TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Security and Cryptography 1

Stefan Köpsell, Thorsten Strufe

Module 7: Keys and asymmetric encryption

Disclaimer: large parts from Dan Boneh, Mark Manulis, Stefan Katzenbeisser, William Stallings

Dresden, WS 17/18

Reprise from the last modules

You recall „the basics“

- Threats
- Security Goals
- Security Services
- Adversaries in general

You know historic ciphers and their cryptanalysis

You remember how the security of ciphers is defined, and how it usually is proven

You can explain secure PRGs and stream ciphers

The Feistel network, 3DES and AES are no secret to you and you know operation modes

You know the differences of confidentiality and integrity and schemes to achieve both

You recall secure hashes and the Merkle Damgard construction

You know MACs, and of course the HMAC construction and Keccak (SHA3)

Module Outline

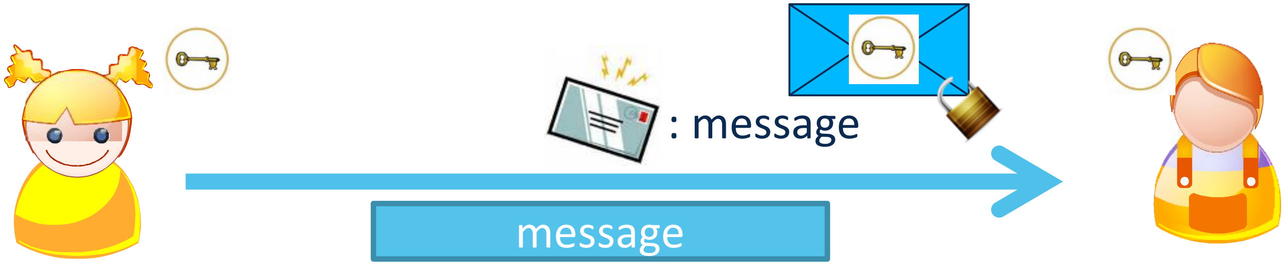
Some basic background on key agreement

Some basic background on number theory

The discrete logarithm and the Diffie Hellmann key agreement

Factoring and the RSA asymmetric crypto scheme

Key Agreement (/ Authentication)



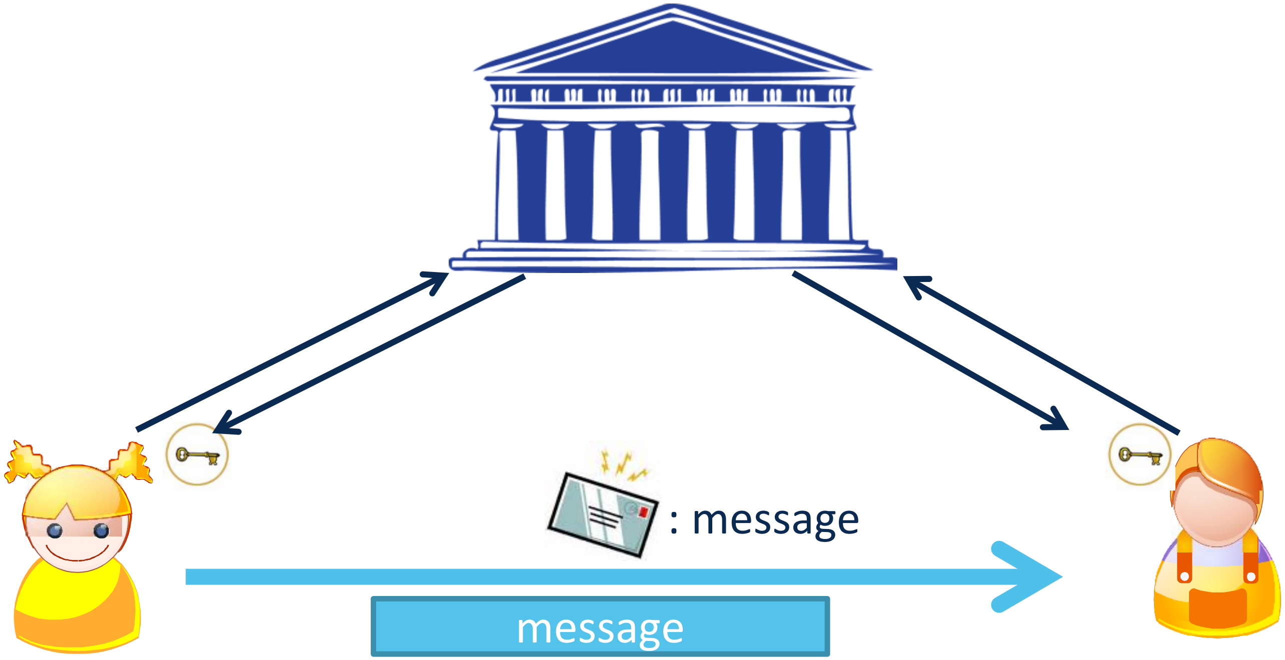
„Key Distribution“:

- ***Secure*** Channel

How many keys have to be exchanged in a system with *n* participants?

What if Alice‘s machine was compromised? If a key has to be revoked? If a key has to be updated?

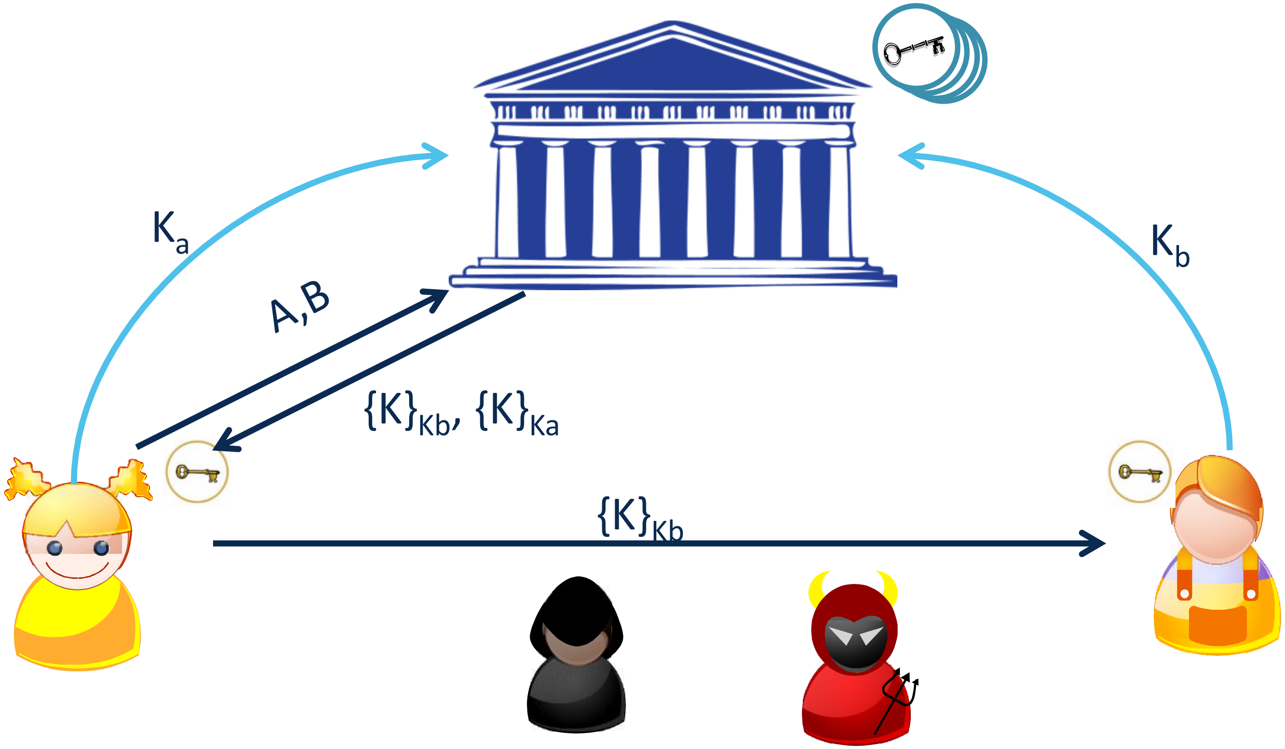
Trusted 3rd Parties/Key Distribution Centers



Key Distribution:

- Pairwise key with KDC („TTP“)
- But what has to be exchanged?

KDC: a Very Simple Protocol



Simple Key Exchange:

- TTP knows / generates all keys
- Eve won‘t break encryption, but Mallory may actively interfere
- …

Interlude: The Dolev - Yao Adversary Model

Mallory has full control over the communication channel

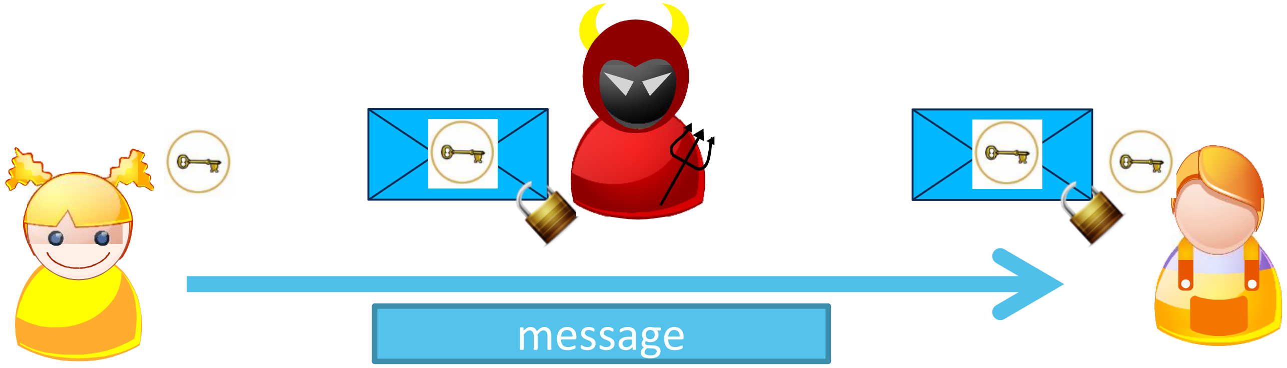
- Intercept/eavesdrop on messages (passive)
- Relay messages
- Suppress message delivery
- Replay messages
- Manipulate messages
- Exchange messages
- Forge messages

But:

- Mallory ***can't*** break (secure) cryptographic primitives!

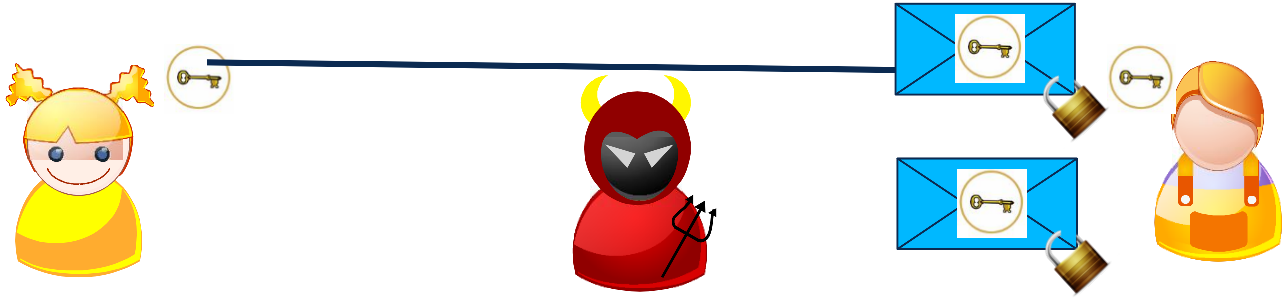
e.g. reverse a secure hash, find collisions, break AES...

