Computer Science Fundamentals

Cryptography – Modern Methods

Technische Hochschule Rosenheim
Winter 2021/22
Prof. Dr. Jochen Schmidt

Overview

Technische Hochschule Rosenheim

- Modern (symmetric) block ciphers
  - DES
  - AES
- Asymmetric Encryption
  - Diffie-Hellman key exchange
  - RSA
  - Elliptic curve cryptography (ECC)
- Cryptographic hash functions

- More details: see literature, e.g.,

C. Paar, J. Pelzl. ***Kryptografie verständlich: Ein Lehrbuch für Studierende und Anwender***. Springer Vieweg, 2016.

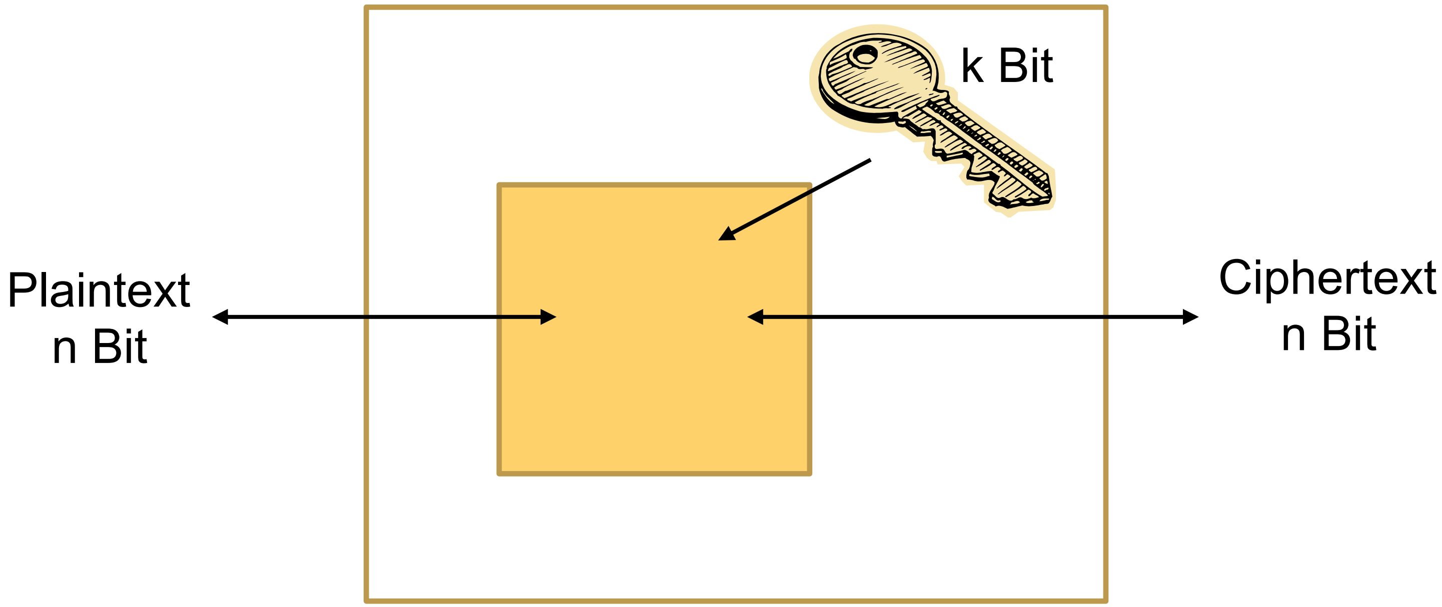
D. Wätjen. ***Kryptographie: Grundlagen, Algorithmen, Protokolle***. Springer Vieweg, 3. Aufl. 2018.

S . Rubinstein-Salzedo. ***Cryptography***. Springer Undergraduate Mathematics Series. Springer, 2018.

C. Paar, J. Pelzl. ***Understanding Cryptography: A Textbook for Students and Practitioners***. Springer, 2010.

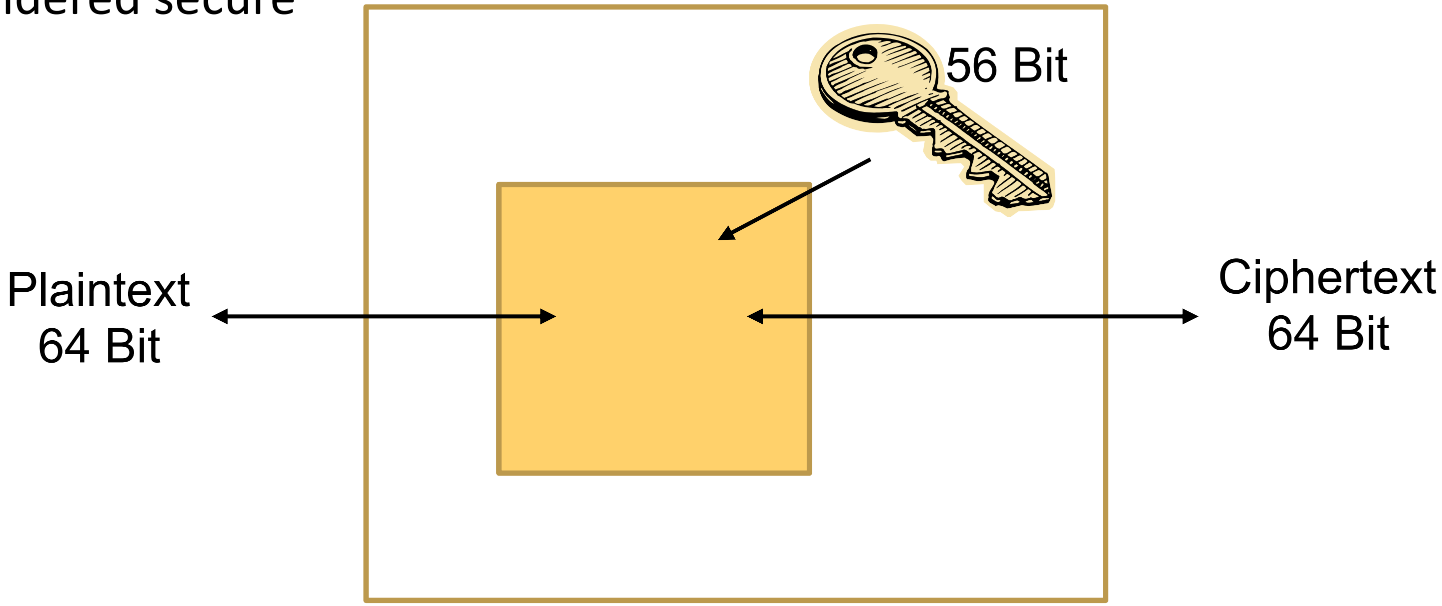
Modern Block Ciphers

- are symmetric encryption methods
- that encrypt the plaintext block-wise



DES

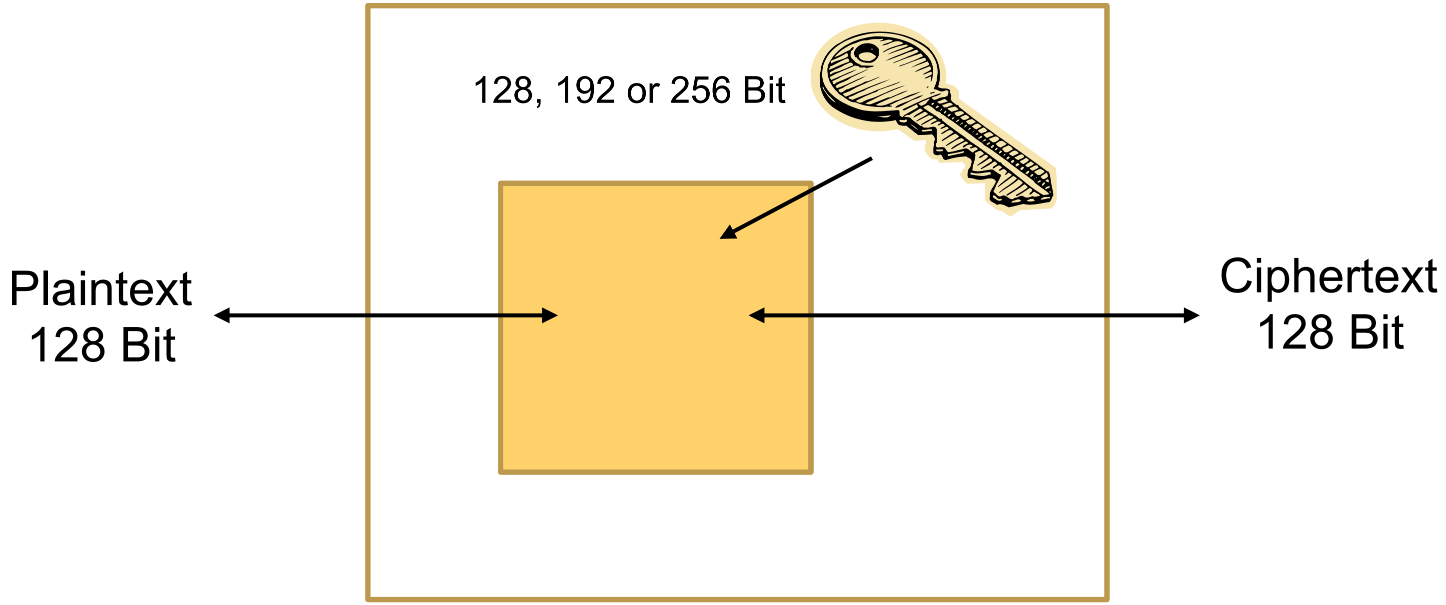
- Data Encryption Standard (DES)
- 1973-77: Development and publication
- extremely widespread since then
- no longer secure
  - 1994 broken for the first time (50 days using 12 computers)
  - 1998 using a custom chip of the Electronic Frontier Foundation (EFF), less than 3 days of computing time
  - 1999 DES-Challenge: 22:15h distributed on 100,000 PCs plus EFF-computer
- the variant 3DES („Triple DES“) is still considered secure



