COSIC

KU LEUVEN

Public Key Encryption

Prof. Bart Preneel
COSIC – KU Leuven - Belgium
Firstname.Lastname(at)esat.kuleuven.be
http://homes.esat.kuleuven.be/~preneel

November 2023

1

1

Goals

- Understand how the two most important public-key encryption algorithms work: RSA and ElGamal
- Understand the Diffie-Hellman key agreement protocol
- Understand need for post-quantum cryptography
- Understand advantages and disadvantages of public-key encryption

2

2

Outline

- introduction
- mathematical background: see previous slides and article
- public-key encryption
  – one-way functions and Diffie-Hellman
  – trapdoor one-way functions and RSA
  – ElGamal

3

3

Secret key vs. Public key

1. key agreement
   – How can 2 people who have never met construct a key which is only known to themselves?
2. digital signature
   – is it possible to "digitally sign" an electronic message so that anyone can verify the signature

4

4

Limitation of symmetric cryptology

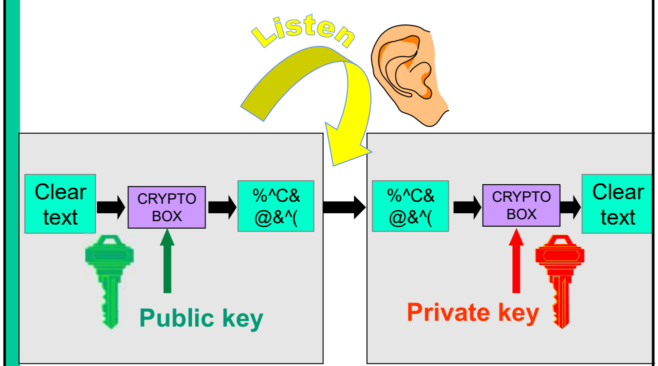
- reduce security of information to security of keys

- but: how to establish these secret keys?
  – Cumbersome and expensive
  – Or risky: all keys in 1 place
- do we really need to establish secret keys?

5

5

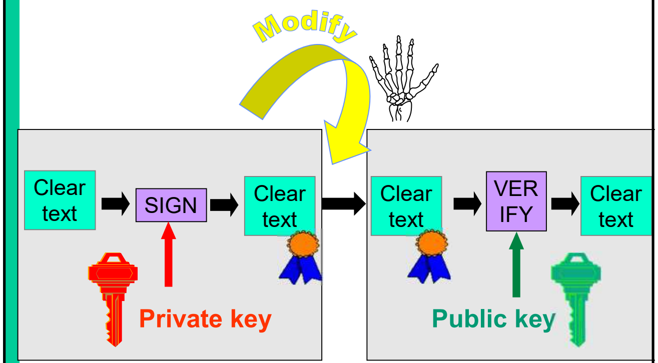
Public-key cryptology: encryption



6

6

Public key cryptology: digital signature



7

7

One-way functions: definition

f : X → Y ; x ↦ f(x) = y is a one-way function ⇔

- f(x), ∀ x ∈ X is easy to compute
- given y ∈ Y, finding an x ∈ X, with f(x) = y is a hard problem (computationally infeasible)

do such functions exist???

= open problem

8

8

Candidate one-way functions

multiplication

given 2 large primes p, q, compute n = p · q (easy)

given a large n, product of 2 primes of about the same size

→ find these primes (e.g., 46,208,777)

modular exponentiation

given a (basis), n with a ∈ [1, n − 1] y = a^x mod n can be computed efficiently (square and multiply)

inverse operation = discrete logarithm:

given a, n and y, find x such that a^x mod n ≡ y

9

9

One-way functions: example

5^4 mod 21 ≡ 16

4 is a solution for the discrete log of 16 w.r.t. the basis 5 modulo 21

but: there is no general polynomial time (efficient) algorithm to compute discrete logs

one-way functions:

- cannot be used directly for encryption of x, because Bob cannot recover x from f(x)
- useful for Diffie-Hellman key agreement protocol and the protection of passwords

10

10

A public-key agreement protocol: Diffie-Hellman

before: Alice and Bob have never met and share no secrets; they agree on a commutative one-way function f(.,.)

generate x
compute f(x, α)

$\xrightarrow{f(x,\alpha)}$

generate y
compute f(y, α)

$\xleftarrow{f(y,\alpha)}$

compute k= f(x, f(y, α))

compute k= f(y, f(x, α))

after: Alice and Bob share a short term key

k = f(x, f(y, α)) = f(y, f(x, α))