Attacks on Key Agreement



Identify/ authenticate parties pro-actively/externally

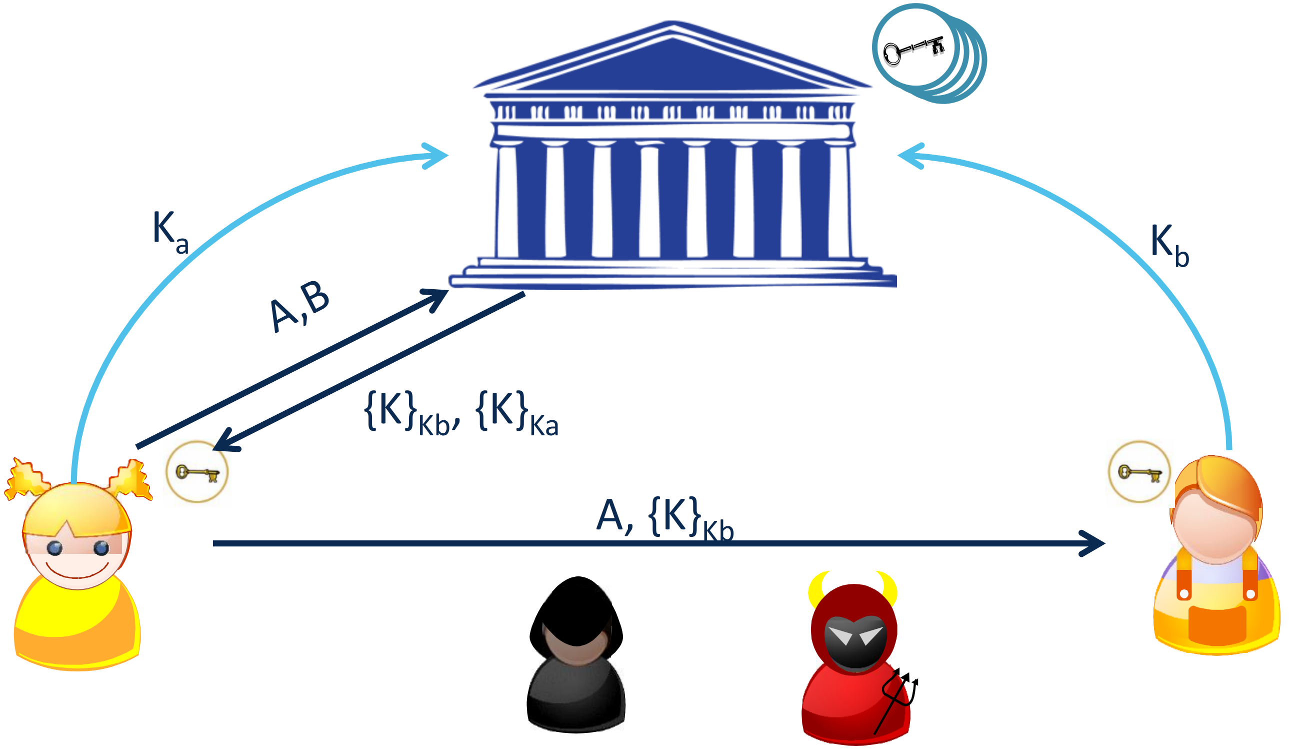
- Man-in-the-middle attack



Ensure freshness!

- Replay attack

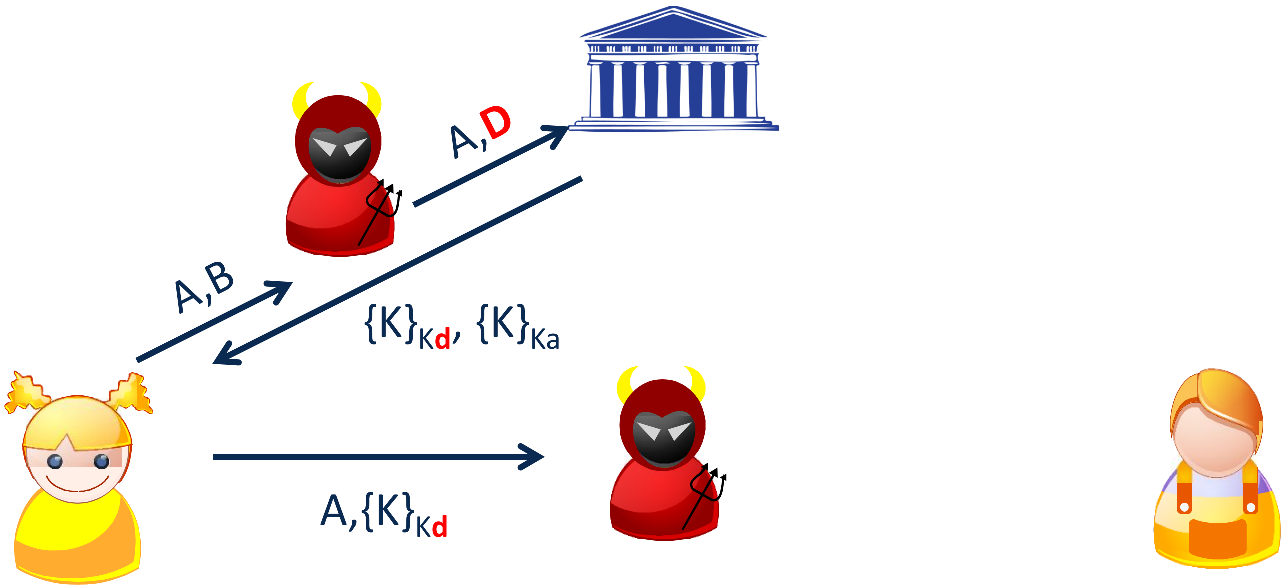
KDC: a Very Simple Protocol



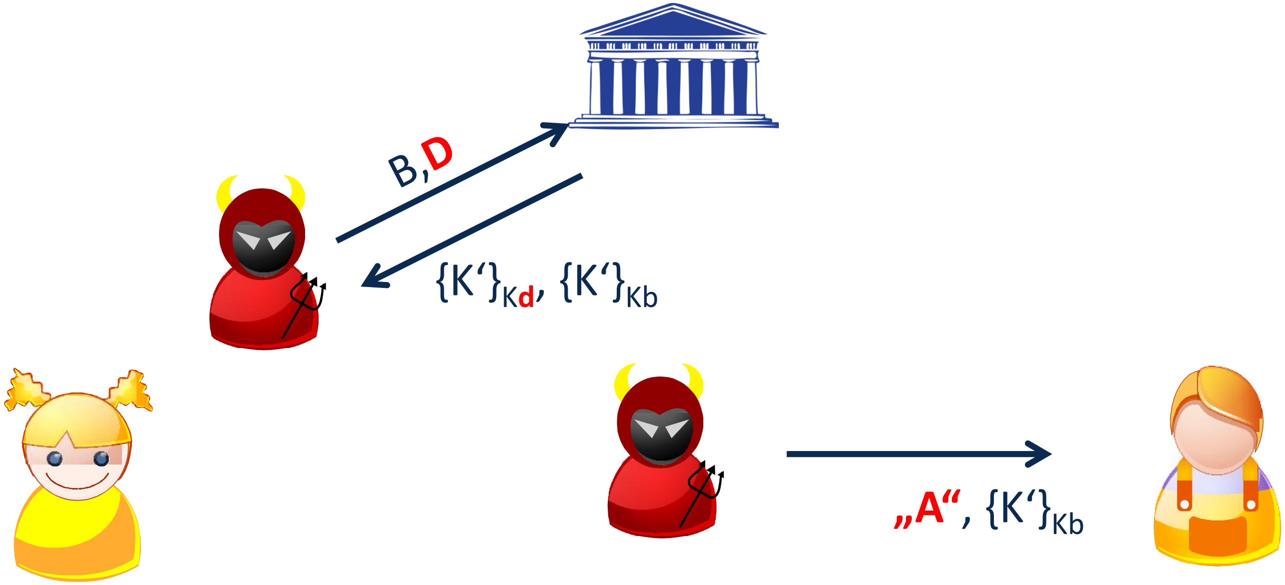
Simple Key Exchange:

- TTP knows / generates all keys
- Eve won‘t break encryption, but Mallory may actively interfere
- ***So what could possibly go wrong??***

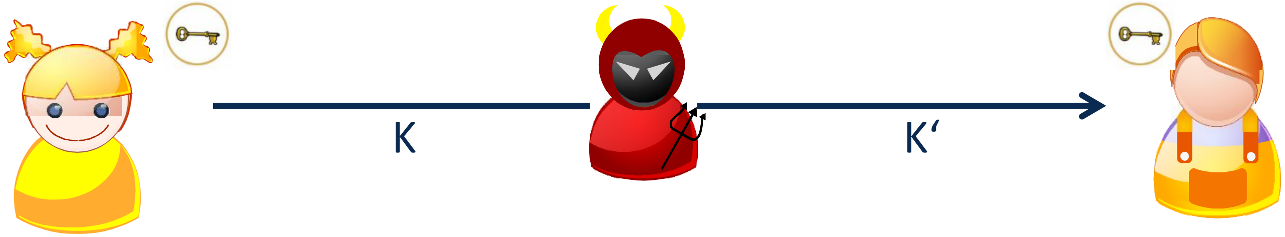
Impersonation / MitM – Step 1



Impersonation / MitM – Step 2

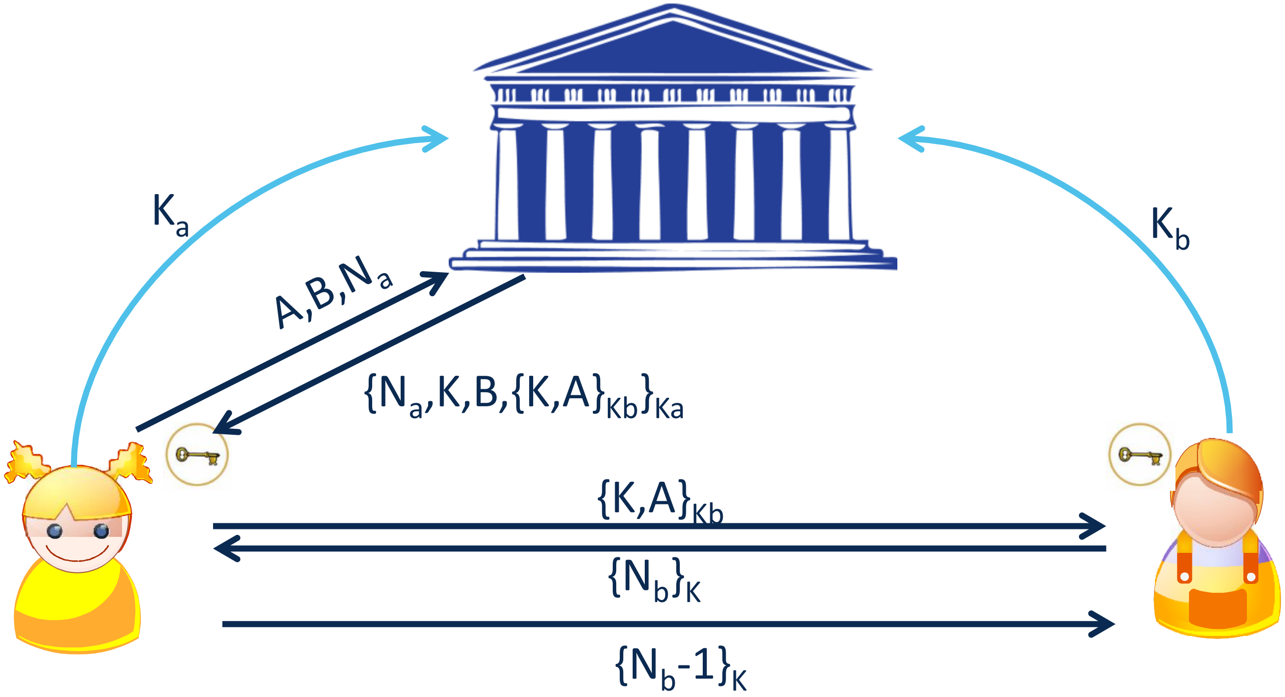


Impersonation / MitM - Result



- Hence: Prevent MitM/replay -> *authenticated key exchange*
- -> Authenticate both parties (*requires trust in KDC*)
- -> ensure „freshness“ of messages
- (and exchange a key…)

Needham Schroeder Key Exchange



- Both prove that they *know key K* (premise: requires K_a or K_b => A/B/KDC)
- Impersonation/MitM prevented by *explicit addressing* (Alice and Bob)
- Replay prevented by *nonces*

- *If Mallory has broken old key, she can impersonate Alice (potentially prevented by timestamps)*

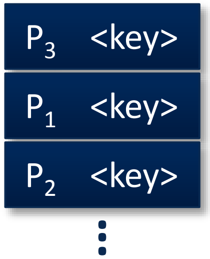
Key *Agreement*

Ralph Merkle, Martin Hellman, Whitfield Diffie

Don‘t exchange keys, calculate them!

Goal:

Exchanged information should be public



Initial idea (due to Merkle, ’74):

- Alice creates 2^{32} puzzles (containing index P_i and key) (O(n))
- Alice shuffles them and sends them to Bob
- Bob selects random puzzle, „calculates“ index P_j and key (O(n))
- Bob informs Alice of P_j, both know key.