AES

Technische Hochschule Rosenheim

- Advanced Encryption Standard (AES)
- 1997: Call for a development competition
- 2000/2001 AES is standardized
  - using Rijndael algorithm
  - derived from the names of the Belgian developers J. Daemen and V. Rijmen
- more secure and more efficient than 3DES
  - approx. 3x faster than DES
  - approx. 9x faster than 3DES

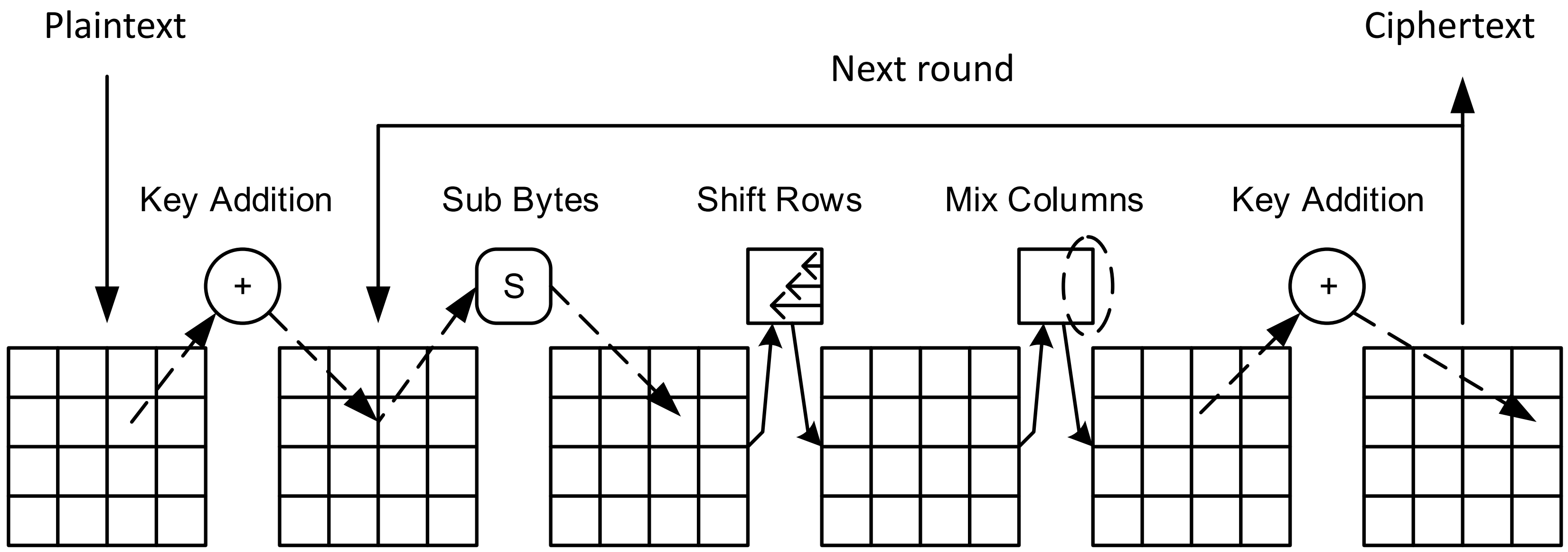


AES – Structure

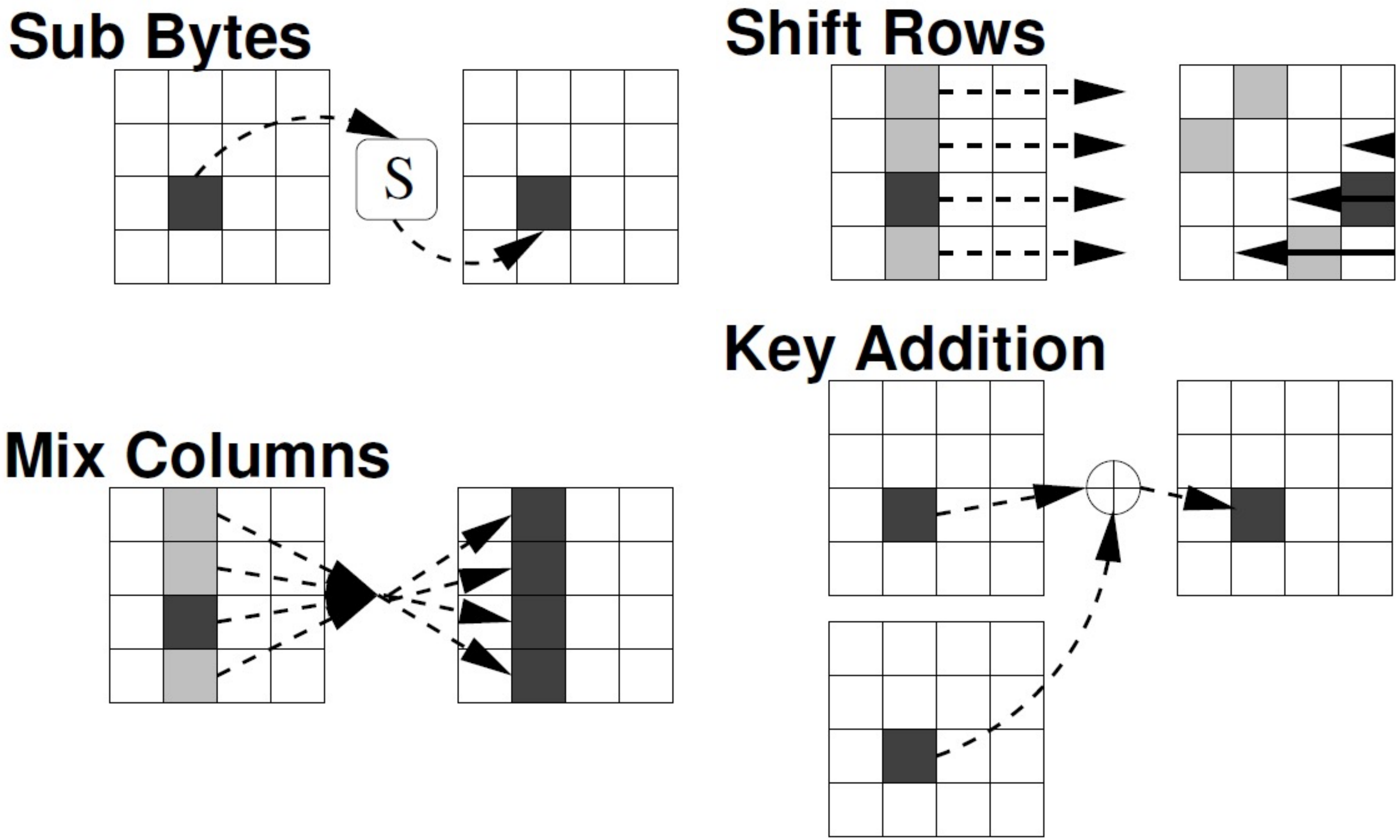
- Multi-round encryption using a substitution-permutation network
- Four basic operations
  - these are combined in each round
- a separate round key for each round
  - „Key Schedule“
  - generated from the encryption key (128-256 Bit)
    11-15 round keys (128 Bit each)
- Number of rounds r depending on block size n and key length k:

r	n = 128	n = 192	n = 256
k = 128	10	12	14
k = 192	12	12	14
k = 256	14	14	14

AES – Round Structure



AES – Basic Operations



AES – Applications

Technische Hochschule Rosenheim

- Used, e.g., in the following protocols
  - SSH (Secure Shell): remote login
  - TLS (Transport Layer Security): https
  - IPSec (Internet Protocoll Security): VPN

