	9	3	12	6
6	3	6	9	12
9	12	9	6	3
12	6	12	3	9

Quellen: Purwanto, Hidayah, I. N., and Hasanah, D.: Results and Problems on Constructing Multiplicative Groups in Modular Arithmetic. http://fmipa.um.ac.id/wp-content/uploads/2019/10/MATEMATIKA_PURWANTO-Rev-14-23.pdf (Zugriff am 25.8.2022), 2019.

[1] Purwanto, Hidayah, I. N., and Hasanah, D.: Results and Problems on Constructing Multiplicative Groups in Modular Arithmetic. http://fmipa.um.ac.id/wp-content/uploads/2019/10/MATEMATIKA_PURWANTO-Rev-14-23.pdf (Zugriff am 25.8.2022), 2019.

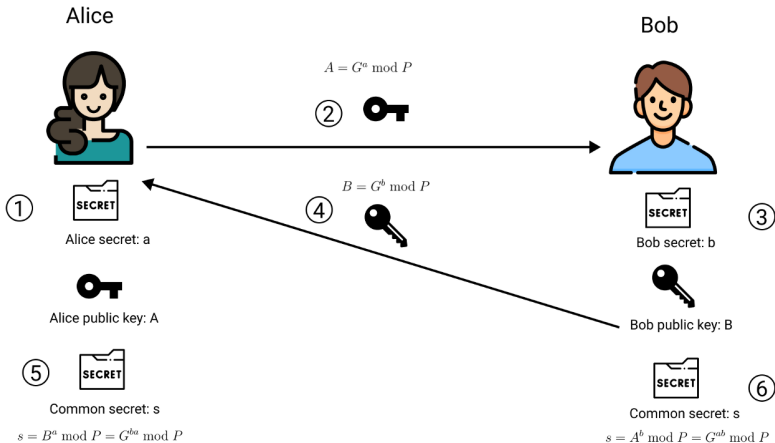
Einschub: Multiplikative Gruppen III

- Exponentiation für Gruppen (analog zu „herkömmlicher“ Exponentiation)
 - Definiert als wiederholte Multiplikation: $g^m := \underbrace{g \cdot g \cdot \dots \cdot g}_{m \text{ Mal}}$
 - $g^0 := 1$ (wobei 1 ein Gruppenelement sein muss)
 - Bekannte Regeln gelten, z.B. $(g^a)^b = g^{a \cdot b}$
- Spezielle Gruppen $\mathbb{Z}_n^* := \{1, 2, \dots, n-1\}$ mit Multiplikation modulo $n \in \mathbb{P}$ (aktuell beschränkt, allgemeine Definition später)
 - n **muss** prim sein (sonst sind einige Elemente nicht umkehrbar)
 - Zur Basis $g \in \mathbb{Z}_n^*$ potenzieren generiert (zyklisch)
Teilmengen/Teilgruppen
 - Beispiel: $g = 3$ in $\mathbb{Z}_5^*$ generiert $\{1, 3, 4, 2\} = \mathbb{Z}_5^*$
 - Kleineres Beispiel: $g = 4$ in $\mathbb{Z}_5^*$ generiert die Teilgruppe $\{1, 4\}$

Anmerkung: Beweise zu den obigen Behauptungen siehe Katz (2021)

