

Cybersicherheit

Kryptographie

Prof. Dr. Daniel Loebenberger

Amberg, 09. Oktober 2024



- Bitte beachten Sie das Urheberrecht!
- Alle Materialien dieser Vorlesung sind – auch wenn sie nicht ausdrücklich gekennzeichnet sind – urheberrechtlich geschützt.
- Sie dienen ausschließlich Ihrem persönlichen Gebrauch im Rahmen dieser Vorlesung.
- Die Materialien dürfen insbesondere nicht weiter verbreitet werden.
- Eigene Aufzeichnungen (Video, Foto, Ton) der Vorlesung sind nicht gestattet.

1. Einführung
2. Kryptographie
3. Netzwerksicherheit
4. Systemsicherheit
5. Anwendungssicherheit
6. Kritische Infrastrukturen und staatlicher Geheimschutz
7. Sicherheitsmodelle und -evaluierung
8. Sicherheit im Unternehmen
9. Hacking und Pentesting (extern)
10. Ausblick

01 | Symmetrische Kryptographie

02 | Public-Key Kryptographie

03 | Hybride Systeme

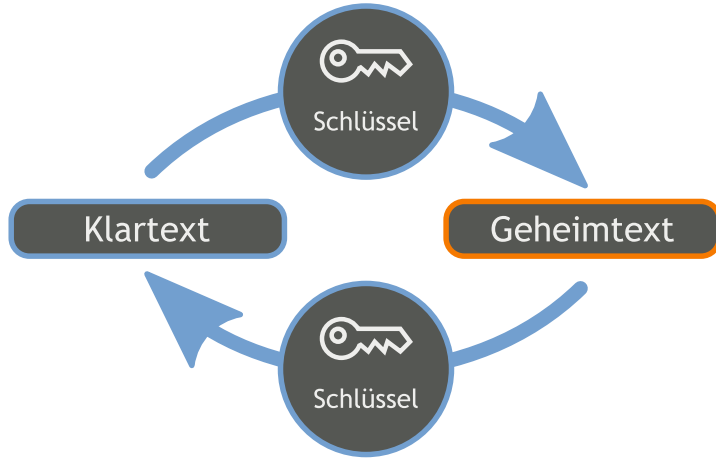
04 | Zufallserzeugung

01 | Symmetrische Kryptographie

02 | Public-Key Kryptographie

03 | Hybride Systeme

04 | Zufallserzeugung



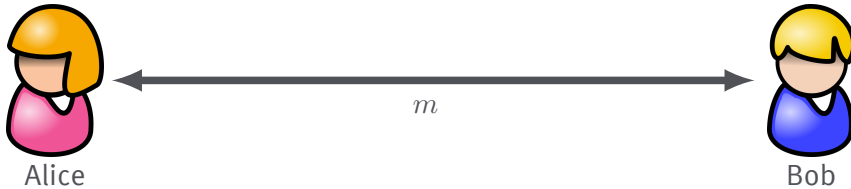


Bild: Andreas Aßmuth

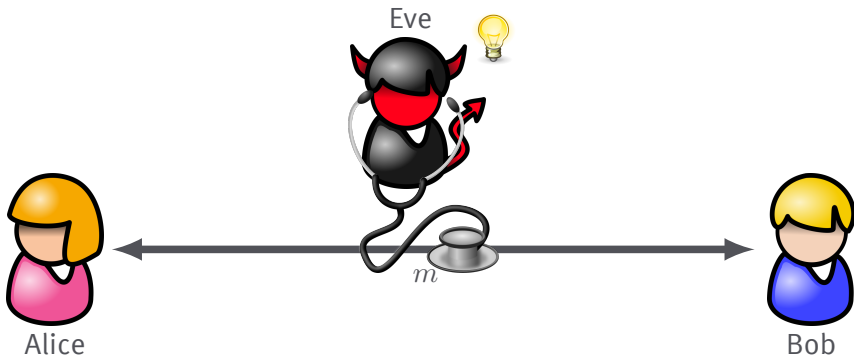


Bild: Andreas Aßmuth

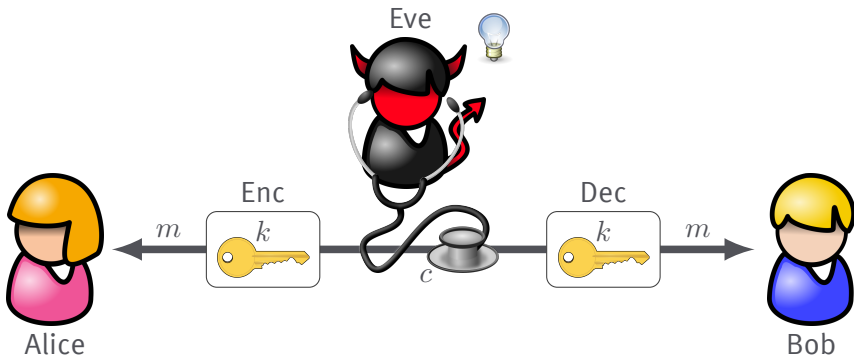
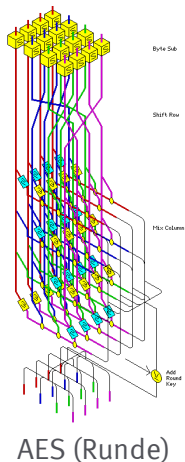


Bild: Andreas Aßmuth

Symmetrische Verschlüsselungsverfahren

Advanced Encryption Standard (AES)

- Standardisiert von der amerikanischen NIST in FIPS PUB 197
- Schlüssellänge von AES-128/192/256 ist 128, 192 bzw. 256 Bit
- Alle Varianten verschlüsseln Nachrichten mit 128 Bit
- AES wird auch vom deutschen Bundesamt für Sicherheit in der Informationstechnik (BSI) in der Richtlinie TR02102 empfohlen
- Sicherheit beruht lediglich auf Geheimhaltung des Schlüssels (Kerckhoffs' Prinzip)
- Generische Angriffe (Brute-Force) müssen den ganzen Schlüsselraum durchsuchen
- Das sog. Sicherheitsniveau entspricht daher (ungefähr) der Schlüssellänge



Definition

Das Sicherheitsniveau eines Verfahrens ist n Bit, wenn ein Angreifer 2^n Schritte benötigt, um das Verfahren zu brechen.

Definition

Die Sicherheitsbehauptung eines Verfahrens ist das Sicherheitsniveau, welches das Verfahren erreichen soll. Findet man Angriffe, die effizienter sind, so ist das Verfahren gebrochen.

Definition

Das Sicherheitsniveau eines Verfahrens ist n Bit, wenn ein Angreifer 2^n Schritte benötigt, um das Verfahren zu brechen.

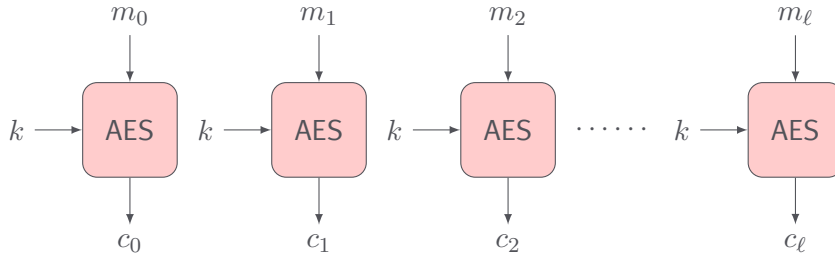
Definition

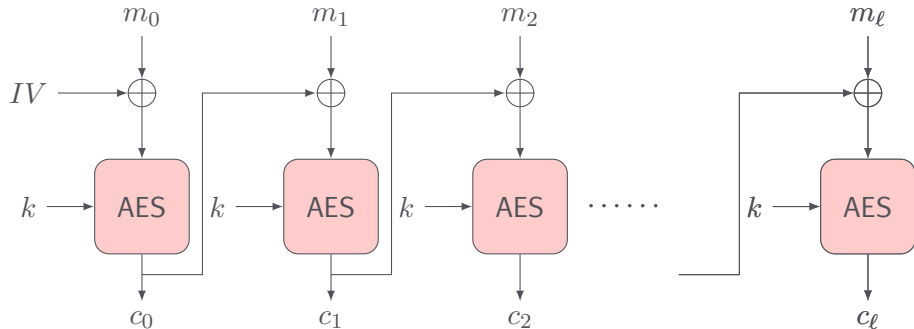
Die Sicherheitsbehauptung eines Verfahrens ist das Sicherheitsniveau, welches das Verfahren erreichen soll. Findet man Angriffe, die effizienter sind, so ist das Verfahren gebrochen.

Heute gilt ein System mit mind. 100 Bit Sicherheitsniveau als sicher.