- WPA (Wi-Fi Protected Access), Wi-Fi encryption
  - WPA2 (since 2004): AES-128
  - WPA3 (since 2018):
    - AES-128 (personal mode)
    - AES-256 (enterprise mode)

Review: Symmetric Encryption

- Properties
  - If you can encrypt, you can also decrypt
  - Each pair of communication partners must exchange a separate common secret key

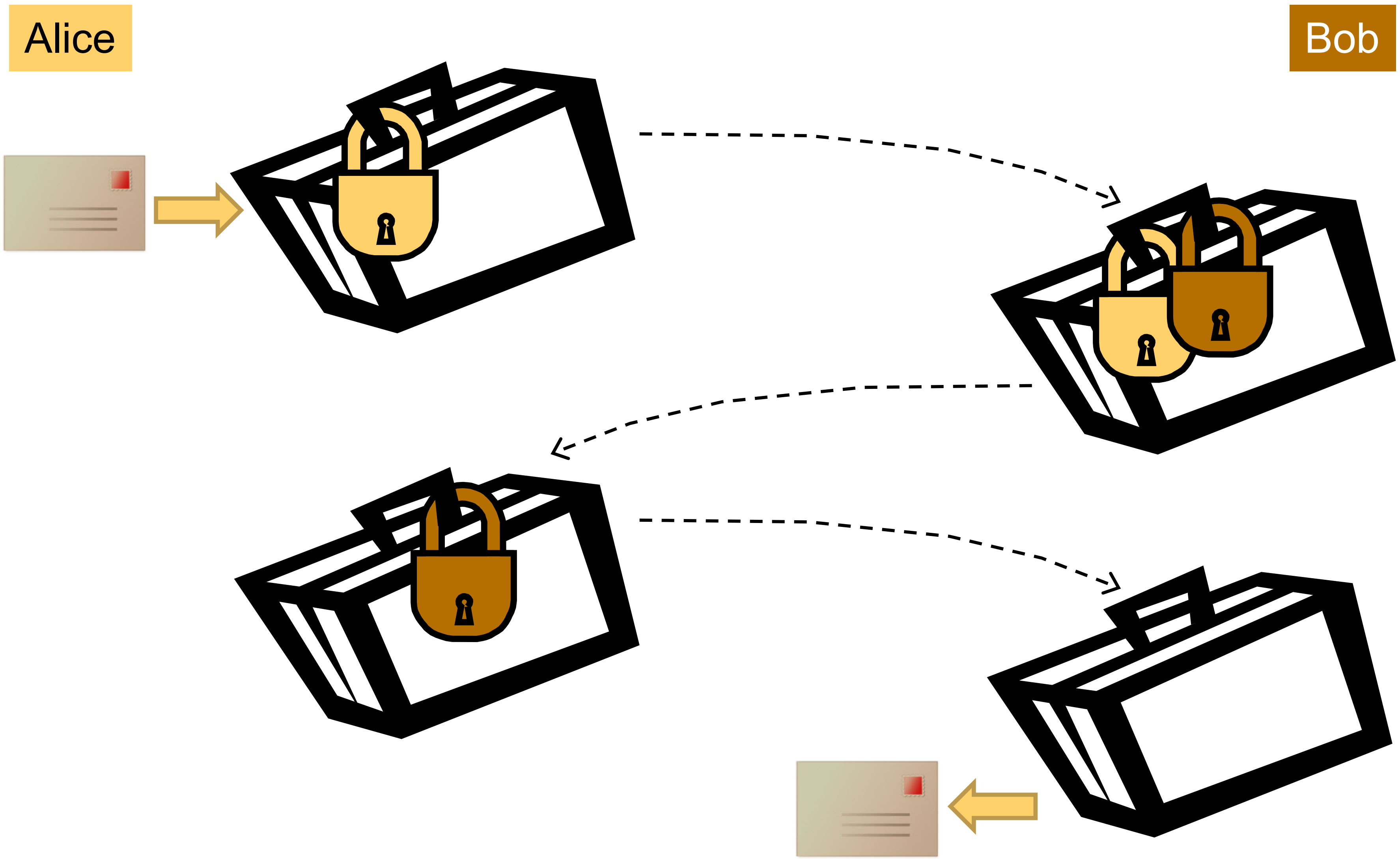
- Assessment
  - Exchange of secret key
    - Secure channel required
    - Often, however, the channel is not secure (e.g., messenger or radio connection)
  - Key management
    - Large number of keys required
    - Problem
      - What to do if sender and recipient have not met before?
      - What if a message is to be sent to several recipients at the same time?
  - Authenticity is not guaranteed (both, sender and recipient use the same key)
- Solution: Asymmetric crypto-systems

Diffie-Hellman Key Exchange

- First public key system
- By Diffie and Hellman 1976
- Also discovered in 1975 by Ellis, Cocks, Williamson at the British GCHQ, but kept secret
- Solves the problem of key exchange over an insecure channel

- Used, e.g., in the following protocols
  - SSH (Secure Shell)
  - TLS (Transport Layer Security)
  - IPSec (Internet Protocol Security)

Diffie-Hellman – Idea



Diffie-Hellman Key Exchange

Choose two public numbers

- a prime number p
- and an integer $g \in \{2, 3, \ldots, p-2\}$

1. Alice randomly chooses an integer $x_A \in \{2, 3, \ldots, p-2\}$ $\quad y_A = g^{x_A} \bmod p$

   x_A remains secret, y_A will be sent to Bob

2. Bob randomly chooses an integer $x_B \in \{2, 3, \ldots, p-2\}$ $\quad y_B = g^{x_B} \bmod p$

   x_B remains secret, y_B will be sent to Alice

3. Alice calculates $\quad k_{AB} = {y_B}^{x_A} \bmod p = (g^{x_B} \bmod p)^{x_A} \bmod p = g^{x_B x_A} \bmod p$
4. Bob calculates $\quad k_{AB} = {y_A}^{x_B} \bmod p = (g^{x_A} \bmod p)^{x_B} \bmod p = g^{x_A x_B} \bmod p$

The key used to exchange messages is k_{AB} (or will be derived therefrom)

Diffie-Hellman – Security

g should be a primitive root modulo p (*primitive Wurzel*)

- it must have order (*Ordnung*) $p-1$:

  $g^{p-1} = 1 \bmod p$ and $g^a \neq 1 \bmod p$ for all $a < p-1$

- i.e., g is a generator (*Generator, erzeugendes Element*)
  - repeated multiplication generates all elements of the field (*Körper*) except zero

- the total number of such elements is $\phi(p-1)$

- g is a primitive root if and only if

  $$g^{\frac{p-1}{r}} \neq 1 \bmod p$$

  for each prime factor r of $p-1$

Reminder: Euler's ϕ-Function

- The function's value is the number of natural numbers
  - that are smaller than n
  - and are relatively prime to n
  - ϕ(n) = | {1 ≤ x ≤ n | gcd(x, n) = 1} |

- Computation (p, q are prime numbers $p \neq q$)
  - $\phi(p) = p - 1$ all integers from 1 to $p - 1$ are relatively prime to p
  - $\phi(pq) = \phi(p)\phi(q) = (p-1)(q-1)$
  - $\phi(p^i) = p^{i-1}(p-1)$
  - $\phi(p^i q^j) = \phi(p^i)\phi(q^j) = p^{i-1}(p-1)\, q^{j-1}(q-1)$

- Examples
  - $\phi(5) = 4$
    - there are four numbers < 5 that are relatively prime to 5, namely 1, 2, 3, 4
  - $\phi(15) = \phi(3 \cdot 5) = \phi(3)\phi(5) = 2 \cdot 4 = 8$
  - $\phi(27) = \phi(3^3) = 3^2 \cdot (3-1) = 9 \cdot 2 = 18$
    - the numbers that are relatively prime to 27 are: 1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19, 20, 22, 23, 25, 26
  - $\phi(72) = \phi(2^3 \cdot 3^2) = 2^2 \cdot (2-1) \cdot 3^1 \cdot (3-1) = 4 \cdot 3 \cdot 1 \cdot 2 = 24$

Diffie-Hellman – Security

Technische Hochschule Rosenheim

- g should be a primitive root modulo p
  - it must have order $p-1$: $g^{p-1} = 1 \bmod p$ and $g^a \neq 1 \bmod p$ for all $a < p-1$
  - i.e., g is a generator, repeated multiplication generates all elements of the field except zero
  - the total number of such elements is $\phi(p-1)$

- p should be a safe prime (*sichere Primzahl*)
  - $p = 2q + 1$, where q is also prime (q is called a Sophie Germain prime)
  - otherwise, there are messages that are not changed at all: $y_A = g^{x_A} \bmod p = g$

- in this case, there exist $\phi(p-1) = \phi(2q) = \phi(2)\phi(q) = q - 1 = \frac{p-3}{2}$ primitive roots
  - the field has p elements $\longrightarrow$ probability that a randomly selected number is a primitive root is about 50%

- to be secure against attacks, we need to use numbers with length larger than 2000 Bits
  - p must be greater than $2^{2000} \approx 10^{602}$ $\longrightarrow$ prime number with 602 decimal digits!