What is the complexity for Mallory?

Can we do better?
Polynomial advantage?

A little background…

We‘ll use n ∈ ℕ to denote any positive integer, *p* to denote a prime

Definition:

We say „*a divides n*“ (or write: $a \mid n$), if there exists an integer *k*, such that: $k \cdot a = n$

Basic rules apply:

$a \mid b$ and $b \mid c$ ==> $a \mid c$ (transitivity)

$a \mid b$ ==> $c \cdot a \mid c \cdot b$

$a \mid b$ and $a \mid c$ ==> $a \mid x \cdot b + y \cdot c$ for all x,y

…

Euclidian Division, Equivalence Classes

Given integers a,b ∈ $\mathbb{N}$ (b ≠ 0); there exist integers q,r (0 ≤ r < b) such that: $a = q \cdot b + r$

Typically: $r = a \ (\mathrm{mod}\ b)$ $(a \% b)$

$q = \lfloor a / b \rfloor$

Definition:

If for three integers *a, b, n* it holds that: $a \% n = b \% n$

or: $n \mid (a - b)$

we call *a,b* congruent modulus *n* (or write: $a \equiv b \pmod{n}$)

==> there exists an integer *k* such that: $a - b = k \cdot n$

Modular Arithmetics

Expected math works ;-)

- $a \equiv a \ (mod\ n)$
- $a\ (mod\ n) = b\ (mod\ n)$ ==> $a \equiv b\ (mod\ n)$
- $a \cdot (b + c)\ \ (mod\ n) = (a \cdot b) + (a \cdot c)\ \ (mod\ n)$

We‘ll use $\mathbb{Z}_n$ to denote the set of residues (or: equivalence classes) {0,1,…,n-1}
(We use „$a+b$ in $\mathbb{Z}_n$“ and „$a + b\ (mod\ n)$" interchangeably)

Equivalence classes in n:

$$[a]_n = \{k \in \mathbb{Z}_n \mid k \equiv a\ (mod\ n))$$

And expected maths work, again:

$$[a]_n + [b]_n = [\ a + b\ ]_n$$
$$[a]_n \cdot [b]_n = [\ a \cdot b\ ]_n$$

Btw: $a^9\ (mod\ n) = \qquad (\ a^2\ (mod\ n)\ )$

Greatest Common Divisor

Definition:

For $x,y \in \mathbb{N}$: $d = \boldsymbol{gcd(x,y)}$ is called the ***greatest common divisor***,
if $d \mid x$ and $d \mid y$ and for all divisors *s* of *x* and *y:* $s \mid d$

Examples:

gcd(5,3) = 1; gcd(12,18) = 6; gcd(13,26) = 13

If $gcd(x,y) = 1$ we call *x* and *y* ***coprime*** (***relatively prime***)

For all $x,y \in \mathbb{N}$ there exist integers a,b such that:

$$\boxed{a \cdot x + b \cdot y = gcd(x,y)}$$

(Extended) Euclidian Algorithm

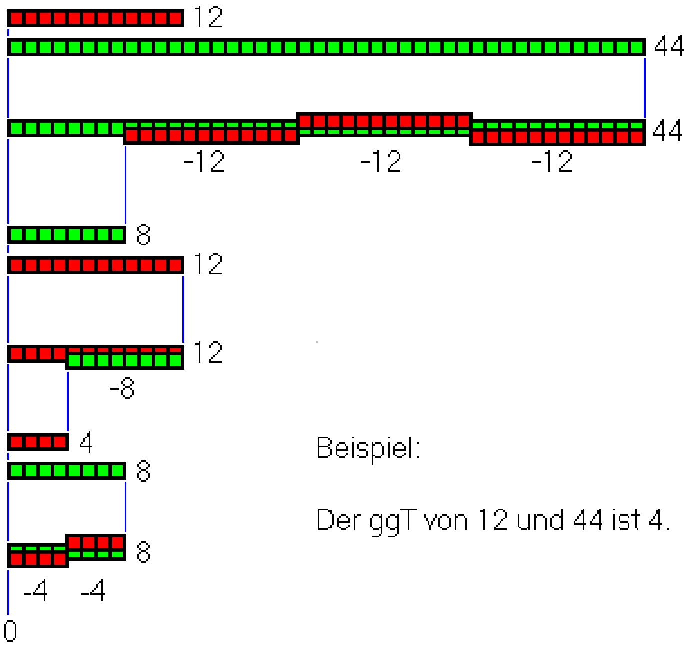
gcd(x,y) can efficiently be computed using the algorithm of Euclid:

> *„If CD does not measure AB, then, when the less of the numbers AB and CD being continually subtracted from the greater, some number is left which measures the one before it.“*

More efficient algorithm directly takes the residue of the integer division of AB and CD…

"[The Euclidean algorithm] is the granddaddy of all algorithms, because it is the oldest nontrivial algorithm that has survived to the present day."

- Donald Knuth, The Art of Computer Programming



Beispiel:

Der ggT von 12 und 44 ist 4.

source: wikimedia

TECHNISCHE UNIVERSITÄT DRESDEN

(Extended) Euclidian Algorithm

gcd(x,y) can efficiently be computed:

```
def gcd(x,y):
    # 0 < x < y
    while (x > 0):
        g = x
        print y, x, y/x, y%x
        x = y % x
        y = g
    print g
```

```
>>> gcd(5,72)
72 5 14 2
5 2 2 1
2 1 2 0
1
```

Extending the algorithm, we can determine the factors a and b:

$$1 = 5 - (2 \cdot 2) = 5 - ((72 - 14 \cdot 5) \cdot 2) = \mathbf{29} \cdot 5 + (-\mathbf{2}) \cdot 72$$

Modular Inversion

Over the rationals, inverse of 2 is ½ . What about $\mathbb{Z}_n$?

Definition:

The **inverse** of *x* in $\mathbb{Z}_n$ is an element *y* in $\mathbb{Z}_n$ s.t.:

$$x \cdot y = 1 \text{ in } \mathbb{Z}_n$$

y is denoted x^{-1} *(x is y^{-1})*

Claim: x in $\mathbb{Z}_n$ has an inverse x^{-1} iff gcd(x,n) = 1

gcd(x,n)=1 ⇒ ∃ a,b: $\mathbf{a \cdot x + b \cdot n = 1}$ *note: $b \cdot n = 0$ in $\mathbb{Z}_n$*

⇒ $a \cdot x = 1$ in $\mathbb{Z}_n$

We'll use $\mathbb{Z}_n^*$ to denote the set of invertible elements in $\mathbb{Z}_n$

Note: $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\} = \{1,2,3,\ldots, (p-1)\}$

Fermat, Primes, and Modular Inversion

Fermat's theorem:

$$\forall x \in \mathbb{Z}_p^* : \quad x^{p-1} = 1 \text{ in } \mathbb{Z}_p^*$$

Example: $p=5$ $\quad 3^4 = 81 = 1$ *in* $\mathbb{Z}_5$

Another (less efficient) way to determine $\mathbf{x^{-1}}$:

$$x \in \mathbb{Z}_p^* \quad \Rightarrow \quad x^{p-1} = 1 \quad \Rightarrow \quad x \cdot x^{p-2} = 1 \quad \Rightarrow \quad x^{p-2} = x^{-1}$$

The structure of $\mathbb{Z}_n^*$

Given that

$$\exists\, g \in \mathbb{Z}_n^* \quad \text{such that} \quad \{1, g, g^2, g^3, \ldots, g^{a-1}\} = \mathbb{Z}_n^*$$

- $\mathbb{Z}_n^*$ is called a cyclic group and
- g is called a generator of $\mathbb{Z}_n^*$

⇒ g^i generates a looping sequence *{1,….}*

The number of elements of this group <g> is its order:

$$ord_n(g) = |<g>| = \text{the smallest a s.t.} \quad g^a = 1 \text{ in } \mathbb{Z}_n^*$$

Examples of <g>: $\mathrm{ord}_7(3) = 6$

$Z_7^* = \{1,2,3,4,5,6\}$

$3^0 = 1$

$3^1 = 3^0 \cdot 3 = 3$

$3^2 = 3^1 \cdot 3 = 9 \equiv 2 \bmod 7$

$3^3 = 3^2 \cdot 3 = 6$

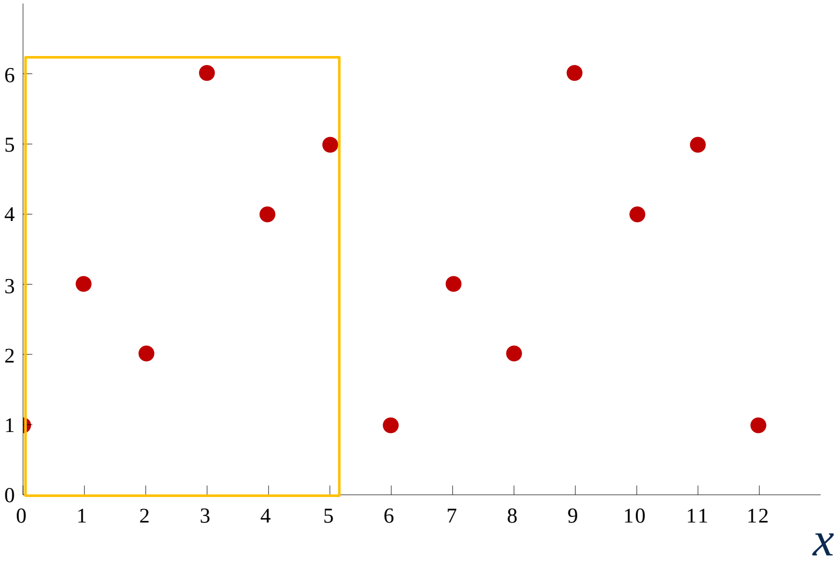
$3^4 = 3^3 \cdot 3 = 18 \equiv 4 \bmod 7$

$3^5 = 3^4 \cdot 3 = 12 \equiv 5 \bmod 7$

$3^6 = 3^5 \cdot 3 = 15 \equiv 1 \bmod 7$

…

$y = 3^x \bmod 7$



$3^6 \equiv 1 \bmod 7$ → order 3 = 6 → 3 generates $\mathbb{Z}_7^*$

Example of <g>: $ord_7(2) = 3$

$Z_7^* = \{1,2,3,4,5,6\}$

$2^0 = 1$

$2^1 = 2^0 \quad 2 = 2$

$2^2 = 2^1 \quad 2 = \quad 4$

$2^3 = 2^2 \quad 2 = \quad 8 \equiv 1 \bmod 7$

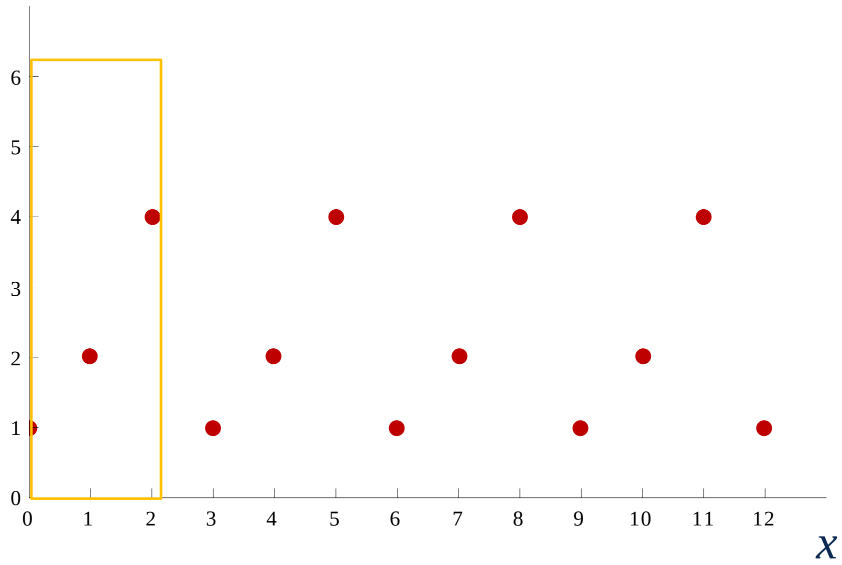
$2^4 = 2^3 \quad 2 = \quad 2$

$2^5 = 2^4 \quad 2 = \quad 4$

$2^6 = 2^5 \quad 2 = \quad 8 \equiv 1 \bmod 7$

…

$y = 2^x \bmod 7$



$2^3 \equiv 1 \bmod 7$ → order 2 = 3 → 2 does not generate $\mathbb{Z}_7^*$

Euler`s Phi function

Let φ(n) (Euler‘s Phi function) denote the number of elements in $\mathbb{Z}_n$*:

$$\varphi\ (n) = |\ \mathbb{Z}_n^*\ |$$

Euler generalized Fermat’s theorem to:

$$\forall\, x \in \mathbb{Z}_n^* : \quad x^{\varphi(n)} = 1 \quad \text{in } \mathbb{Z}_n^*$$