- Diskreter Logarithmus (analog zum „herkömmlichen“ Logarithmus)
 - Umkehroperation zur Exponentiation in einem Modul
 - Definition: $\log_g(h) = x$, wenn $g^x \equiv h \pmod n$ in $\mathbb{Z}_n^*$
 - Beispiel: $\log_3(4) \equiv 2 \pmod 5$
 - Für einige g in zyklischen Gruppen ist $\log_g$ einfach zu berechnen
 - Für einige g wird **angenommen**, dass $\log_g$ schwierig zu berechnen ist
 - Diffie-Hellman-Vermutung (Computational-Diffie-Hellman-Problem):
 - Aufgabe: Gegeben $X := g^x$ und $Y := g^y$ mit bekanntem Generator g und Modul n , bestimme $g^{x \cdot y} = (g^x)^y = (g^y)^x$
 - Wenn $\log_g$ für g in $\mathbb{Z}_n^*$ einfach zu berechnen ist, ist $g^{x \cdot y}$ einfach als $X^{\log_g(Y)} = (g^x)^y = g^{x \cdot y}$ zu berechnen
 - **Wenn** $\log_g$ für g in $\mathbb{Z}_n^*$ schwierig zu berechnen ist, ist $g^{x \cdot y}$ schwierig zu berechnen
- Verwende diese Vermutung, um ein Schlüsseltauschprotokoll zu konstruieren

Diffie-Hellman-Schlüsseltausch II



Quelle: Kosolov, P.: Diffie-Hellman Key Exchange via REST.

https://medium.com/@razumovsky_r/diffie-hellman-key-exchange-via-rest-b7a91c9df7b1 (Zugriff am 25.8.2022), 2022.

Diffie-Hellman-Schlüsseltausch III

- Schritte (G und P werden als öffentlich bekannt angenommen):
 - ① Alice generiert ein zufälliges Gruppenelement a (außerhalb des Umf.)
 - ② Alice sendet $A = G^a \bmod P$ an Bob
 - ③ Bob generiert ein zufälliges Gruppenelement b (außerhalb des Umf.)
 - ④ Bob sendet $B = G^b \bmod P$ an Alice
 - ⑤ Alice berechnet das gemeinsame Geheimnis/den Schlüssel
$$s = B^a = (G^b)^a = G^{a \cdot b}$$
 - ⑥ Bob berechnet das gemeinsame Geheimnis/den Schlüssel
$$s = A^b = (G^a)^b = G^{a \cdot b}$$
- Eve kann $A = G^a$ und $B = G^b$ sehen, aber aus dieser Information alleine $s = G^{a \cdot b}$ nicht berechnen (Diffie-Hellman-Vermutung)
- Es könnte andere Wege geben, s zu berechnen $\rightarrow$ stärkere Sicherheitsdefinition über Decisional-Diffie-Hellman-Problem (außerhalb des Umfanges)
- Ein „man in the middle“ könnte immer noch die Kommunikation abhören, d.h. zusätzlicher Schutz ist notwendig (ohne Details)

- Forward Secrecy
 - Wenn ein Angreifer irgendwie das gemeinsame Geheimnis herausfindet, kann er nur die aktuelle, nicht aber vergangene oder zukünftige Konversationen zwischen Alice und Bob mitlesen
 - Erfordert, dass Alice und Bob jedes Mal neue Schlüssel generieren/austauschen, wenn sie kommunizieren (auch Diffie-Hellman Ephemeral genannt, von engl. *ephemeral* für kurzlebig, flüchtig)
- Wie gute Generatoren und Gruppen wählen
 - Für $\mathbb{Z}_{n \in \mathbb{P}}^*$ ist diskreter Logarithmus in einigen Fällen leicht zu berechnen
→ „Sichere“ Primzahlen und bestimmte große Teilgruppen verwenden (außerhalb des Umfangs)
 - Zusätzlich auf angreifbare g und n überprüfen (außerhalb des Umfangs)
- Wie die Größe des Moduls n wählen
 - Symmetrische Schlüsselgrößen können **nicht** mit (Teil-)Gruppengrößen verglichen werden
 - Empfehlungen unter <https://www.keylength.com/en/compare/>

[2] IBM Corporation: Variants of Diffie-Hellman.

https://www.ibm.com/docs/en/zvse/6.2?topic=SSB27H_6.2.0/fa2ti_openssl_variants_of_diffie_hellman.html