Diffie-Hellman – Example

Choose two public numbers

- a prime number $p = 23 = 2 \cdot 11 + 1 \longrightarrow$ safe prime, there are 10 primitive roots
- and an integer $g \in \{2, 3, \ldots, 21\}$: $g = 5$

- 5 is a primitive root as
  - $5^{\frac{22}{2}} = 5^{11} = 22 \bmod 23$ and $5^{\frac{22}{11}} = 5^{2} = 25 = 2 \bmod 23$
  - $\longrightarrow$ 5 continuously multiplied by itself generates all numbers from 1 to 22: {5, 2, 10, 4, 20, 8, 17, 16, 11, 9, 22, 18, 21, 13, 19, 3, 15, 6, 7, 12, 14, 1}

- 2 is **not** a primitive root as
  - $2^{\frac{22}{2}} = 2^{11} = 1 \bmod 23$
  - $\longrightarrow$ 2 continuously multiplied by itself **does not** generate all numbers from 1 to 22: {2, 4, 8, 16, 9, 18, 13, 3, 6, 12, 1}

Diffie-Hellman – Example

$p = 23, g = 5$

1. Alice randomly chooses an integer $x_A \in \{2, 3, \ldots, 21\} \longrightarrow 3$ $\quad y_A = 5^3 \bmod 23 = 10$
   3 remains secret, 10 will be sent to Bob

2. Bob randomly chooses an integer $x_B \in \{2, 3, \ldots, p-2\} \longrightarrow 7$ $\quad y_B = 5^7 \bmod 23 = 17$
   7 remains secret, 17 will be sent to Alice

3. Alice calculates $\quad k_{AB} = 17^3 \bmod 23 = 14$
4. Bob calculates $\quad k_{AB} = 10^7 \bmod 23 = 14$

The key used to exchange messages is 14 (or will be derived therefrom)

Diffie-Hellman – Security

- Security is based on using a one-way function (*Einwegfunktion*)
- Definition One-way function $f: X \rightarrow Y$
  - $y = f(x)$ can be computed efficiently for all $x \in X$
  - x cannot be computed efficiently when you know y
  - i.e., the inverse function $x = f^{-1}(y)$ can only be determined with unrealistic effort

- Diffie-Hellman:
  - discrete exponentiation is simple

$$y_A = g^{x_A} \bmod p$$

  - inversion requires calculation of discrete logarithm $\longrightarrow$ very hard (or at least we believe so …)
  - a way to break Diffie-Hellman without discrete logarithm is not yet known

One-way Functions

Technische Hochschule Rosenheim

- whether one-way functions exist at all is unknown!
  - a proof of this would include the proof that P ≠ NP (the reverse is not true) (more on P/NP ⟶ course Theoretical Computer Science next semester)

- Examples of functions that may meet the conditions
  - discrete exponentiation
  - (cryptographic) hash functions
    - MD5 (Message Digest, 128 Bit length)
    - SHA-1 (Secure Hash Algorithm, 160 Bit)
    - SHA-2/SHA-3 (224 to 512 Bit)
    - typical application: Encryption of passwords
    - MD5 and SHA-1 are no longer considered secure
  - Primes
    - Multiplication is easy
    - Factorization is difficult

Trapdoor Functions (*Falltürfunktionen*)

Technische Hochschule Rosenheim

- Special case of using one-way functions
  - Using additional information (a key)
  - the inverse functions can be computed efficiently

- Example: Integer Factorization
  - easy, if one of the two factors is known
  - ⟶ RSA

RSA Encryption

- 1978 developed by R. **R**ivest, A. **S**hamir, and L. **A**dleman

- Based on the assumption that
  - factorization of large numbers (decomposition into prime factors) is very time-consuming
  - generating such a large number by multiplying two prime numbers is very easy

RSA – Key Generation

1. Choose two large prime numbers p and q

2. Determine RSA modulus $\qquad n = pq$
   - n should have at least 600 (decimal) digit/2048 bits

3. Calculate Euler's function of n: $\qquad \phi(n) = (p-1)(q-1)$

4. Choose an encryption exponent c with
   - $1 < c < \phi(n)$
   - c has no common divisor with Euler's function: $\qquad \gcd\big(c, \phi(n)\big) = 1$

5. Calculate decryption exponent d as modular inverse of c wrt $\phi(n)$: $\qquad cd \bmod \phi(n) = 1$
   - e.g., using the extended Euclidean algorithm

(c, n) form the public key, d is the private key

RSA – Encryption & Decryption

„Alice wants to send message to Bob “

- Look up Bob's public key in Key Directory: $(c_{\mathrm{Bob}}, n_{\mathrm{Bob}})$
- Split the message into chunks of equal size $x_1, x_2, x_3, \ldots$ (all $x_i < n_{\mathrm{Bob}}$)
- Encrypt chunks $y_i = x_i^{\,c_{\mathrm{Bob}}} \bmod n_{\mathrm{Bob}}$
- Transmit y_i
- Decryption by Bob using the private key d_{Bob} known only to him

$$x_i = y_i^{\,d_{\mathrm{Bob}}} \bmod n_{\mathrm{Bob}}$$

RSA – (Main Part of the) Proof

- Euler‘s theorem:

$$a^{\phi(n)} \bmod n = 1 \qquad \text{if } \gcd(a, n) = 1$$

- RSA

$$x^{cd} \bmod n$$

$$cd \bmod \phi(n) = 1 \qquad \Rightarrow cd = 1 + k\phi(n)$$

$$x^{cd} \bmod n =$$
$$x^{1 + k\phi(n)} \bmod n =$$
$$x\, x^{k\phi(n)} \bmod n =$$
$$x\, (x^{\phi(n)})^{k} \bmod n = x$$

Note: There is a small amount of numbers where this proof is invalid (namely multiples of the prime factors).
Decryption still works for these cases; a proof can be given based on Fermat’s little theorem and is omitted here.

RSA – Example

Alice wants to send an encrypted message to Bob

- We will use only the 26 Latin letters, in a decimal representation:
  - Each letter is assigned its position in the alphabet (A $\longrightarrow$ 1, …, Z $\longrightarrow$ 26)
- Split the message into chunks, containing a single letter each

RSA – Key Generation Example

1. Choose two prime numbers $p = 5$ and $q = 11$

2. Determine RSA modulus $n = 5 \cdot 11 = 55$

3. Calculate Euler's function of n: $\phi(n) = (p-1)(q-1) = 4 \cdot 10 = 40$

4. Choose an encryption exponent c with
   - $1 < c < 40$
   - c has no common divisor with Euler's function: $\gcd(c, 40\ ) = 1$ $\longrightarrow$ e.g., $c = 3$

5. Calculate decryption exponent d as modular inverse of c wrt $\phi(n)$: $3 \cdot d \bmod 40 = 1$
   - e.g., extended Euclidean algorithm (or, as shown here, Euler's theorem):
   - $d = c^{-1} = c^{\phi(\phi(n))-1} \bmod \phi(n)$
   - $d = 3^{\phi(40)-1} \bmod 40 = 3^{15} \bmod 40 = 27$
   - $\phi(40) = \phi(2^3 \cdot 5) = 2^2 \cdot 1 \cdot 4 = 16$

RSA – Encryption & Decryption Example

Encryption of the text `CLEO`

- Determine numerical representation: 3, 12, 5, 15
- Encrypt using public key $c = 3, n = 55$
  - `C`: $y_1 = 3^3 \bmod 55 = 27$
  - `L`: $y_2 = 12^3 \bmod 55 = 1728 \bmod 55 = 23$
  - `E`: $y_3 = 5^3 \bmod 55 = 125 \bmod 55 = 15$
  - `O`: $y_4 = 15^3 \bmod 55 = 3375 \bmod 55 = 20$
- Send 27, 23, 15, 20