For prime p:

$\varphi(p) = p\text{-}1$ (so the number of elements in $\mathbb{Z}_p$* is p-1)

For primes p,q, and n=p·*q:*

$$\varphi(n) = n - p - q + 1 = pq - p - q + 1 = (p\text{-}1)\,(q\text{-}1)$$

(calculating φ(n) requires factors p,q; factoring is considered hard)

The Discrete Logarithm Problem

Discrete logarithm:

Let $\mathbb{Z}_p{}^*$ be a cyclic group and integer *g* be a generator, then

$$\forall y \in \mathbb{Z}_p{}^*: \qquad \exists x: \ 0 \leq x \leq p\text{-}2 \qquad \text{s.t.}$$

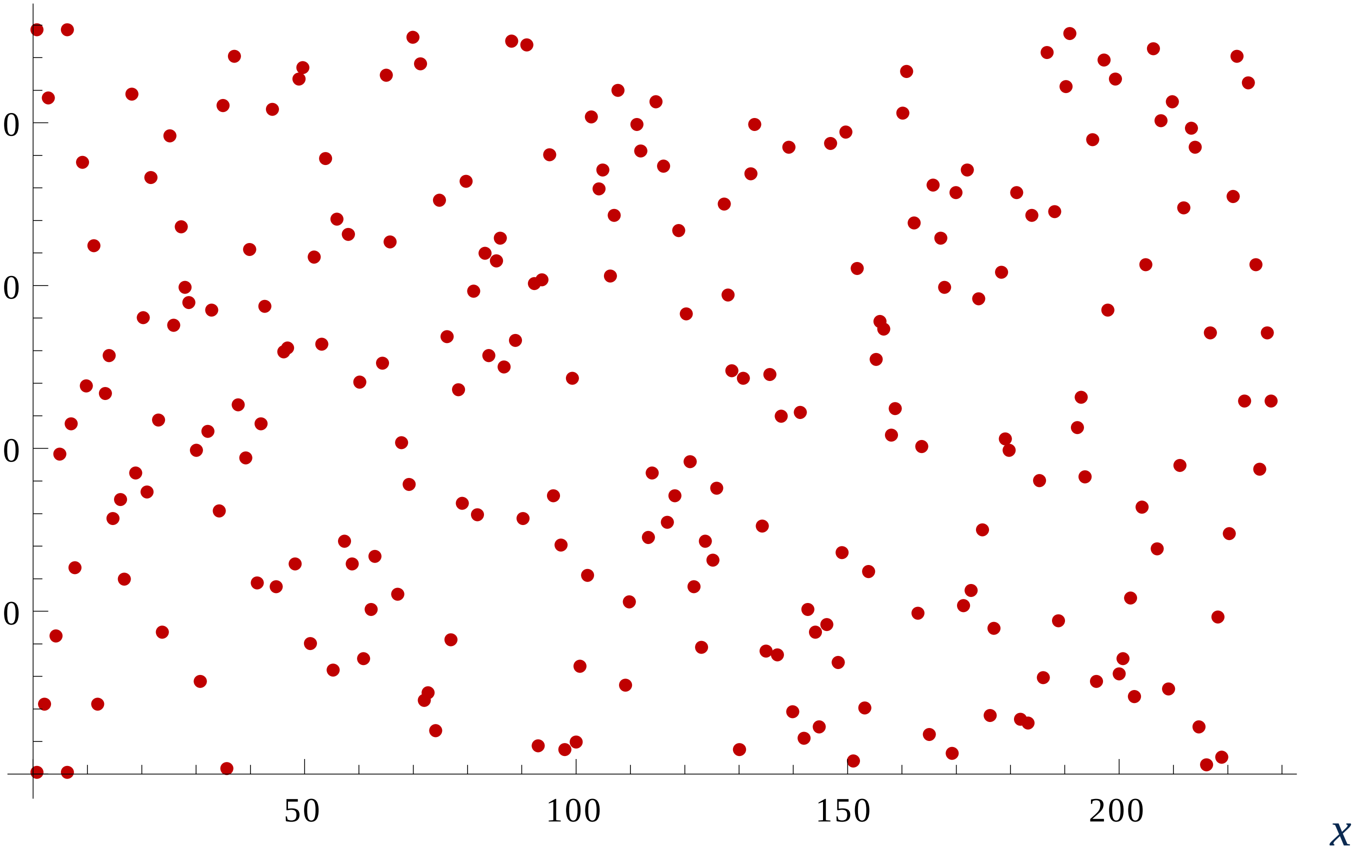
$$g^x \bmod p = y$$

x is called the *discrete logarithm of y to base g mod p* (or „in $\mathbb{Z}_p{}^*$”);

$$x = \log_g y \ (\mathrm{mod}\ p)$$

Discrete Logarithm for p = 229, g = 6

$y = \log_6 x \bmod 229$



Hardness of DLOG

Formally:

DLOG is considered a hard problem if for all eff. alg. A:

$$\mathbf{PR}_{g \longleftarrow G,\, x \longleftarrow \mathbb{Z}p*}[A(G,p,g,\, g^{x}) = x] \leq \varepsilon$$

DLOG currently is considered a hard problem (NP ∩ BQP)

in $\mathbb{Z}_p$* for large p (and in elliptic curve groups mod p)

Computing DLOG

Best known algorithm is the general number field sieve (GNFS)

GNFS finds DLOG of b-bit number in $O(\exp(\sqrt[3]{b} \cdot loglog(b)))$

Key lengths (considered equivalent)

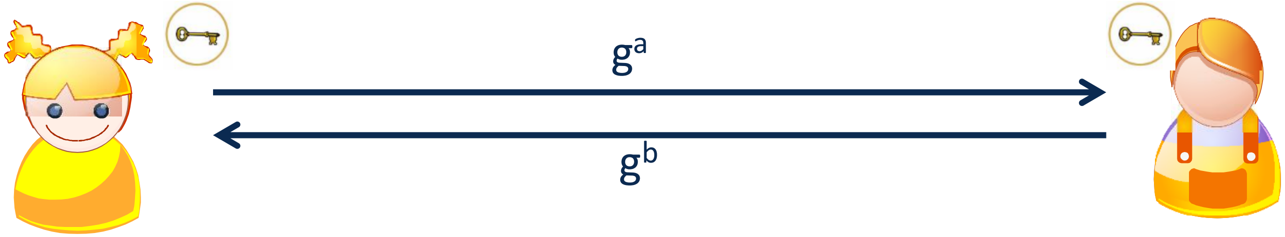
Cipher key length	modulus size	elliptic curve group size
80 bits	1024 bits	160 bits
128 bits	3072 bits	256 bits
256 bits	15360 bits	512 bits

…transition from $\mathbb{Z}_p$* to elliptic curve groups

Back to Keys: Diffie-Hellman(-Merkle)

Consider $\mathbb{Z}_p{}^*$ generated by g, and $\varphi(p) = p-1$

Alice chooses a $\overset{R}{\longleftarrow} \{1,\ldots,(p-1)\}$, Bob chooses b $\overset{R}{\longleftarrow} \{1,\ldots,(p-1)\}$



Alice can calculate: $(g^b)^a$ = g^{ab} = Bob calculates: $(g^a)^b$

Side note Elliptic Curve DH (on the black board)

Security of DH

What about a man in the middle?

Eve sees: p, g, A (== g^a *mod p), B (==* g^b mod p)

How does she compute *g^{ab} mod p?*

Computational Diffie Hellmann Problem (CDH (ECDH)):

- *Given p, g, g^a, g^b* *Curve, G, Q_A, Q_B*
- *Output g^{ab}* *d_B d_A G*

TECHNISCHE UNIVERSITÄT DRESDEN

Modular e‘th roots

Let p be a prime and c ∈ $\mathbb{Z}_p$:

What about higher degree modular polynomials?

Definition:

$x \in \mathbb{Z}_p$ *s.t.* $x^{e} = c$ *in* $\mathbb{Z}_p$ is called the ***e‘th root*** of c

Examples: $7^{1/3} = 6$ in $\mathbb{Z}_{11}$

$3^{1/2} = 5$ in $\mathbb{Z}_{11}$

$1^{1/3} = 1$ in $\mathbb{Z}_{11}$

But: $2^{1/2}$ does not exist in $\mathbb{Z}_{11}$:

0	***0***
1	***1***
2	***4***
3	***9***
4	***5***
5	***3***
6	***3***
7	***5***
8	***9***
9	***5***
10	***1***

Some times it works…

When does $c^{1/e}$ exist ?

Can we compute it efficiently?

The easy case: suppose $\gcd(e, n-1) = 1$ in $\mathbb{Z}_n$

Then for all c in $\mathbb{Z}_n$: $c^{1/e}$ exists in $\mathbb{Z}_n$ and can easily be computed

Proof: let $d = e^{-1}$ in $\mathbb{Z}_{n-1}$, then: $c^{1/e} = c^{d}$ in $\mathbb{Z}_n$:

$$d \cdot e = 1 \text{ in } \mathbb{Z}_{n-1} \Rightarrow \exists k \text{ in } \mathbb{Z}_{n-1}: \quad d \cdot e \quad = \quad k \cdot (n-1) + 1$$

$$\Rightarrow (c^{d})^{e} = c^{de} = c^{k \cdot (n-1) + 1} = (c^{(n-1)})^{k} \cdot c = 1^{k} \cdot c = c$$

Some it doesn‛t

If *n* is an odd prime *(p)*, then:

$\exists$ a in $\mathbb{Z}_p$: gcd(a,p-1) ≠ 1

⇒ some roots don’t exist

And more general:

Let N be a composite number and e>1

Does $c^{1/e}$ exist in $\mathbb{Z}_n$ and can we compute it efficiently?

⇒ can be answered having the prime factors of N (gcd, EEA, above)

Factoring (prime decomposition)

Theorem: all integers > 1 are either prime or a product of primes.

Factoring:

Consider set of integers $\mathbb{Z}_{(2)}(n)$= { $N=pq$, where p,q are n-bit primes}

Task: Find the prime factors (*p and q*) of a random *N* in $\mathbb{Z}_{(2)}(n)$

Best known algorithm (NFS): exp($\tilde{O}(\sqrt[3]{n})$) for n-bit integers

Current world record: RSA-768 (232 digits) (200 machine years)

Consumed enough energy to heat to boiling point 2 olympic pools…

(Breaking RSA-2380 equivalent to evaporating all water on earth)

Lenstra, Kleinjung, Thomé

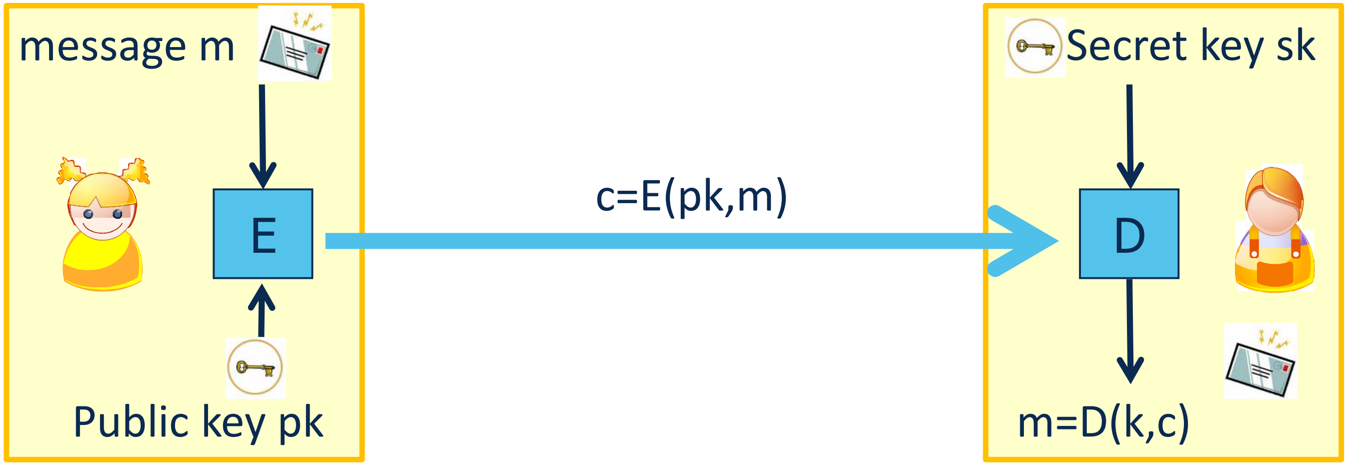
Back to Crypto!