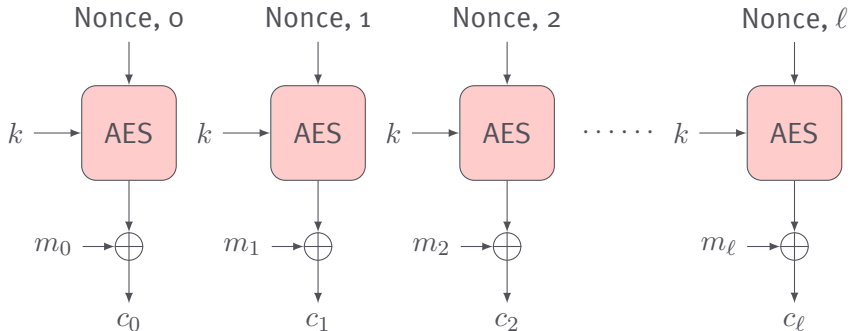
Operationsmodi

Electronic Codebook Mode

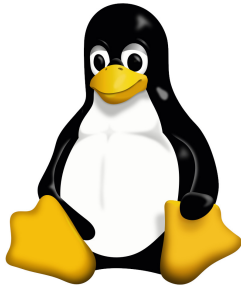




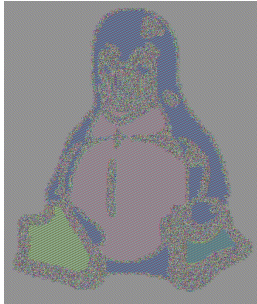
IV Initialisierungsvektor (z.B. 0)



Nonce: Einmalig genutzte Zahl (engl. „Number to be used ONCE“)



Klartext



ECB Modus



CBC/CTR Modus

Quelle: Tux the Penguin, the Linux mascot. Created in 1996 by Larry Ewing with The GIMP.

- Daten werden vor dem Verschlüsseln mit einem Padding versehen
- Danach entspricht die Länge des Klartextes einem Vielfachen der Blocklänge
- Das sind bei AES immer 16 Byte
- Der Wert jedes hinzugefügten Bytes ist die Anzahl der hinzugefügten Bytes
- Das heißt: es werden N Bytes mit jeweils dem Wert N hinzugefügt
- Das Padding ist demnach immer eines der Folgenden:

01

02 02

03 03 03

04 04 04 04

...

0f 0f 0f 0f ... 0f

10 10 10 10 ... 10 10

PKCS#7 Padding

Beispiel für AES

Nachricht 43 79 62 65 72 21 0a:

- Nachricht hat Länge 7.
- Füge 9 Bytes hinzu, um 1 · 16 Bytes zu erhalten:
- Resultierende Nachricht mit Padding:

43 79 62 65 72 21 0a 09 09 09 09 09 09 09 09

Nachricht 43 79 62 65 72 21 0a:

- Nachricht hat Länge 7.
- Füge 9 Bytes hinzu, um $1 \cdot 16$ Bytes zu erhalten:
- Resultierende Nachricht mit Padding:

43 79 62 65 72 21 0a 09 09 09 09 09 09 09 09

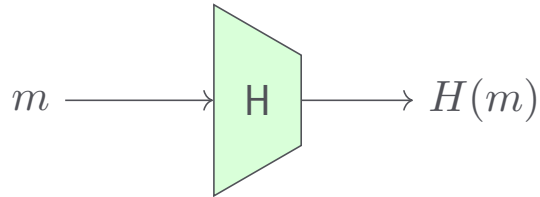
Nachricht 43 79 62 65 72 73 69 63 68 65 72 68 65 69 74 0a:

- Nachricht hat Länge 16.
- Füge 16 Bytes hinzu, um $2 \cdot 16$ Bytes zu erhalten:
- Resultierende Nachricht mit Padding:

43 79 62 65 72 73 69 63 68 65 72 68 65 69 74 0a

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

- Es wird immer mindestens 1 Byte hinzugefügt!



Eine Hash-Funktion H bildet Nachrichten m beliebiger Länge auf Hash-Werte $H(m)$ fester Länge ab.

Effizienz Die Funktion ist leicht zu berechnen

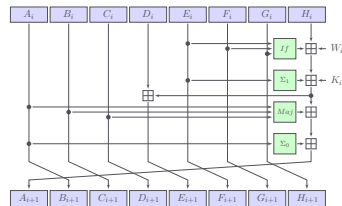
Lawinen-Eigenschaft Eine kleine Änderungen der Eingabe führt zu möglichst großen Änderungen der Ausgabe

Urbildresistenz Es ist schwer, ein Eingabe zu berechnen, die einen vorgegebenen Hash-Wert hat

Zweite Urbildresistenz Gegeben eine Eingabe, ist es schwer, eine zweite Eingabe zu finden, die den selben Hash-Wert hat

Kollisionresistenz Es ist schwer, zwei unterschiedliche Eingaben zu finden, deren Hash-Wert identisch ist

- Verschiedene Funktionen im Praxiseinsatz
- Standardisiert in
 - ▶ FIPS PUB 180-1 (SHA-1, gebrochen!)
 - ▶ FIPS PUB 180-4 (SHA-256, SHA-384, SHA-512)
 - ▶ FIPS PUB 202 (SHA-3 Familie)
- Bildet Daten auf Hash-Werte fester Länge ab:
 - ▶ SHA-1 produziert 160 Bit
 - ▶ SHA-(3)-256/384/512 produzieren 256, 384 bzw. 512 Bit
- SHA-(3)-256/384/512 werden in der TR02102 empfohlen
- Sicherheit: Hash-Funktionen müssen kollisionsresistent sein
- Generischer Angriff durch Geburtstagsangriff



SHA-256 (Runde)

Wie viele Personen müssen in einem Raum sein, damit es eine hohe Wahrscheinlichkeit gibt, dass mindestens zwei Gäste am gleichen Tag Geburtstag haben?

Wie viele Personen müssen in einem Raum sein, damit es eine hohe Wahrscheinlichkeit gibt, dass mindestens zwei Gäste am gleichen Tag Geburtstag haben?

Antwort: 23

- Bei 23 Personen ist die Wahrscheinlichkeit eine Kollision ungefähr 50 Prozent
- Bei 40 Personen liegt sie bei etwa 90 Prozent
- Beim Suchen von Kollisionen einer Hash-Funktion liegt die gleiche Situation vor
- Allerdings gibt es nicht 365 mögliche Werte, sondern $m = 2^n$, wobei n die Ausgabelänge der Hash-Funktion ist
- Wie viele Nachrichten t muss Eve also hashen?
- Für Erfolgswahrscheinlichkeit $1/2$ sind $t \approx 1.18\sqrt{m} = 1.18 \cdot 2^{n/2}$ Versuche nötig
- Das heißt: Sicherheitsniveau der Hash-Funktion ist die Hälfte der Ausgabelänge

- Realisierung von Integritätsschutz
- Aufruf der Hash-Funktion unter Nutzung eines Schlüssels k :

$$\text{MAC}_k(m) = H(k||m)$$

- Hier ist $||$ das Konkatenieren von Bitstrings
- Problem ist, dass es Angriffe auf diese Konstruktion gibt

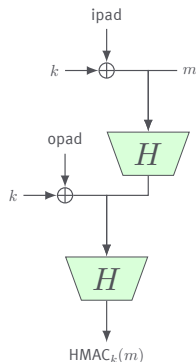
- Realisierung von Integritätsschutz
- Aufruf der Hash-Funktion unter Nutzung eines Schlüssels k :

$$\text{MAC}_k(m) = H(k\|m)$$

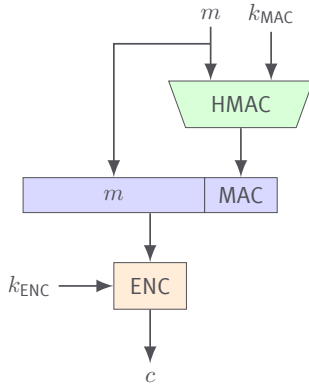
- Hier ist $\|$ das Konkatenieren von Bitstrings
- Problem ist, dass es Angriffe auf diese Konstruktion gibt
- Sichere Konstruktion:

$$\text{HMAC}_k(m) = H((k \oplus \text{opad})\|H((k \oplus \text{ipad})\|m))$$

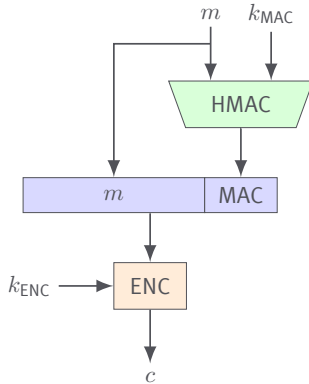
- Konstanten $\text{ipad} = 0x5C \dots 5C$, $\text{opad} = 0x36 \dots 36$
- Sicherheit: Ist H pseudozufällig, so auch $\text{HMAC}_k(m)$