11

11

A public-key agreement protocol: Diffie-Hellman

before: Alice and Bob have never met and share no secrets; they know a public system parameter *α* and a prime *p*

generate x compute
α^x mod p

$\xrightarrow{\alpha^x \bmod p}$

generate y compute
α^y mod p

$\xleftarrow{\alpha^y \bmod p}$

compute k=$(\alpha^y)^x$ mod p

compute k=$(\alpha^x)^y$ mod p

after: Alice and Bob share a short term key *k*

The Diffie-Hellman protocol can be broken if one can solve the **discrete logarithm (DLOG)** problem: compute x from *α^x mod p*

12

12

Diffie-Hellman (D-H): example

- prime p = 37; generator α = 2:
  - $2^0=1, 2^1=2, \ldots, 2^5=32, 2^6=27, \ldots, 2^{35}=19, 2^{36}=1$
- $x = 10$: $\alpha^x = 2^{10} \bmod 37 = 25$
- $y = 13$; $\alpha^y = 2^{13} \bmod 37 = 15$
- $k=(\alpha^y)^x = 15^{10} \bmod 37 = 15^{8+2} \bmod 37 = 21$
- $k=(\alpha^x)^y = 25^{13} \bmod 37 = 25^{8+4+1} \bmod 37 = 21$

x	15^x mod 37	25^x mod 37
1	15	22
2	3	33
4	9	16
8	7	34

Try it yourself?
http://users.wpi.edu/~martin/mod.html

13

13

Diffie-Hellman (D-H): security

security of the D-H protocol is based on the **computational D-H assumption**: it is hard to compute α^{xy} from α^x and α^y

"the D-H protocol is secure because the D-H assumption holds"

note: one **cannot** compute α^{xy} by multiplying α^x and α^y !!!

solving the D-H problem cannot be harder than solving the DLOG problem

- indeed: if one can compute x from α^x one can subsequently compute k as $(\alpha^y)^x$
- so the D-H assumption is a stronger assumption than the DLOG assumption ("solving DLOG is hard")

in practice: check also whether α^x and $\alpha^y \notin \{0,1,p-1\}$

14

14

The discrete logarithm problem in practice

DLOG in group $<Z_p^*,.>$

- harder if p is a 'safe prime': (p − 1)/2 prime
- security: 1024 bits ≃ a few months; 2048 bits ≃ 4-6 years; 3072 bits ≃ 10+ years (comparable to factoring an RSA modulus of the same size); algorithms: index calculus, (general) number field sieve
- for a large class of prime numbers, it has been shown that the DLOG problem and the D-H problem are equivalent

DLOG in other groups

- D-H protocol only needs group structure: less algebraic structure implies that DLOG is probably harder
- example: group of Elliptic Curves over a finite field (ECC) Group operations more complex, but keys of 256…512 bits are sufficient

15

15

Trapdoor one-way functions

- one-way functions which can be inverted using addition information (the trapdoor information)
- modular exponentiation with exponent e and modulus n, with fixed values for e, n: $y = x^e \bmod n$.
- inverse operation: modular eth root of y: given e, n and y, find x such that $x^e \bmod n \equiv y$
  - example: $17^3 \bmod 55 \equiv 18$ and $18^{1/3} \bmod 55 = 17$
- trapdoor: there exists an efficient algorithm to extract eth roots mod n if the prime factorization of n is known

16

16

Public-key encryption

send a confidential message protected with a public key (trapdoor one-way functions) $D_{SB}[E_{PB}(m)] = m$

attacks

- chosen plaintext is not meaningful: everyone can compute the ciphertext for any plaintext
- chosen ciphertext: choose c_i and get $m_i = D_{SB}(c_i)$

in a **secure** public-key encryption system, it should be computationally infeasible for an opponent who can launch chosen ciphertext attacks

- to compute SB from PB
- to compute the plaintext m corresponding to a new ciphertext

17

17

Pohlig-Hellman symmetric cryptosystem

key generation

- choose a "large" prime number p
- choose e relatively prime w.r.t. p-1
- compute $d = e^{-1} \bmod p-1$

encryption: $c = m^e \bmod p$
decryption: $m = c^d \bmod p$

note: once p is known, it is very easy to compute d from e and vice versa

18

18

Pohlig-Hellman: example

key generation

- p = 23
- e = 13 (gcd(e,p-1) = 1)
- d = 13^{-1} mod 22 = -5 mod 22 = 17 (Euclides)

encryption: c = 20^{13} mod 23 = 15

decryption: m = 15^{17} mod 23 = … = 20

19

19

RSA asymmetric cryptosystem (1978)

key generation

- choose 2 "large" prime numbers p and q
- compute modulus n = p.q
- compute λ(n) = lcm(p-1,q-1)
- choose e relatively prime w.r.t. λ(n)
- compute d = e^{-1} mod λ(n)

public key = (e,n)

private key = d or (p,q)