Ron Rivest, Adi Shamir, Leonard Adleman

Goal:

Direct asymmetric encryption

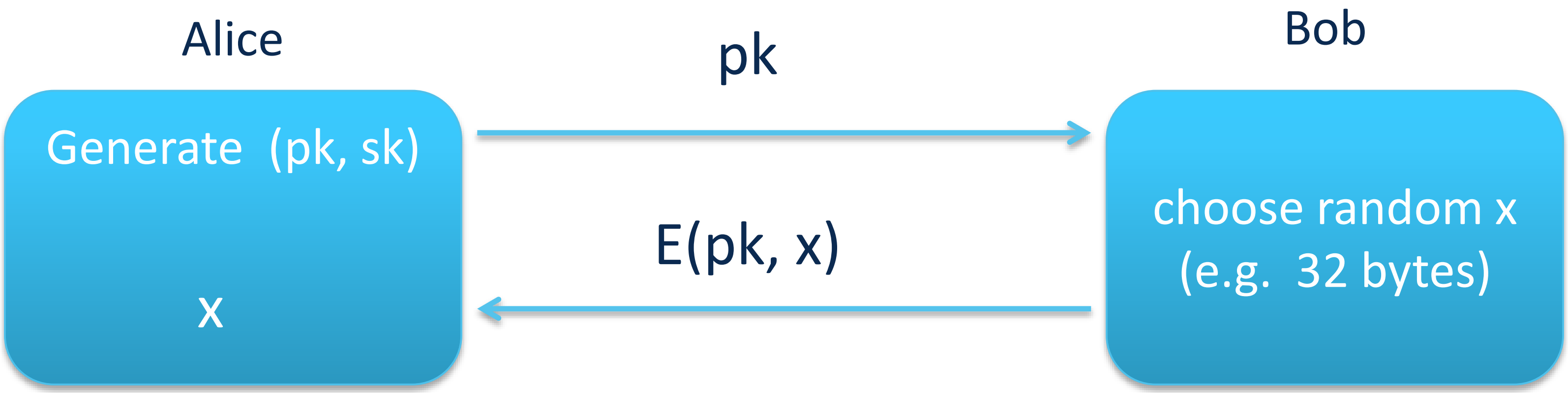
⇒ public key, instead of key-agreement



Ronald L. Rivest, Adi Shamir, Leonard M. Adleman: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM, vol. 21, no. 2, 1978, 120-126.

Common Applications of Asymmetric Enc.

Session setup (for now, only eavesdropping security)

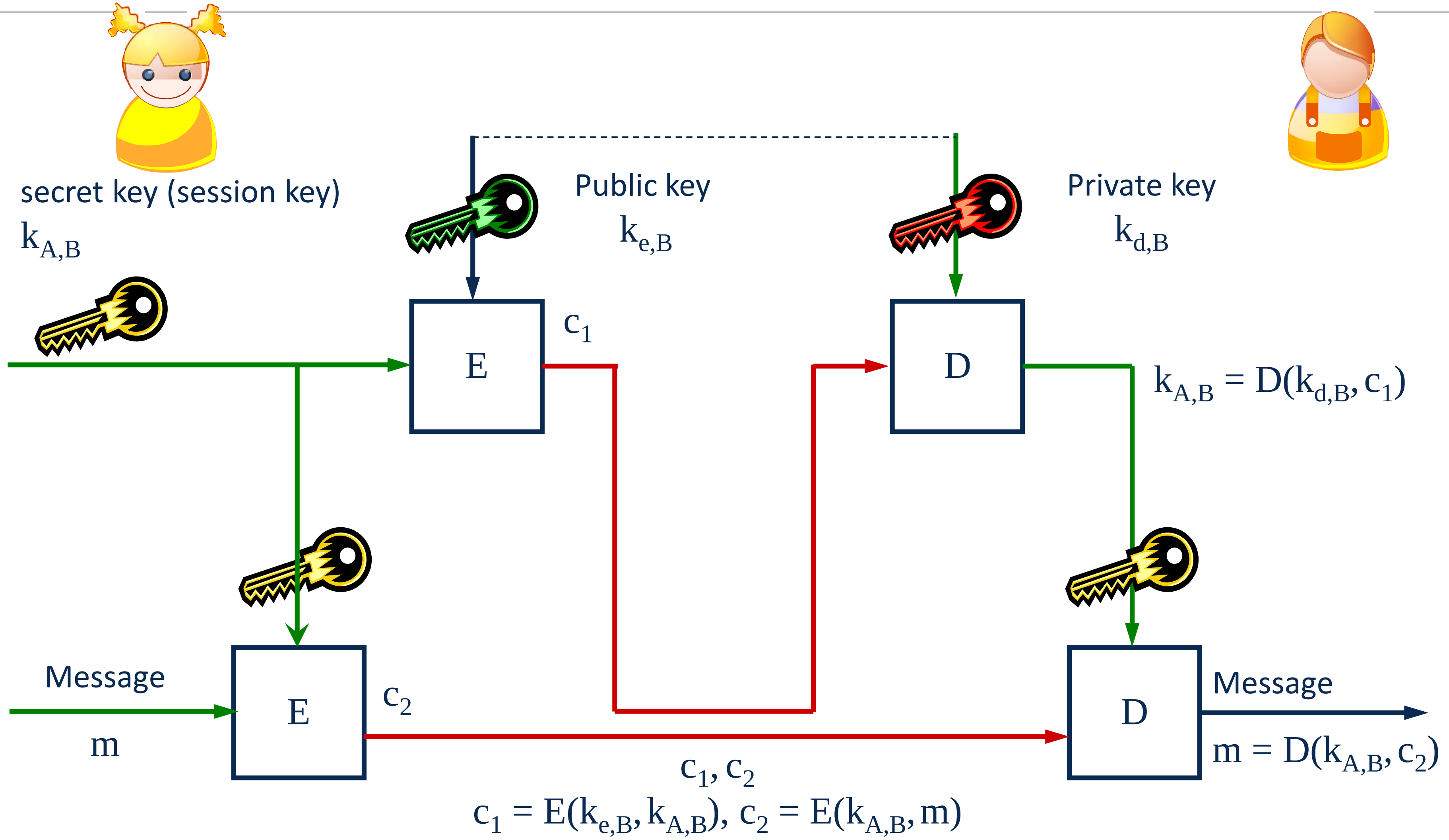


Non-interactive applications: (e.g. Email)

Bob sends email to Alice encrypted using pk_{alice}

Note: Bob needs pk_{alice} (public key management)

Hybrid Crypto System



Public key encryption

A public-key encryption system is a triple of algorithms (G, E, D):

- G(): randomized alg. outputs a key pair (pk, sk)
- E(pk, m): randomized alg. that takes $m \in M$ and outputs $c \in C$
- D(sk,c): det. alg. that takes $c \in C$ and outputs $m \in M$ or $\perp$

Correctness: $\forall$(pk, sk) output by G :

$$\forall m \in M: \quad D(sk, E(pk, m)) = m$$

RSA – The core idea

Observation 1:

- For large primes p and q, $n = p \cdot q$ ***is simple***
- Factoring n to p and q ***is hard***

Observation 2:

- Given p,q, finding e,d, s.t. $x^{e^d} = x^{e \cdot d} = x^1$ in $\mathbb{Z}_n$* ***is simple***
- Extracting the e-th root in $\mathbb{Z}_n$ ***is hard***

Assumption: RSA is a one-way permutation:

For all eff. algs. A:

$$\mathbf{PR[\ A(N,e,y) = y^{1/e}] \leq \varepsilon}$$

with p,q ← n-bit primes, n ← pq, y ← $\mathbb{Z}_N$*

RSA – Key Generation

Each participant

- Chooses two independent, large random primes p, q
- Calculates $N = p \cdot q$ and $\varphi(N) = N\text{-}p\text{-}q+1 = (p-1)(q-1)$
- Chooses random $\boldsymbol{e}$, with $2 < e < \varphi(N)$, $gcd(e, \varphi(N)) = 1$
- And calculates $\boldsymbol{d}$ such that $e \cdot d = 1 \bmod (\varphi(N))$

Subsequently:

Store (p,q,d) (as secret key sk)

Publish (N,e) (as public key pk)

RSA – Encryption and Decryption

Permute („Encryption“)

Given *pk = (N,e):*

RSA (pk, m): $\mathbb{Z}_N^* \longrightarrow \mathbb{Z}_N^*$; c = RSA(e,m) = m^e (in $\mathbb{Z}_N$)

Invert ("Decryption")

Given sk (p,q,d):

m = RSA^{-1} (pk, c) $= c^{1/e}$ $= c^d$ = RSA(d,c) (in $\mathbb{Z}_N$)

$$c^d = \mathbf{RSA(m)^d} = m^{ed} = m^{k\varphi(N)+1} = \left(m^{\varphi(N)}\right)^k \cdot \mathbf{m} = m$$

Bonus: „Signing“ a message

Given sk (p,q,d):

tag = RSA^{-1} (pk, h(m)) = RSA(d,h(m))

TECHNISCHE UNIVERSITÄT DRESDEN

RSA – Encryption and Decryption

Encryption

Given *pk = (N,e):*

RSA (pk, m): $\mathbb{Z}_N{}^* \longrightarrow \mathbb{Z}_N{}^*$; c = RSA(e,m) = m^e (in $\mathbb{Z}_N$)

Decryption

Given sk (p,q,d):

m = RSA^{-1} (pk, c) ,c) (in $\mathbb{Z}_N$)

This „textbook RSA“ is ***insecure***

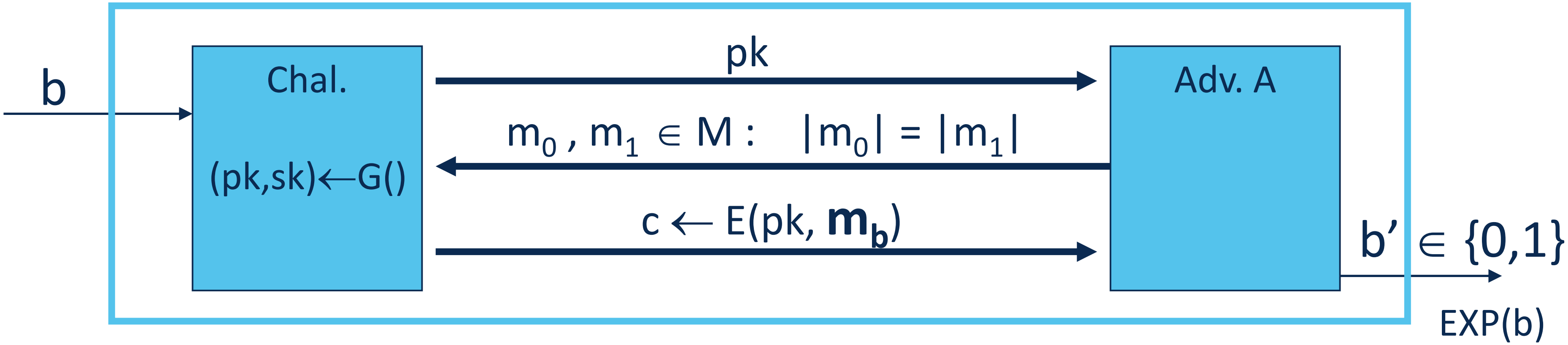
$$c^d = \mathbf{RSA(m)}^d = m^{ed} = m^{k\varphi(N)+1} = \left(m^{\varphi(N)}\right)^k \cdot \mathbf{m} = m$$

Bonus: Signing a message

Given sk (p,q,d):

tag = RSA^{-1} (pk, h(m)) = RSA(d,h(m))

Security against eavesdropping


b
Chal.
(pk,sk)⟵G()
pk
$m_0, m_1 \in M$: $|m_0| = |m_1|$
$c \leftarrow E(pk, \mathbf{m_b})$
Adv. A
$b' \in \{0,1\}$
EXP(b)