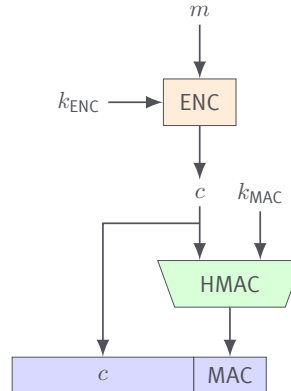
Quelle: Dr. Hugo Krawczyk, Mihir Bellare und Ran Canetti.
HMAC: Keyed-Hashing for Message Authentication.
RFC 2104. Feb. 1997. DOI: 10.17487/RFC2104. URL:
<https://rfc-editor.org/rfc/rfc2104.txt>



MAC-then-Encrypt



MAC-then-Encrypt



Encrypt-then-MAC

(Historisches) Interludium

American Standard Code for Information Interchange (ASCII)

- Historischer Standard-Code für den Informationsaustausch
- Erste Publikation im Jahr 1963
- 7 Bit pro Zeichen, ermöglicht achtes Prüfbitt
- Grundlage für spätere, auf mehr Bits basierende Kodierungen für Zeichensätze
- Die 95 druckbaren Zeichen umfassen:
 - ▶ das lateinische Alphabet in Groß- und Kleinschreibung
 - ▶ die zehn Ziffern
 - ▶ Interpunktion und Sonderzeichen
- Zeichenvorrat entspricht weitgehend dem einer Tastatur oder Schreibmaschine für die englische Sprache
- Die nicht druckbaren Steuerzeichen enthalten
Ausgabezeichen wie Zeilenvorschub, Tabulatorzeichen etc

USASCII code chart

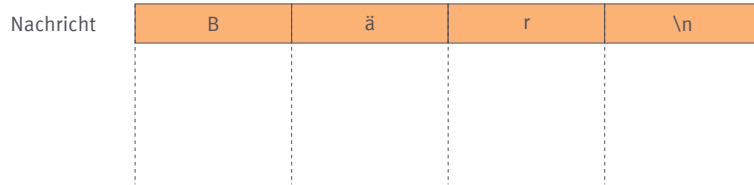
								Column							
								Row							
b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	NUL	DLE	SP	@	P	\	p	
0	0	0	0	0	0	1	1	SOM	DC1	!	!	A	Q	a	q
0	0	0	0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	0	0	1	1	0	3	ETX	DC3	#	3	C	S	c	s
0	0	0	1	0	0	0	4	EOT	DC4	\$	4	D	T	d	t
0	0	0	1	0	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	0	0	1	0	1	0	6	ACK	SYN	&	6	F	V	f	v
0	0	0	1	0	1	1	7	BEL	ETB	'	7	G	W	g	w
0	0	1	0	0	0	0	8	BS	CAN	(8	H	X	h	x
0	0	1	0	0	0	1	9	HT	EM)	9	I	Y	i	y
0	0	1	0	0	1	0	10	LF	SUB	*	:	J	Z	j	z
0	0	1	0	0	1	1	11	VT	ESC	+	;	K	[k	{
0	0	1	0	1	0	0	12	FF	FS	.	<	L	\	l	
0	0	1	0	1	0	1	13	CR	GS	-	=	M]	m	~
0	0	1	0	1	1	0	14	SO	RS	.	>	N	^	n	~
0	0	1	0	1	1	1	15	SI	US	/	?	O	_	o	DEL

US-ASCII Code Chart. Scanner copied from the material delivered with TerminiNet 300 impact type printer with Keyboard, February 1972, General Electric Data communication Product Dept., Waynesboro, Virginia

- Verfahren zur Kodierung von 8-Bit-Binärdaten als 7-Bit ASCII Charakter
- Keine Verschlüsselung im kryptographischen Sinne!
- Nutzung von Codebuchstaben aus einem Vorrat der folgenden 64 Zeichen
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
codiert durch sechs Bits $A \hat{=} 000000$, $B \hat{=} 000001$, ..., $+ \hat{=} 111110$, $/ \hat{=} 111111$
- Übersetzung von je drei Byte in vier 6 Bit Blöcke
- Dazu Padding der Nachricht auf eine durch drei teilbare Länge
- Anzahl der Padding-Bytes wird mit null, einem oder zwei Symbolen = an das Ende der codierten Nachricht gestellt

Base64 Codierung

Beispiel



Base64 Codierung

Beispiel

Nachricht	B	ä	r	\n
ISO-8859-1 Hexcode	0x42	0xE4	0x72	0x0A

Base64 Codierung

Beispiel

Nachricht	B	ä	r	\n
ISO-8859-1 Hexcode	0x42	0xE4	0x72	0x0A
Bitstring	0 1 0 0 0 0 1 0 1 1 1 0 0 1 0 0 0 1 1 1 0 0 1 0 0 0 0 0 1 0 1 0			

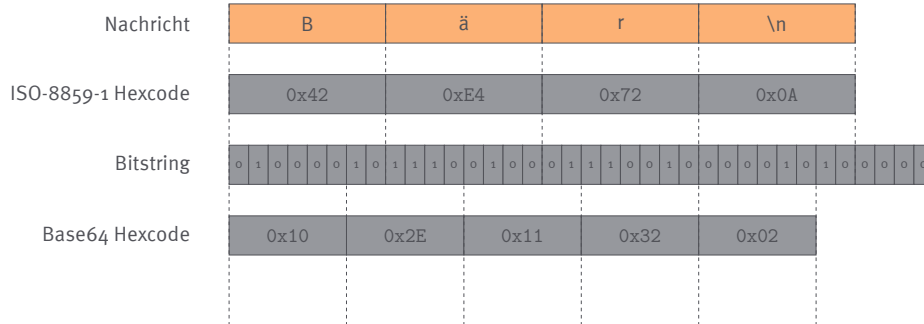
Base64 Codierung

Beispiel

Nachricht	B	ä	r	\n	
ISO-8859-1 Hexcode	0x42	0xE4	0x72	0x0A	
Bitstring	0 1 0 0 0 0 1 0 1 1 1 0 0 1 0 0 0 1 1 1 0 0 1 0 0 0 0 0 0 1 0 1 0				
Base64 Hexcode	0x10	0x2E	0x11	0x32	0x02

Base64 Codierung

Beispiel



Base64 Codierung

Beispiel

Nachricht	<div>Bärrn</div>					
ISO-8859-1 Hexcode	<div>0x420xE40x720x0A</div>					
Bitstring	<div>0100001011100100011100100000101000010100000</div>					
Base64 Hexcode	<div>0x100x2E0x110x320x020x20</div>					

Base64 Codierung

Beispiel

					Padding																							
Nachricht	B	ä	r	\n	—	—																						
ISO-8859-1 Hexcode	0x42	0xE4	0x72	0x0A	—	—																						
Bitstring	0 1 0 0 0 0 1 0 1 1 1 0 0 1 0 0 0 1 1 1 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 - - - - - - - - - - - -																											
Base64 Hexcode	0x10	0x2E	0x11	0x32	0x02	0x20	—	—																				

Base64 Codierung

Beispiel

					Padding																							
Nachricht	B	ä	r	\n	—	—																						
ISO-8859-1 Hexcode	0x42	0xE4	0x72	0x0A	—	—																						
Bitstring	0 1 0 0 0 0 1 0 1 1 1 0 0 1 0 0 0 1 1 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 - - - - - - - - - - - -																											
Base64 Hexcode	0x10	0x2E	0x11	0x32	0x02	0x20	—	—																				
Base64 Coding	Q	u	R	y	C	g	=	=																				

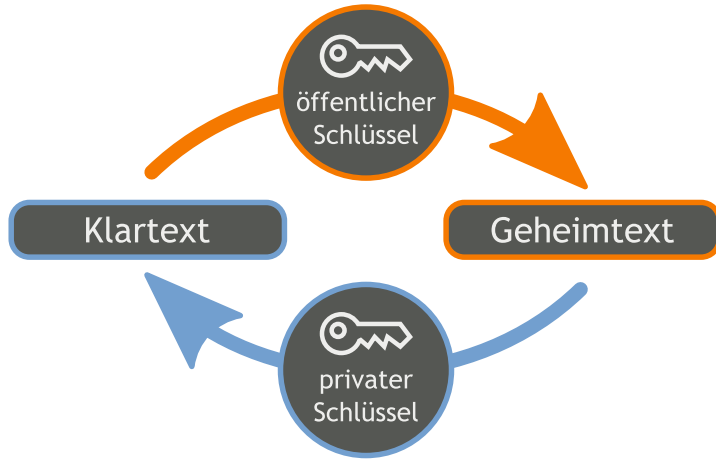
- 2.1.1. Installieren Sie eine Systemvirtualisierung wie `virtualbox` auf Ihrer Maschine und installieren Sie eine aktuelle Version einer Linux-Distribution Ihrer Wahl.
- 2.1.2. Erstellen Sie eine Datei mit dem Inhalt „Hallo Welt!“, z.B. mit `nano` oder `vim`.
- 2.1.3. Das Kommando `openssl` stellt Ihnen Zugang zu einer großen Reihe von kryptographischen Verfahren auf Ihrer Maschine bereit. Verschlüsseln Sie Ihre Datei mit `openssl` und der Chiffre AES-CBC-128 mit einem Schlüssel K Ihrer Wahl und $IV = 0000 \dots 00$. Hinweis: <https://stackoverflow.com/> und `openssl -help` sind nützlich, um die korrekten Parameter herauszufinden.
- 2.1.4. Vollziehen sie auf Byte-Ebene nach was bei der Verschlüsselung passiert, beachten Sie insbesondere das Verhalten beim Padding. Nutzen Sie dazu das Werkzeug `hexdump -C`.
- 2.1.5. Entschlüsseln Sie die Datei wieder.