The security of RSA is based on the "fact" that it is easy to generate two large primes, but that it is hard to factor their product

encryption: c = m^e mod n

decryption: m = c^d mod n

try to factor 2419

20

20

RSA example

key generation

- p = 19, q = 23
- n= 437
- λ(n) = lcm(18,22) = 18 · 22/ gcd(18, 22) = 9 · 22 = 198
- e= 13
- d = e^{-1} mod 198 **(and NOT mod 437 !!!!)** with Euclid: d = 61

Frequent mistake!

public key = (13,437)

private key = 61 or (19,23)

encryption: c = 123^{13} mod 437 = 386

decryption: m = 386^{61} mod 437 = ... = 123

21

21

RSA: implementation aspects

- generation of large random primes p and q of 1000…2000 bits: Rabin-Miller test
- make sure difference between p and q is sufficiently large (otherwise factoring n is too easy)
- requires efficient implementation of large integer arithmetic (2048…4096 bits)
- choose small public exponent e (3?, 4099, 65537) for faster encryption (questionable if less than 16..32 bits; ok for signatures)
- p, q known (by owner of private key): faster exponentiation with Chinese Remainder Theorem (CRT) (2.5 . . . 3x faster)

22

22

RSA: software performance on a 4-core 3 GHz processor

(KabyLake (906e9); 2017 Intel Xeon E3-1220 v6)
source: https://bench.cr.yp.to/index.html

- DES: 30-40 cycles/byte, about 80 Mbyte/sec, 280 cycles for one 64-bit block (note: estimated not measured)
- AES: 0.9 cycles/byte, about 3.3 Gbyte/sec, 14 cycles for one 128-bit block
  - 8x slower without AES instruction in processors (Intel before 2010)
- SHA-2/SHA-3: 7.5/8.6 cycles/byte, about 400/348 Mbyte/sec, 480/1170 cycles for one 512/1088-bit block
- RSA-3072 encryption: 162,000 cycles for one 3072-bit block; decryption 8.5 million cycles for one 3072-bit block (422 and 22.000 cycles/byte)
  - RSA encryption with exponent e=65537 is 50-100 times faster

23

23

RSA: proof (1/2)

to be shown: ed ≡ 1 mod λ(n) ⇒ m^{ed} ≡ m mod n

case 1: gcd(m, n) = 1

λ(n) = lcm(φ(p)· φ(q))

φ(p) | λ(n) and λ(n) | ed − 1, thus φ(p) | ed − 1

or φ(p) · r = ed − 1 for some r

thus $m^{ed-1} = (m^{\varphi(p)})^r \equiv 1$ mod p

similarly $m^{ed-1} = (m^{\varphi(q)})^r \equiv 1$ mod q

p is a factor of $m^{ed-1} - 1$ and q is a factor of $m^{ed-1} - 1$

thus pq is a factor of $m^{ed-1} - 1$

or $m^{ed-1} - 1 \equiv 0$ mod pq or $m^{ed} \equiv m$ mod n

24

24

RSA: proof (2/2)

to be shown: $ed \equiv 1 \bmod \lambda(n) \Rightarrow m^{ed} \equiv m \bmod n$

case 2: $gcd(m, n) \neq 1$

three subcases: m = 0, gcd(m, n) = p, gcd(m, n) = q

- m = 0: trivial
- gcd(m, n) = p:
  - $m \equiv 0 \bmod p$ implies that $m^{ed} \equiv m \bmod p$
  - see case 1: q is a factor of $m^{ed-1} - 1$ or $m^{ed} \equiv m \bmod q$
  - CRT implies: $m^{ed} \equiv m \bmod n$
- gcd(m, n) = q: similar to previous case