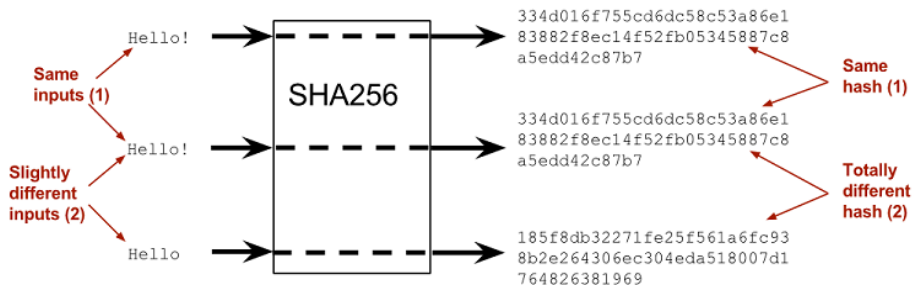
(Zugriff am 25.8.2022), 2021.

Überblick über kryptografische Hashes I

- Eine Hashfunktion $H(m)$
 - erzeugt einen Fingerabdruck/Hash einer gegebenen Nachricht m
 - ordnet einer variabel langen Eingabe eine Ausgabe fester Länge zu
 - ist eine „Einbahn“, d.h. rechnerisch unzumutbar sie umzukehren
- Anwendungen
 - Nachrichtenintegrität: Verfälschungen/Sabotage erkennen
 - Digitale Signaturen (später): Hash statt voller Nachricht signieren
 - Pseudozufallszahlenerzeugung (außerhalb des Umfangs)
- Beispielhashfunktion: Secure Hash Algorithm 2

Kryptografische Hashes II

- Dieselbe Eingabe liefert immer dieselbe Ausgabe, d.h.
 $H(m_1) = H(m_2)$, wenn $m_1 = m_2$
- Verschiedene Eingaben liefern idealerweise verschiedene Ausgaben (später mehr)



Quelle: Manning Publications: Cryptographic Hashes and Bitcoin.

<https://freecontent.manning.com/cryptographic-hashes-and-bitcoin/> (Zugriff am 23.8.2022), 2017.

- Eine Einwegfunktion
 - ist einfach zu berechnen
 - ist schwer umzukehren
 - baut auf die Existenz eines Problems auf, das einfach in eine und schwierig in die Gegenrichtung zu berechnen ist
- Beispiel: Faktorisierung als Einwegfunktion
 - Gegeben beliebige und große $p, q \in \mathbb{P}$, multipliziere $N = p \cdot q$ und gib letzteres aus
 - Umgekehrtes Problem: gegeben ein beliebiges und großes N , das das Produkt zweier Primzahlen ist, faktorisiere p und q , sodass $p \cdot q = N$
 - **Nimmt an**, dass die Faktorisierungsvermutung wahr ist (Multiplizieren ist rechnerisch „leicht“, aber Faktorisieren ist rechnerisch „schwierig“)
- Einwegfunktionen alleine sind unzureichend, um robuste Hashfunktionen zu konstruieren
- Praktische Hashfunktionen begründen ihre Sicherheit nicht auf beweisbarer Reduktion auf Einwegfunktionen, sondern Heuristiken

Secure Hash Algorithm 2 I

- Secure Hash Algorithm 2 (SHA-2)

- Nach SHA-1-Schwachstellen standardisiert (2001)
- Weit verbreitet und unterstützt
- Unterstützt 224-, 256-, 384- und 512-Bit-Ausgaben (siehe <https://www.keylength.com/en/compare/> für empfohlene Ausgabegrößen)
- Bekannte Angriffe auf vereinfachte Variationen von SHA-2, aber kein besserer Weg als generische Angriffe, die alle kryptografischen Hashes betreffen (später), bekannt, um volles SHA-2 zu brechen