01 | Symmetrische Kryptographie

02 | Public-Key Kryptographie

03 | Hybride Systeme

04 | Zufallserzeugung



Public-Key-Verschlüsselung

Öffentliche und private Schlüssel

Liste von public keys



Alice



Bob



Charly



Dave

Bild: Andreas Aßmuth

Public-Key-Verschlüsselung

Öffentliche und private Schlüssel

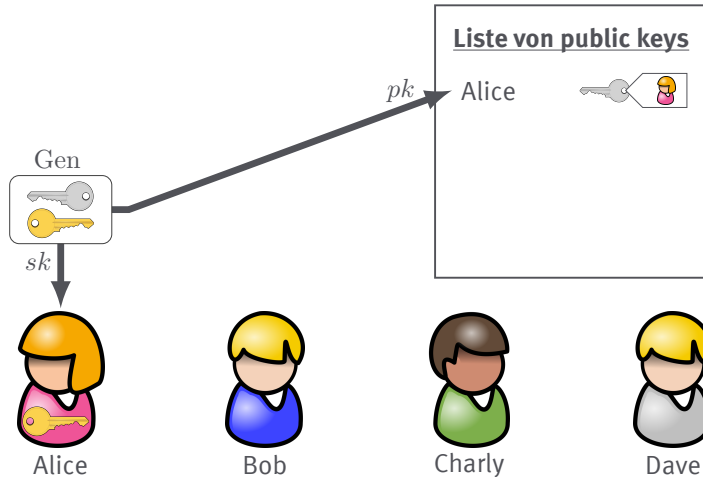


Bild: Andreas Aßmuth

Public-Key-Verschlüsselung

Öffentliche und private Schlüssel

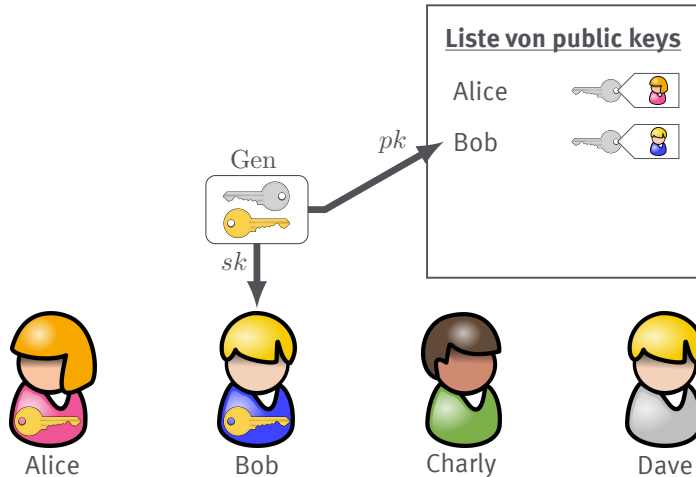



Bild: Andreas Aßmuth

Public-Key-Verschlüsselung

Öffentliche und private Schlüssel

Liste von public keys

Alice	
Bob	
Charly	
Dave	



Alice



Bob



Charly



Dave

Bild: Andreas Aßmuth

Public-Key-Verschlüsselung

Verschlüsselte Nachrichten



Alice



Charly

Liste von public keys	
Alice	
Bob	
Charly	
Dave	



Bob

Bild: Andreas Aßmuth

Public-Key-Verschlüsselung

Verschlüsselte Nachrichten

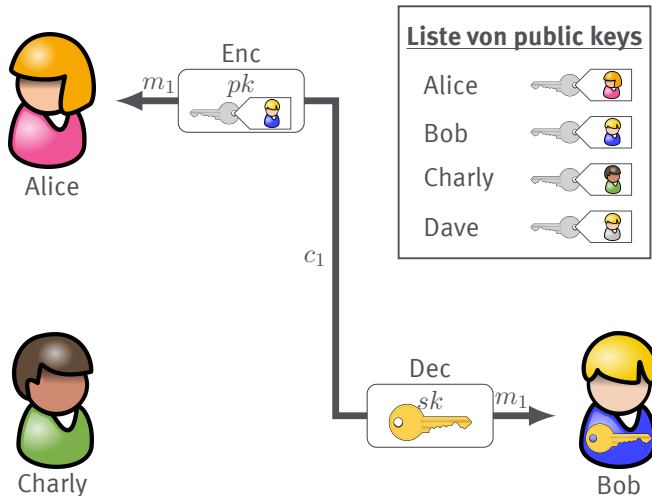






Bild: Andreas Aßmuth



Liste von public keys

Alice	
Bob	
Charly	
Dave	

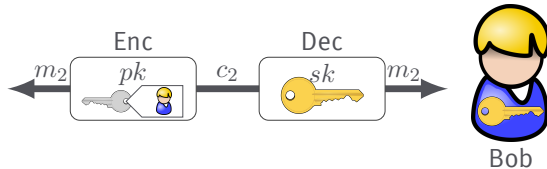
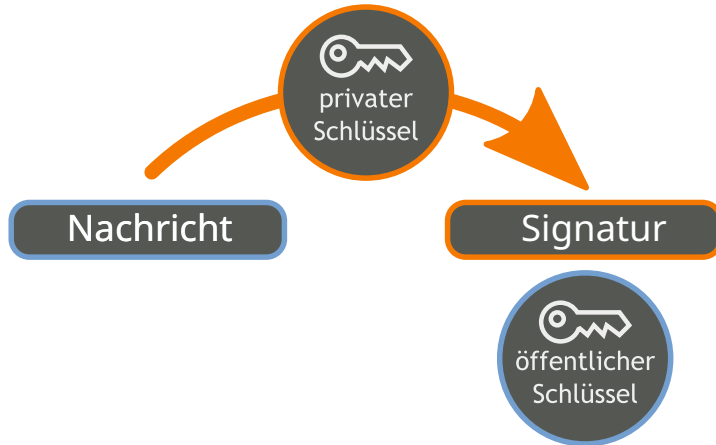


Bild: Andreas Aßmuth

- Standardisiert u.a. in PKCS#1
- Realisiert Public-Key Verschlüsselung
 - ▶ Verschlüsselung erfolgt mittels öffentlich bekanntem Schlüssel
 - ▶ Entschlüsselung nutzt dazu gehörigen geheimen Schlüssel
- Löst damit das Schlüsselaustauschproblem für symmetrische Chiffren:
 - ▶ Symmetrischer Schlüssel wird zufällig erzeugt
 - ▶ Resultat wird mit öffentlichem Schlüssel des Empfängers verschlüsselt
 - ▶ Chiffre wird übertragen
 - ▶ Empfänger kann das Chiffre entschlüsseln und erhält so den symmetrischen Schlüssel
 - ▶ Damit können beide Parteien symmetrisch verschlüsseln und authentisieren
 - ▶ Angreifer ohne Zugang zum geheimen Schlüssel des Empfängers können den symmetrischen Schlüssel nicht ableiten
- Public-Key Verschlüsselung setzt das Schutzziele der Vertraulichkeit technisch um



Lorem ipsum dolor sit amet, con-
setetur sadipscing elitr, sed diam
nonumy eirmod tempor invidunt
ut labore et dolore magna ali-
quyam erat, sed diam voluptua.

Hash

0xd41d8cd98f00b204e9800998ecf8427e

Sig

Lorem ipsum dolor sit amet, con-
setetur sadipscing elitr, sed diam
nonumy eirmod tempor invidunt
ut labore et dolore magna ali-
quyam erat, sed diam voluptua.

0x68b329da9893e ...c7d8ad5cb9c940

0x68b329da9893e ...c7d8ad5cb9c940

- International standardisierte Signaturverfahren
- Standardisiert in FIPS PUB 186-4
- Realisiert Digitale Signaturen
 - ▶ Signatur von Dokumenten erfolgt mittels geheimen Schlüssel
 - ▶ Prüfung der Korrektheit einer Signatur nutzt dazu gehörigen öffentlichen Schlüssel
- Durch digitale Signaturen erhalten wir Schutzziele wie
 - ▶ Integrität
 - ▶ Authentizität
 - ▶ Verbindlichkeit
 - ▶ Zurechenbarkeit
- Problem: Wie stellt man den Ersteller einer digitalen Signatur fest?