Def: E = (G,E,D) is sem. secure (IND-CPA) if for all efficient A:

$$Adv_{SS}[A,E] = |\ PR[Exp(0)=1] - PR[Exp(1)=1]\ | \leq \boldsymbol{\varepsilon}$$

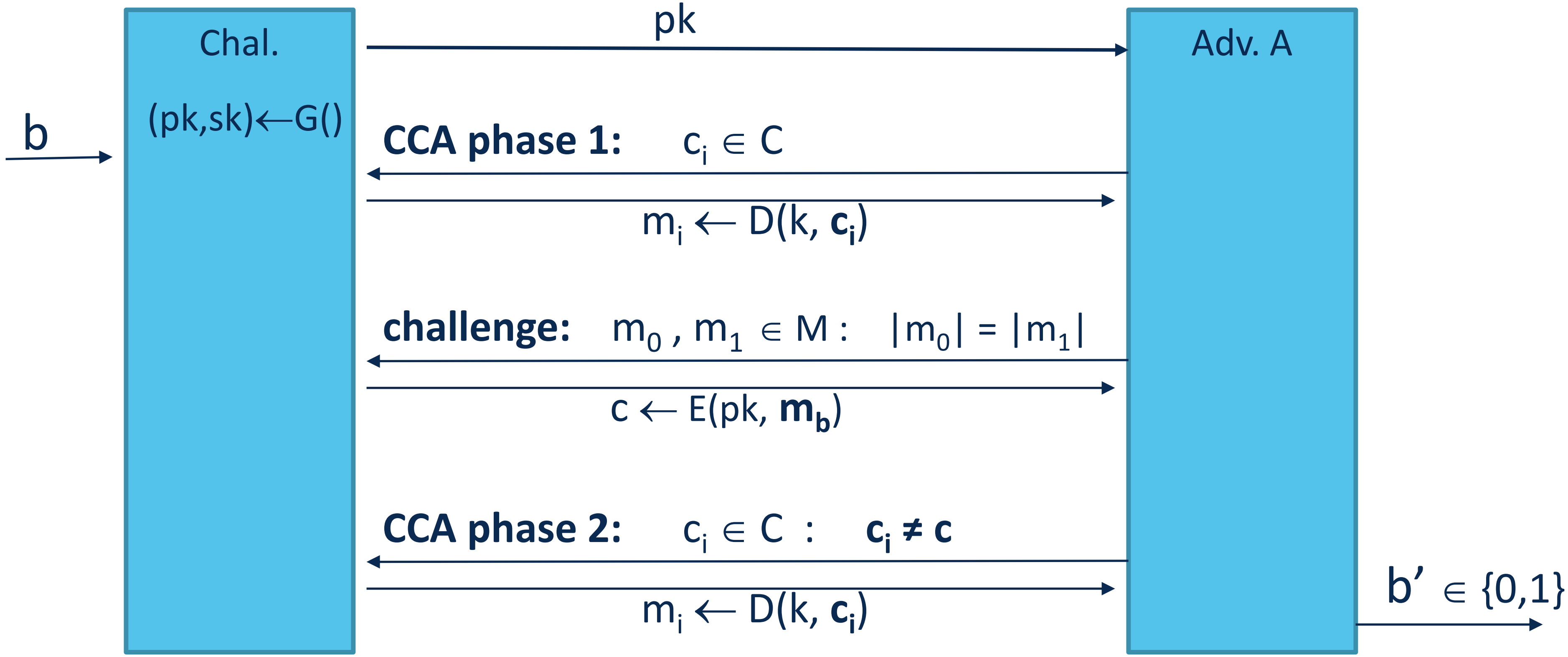
Symmetric crypto: One-time vs. many time security

Asymmetric crypto: Always many time (adversary has PK)

Public key encryption ***must*** be randomized (non-deterministic)

AE – Chosen Ciphertext Security

E = (G,E,D) public key encryption over (M,C). Exp (b=0,1):


Chal.
$(pk,sk) \leftarrow G()$
b
pk
Adv. A
CCA phase 1: $c_i \in C$
$m_i \leftarrow D(k, \mathbf{c_i})$
challenge: $m_0 , m_1 \in M : \ |m_0| = |m_1|$
$c \leftarrow E(pk, \mathbf{m_b})$
CCA phase 2: $c_i \in C : \ \mathbf{c_i \neq c}$
$m_i \leftarrow D(k, \mathbf{c_i})$
$b' \in \{0,1\}$

Def.: E is CCA secure (IND-CCA) if for all efficient A:

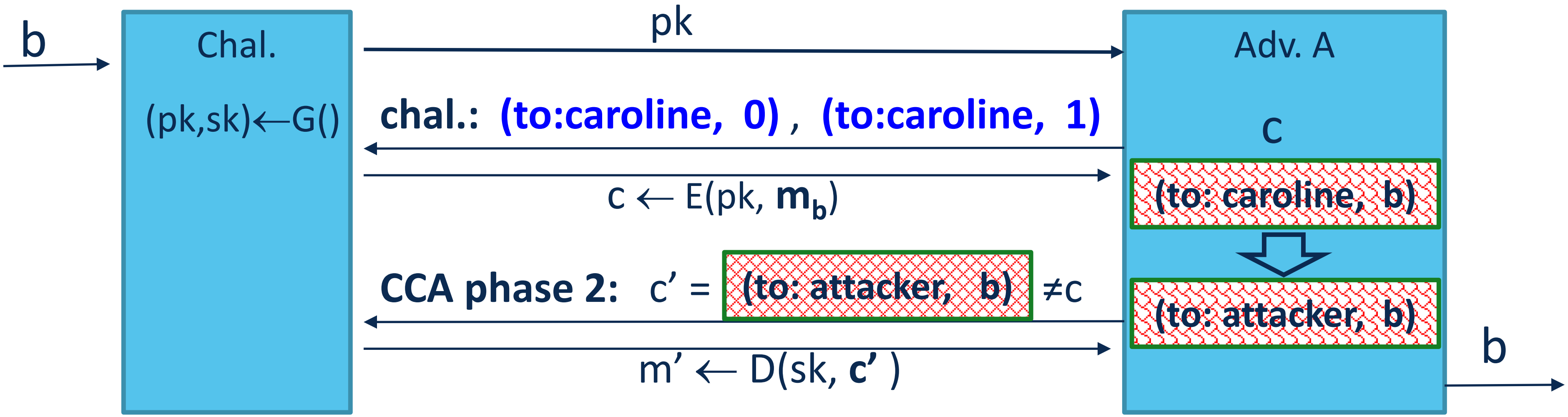
$$Adv_{CCA}[E,A] = |\ PR[Exp(0)=1] - PR[Exp(1)=1]\ | < \boldsymbol{\varepsilon}$$

AE – IND-CCA and malleability

Claim:

E cannot be IND-CCA if it is malleable

Simple example, suppose: to: caroline, body ⟶ to: attacker, body



b
Chal.
(pk,sk)←G()
pk
chal.: (to:caroline, 0) , (to:caroline, 1)
c ← E(pk, $\mathbf{m}_b$)
CCA phase 2: c' = (to: attacker, b) ≠c
m' ← D(sk, **c'**)
Adv. A
c
(to: caroline, b)
(to: attacker, b)
b

Reminder: Trapdoor Functions (TDF)

Def: a trapdoor func. $X \longrightarrow Y$ is a triple of efficient algs. (G, F, F^{-1})

G(): randomized alg. outputs a key pair (pk, sk)

F(pk,·): det. alg. that defines a function $X \longrightarrow Y$

F^{-1}(sk,·): defines a function $Y \longrightarrow X$ that inverts F(pk,·)

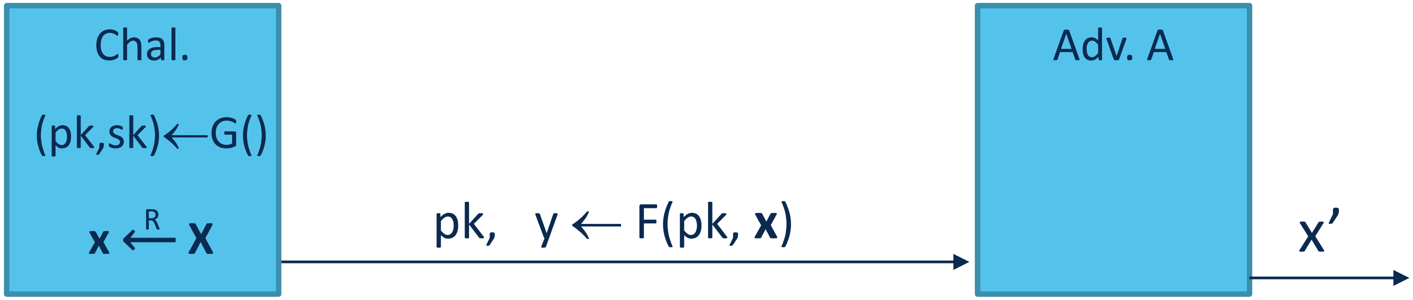
More precisely: ∀(pk, sk) output by G

$$\forall x \in X: \quad F^{-1}(sk, F(pk, x)) = x$$

Secure TDF

(G, F, F^{-1}) is secure if $F(pk, \cdot)$ is a "one-way" function:

can be evaluated, but cannot be inverted without sk


Chal.
$(pk,sk) \leftarrow G()$
$\mathbf{x} \overset{R}{\leftarrow} \mathbf{X}$
pk, $y \leftarrow F(pk, \mathbf{x})$
Adv. A
x'

Def: (G, F, F^{-1}) is a secure TDF if for all efficient A:

$$\mathrm{Adv}_{OW}[A,F] = \mathbf{Pr}[\, \mathbf{x} = \mathbf{x'} \,] < \boldsymbol{\varepsilon}$$

Public key encryption using S-TDF (ISO)

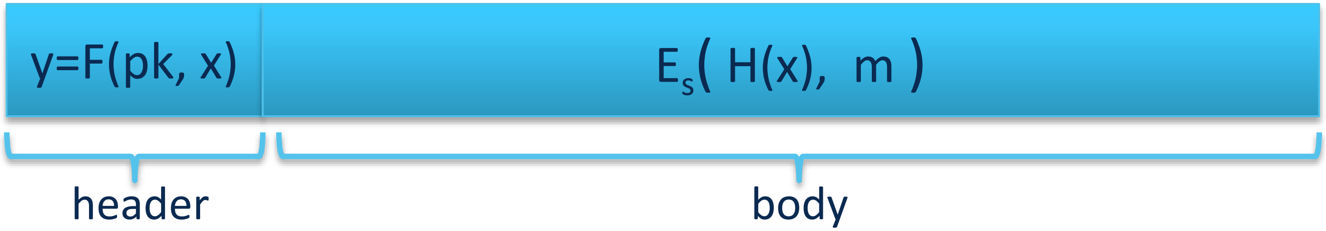
- (G, F, F^{-1}): secure TDF $X \longrightarrow Y$
- (E_s, D_s) : symmetric auth. encryption defined over (K,M,C)
- $H: X \longrightarrow K$ a hash function

E(pk, m) :

```
    $x \xleftarrow{R} X$,      $y \longleftarrow F(pk, x)$
    $k \longleftarrow H(x)$,   $c \longleftarrow E_s(k, m)$
    output   $(y, c)$
```

D(sk, (y,c)) :

```
    $x \longleftarrow F^{-1}(sk, y)$,
    $k \longleftarrow H(x)$,   $m \longleftarrow D_s(k, c)$
    output   $m$
```



$y=F(pk, x)$ | $E_s(\,H(x),\; m\,)$

header | body

Incorrect Encryption with S-TDF

Never encrypt by applying F directly to plaintext:

E(pk, m) :

output $c \longleftarrow F(pk, m)$

D(sk, c) :

output $F^{-1}(sk, c)$

Deterministic: cannot be semantically secure

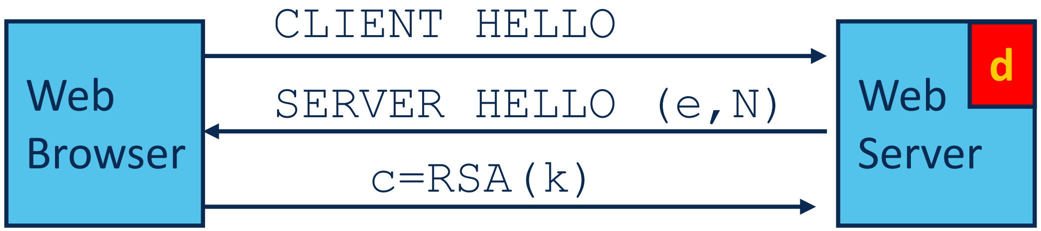
This leads us to: textbook RSA is only a

malleable, deterministic trapdoor permutation!