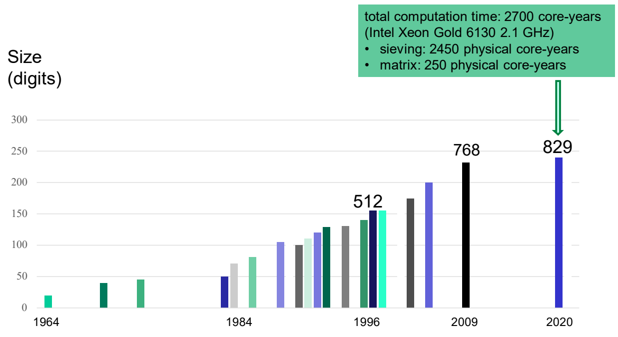
25

25

Factorisation records

https://en.wikipedia.org/wiki/RSA_Factoring_Challenge

2020: 829 bits or 250 digits



26

26

RSA: security in practice (1)

- best known attack: factor n (or discrete log mod n): $O(L_n[a, b])$
  with $L_n[a, b] = \exp [(b+O(1)) (\ln(n))^a \cdot (\ln(\ln(n))^{1-a}]$
- 1970–1992: b = 1, a = 1/2 (quadratic sieve)
  - record (1994): 429 bits (129 digits)
- 1992–. . . : b = 1.923, a = 1/3 (number field sieve)
  - record (2020): 829 bits (250 digits)
- security for 4-6 years: 2048 bits
- security for 10 years and more: need 3072..4096 bits

27

27

RSA: security in practice (2)

second best attack: find $\lambda(n)$

but: if one can find $\lambda(n)$, one can factor n

- we will prove a simpler statement:
  if one can find $\varphi(n)$ one can factor n
  - $n - \varphi(n)+1 = p+q$
    (indeed: $(n - \varphi(n)+1 = pq - (p - 1) \cdot (q - 1) +1)$
  - $n = pq$
- easy to solve these equations for p and q (quadratic equation)

28

28

RSA: security in practice (3)

one can also try to extract eth roots without factoring!

- RSA(0)=0, RSA(1)=1, and RSA(-1)=-1
  - add redundancy to m (or encode m)
- no reduction for small m, which implies that for these values computing eth roots is easy (Newton-Raphson)
  - add redundancy to m (or encode m)
- widely used redundancy = PKCS#1 v1.5
  - Encoding: convert_to_integer(EM = 0x00 || 0x02 || PS || 0x00 || M) with PS at least 8 (pseudo-)random bytes
  - Chosen ciphertext attacks based on error messages allows to recover the plaintext (needs a few thousand to a few million chosen ciphertexts)
  - Better solutions?
    - PKCS#1 v2.0 based on Optimal Asymmetric Encryption (OAEP)
      - but OAEP also has its problems: move to RSA-KEM
    - KEM/DEM construction

29

29

RSA: security in practice (4)

optimizations are often problematic in practice!

- each user needs to have a different modulus
  - otherwise each user can decrypt messages of other users
- too small secret exponents d are not secure (< 29% of modulus length)
- small public exponent e can have problems too (for encryption)
  - if a fraction $(e - 1)/e$ of the plaintext bits is known, the remaining bits can be determined
  - if identical plaintext is sent to e users (with moduli $n_1, n_2, \ldots, n_e$): $c_i \equiv m^e \bmod n_i$ (note n_i are relatively prime)
    - idea: use CRT to find $\underline{c} \equiv m^e \bmod n_1 \cdot n_2 \cdots n_e$
    - as $m^e < n_1 \cdot n_2 \cdots n_e$ solving for m is easy (Newton-Raphson)
    - note: can be extended to different but 'related' messages

30

30

ElGamal encryption (1985)

idea: Diffie-Hellman can be turned into public-key encryption if Bob chooses a fixed private key y and publishes the value α^y mod p

Bob's public key α^y mod p

Bob's private key: y

generate x compute α^x mod p

compute $k=(\alpha^y)^x$ mod p

compute ciphertext c = k.m mod p

α^x mod p

c = k.m mod p

compute $k=(\alpha^x)^y$ mod p

compute plaintext m = c. k^{-1} mod p

Note: the notation in the literature is different, but it is an easy exercise to make the mapping (see next slide)

31

31

ElGamal encryption (1985)

key generation

- general parameters: (safe) prime p and generator α
- private key: x ($1 < x < p - 1$)
- public key: $y = \alpha^x$ mod p

encryption

- generate random k ($1 < k < p - 1$) with gcd(k, p − 1) = 1
- $r = \alpha^k$ mod p (k and r are temporary private/public key)
- $s = y^k \cdot m$ mod p ($0 \le m \le p - 1$)
- ciphertext c = (r, s)

decryption

- $m = s \cdot r^{-x}$ mod p
- indeed $r^{-x} = \alpha^{-kx} = y^{-k}$ mod p

32

32

ElGamal: properties

- security relies on the discrete log problem and not on factoring
  - can also be used with elliptic curves (ECIES)
- ciphertext twice as long as the plaintext
- secure random number generator required for k
- 'non-deterministic' or 'probabilistic' encryption: the same plaintext will always result in different ciphertexts