- Etabliertes Schlüsselaustauschverfahren
- Standardisierung in RFC 2631 schon im Jahr 1999
- Einsatz in den meisten gängigen Internetprotokollen
- Löst ebenfalls das Schlüsselaustauschproblem für symmetrische Chiffren:
 - ▶ Beide Kommunikationspartner wählen eine geheime Hälfte des Geheimnisses
 - ▶ Die dazu gehörigen öffentlichen Hälften werden jeweils verschickt
 - ▶ Beide Seiten sind anschließend in der Lage das gemeinsame Geheimnis zu rekonstruieren
 - ▶ Dieses bildet die Grundlage für die Ableitung des symmetrischen Schlüssels
 - ▶ Angreifer ohne Zugang zu den geheimen Hälften können den symmetrischen Schlüssel nicht ableiten

Schlüsselaustausch

Diffie-Hellman Protokoll (DH)

Parameter: p Primzahl, g Generator von \mathbb{Z}_p^\times

Alice

Bob

cf. Whitfield Diffie und Martin Hellman. "New directions in cryptography". In: IEEE transactions on Information Theory 22.6 (1976), S. 644–654

Schlüsselaustausch

Diffie-Hellman Protokoll (DH)

Parameter: p Primzahl, g Generator von \mathbb{Z}_p^\times

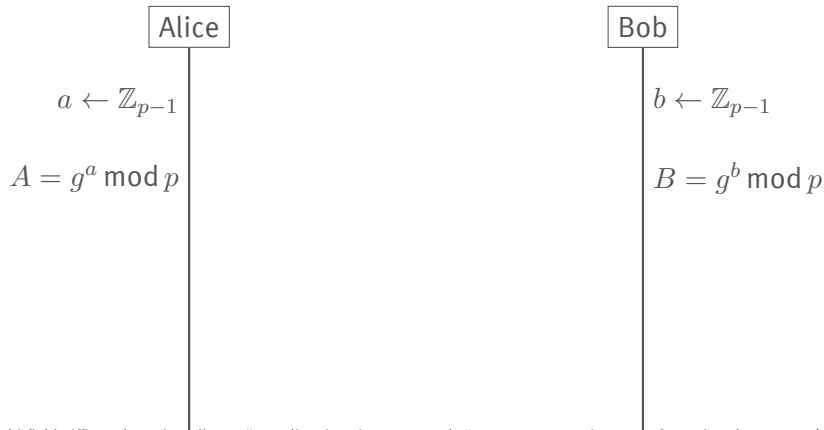


cf. Whitfield Diffie und Martin Hellman. "New directions in cryptography". In: IEEE transactions on Information Theory 22.6 (1976), S. 644–654

Schlüsselaustausch

Diffie-Hellman Protokoll (DH)

Parameter: p Primzahl, g Generator von \mathbb{Z}_p^\times

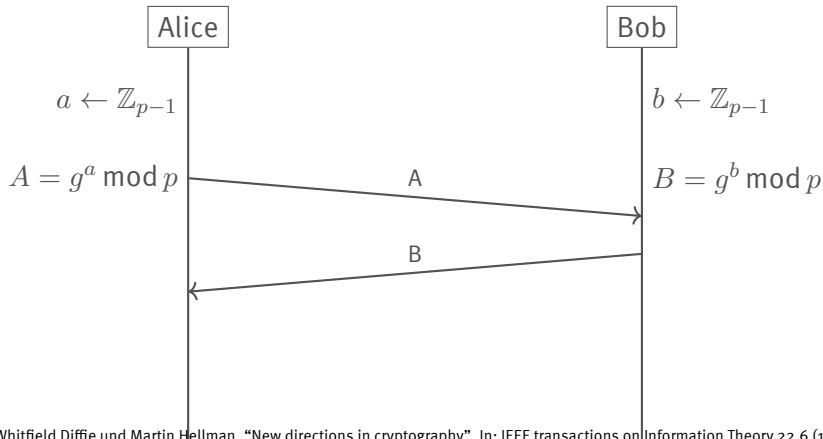


cf. Whitfield Diffie und Martin Hellman. "New directions in cryptography". In: IEEE transactions on Information Theory 22.6 (1976), S. 644–654

Schlüsselaustausch

Diffie-Hellman Protokoll (DH)

Parameter: p Primzahl, g Generator von \mathbb{Z}_p^\times

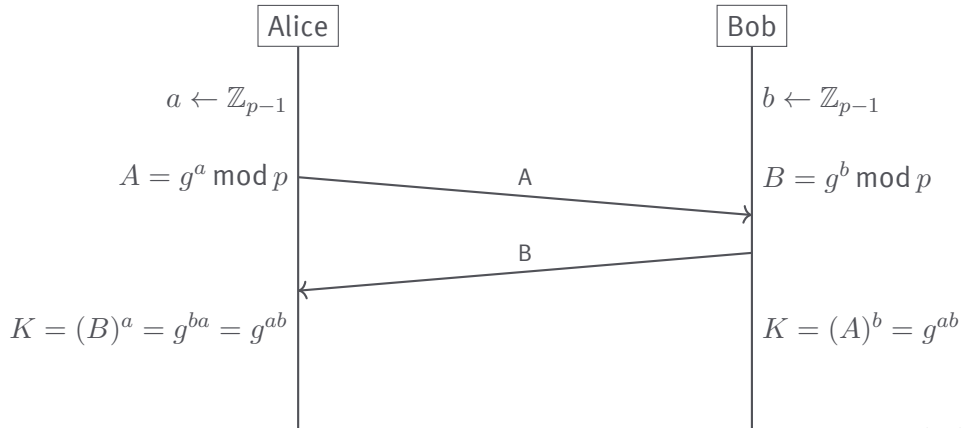


cf. Whitfield Diffie und Martin Hellman. "New directions in cryptography". In: IEEE transactions on Information Theory 22.6 (1976), S. 644–654

Schlüsselaustausch

Diffie-Hellman Protokoll (DH)

Parameter: p Primzahl, g Generator von \mathbb{Z}_p^\times

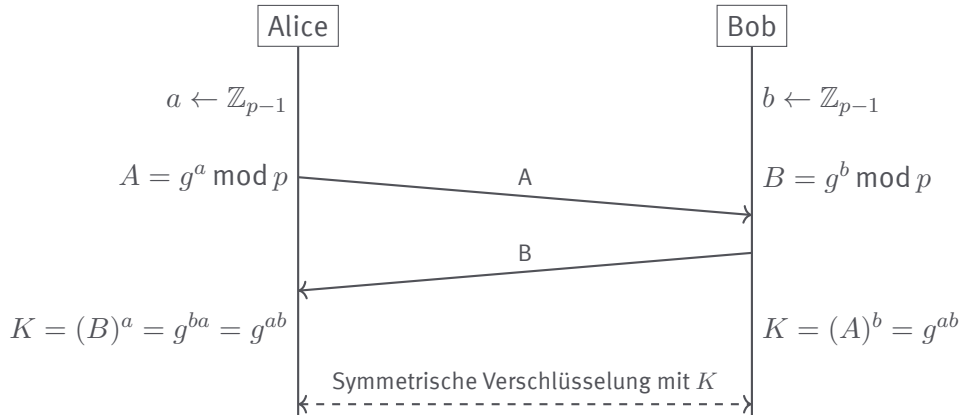


cf. Whitfield Diffie und Martin Hellman. "New directions in cryptography". In: IEEE transactions on Information Theory 22.6 (1976), S. 644–654

Schlüsselaustausch

Diffie-Hellman Protokoll (DH)

Parameter: p Primzahl, g Generator von \mathbb{Z}_p^\times



cf. Whitfield Diffie und Martin Hellman. "New directions in cryptography". In: IEEE transactions on Information Theory 22.6 (1976), S. 644–654