„Homomorphic property“ of RSA

Assume $m = m1 \cdot m2$

$\Rightarrow$ RSA(m) = RSA(m1) $\cdot$ RSA(m2)



$p = 3$, $q = 11 \Rightarrow N = 33$; $e = 7 \Rightarrow d = 3$

$m = 15 \quad (= 5 \cdot 3)$

$RSA(15) = 15^7 \bmod 33 = 27$

$RSA(5) \cdot RSA(3)$

$= 5^7 \bmod 33 \cdot 3^7 \bmod 33$

$= 14 \cdot 9 \quad \bmod 33$

$= 27$

Suppose k is 64 bits: $k \in \{0,\ldots,2^{64}\}$. Eve sees: $c = k^e$ in Z_N

If $\mathbf{k = k_1 \cdot k_2}$ where $k_1, k_2 < 2^{34}$ (prob. ≈20%) then $\boxed{\mathbf{c/k_1^e = k_2^e}}$ in Z_N

Step 1: build table: $c/1^e, c/2^e, c/3^e, \ldots, c/2^{34e}$. time: 2^{34}

Step 2: for $k_2 = 0,\ldots, 2^{34}$ test if k_2^e is in table. time: 2^{34}

Output matching (k_1, k_2). Total attack time: $\boxed{\approx 2^{40} << 2^{64}}$

Secure RSA (ISO)

(E_s, D_s): symmetric enc. scheme providing auth. encryption.

H: $Z_N \rightarrow K$ where K is key space of (E_s,D_s)

G(): generate RSA params: $pk = (N,e)$, $sk = (N,d)$

E(pk, m):

(1) choose random x in Z_N

(2) $y \leftarrow RSA(x) = x^e$, $k \leftarrow H(x)$

(3) output $(y, E_s(k,m))$

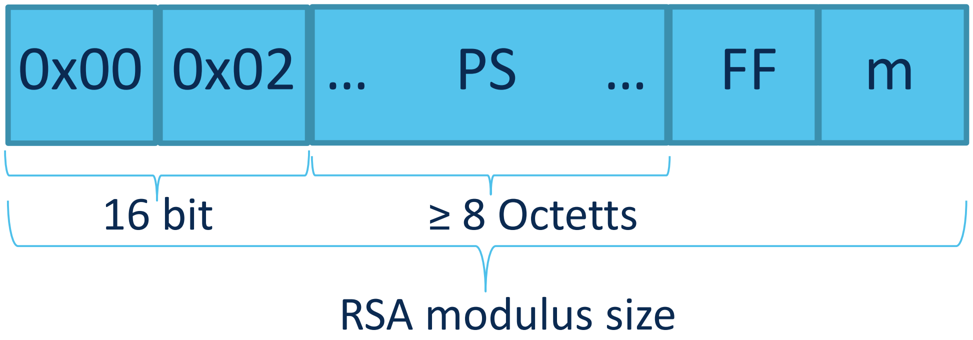
D(sk, (y, c)): output $D_s\left(H\left(RSA^{-1}(y)\right), c\right)$

… in reality, however…

PKCS1 v1.5

Public Key Cryptography Standard (RSA Labs) for RSA encryption

EM =



0x00 0x02 … PS … FF m

16 bit

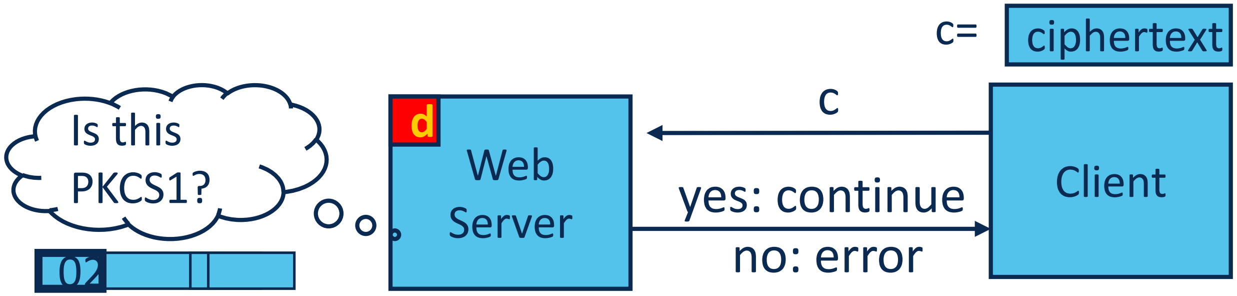
≥ 8 Octetts

RSA modulus size

PS: random non-zero octets

E(pk,m) = RSA(EM)

Widely used, e.g. HTTPS (exchange pre-master secret):



c= ciphertext

Is this PKCS1?

02

d

Web Server

c

yes: continue

no: error

Client

Bleichenbacher Attack on PKCS1 v1.5

Attacker can test if 16MSBs of M are „0x00 0x02“

Recall homomorphic multiplication…

Dan Boneh‘s „Baby Bleichenbacher“ attack:

Suppose N is $N = 2^n$ (an invalid RSA modulus). Then:

Sending c reveals msb(m)

Sending $\mathbf{2^e \cdot c = (2m)^e}$ *in* Z_N *reveals msb(2x mod N) =* $\mathbf{msb_2(m)}$

Sending $\mathbf{4^e \cdot c = (4m)^e}$ *in* Z_N *reveals msb(4x mod N) =* $\mathbf{msb_3(m)}$

… and so on to reveal all of m

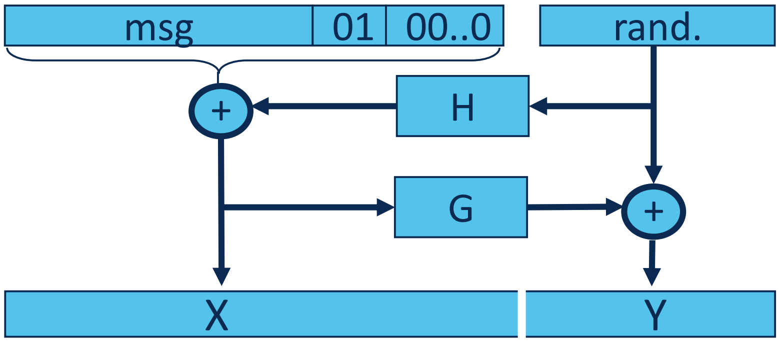
PKCS1 v1.5 defenses

RFC 5246 (HTTPS):

-> always continue with protocol (in encrypted domain with submitted or random pre-master secret), may fail later…

-> PKCS1 v2.0: OAEP

H,G „random oracles“



RSA in Reality (low public exponent)

Avoid costly exponentiation by ***repeated squaring algorithm***

Suppose $x = 53 = (110101)_2 = 32+16+4+1$

Then: $g^{53} = g^{32+16+4+1} = g^{32} \cdot g^{16} \cdot g^{4} \cdot g^{1}$

$$\mathbf{g} \longrightarrow \mathbf{g^{2}} \longrightarrow \mathbf{g^{4}} \longrightarrow \mathbf{g^{8}} \longrightarrow \mathbf{g^{16}} \longrightarrow \mathbf{g^{32}} \qquad \mathbf{g^{53}}$$

To speed up RSA encryption use a small e: $c = m^e \pmod N$

Minimum value: **e=3** ($\gcd(e, \varphi(N)) = 1$)

“Recommended value”: $\mathbf{e=65537=2^{16}+1}$

Encryption: 17 multiplications

Yields asymmetry of RSA: fast enc. / slow dec.

Encryption from DH: ElGamal

Two families of public-key encryption schemes:

Above: based on trapdoor functions (such as RSA)

- Schemes: ISO standard, PKCS1 v1.5, OAEP+, …

To follow: based on the Diffie-Hellman protocol

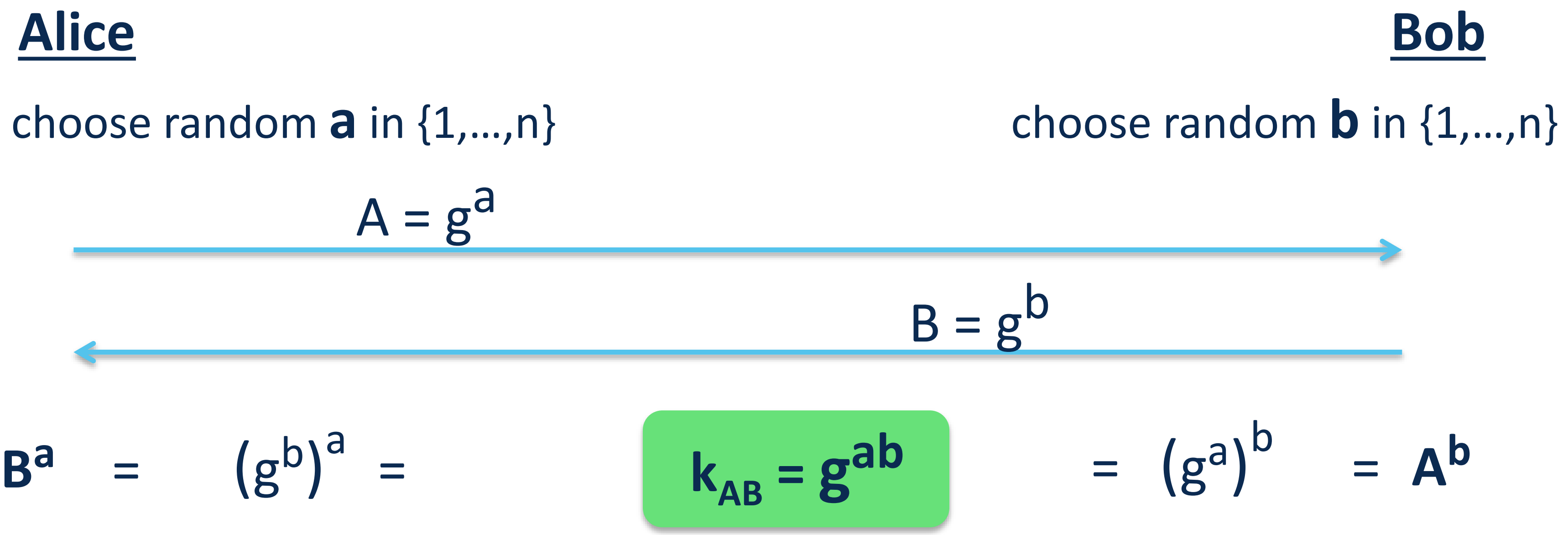
- Schemes: ElGamal encryption and variants (e.g. used in GPG)

Security goals: chosen ciphertext security

Recall the Diffie-Helman Protocol

Fix a finite cyclic group G (e.g G = $(Z_p)^*$) of order n

Fix a generator g in G (i.e. G = $\{1, g, g^2, g^3, \ldots, g^{n-1}\}$)