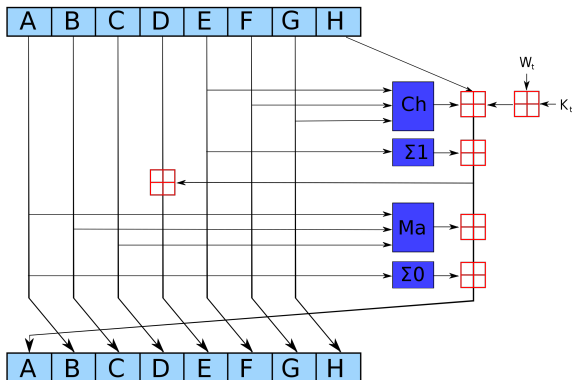
- Beispiel: SHA-512 (SHA-2 mit 512-Bit-Ausgabe)

- 80 Runden Aktualisierung von internen 32-Bit-Statusvariablen A-H mit 64-Bit-Wort W basierend auf der Eingabenachricht (vereinfacht)
- Automatisches Auffüllen für Nachrichten, deren Länge kein ganzzahliges Vielfaches von 64 Bit ist

[3] Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In: FSE 2012: Fast Software Encryption, 2012.

Secure Hash Algorithm 2 II

- Anmerkung: Das rote Plus bezeichnet 64-Bit-Addition, die blauen Funktionsblöcke führen logische und Bitshift-Operationen durch, wie in der Bildquelle ausgeführt



Source: kockmeyer: A schematic that shows the SHA-2 algorithm.

<https://commons.wikimedia.org/wiki/File:SHA-2.svg#/media/File:SHA-2.svg> (Zugriff am 23.8.2022), 2007.

[3] Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In: FSE 2012: Fast Software Encryption, 2012.

- Grundlegende Definitionen:

- Kollision: Zwei Eingaben (Nachrichten), die dieselbe Ausgabe (Hash) liefern: $H(m_1) = H(m_2)$, wenn $m_1 \neq m_2$
- Kollisionsresistenz: Es ist rechnerisch unzumutbar Kollisionen zu finden

- Kollisionen **müssen** auftreten, wenn beliebig lange Eingaben bei Ausgaben fester Länge erlaubt werden (Schubfachprinzip)

→ Schwächere Sicherheitsstufen in der Praxis:

- Second-Preimage-Resistenz: Gegeben eine Nachricht m , ist es unzumutbar ein $m' \neq m$ zu finden, sodass $H(m') = H(m)$
- Preimage-Resistenz: Gegeben ein Hash $h = H(m)$, ist es unzumutbar ein m' zu finden, sodass $H(m') = h$

Angriffe auf kryptografische Hashes – Überblick

- Brute-Force-Angriff ((Second-)Preimage-Angriff)
 - Versuche ein Urbild (engl. *preimage*) m oder ein anderes m' zu finden, sodass $H(m) = H(m') =: h$
 - Generiere Nachrichten $m_1, m_2, \dots$
 - Für n -Bit-Hashes ist die Wahrscheinlichkeit, dass ein beliebiges m_i zu h gehasht wird $\frac{1}{2^n}$
→ 2^n Versuche notwendig
- Geburtstagsangriff (Kollisionsangriff)
 - Versuche zwei Urbilder $m \neq m'$ zu finden, sodass $H(m) = H(m')$
 - Generiere **unterschiedliche** Nachrichten $m_1, m_2, \dots$ **zufällig gleichverteilt**
 - Für n -Bit-Hashes ist die Wahrscheinlichkeit, dass zwei beliebige m_i und m_j ($i \neq j$) zur selben Ausgabe gehasht werden, durch das Geburtstagsparadoxon bestimmt
→ $\approx 2^{\frac{n}{2}}$ Versuche notwendig
- Verlängerungen (engl. *length extensions*, außerhalb des Umfangs)
- Teilnachrichtangriff (engl. *partial-message attack*, außerhalb d. Umf.)

Danke für die Aufmerksamkeit!

Fragen?