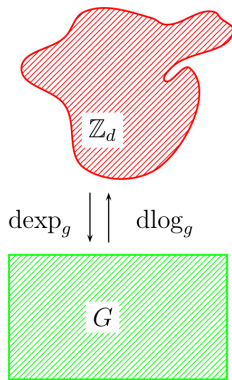
- Primzahl $p =$

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9
DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510
15728E5A 8AACAA68 FFFFFFFF FFFFFFFF
```

- Generator $g = 2$

Quelle: T. Kivinen und M. Kojo. More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE). RFC 3526 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Mai 2003. DOI: 10.17487/RFC3526. URL: <https://www.rfc-editor.org/rfc/rfc3526.txt>

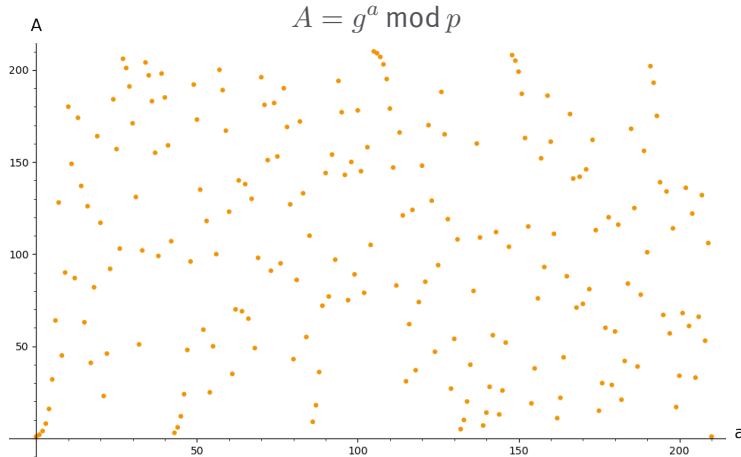
- Diskrete Exponentiation (berechne $A = g^a$ gegeben g und a) effizient realisierbar mittels sog. schneller Exponentiation
- Umkehrung, der diskrete Logarithmus (berechne $a = \text{dlog}_g(A)$ gegeben g und A , gilt als schwierig:
 - ▶ Für eine Primzahl p gibt es für die Gruppe $G = \mathbb{Z}_p^\times$ subexponentielle Algorithmen
 - ▶ In anderen Gruppen oft nur generische Angriffe mit exponentieller Laufzeit
- Sicherheitsniveau hängt daher von der Gruppe ab



cf. Joachim von zur Gathen. CryptoSchool. Berlin; Heidelberg: Springer, 2015. ISBN: 978-3-6624-8423-4. DOI: 10.1007/978-3-662-48425-8, Kapitel 4

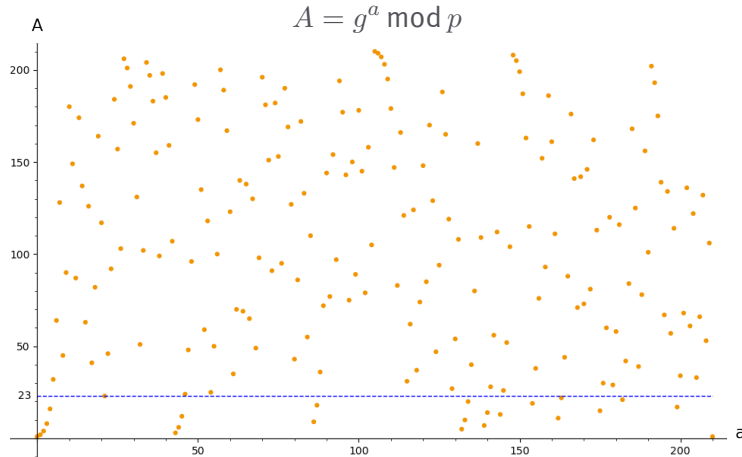
Interludium: Exponentiation und diskrete Logarithmen

Spielzeug-Beispiel: $g = 2, p = 211$



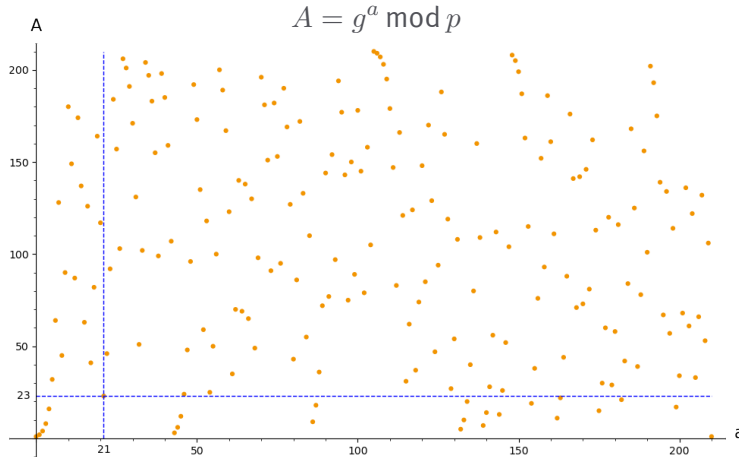
Interludium: Exponentiation und diskrete Logarithmen

Spielzeug-Beispiel: $g = 2, p = 211$



Interludium: Exponentiation und diskrete Logarithmen

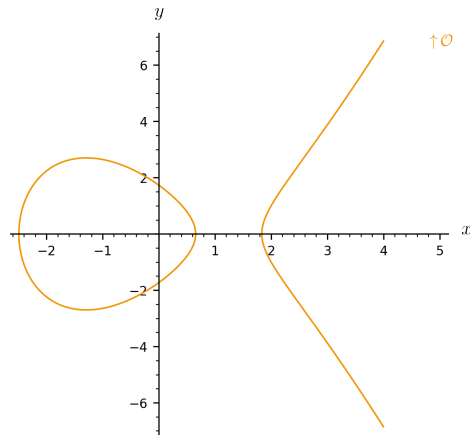
Spielzeug-Beispiel: $g = 2, p = 211$



Fakt

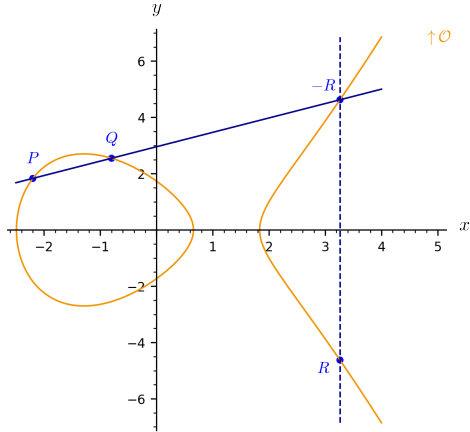
Wer diskrete Logarithmen berechnen kann, kann den Diffie-Hellman Schlüsselaustausch brechen.

- Elliptische Kurven sind eine weitere Domäne, in der man rechnen kann
- Die Kurven bestehen aus Kurven-Punkten und bilden eine additive Gruppe
- Das heißt, es gibt ein neutrales Element \mathcal{O} und man kann Punkte negieren und addieren
- Alle relevanten Konzepte (wie Punkt- oder Gruppenordnung) lassen sich auf diese Domäne übertragen
- Daher sind viele kryptographische Protokolle einfach auf elliptische Kurven übertragbar
 - ▶ Elliptic Curve Diffie Hellman (ECDH)
 - ▶ Elliptic Curve Digital Signature Algorithm (ECDSA)



Punktaddition auf elliptischen Kurven

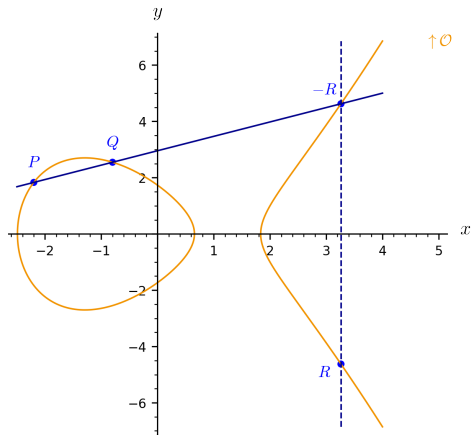
Illustration



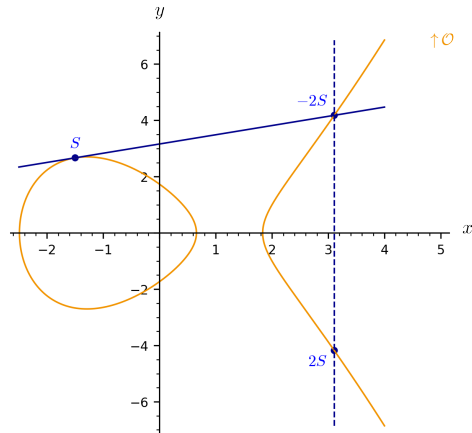
Punktaddition

Punktaddition auf elliptischen Kurven

Illustration

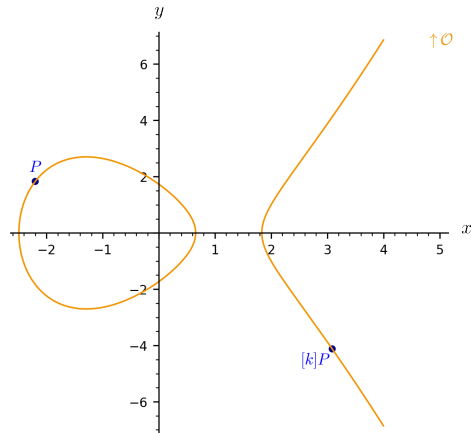


Punktaddition



Punktverdopplung

- Gegeben ist $k \in \mathbb{N}_{\geq 1}$
- Berechne $[k]P = \underbrace{P + \dots + P}_{k \text{ mal}}$
- Analog zur schnellen Exponentiation lässt sich $[k]P$ aus P und k leicht berechnen
- Die Umkehrung, der diskrete Logarithmus, d.h. k aus P und $[k]P$ zu berechnen, ist hingegen schwer
- Tatsächlich ist kein subexponentieller Algorithmus hierfür bekannt
- Daher gleiches Sicherheitsniveau bei viel kleineren Schlüsseln möglich!



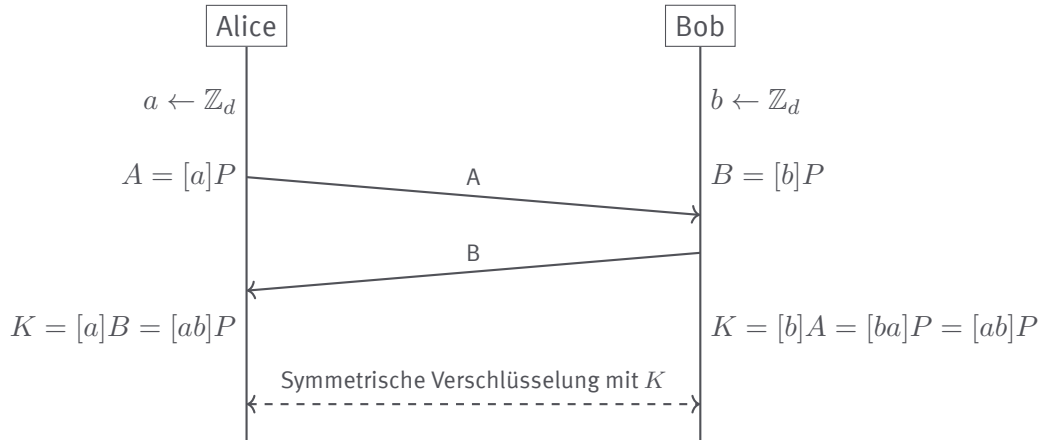
Zahlreiche unterschiedliche Standards:

- NIST P Kurven, nach FIPS 186-4 (von der NSA!)
- SecP Kurven des SEG Konsortiums nach SEC 2 Standard (genutzt in Bitcoin)
- Brainpool Kurven des BSI nach RFC 5639
- Moderne Kurven wie Curve25519 oder Curve448 nach RFC 7748
- ...

Schlüsselaustausch

Elliptischer Kurven Diffie-Hellman (ECDH)

Parameter: E elliptische Kurve der Ordnung d , P Basispunkt, $[\cdot]$ Skalarmultiplikation



Resultierender „Werkzeugkasten“

Verfahrensklassen

- Symmetrische Verschlüsselungsverfahren
- Hash-Funktionen
- Message Authentication Codes
- Asymmetrische Verschlüsselungsverfahren
- Signaturverfahren
- Schlüsseleinigungsverfahren
- ...



- 2.2.1. Erzeugen Sie mittels `openssl` ein 3072 Bit RSA Schlüsselpaar. Dieses Schlüsselpaar ist auch implizit der private Schlüssel.
- 2.2.2. Speichern Sie den öffentlichen Schlüssel aus dem privaten Schlüssel.
- 2.2.3. Signieren Sie eine Datei Ihrer Wahl mit dem privaten Schlüssel. Als Hash-Funktion wählen Sie SHA256.
- 2.2.4. Verifizieren Sie die Korrektheit der Signatur.

01 | Symmetrische Kryptographie

02 | Public-Key Kryptographie

03 | Hybride Systeme

04 | Zufallserzeugung

- Richtlinie TR02102 des BSI
- Internationale Empfehlung (z.B. NIST Special Publication 800-175B)