Alice

Bob

choose random **a** in {1,…,n}

choose random **b** in {1,…,n}

$A = g^a$

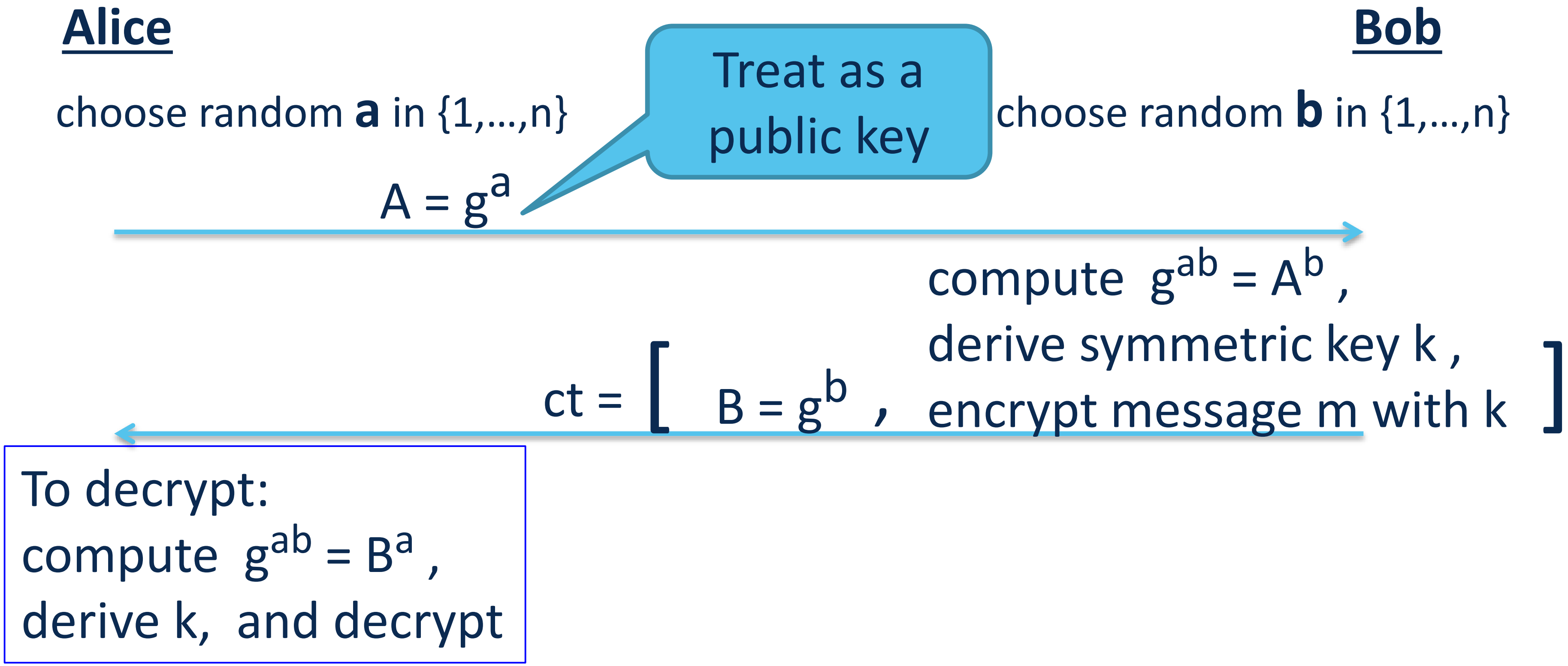
$B = g^b$

$$\mathbf{B^a} = (g^b)^a = \boxed{\mathbf{k_{AB} = g^{ab}}} = (g^a)^b = \mathbf{A^b}$$

ElGamal: PKC from Diffie-Helman

Fix a finite cyclic group G (e.g $G = (Z_p)^*$) of order n

Fix a generator g in G (i.e. $G = \{1, g, g^2, g^3, \dots, g^{n-1}\}$)



Alice

choose random **a** in {1,…,n}

$A = g^a$

Treat as a public key

Bob

choose random **b** in {1,…,n}

compute $g^{ab} = A^b$,
derive symmetric key k ,
encrypt message m with k

ct = $[\; B = g^b$, compute $g^{ab} = A^b$, derive symmetric key k , encrypt message m with k $]$

To decrypt:
compute $g^{ab} = B^a$,
derive k, and decrypt

The ElGamal System (modern)

G: finite cyclic group of order n

(E_s, D_s) : symmetric auth. encryption defined over (K,M,C)

$H: G^2 \longrightarrow K$ a hash function

Construct a pub-key encryption system (Gen, E, D):

Key generation Gen:

- choose random generator g in G and random a in Z_n
- output sk = a , $pk = (g, h=g^a)$

E(pk=(g,h), m) :

$b \overset{R}{\leftarrow} Z_n$, $u \longleftarrow g^b$, $v \longleftarrow h^b$

$k \longleftarrow H(u,v)$, $c \longleftarrow E_s(k, m)$

output (u, c)

D(sk=a, (u,c)) :

$v \longleftarrow u^a$

$k \longleftarrow H(u,v)$, $m \longleftarrow D_s(k, c)$

output m

Semantic security of ElGamal

G: finite cyclic group of order n

Comp. DH (CDH) assumption holds in G if: $g,\ g^a,\ g^b \not\Rightarrow g^{ab}$

for all efficient algs. A:

$$\Pr[\ A(g, g^a, g^b) = g^{ab}\ ] < \boldsymbol{\varepsilon}$$

where $g \longleftarrow \{\text{generators of } G\}$, $a, b \longleftarrow Z_n$

Hash DH (HDH) assumption holds for G and $H: G^2 \longrightarrow K$ if:

$$\big(g,\ g^a,\ g^b,\ H(g^b, g^{ab})\big) \approx_p \big(g,\ g^a,\ g^b,\ R\big)$$

where $g \longleftarrow \{\text{generators of } G\}$, $a, b \longleftarrow Z_n$, $R \longleftarrow K$

Claim: ElGamal is SS under HDH

KeyGen: $g \longleftarrow \{\text{generators of } G\}$, $a \longleftarrow Z_n$

output $pk = (g, h{=}g^a)$, $sk = a$

E(pk=(g,h), m) : $b \longleftarrow Z_n$

$k \longleftarrow H(g^b, h^b)$, $c \longleftarrow E_s(k, m)$

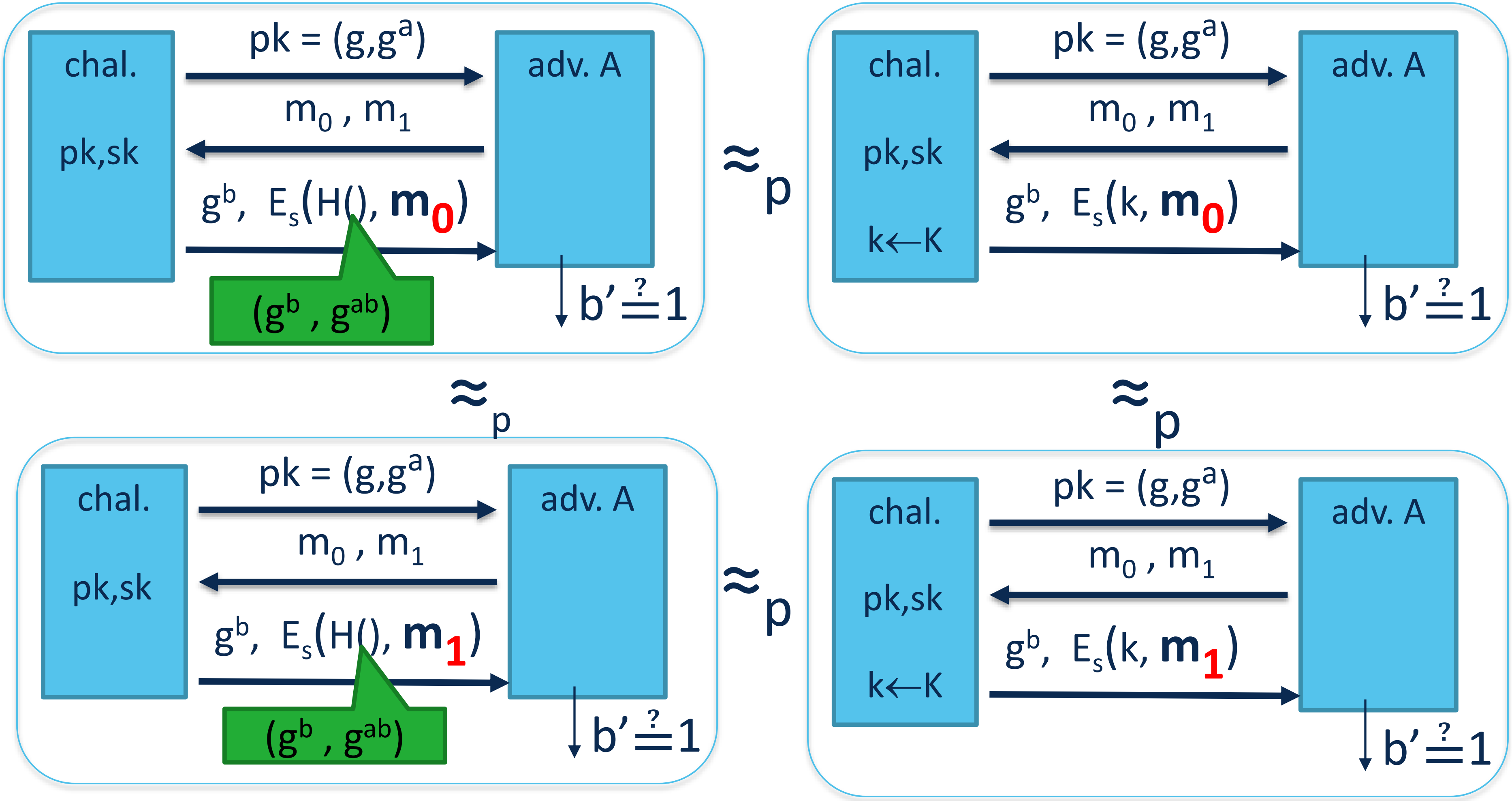
output (g^b, c)

D(sk=a, (u,c)) :

$k \longleftarrow H(u, u^a)$, $m \longleftarrow D_s(k, c)$

output m

Proof idea of semantic security:



chal.

pk,sk

$pk = (g, g^a)$

m_0, m_1

g^b, $E_s(H(), \mathbf{m_0})$

(g^b, g^{ab})

adv. A

$b' \overset{?}{=} 1$

$\approx_p$

chal.

pk,sk

$k \leftarrow K$

$pk = (g, g^a)$

m_0, m_1

g^b, $E_s(k, \mathbf{m_0})$

adv. A

$b' \overset{?}{=} 1$

$\approx_p$

$\approx_p$

chal.

pk,sk

$pk = (g, g^a)$

m_0, m_1

g^b, $E_s(H(), \mathbf{m_1})$

(g^b, g^{ab})

adv. A

$b' \overset{?}{=} 1$

$\approx_p$

chal.

pk,sk

$k \leftarrow K$

$pk = (g, g^a)$

m_0, m_1

g^b, $E_s(k, \mathbf{m_1})$

adv. A

$b' \overset{?}{=} 1$

Chosen Ciphertext Security of ElGamal

IND-CCA of ElGamal is slightly more complicated…

…suffice to say, it can be shown

- Under slightly adapted assumptions
  - „Interactive DH assumption“ (M can check tuples repeatedly)
  - Groups in which IDH = CDH („bilinear“ groups)
- Change algorithm (use tuple as secret key, „twin ElGamal“)
  - pk = $(g, h_1=g^{a1}, h_2=g^{a2})$, sk = $(a1, a2)$

```
**E( pk=(g,h_1,h_2), m) :**    $b \longleftarrow Z_n$

    $k \longleftarrow H(g^b, h_1^b, h_2^b)$

    $c \longleftarrow E_s(k, m)$

    output  $(g^b, c)$
```

```
**D( sk=(a1,a2), (u,c) ) :**

    $k \longleftarrow H(u, u^{a1}, u^{a2})$

    $m \longleftarrow D_s(k, c)$

    output  m
```

Intermediate Summary of AE

Using One-Way functions for asymmetric encryption:

- $f: X \longrightarrow Y$ and efficient algorithm to evaluate $f(\cdot)$, but
- Inverting f is hard
- Trapdoor one-way function: inverting feasible w/ trapdoor information

One-Way functions seen in class:

- DLOG one-way function:
  - With g being generator of cyclic group G
  - $f: \mathbb{Z}_N \longrightarrow G :$ $f(x) = g^x \in G$
  - DLOG hard in G ⇒ f is one-way
- RSA one-way function:
  - $N=p \cdot q$ (p,q large primes), integers e,d: $e \cdot d = 1 \pmod{\varphi(N)}$
  - $f: \mathbb{Z}_N{}^* \longrightarrow \mathbb{Z}_N{}^* :$ $f(x) = x^e$ in $\mathbb{Z}_N{}^* \longrightarrow \mathbb{Z}_N{}^*$
  - F is one-way under RSA assumption, and **f has a trapdoor**

TECHNISCHE UNIVERSITÄT DRESDEN

Summary

You recall the key exchange problem

You can give simple examples of Dolev-Yao adversaries

You understand the idea of key agreement and Merkle puzzles

You recall the basics of modular arithmetics, gcd, and $\varphi (N)$

You can explain the DLOG problem and why it is hard

The Diffie-Helman key agreement is easy for you

You know prime decomposition and the factoring problem

You can explain asymmetric (and hybrid) crypto

You know textbook RSA, you can show that it‘s not secure and how to make it secure

You know ISO encryption and PKCS1 v1.5, and of course ElGamal and its security