Decryption of 27, 23, 15, 20 using the receiver's private key $d = 27$

$x_1 = 27^{27} \bmod 55 = 3 \longrightarrow$ `C`

$x_2 = 23^{27} \bmod 55 = 12 \longrightarrow$ `L`

$x_3 = 15^{27} \bmod 55 = 5 \longrightarrow$ `E`

$x_4 = 20^{27} \bmod 55 = 15 \longrightarrow$ `O`

RSA – Notes

Technische Hochschule Rosenheim

- approx. 1000x slower than common symmetric encryption methods (e.g., AES)

- Therefore: Used as a hybrid method
  - RSA to encrypt a shared (symmetric) key
  - Transmission of the encrypted symmetric key
  - Actual data exchange using symmetric encryption

- Application examples
  - Protocols SSH, TLS (in https)
  - RFID-Chip in German passports

RSA Factoring Challenge

- Competition initiated by the company RSA Security
  - idea: show security of RSA encryption
  - started 18.3.1991
  - discontinued 2007

- given: Integer that was calculated as product of exactly two prime numbers
- wanted: the two prime factors

RSA Factoring Challenge

RSA Number	#digits decimal	#digits binary	price money	date of factorization	notes
RSA-100	100	330	**$1.000**	1.4.1991	Lenstra, Uni Amsterdam, a few days
RSA-110	110	364	**$4.429**	14.4.1992	Lenstra, Uni Amsterdam, 1 month
RSA-155	155	512	**$9.383**	22.8.1999	te Riele et al., CWI Amsterdam, 8000 MIPS years
RSA-576	174	576	**$10.000**	3.12.2003	Franke et al., Uni Bonn
RSA-220	220	729	-	13.5.2016	S. Bai, P. Gaudry, A. Kruppa, E. Thomé, P. Zimmermann, Australian National University, ~370 CPU years (Xeon E5-2650, 2GHz)
RSA-230	230	762	-	15.8.2018	Samuel S. Gross, Noblis Inc.
RSA-640	193	640	**$20.000**	2.11.2005	Franke et al., Uni Bonn, 5 months on 80 AMD Opteron 2.2 GHz
RSA-704	212	704	$30.000	2.7.2012	S. Bai, E. Thomé, P. Zimmermann, Australian National University, ~14 months
RSA-768	232	768	$50.000	12.12.2009	Kleinjung (Lausanne) et al. 2000 CPU years (single-core AMD Opteron 2.2 GHz) http://eprint.iacr.org/2010/006.pdf
RSA-250	250	829	-	28.2.2020	F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé, P. Zimmermann (INRIA, F), ~2700 CPU-core years (Intel Xeon, 2.1GHz)
RSA-1024	309	1024	$100.000	-	~1000x harder than RSA-768
RSA-1536	463	1536	$150.000	-	
RSA-2048	617	2048	$200.000	-	

Public Key Cryptography – Man-in-the-Middle Attack

- With public keys: Use central system for key management (key server)
- Susceptible to man-in-the-middle attacks
  - Attacker breaks into key server
  - Attacker returns her own key instead of the real one when someone requests a public key
  - Attacker
    - intercepts sent message,
    - decrypts it using her own key,
    - and encrypts a changed message using the original receiver's public key
  - Encrypted message is forwarded to original receiver
    - who has no clue there was an attack
    - and assumes the message coming from the original sender
- Possible remedy: Digital signatures and the Web of Trust

Web of Trust – Idea

- Authenticity of public keys is ensured by a network of mutual confirmations

- Certificate
  - = Digital signature on a key
  - Submitted by a person who also participates in the Web of Trust…
  - … after this person has assured himself of the identity of the key holder

- Keys can also be authenticated by signatures of Certification Authorities (CA)
  - they act as a trusted 3rd party

Web of Trust – Example

Technische Hochschule Rosenheim

- Alice
  - generates a key pair (public and private key)
  - sends public key to key server
- Bob wants to communicate with Alice in encrypted form
  - gets Alice’s public key from the key server
  - asks Alice about details of her public key (e.g., personal contact: meeting, telephone, …)
  - compares the information with that on the key server
  - digitally signs Alice's public key if there is a match
  - sends this signature back to the key server
- Karl wants to communicate with Alice in encrypted form
  - gets Alice's public key from the key server
  - notes that Bob has already checked and signed the key
  - if Karl trusts Bob, he will trust Alice's key
    - and does not have to perform an additional check of Alice’s key

Digital Signature

Ensuring authenticity – Alice sends signed message to Bob

- Calculation of an intermediate result s
  - from the original message x to be conveyed
  - using Alice's own private key d_{Alice}

  $s = x^{d_{Alice}} \bmod n_{Alice}$

- Encryption of the intermediate result s
  - with the public key of Bob c_{Bob}

  $y = s^{c_{Bob}} \bmod n_{Bob}$

Digital Signature

- After receiving the signed message y
  - Bob applies his private key to decrypt
  - He obtains the intermediate result s

$$s = y^{d_{Bob}} \bmod n_{Bob}$$

- Bob looks up Alice's public key in the key directory and applies it to s

$$x = s^{c_{Alice}} \bmod n_{Alice}$$

- „Reasonable“ result for x = Bob can be sure that the message comes from the correct sender

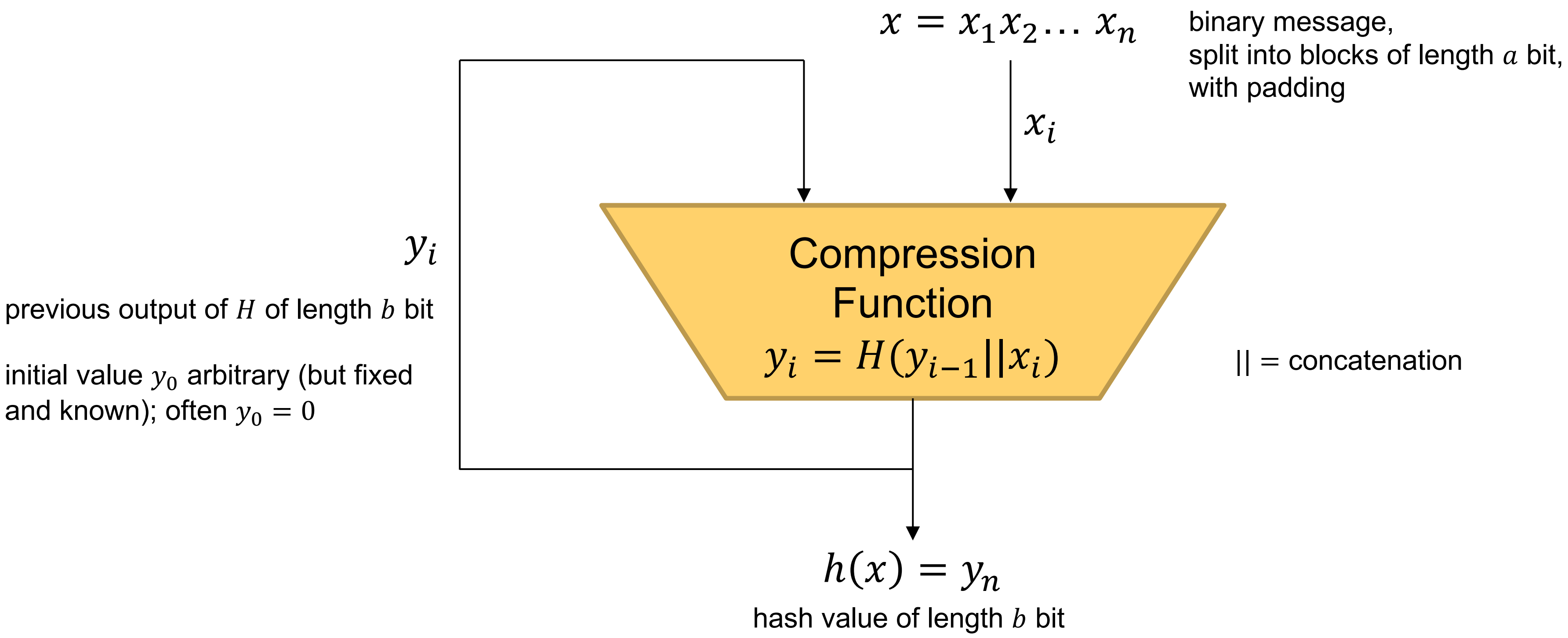
- In real applications checking for “reasonable” is not feasible:
  - instead of signing the whole message x: Generate a fixed-length hash value from the message
  - sign the hash value
  - encrypt message as usual (or: send plain text message with separately attached digital signature)