- Einsatz jeweils auf wenige Jahre empfohlen!
- Vergleich verschiedener Empfehlungen <https://www.keylength.com/>
- Ziel: Immer vergleichbares Sicherheitsniveau für alle eingesetzten Verfahren

- Richtlinie TR02102 des BSI
- Internationale Empfehlung (z.B. NIST Special Publication 800-175B)
- Einsatz jeweils auf wenige Jahre empfohlen!
- Vergleich verschiedener Empfehlungen <https://www.keylength.com/>
- Ziel: Immer vergleichbares Sicherheitsniveau für alle eingesetzten Verfahren

Zeitraum	Symmetrische Verfahren		Asymmetrische Verfahren		
	Blockchiffre	Hash-Funktion	RSA	DSA/DH	ECDSA/ECDH
bis 2022	128	256	2000	2000	250
ab 2023	128	256	3000	3000	250

Empfohlene Längen nach TR02102-1 in Bit

- Kryptographie bildet die Grundlage jeglicher IT-Sicherheit
- Wird die Kryptographie gebrochen, ist jeglicher Schutz hinfällig
- Wir unterscheiden symmetrische und asymmetrische Kryptographie
- Die symmetrische Kryptographie erlaubt die Realisierung aller Schutzziele der IT-Sicherheit, wenn vorher ein gemeinsames Geheimnis vereinbart wurde
- Die asymmetrische Kryptographie dient in der Praxis im wesentlichen dazu, das (authentisierte) Schlüsselaustauschproblem zu lösen
- Lösung muss aber auch vertraulich, integritätsgeschützt und authentisiert geschehen

- 2.3.1. Konsultieren Sie die Webseite des BSI hinsichtlich der Empfehlung kryptographischer Verfahren in der Richtlinie TR02102.
- (a) Welchen Hauptzweck erfüllt die Technische Richtlinie TR-02102?
 - (b) An wen richtet sich die TR-02102 hauptsächlich und warum?
 - (c) Wie häufig werden die in dieser Richtlinie ausgesprochenen Empfehlungen überprüft und aktualisiert?
 - (d) Warum erhebt das BSI keinen Anspruch auf Vollständigkeit bezüglich der in der Richtlinie aufgeführten kryptographischen Verfahren?
 - (e) Nennen Sie jeweils einen Vor- und einen Nachteil verbindlicher nationaler Empfehlungen kryptographischer Verfahren.
 - (f) Welche Blockchiffren werden empfohlen? Welche Hash-Funktionen werden empfohlen?
 - (g) Wieso gibt es unterschiedliche Empfehlungen zum selben Verfahren mit unterschiedlichen Parametern?
 - (h) Erläutern Sie, warum Empfehlungen kryptographischer Verfahren seriös immer nur wenige Jahre gelten können.

01 | Symmetrische Kryptographie

02 | Public-Key Kryptographie

03 | Hybride Systeme

04 | Zufallserzeugung

- Zufall: keine kausale Erklärung für ein einzelnes Ereignis
- Zufall wird an vielen Stellen der Informatik genutzt: Randomisierung, Simulation etc.
- Hierzu benötigt man oft zufällige Bits/Bytes
- Wir betrachten lediglich kryptographisch sicheren Zufall:
- Kryptographisch sicherer Zufall muss sich wie echter Zufall verhalten
 - ▶ Uniform (gleichverteilt)
 - ▶ Unabhängig
 - ▶ Unvorhersagbar
- Wesentliche Charakteristika sind:
 - ▶ Die Geschwindigkeit der Erzeugung (in MBit/s)
 - ▶ Die „Qualität“ des Zufalls (in ???)
- Einsatzszenarien: Schlüsselerzeugung, Cookies und Nonces, Salts, ...
- Merke: Wer den Zufall beeinflussen kann, kann prinzipiell jedes kryptographische System schwächen bzw. brechen.

- Es gibt unterschiedliche Arten der Erzeugung von Zufall
- Man unterscheidet folgende Klassen:
 - ▶ Physical True Random Number Generators (PTG)
 - ▶ Non-physical true Random Number Generators (NTG)
 - ▶ Deterministic Random Number Generators (DRG)
- PTGs und NTGs nennt man echte Zufallsgeneratoren, DRGs Pseudozufallsgeneratoren
- Der Generator produziert nach Aktivierung/Aufruf eine Anzahl an Bits/Bytes



Bild: Joachim von zur Gathen

- Beispiel: Wurf einer Münze
- Das Ergebnis des Münzwurfs (Kopf bzw. Zahl) interpretieren wir als Bit
- Qualität der erzeugten Bits misst man mittels der sog. Entropie des Erzeugungssystems
- Formal: Sei X eine diskrete Zufallsvariable mit

$$\text{prob}(X = \text{Kopf}) = p \text{ und } \text{prob}(X = \text{Zahl}) = q = 1 - p.$$

Dann ist die Entropie von X definiert als

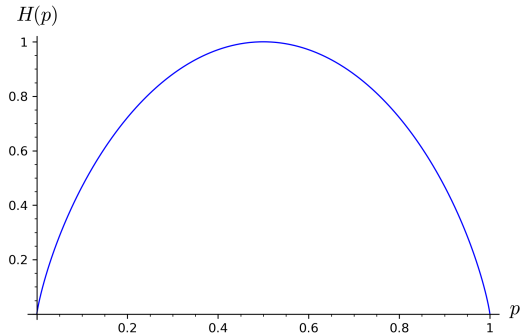
$$H(X) = p \log_2 1/p + q \log_2 1/q.$$

- Die Münze heißt fair, wenn $p = q = 1/2$.

- Sei X eine diskrete Zufallsvariable
- Mögliche Werte von X sind x_1, \dots, x_n
- Schreibe $p_i = \text{prob}(X = x_i)$
- Dann ist die Entropie von X definiert als

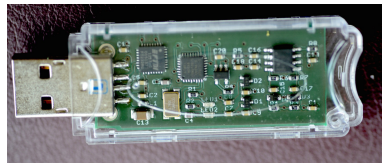
$$H(X) = H(p_1, \dots, p_n) := - \sum_{i=1}^n p_i \log_2 p_i$$

- Entropie ist maximal, wenn $p_i = p_j$ für alle $1 \leq i, j \leq n$
- Intuitiv: Entropie gibt an wie viel Zufall (in Bit!) in der Zufallsvariablen bzw. dem Erzeuger stecken!



Entropie eines Münzwurfs

- Basis ist eine physikalische Entropiequelle
- Abschätzung durch stochastisches Modell
- Beispiele:
 - ▶ Radioaktiver Zerfall
 - ▶ Rauschende Dioden
 - ▶ Sampling schneller Oszillatoren
- Die Entropiequelle generiert Rohdaten
- Sind diese analog, werden sie digitalisiert
- Anschließend werden die Rohdaten nachbearbeitet
- Mögliche Nachbearbeitung: Hashing, Anwendung von AES etc.



Hardware-Generator



Intel Prozessor mit RDRAND

Bild: IBB Ingenieurbüro Bergman (oben), Intel Corporation (unten)

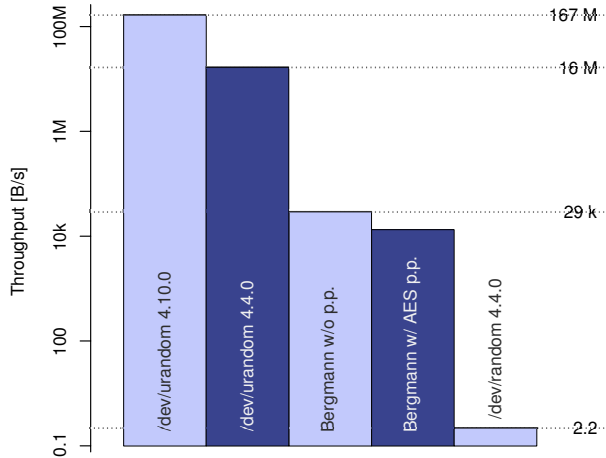
- Nutzung externer Ereignisse, um Zufall zu erzeugen
- Verarbeitung von System-Events:
 - ▶ Lese- und Schreibverhalten
 - ▶ Interrupts
 - ▶ Zeit seit Bootvorgang
 - ▶ Threat-IDs
 - ▶ Nutzereingaben
 - ▶ ...
- Jedes Ereignis hat für sich genommen nur geringe Entropie
- Diese sammelt man in einem „Entropie-Pool“

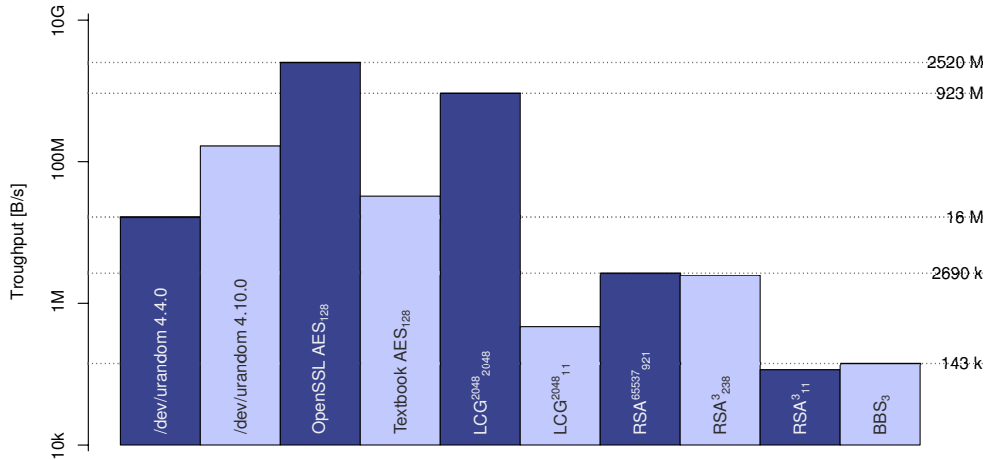
- Der Generator verfügt über einen Entropie-Pool namens „Input-Pool“
 - ▶ Komprimiert Entropie aus verschiedenen Rauschquellen (s.o.)
 - ▶ Arbeitet deterministisch
 - ▶ Verwendet eine Hash-Funktion (nach RFC7693), um die Daten zu verarbeiten
 - ▶ Entropie wird akkumuliert, indem Hardware-Ereigniszeiten und -werte gehashed werden
- Der Entropie-Pool wird als Eingabe für einen DRG genutzt (s.u.)
- Dieser sekundäre DRG liefert die eigentlichen Zufallszahlen
- Zugriff durch Lesen der Dateien `/dev/random` bzw. `/dev/urandom`

cf. Stephan Müller. Documentation and Analysis of the Linux Random Number Generator. Techn. Ber. 5.4. atsec information security GmbH for the Federal Office for Information Security, März 2023. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/LinuxRNG/LinuxRNG_EN_V4_5.pdf

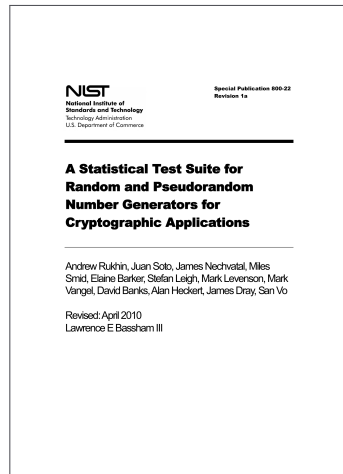
- Idee: Man nehme eine kleine Menge „echten“ Zufall
- Seed wird in einen deterministischen Algorithmus eingebracht
- Der Algorithmus erzeugt zufällig aussehende Zahlen
- Diese Pseudozufallszahlen können dann kryptographisch genutzt werden
- Beispielrealisierung (stark vereinfacht):
 - ▶ Seed ist Schlüssel von AES-CTR
 - ▶ Ausgabe von AES-CTR werden als Pseudozufallszahlen genutzt
 - ▶ korrekte Konstruktion: siehe NIST.SP.800-90A, p. 48ff
- Vorteil: Sehr effizient, keine echte Zufalls-Komponente nötig!

cf. Elaine B. Barker und John M. Kelsey. Recommendation for Random Number Generation Using Deterministic Random Bit Generators. Techn. Ber. 2015. DOI: 10.6028/NIST.SP.800-90Ar1. URL: <https://doi.org/10.6028/NIST.SP.800-90Ar1>





- Prüfung der Qualität von Zufall durch Tests
- Ausgang des Tests:
 - positiv Eingabe als zufällig klassifiziert
 - negativ Eingabe als nicht-zufällig klassifiziert
- Testsuite zur Prüfung von Bitfolgen:
 - ▶ Häufigkeits Test: Anzahl 0en vs. Anzahl 1en
 - ▶ Runs Test: Anzahl aufeinanderfolgender 0en
 - ▶ Entropie Test: (Block-)Entropie
 - ▶ ...
- Gibt ein einziger Test ein negativ-Ergebnis zurück, ist die untersuchte Folge als nicht-zufällig zurückzuweisen!



- 2.4.1. Im Jahr 2013 wurden durch die Aktivitäten des Whistle-Blowers Edward Snowden zahlreiche klassifizierte Dokukumente der Öffentlichkeit zugänglich gemacht. Ein Aspekt war das Brechen kryptographischer Zufallsgeneratoren:

[https://www.theguardian.com/world/2013/sep/05/
nsa-gchq-encryption-codes-security](https://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security)

Lesen Sie den Artikel und beantworten Sie folgende Fragen:

- (a) Wie haben Geheimdienste die Versprechungen von Internetunternehmen an ihre Kunden in Bezug auf die Unentschlüsselbarkeit ihrer Kommunikation und Daten kompromittiert?
- (b) Was sind die Hauptmethoden, die von diesen Agenturen eingesetzt werden, um die weit verbreitete Verschlüsselung im Internet zu bekämpfen?
- (c) Wie hat die NSA Einfluss auf internationale Verschlüsselungsstandards genommen?
- (d) Wie beschreibt die NSA ihre Ansicht über starke Entschlüsselungsprogramme im Kontext des Cyberspace?
- (e) Welche vier großen Dienstanbieter wurden ins Visier genommen, um Wege in ihren verschlüsselten Datenverkehr zu finden?

Gibt es Fragen?