Cryptographic Hash Functions

- Applications
  - Ensuring data integrity (have data been manipulated?)
  - Ensuring authenticity: Digital signatures
    - to ”concentrate" a message to a fixed length („message digest“ = hash value)
    - this also allows integrity checking
  - Storage of passwords

- Definition: Cryptographic hash function h
  - h is a (not injective) one-way function
  - the hash value (or just “hash”) $h(x)$ of a message x is easy to compute
  - weak and strong collision resistance

Cryptographic Hash Functions – Basic Principle



This is the Merkle-Damgård construction

- widely used, e.g., MD5, SHA-1, SHA-2 (as SHA-256, SHA-512)
- SHA-3, BLAKE2, BLAKE3 use a different scheme

Cryptographic Hash Functions – Classes

- Compression functions based on block ciphers (like, e.g., AES)
  - e.g., Whirlpool
  - not very common

- Custom-made compression functions
  - based on logical bit operations (AND, OR, XOR, …)
  - function blocks similar to block ciphers
  - e.g., MD5, SHA

Cryptographic Hash Functions – Example SHA-1

- SHA = Secure Hash-Algorithm
  - Message blocks x_i 512 bit
  - Hash value 160 bit

- for each block: 4 stages, 20 rounds each
  - efficient: only AND, OR, XOR, NOT, shift, addition
  - initial values for A-E (32 bit each) are fixed:

    A = 67452301_{16},
    B = EFCDAB89_{16},
    C = 98BADCFE_{16},
    D = 10325476_{16},
    E = C3D2E1F0_{16}

- the input words W_t are derived from x_i

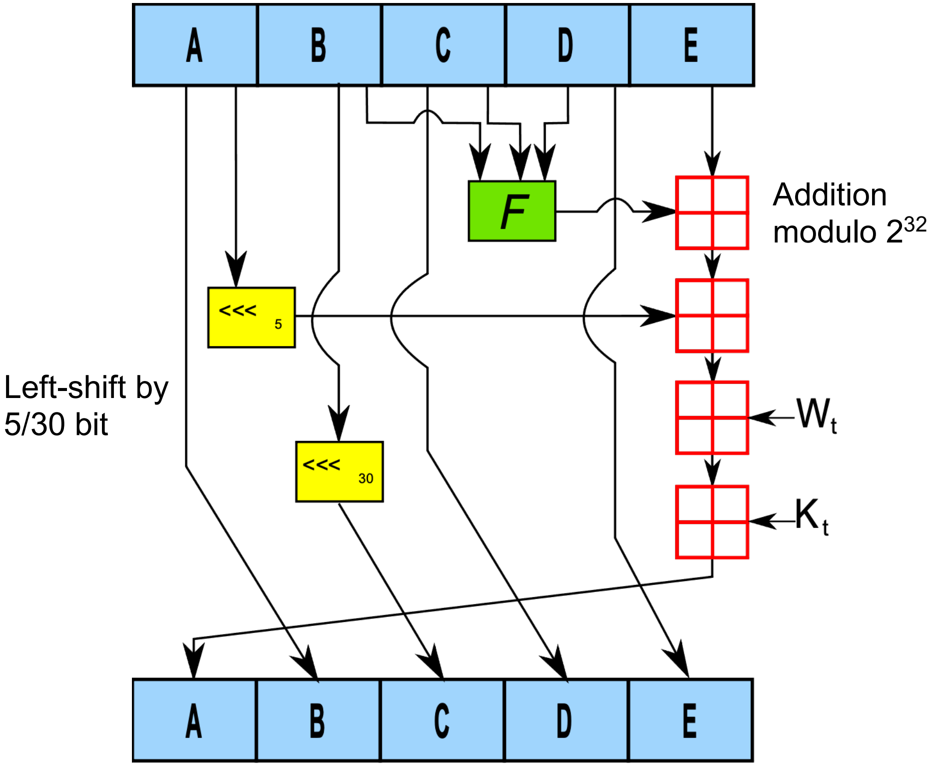


Image: H2g2bob, SHA-1, Wikimedia Commons, CC BY-SA 2.5

Stage t	Round j	Constants K_t	Functions $F(B,C,D)$
1	0…19	$K_1 = \text{5A827999}_{16}$	$F_1(B,C,D) = (B \wedge C) \vee (\bar{B} \wedge D)$
2	20…39	$K_2 = \text{6ED9EBA1}_{16}$	$F_2(B,C,D) = B \oplus C \oplus D$
3	40…59	$K_3 = \text{8F1BBCDC}_{16}$	$F_3(B,C,D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
4	60…79	$K_4 = \text{CA62C1D6}_{16}$	$F_4(B,C,D) = B \oplus C \oplus D$

Cryptographic Hash Functions – Requirements

Technische Hochschule Rosenheim

- One-way property (pre-image resistance)
  - given: $y = h(x)$
  - determining $x = h^{-1}(y)$ efficiently is not possible

- Weak collision resistance (second pre-image)
  - given: Message x_1 and its hash value $y_1 = h(x_1)$
  - finding a message $x_2 \neq x_1$ having the same hash value $h(x_2) = h(x_1)$ efficiently is not possible

- Strong collision resistance
  - finding pairs of messages $x_2 \neq x_1$ having the same hash value $h(x_2) = h(x_1)$ efficiently is not possible
  - in contrast to weak collision resistance, an attacker can freely choose both messages here

Cryptographic Hash Functions – Strong Collision Resistance

Technische Hochschule Rosenheim

- Attacker Oscar generates two messages, e.g.
  - $x_1 =$ „Transfer 10€ to Oscar’s account“
  - $x_2 =$ „Transfer 10,000€ to Oscar’s account“
- now he changes both at invisible places (e.g., add spaces or replace them with tabs),
  - so that the semantics are preserved
  - until $h(x_2) = h(x_1)$
  - (for, e.g., 32 positions that can be changed in a message this results in 2^{32} versions of the same message with 2^{32} hash values)
- The attack: Oscar
  - gets Bob to sign the message x_1
  - intercepts the transfer to the legitimate recipient Alice and
  - exchanges the message x_1 by x_2

Cryptographic Hash Functions – Strong Collision Resistance

Technische Hochschule Rosenheim

- How difficult is it to find such collisions?
- Weak collision resistance
  - for 80 bit hash: Collision latest after 2^{80} checked messages
- Strong collision resistance
  - for 80 bit hash: checking 2^{40} message is sufficient!

- Known as Birthday Attack
  - how many people must be gathered for the probability of 2 people having a birthday on the same day to be greater than 50%?
  - number of possible values: 365
  - it can be shown: 23 persons are sufficient
  - 40 persons are sufficient for a probability greater than 90%

Cryptographic Hash Functions – Birthday Attack

- given: hash function h,
  hashes of length n bit $\longrightarrow 2^n$ possible hash values

- We have to compute approx. $t \approx 2^{(n+1)/2}\sqrt{\ln\frac{1}{1-p}}$ hash values,
  with $p =$ desired probability for at least one collision

- For 50% probability and 80 bit: $t \approx 2^{81/2}\sqrt{\ln\frac{1}{0{,}5}} \approx 2^{40{,}2}$

- For 90% probability and 80 bit: $t \approx 2^{81/2}\sqrt{\ln\frac{1}{0{,}1}} \approx 2^{41{,}1}$

$\longrightarrow$ effectively only $\frac{n}{2}$ bit security with n bit hash

Cryptographic Hash Functions – Passwords

Example: Storage of passwords

User	Password
user1	12345
user2	abc123
user3	abc123

↓

Password file

User	Hash (SHA-256)
user1	5994471abb01112afcc18159f6cc74b4f511b99806da59b3caf5a9c173cacfc5
user2	6ca13d52ca70c883e0f0bb101e425a89e8624de51db2d2392593af6a84118090
user3	6ca13d52ca70c883e0f0bb101e425a89e8624de51db2d2392593af6a84118090

Problem:
- Same password – same hash
- Dictionary attacks with tables containing plaintext–hash are easy to perform

Cryptographic Hash Functions – Passwords

Example: Storing passwords with salt

User	Password
user1	12345
user2	abc123
user3	abc123

User	Salt
user1	d20f49dee7ec062e
user2	0c8ed18a475ae634
user3	6209501594ef2f7c

randomly generated by the system when the user is created, here 8 bytes

Concatenation password/salt
12345d20f49dee7ec062e
abc1230c8ed18a475ae634
abc1236209501594ef2f7c

Hashing

User	Salt	Hash (SHA-256)
user1	d20f49dee7ec062e	59363006faeb0c6cd62f6e67a0f6f1f1d66c43363af7772463697bf0a22aa37a
user2	0c8ed18a475ae634	d012fd6850fb028826b6e4368a10c794892c50b8a2cbdaee95691834f8aa8b3c
user3	6209501594ef2f7c	016cdfec92bba0423bbd20c236521cfefd82783d0cfaaa7c1db1db2599f3f6f1

Password file

Elliptic Curve Cryptography (ECC)

- Independently discovered by N. Koblitz (1987) and V. Miller (1987)
- Public-Key cipher
- Established as a standard cipher (e.g., IPsec, TLS)

- Advantage over RSA:
  - Basically, only the calculation of the discrete logarithm remains as a possible attack
    - this is less efficient with ECC than with RSA
  - Therefore, higher security even with small key lengths
    1024 bit RSA ≈ 160 bit ECC
    3072 bit RSA ≈ 256 bit ECC

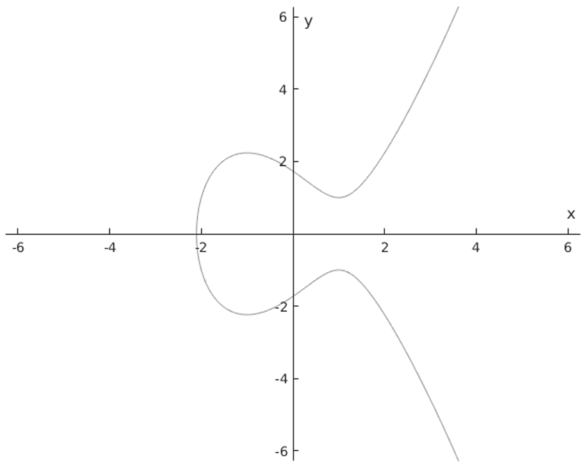
Elliptic Curve – Definition

Technische Hochschule Rosenheim

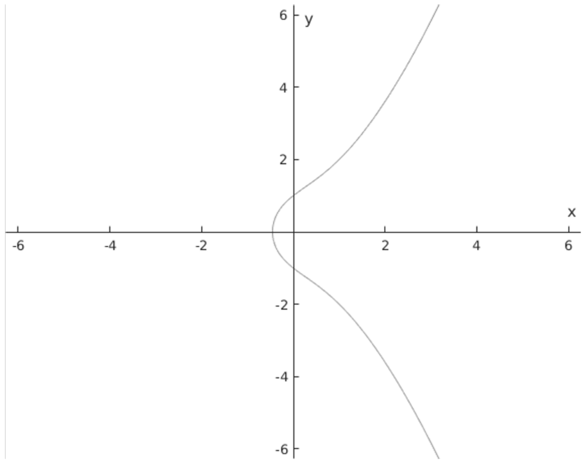
Elliptic curve ≠ ellipse!

Elliptic curve: All points (x, y) that satisfy the following equation: $y^2 = x^3 + ax + b$
with a, b, x, y elements of an arbitrary field (with at least 4 elements) and $4a^3 + 27b^2 \neq 0$

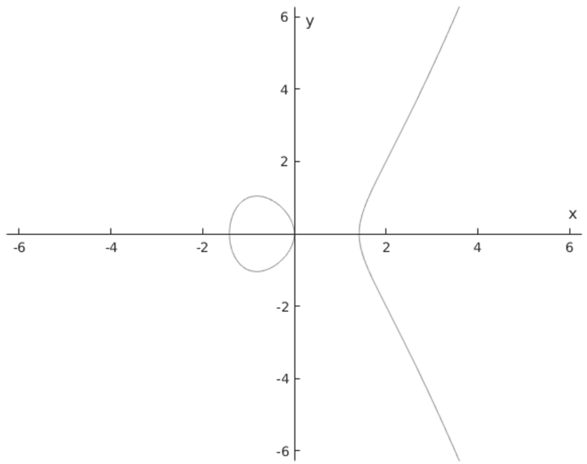
Examples (Plots over the field of real numbers):



$y^2 = x^3 - 3x + 3$



$y^2 = x^3 + 2x + 1$



$y^2 = x^3 - 2x$

Cryptography: use finite field $\mathbb{F}_q$ with $q = p^i$ elements, p prime, $i \in \{1, 2, 3, ...\}$ ($i = 1 \longrightarrow$ calculations modulo p)

ECC – How to Perform Calculations?

- Instead of „normal“ numbers: use points $P = (x, y)$ with $x, y \in \mathbb{F}_q$, that satisfy the equation (we'll use $q = p$ prime)
- Define a commutative group (algebraically closed, associative, neutral element, inverse)

- **Operation** „+“: $P_3 = P_1 + P_2 = (x_1, y_1) + (x_2, y_2)$ (the "+" symbol is arbitrary!), with

$$x_3 = s^2 - x_1 - x_2 \quad \mod p$$
$$y_3 = s(x_1 - x_3) - y_1 \quad \mod p$$

$$y^2 = x^3 + ax + b$$

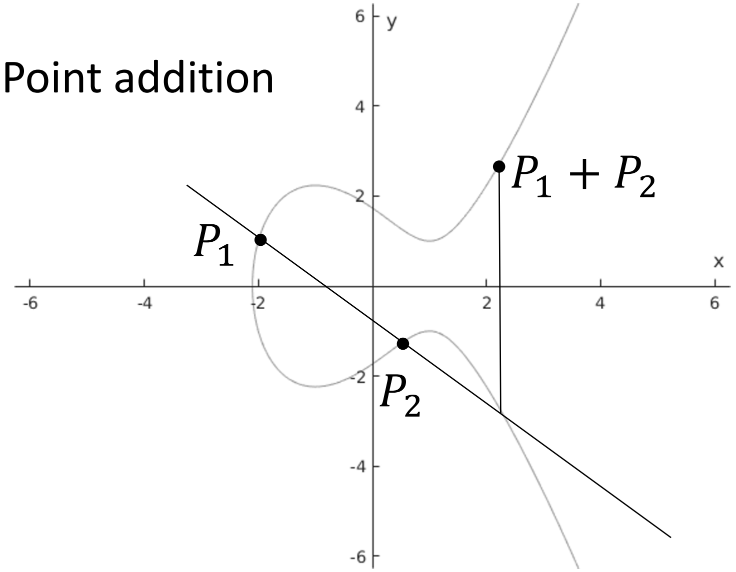
and $s = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \mod p & \text{if } P_1 \neq P_2, x_1 \neq x_2 \text{ (point addition)} \\ \frac{3x_1^2 + a}{2y_1} \mod p & \text{if } P_1 = P_2, y_1 \neq 0 \text{ (point doubling)} \end{cases}$

- **neutral element** $\mathcal{O}$ with $P + \mathcal{O} = \mathcal{O} + P = P$
(an infinitely distant point in the direction of the y-axis)

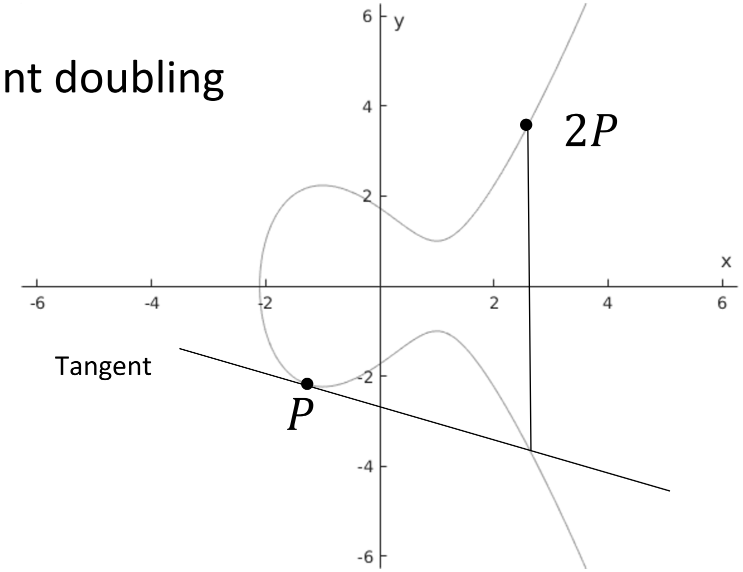
- **Inverse** to $P = (x, y)$ is $-P = (x, -y)$

ECC – Visualization of Operation „+“

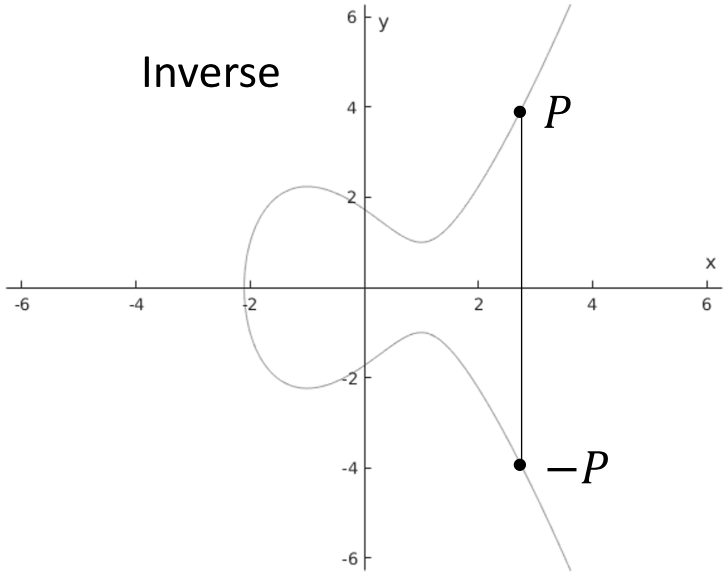
Point addition



Point doubling



Inverse



ECC – Which Points are on the Curve?

- In $\mathbb{F}_p$ (p prime): calculations mod p!
- Insert all possible x values in $y^2 = x^3 + ax + b$
- Equation is satisfied exactly for the quadratic residues (*quadratische Reste*) R_p
  - these are numbers $c = x^3 + ax + b$ for which $c^{\frac{p-1}{2}} \bmod p = 1$ holds
  - in addition, for $c = 0$ the point $(x, 0)$ is on the curve
- For all elements of R_p: Calculate the square root (mod p!)

- Calculation of the root is easy if $4 \mid (p+1)$
  - For $y^2 \bmod p = c$ the solutions are: $y_1 = c^{\frac{p+1}{4}}$ and $y_2 = p - y_1$

- In other cases: probabilistic algorithm, see [Wätjen 2008, Algorithmus 9.1]

- Estimate of number of elements N of the curve: $p + 1 - 2\sqrt{p} \le N \le p + 1 + 2\sqrt{p}$

  i.e., a curve consists of approx. p elements

ECC – Example: $y^2 = x^3 + 3x + 9$ in $\mathbb{F}_{11}$

- Check for all numbers $x \in \{0, 1, 2, \dots, 10\}$ if y^2 are quadratic residues (i.e. in R_{11})
- Determine the square root to obtain y

x	$y^2 = x^3 + 3x + 9 \bmod 11$	y^2 in R_{11}?	y
0	9	✓	3, 8
1	2	–	
2	1	✓	1, 10
3	1	✓	1, 10
4	8	–	
5	6	–	
6	1	✓	1, 10
7	10	–	
8	6	–	
9	6	–	
10	5	✓	4, 7

The commutative group therefore contains a total of 11 points:
The 10 from the table and the point $\mathcal{O}$

Example from Wätjen, 2008

ECC-Diffie-Hellman (ECDH)

Choose (public)

- a prime number p
- an elliptic curve $E\colon\ y^2 = x^3 + ax + b$ with N elements
- an element $g = (x_g,\ y_g) \in E$ (to be secure, it must be a primitive (= generating) element)

1. Alice randomly chooses a number $x_A \in \{2, 3, \dots, N-1\}$, adds g x_A times: $y_A = g + g + \cdots + g = x_A g$

   x_A remains secret, y_A will be sent to Bob

2. Bob randomly chooses a number $x_B \in \{2, 3, \dots, N-1\}$, adds g x_B times: $y_B = g + g + \cdots + g = x_B g$

   x_B remains secret, y_B will be sent to Alice

3. Alice calculates $k_{AB} = x_A y_B \; = x_A x_B g$
4. Bob calculates $k_{AB} = x_B y_A \; = x_B x_A g$

Since calculations are performed in a commutative group, the result is identical.
The key used to exchange messages is k_{AB} (or rather derived therefrom, e.g., from the x-value using a hash function)

ECC-Diffie-Hellman (ECDH) – Example

$p = 11, y^2 = x^3 + 3x + 9, g = (0, 8)$

1. Alice randomly chooses a number $x_A \in \{2, 3, \dots, 10\} \longrightarrow 3$

   $$y_A = 3 \cdot (0, 8) = (0, 8) + (0, 8) + (0, 8) = (3,10) + (0,8) = (6, 10)$$

   3 remains secret, $(6, 10)$ will be sent to Bob

2. Bob randomly chooses a number $x_B \in \{2, 3, \dots, 10\} \longrightarrow 2$

   $$y_B = 2 \cdot (0, 8) = (0, 8) + (0, 8) = (3, 10)$$

   2 remains secret, $(3, 10)$ will be sent to Alice

3. Alice calculates $k_{AB} = 3 \cdot (3, 10) = (2, 10)$
4. Bob calculates $k_{AB} = 2 \cdot (6, 10) = (2, 10)$

The key used to exchange messages is derived from $(2, 10)$, e.g., from the x-value using a hash function

ECDH – Primitive Elements

- ECDH works as presented for any public element g
- To be secure, g must be a primitive element (generator)
  - i.e., g added to itself gets zero only after all group elements have been created
  - This is the same as the primitive root criterion for standard DH
  - except that there we use multiplication and a finite field created modulo a prime (neutral element = 1), group order (= #elements , here of the multiplicative group) is $p - 1$
  - here we use point addition on the curve (neutral element = $\mathcal{O}$), group order is #points on curve + 1

- For group order N and a point g on the curve
  - determine all prime factors r of N
  - if $\frac{N}{r} g$ (= g added to itself $\frac{N}{r}$ times) is not zero ($\mathcal{O}$) for all factors r, g is primitive
  - note: $Ng = \mathcal{O}$ is always true

- In the previous example, the group contains $N = 11$ elements
  - therefore, all elements $\neq \mathcal{O}$ are primitive
  - sequence for (0, 8): (0, 8), (3, 10), (6, 10), (10, 7), (2, 1), (2, 10), (10, 4), (6, 1), (3, 1), (0, 3), $\mathcal{O}$
- It is not so easy in practice to find good curves
  - some may not have any generating elements at all

ECC – Notes

- To break the cipher, x_A or x_B must be determined
  - These are the number of jumps on the curve from the start to the end point
  - This corresponds to the discrete logarithm; the notation using "+" just looks unusual

- This way, other encryption methods can also be converted to elliptic curves: Perform calculations with points of the curve instead of "normal" numbers

- Security also depends on the curve used
  Example: Curve 25519 (Bernstein, 2005)
  - used for Diffie-Hellman
  - $p = 2^{255} - 19, y^2 = x^3 + 486662x^2 + x, \quad g = (9, y)$
  - (an isomorphic curve exists for this curve as a so-called *short Weierstrass Equation*, which then has the form $y^2 = x^3 + ax + b$.)

Security Level of Ciphers

An algorithm has a **security level of n bit** if the best known attack requires 2^n steps.

- typical **symmetric** algorithms with key length n have a security level of n bit
- typical **hash functions** with n bit have
  - a collision security level of $n/2$ bit (birthday attack)
  - a pre-image security level of n bit
- **asymmetric** algorithms are difficult to assess

Algorithm	Example	Security Level (Bit)			
		80 Bit	128 Bit	192 Bit	256 Bit
symmetric	AES	80	128	192	256
factorization	RSA	1024	3072	7680	15360
discrete logarithm	Diffie-Hellman	1024	3072	7680	15360
elliptic curves	ECDH	160	256	384	512

In a Nutshell – What should you use at the moment?

- Hashing:
  - SHA-2 (as SHA-256, SHA-384 or SHA-512)
  - SHA-3 (as SHA3-256, SHA3-384 or SHA3-512)

- Symmetric methods:
  - AES-256,
  - in GCM (Galois/Counter Mode)

- Asymmetric methods
  - RSA with 2048 Bit, for medium-term security 3072 Bit
  - ECC with 256 Bit (e.g., Curve 25519)

see also:

Cryptographic Mechanisms: Recommendations and Key Lengths. BSI – Technical Guideline

https://www.bsi.bund.de/EN/Service-Navi/Publications/TechnicalGuidelines/tr02102/tr02102_node.html

Kryptographische Verfahren: Empfehlungen und Schlüssellängen. BSI – Technische Richtlinie.

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr02102/tr02102_node.html

The Future

- Quantum computing
  - almost all public-key methods break down (in particular, the ones presently used in practice)
    - Shor's algorithm (1994): efficient prime factorization and discrete logarithms
    - primarily concerns key exchange and digital signatures
  - most symmetric methods (especially AES) remain secure
  - most cryptographic hashing methods remain secure

- Post-quantum cryptography required
  see, e.g.
  - https://pqcrypto.org/
  - https://en.wikipedia.org/wiki/Post-quantum_cryptography