33

33

Public-key encryption

RSA/ElGamal 1000 times slower than symmetric encryption and has large data expansion for short blocks

- ok for encryption of PIN between terminal and credit card

in practice mostly **hybrid systems** (see next slide)

- use a public key system to establish a secret key (KEM or Key Encapsulating Mechanism) which is then used in a symmetric system for authenticated encryption (DEM or Data Encapsulating Mechanism that is, authenticated encryption or encryption+MAC)
- widely used: SSL/TLS, IPsec, SSH, EMV, email encryption, messaging (Skype, Whatsapp, Signal)

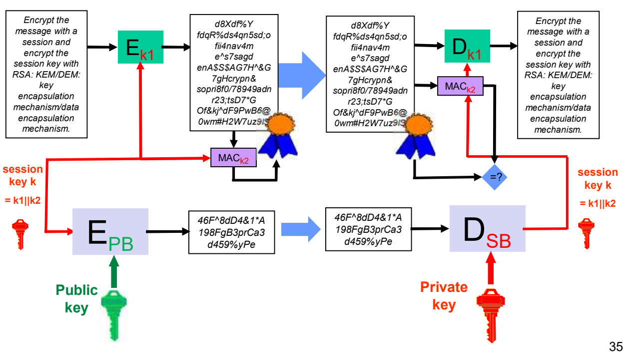
34

34

RSA encryption for long messages (KEM/DEM)

encryption: $c = m^e$ mod n

decryption: $m = c^d$ mod n



35

35

Public-key encryption

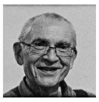
- key management without central Key Distribution Center (rarely used in practice)
  - 2n keys instead of n(n − 1)/2 for symmetric cryptography
- key management with Key Distribution Center:
  - higher security level (no central database of symmetric keys)
  - no need for on-line trusted third party
  - use of certificates
- security: relies on "elegant hard problem":
  - factoring, discrete log mod p or in elliptic curve group
  - what if these problems are solved?
- amazing new protocols possible: ecash, voting, auction,...

36

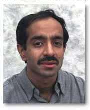
36

If a large quantum computer can be built...

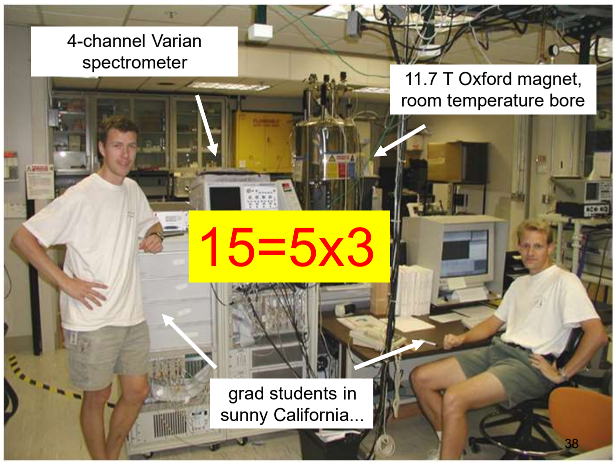
- Yuri Manin 1980 and Richard Feynman 1981



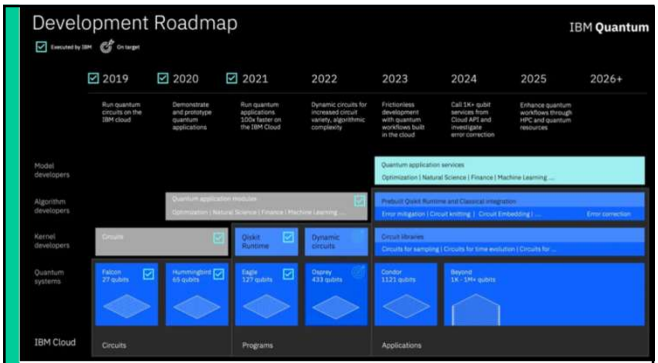

- all schemes based on factoring (RSA) and DLOG are insecure [Shor'94]
  - including elliptic curve cryptography
- symmetric key sizes: x2 [Grover]

37



38



	#qubits
IBM '17:	50
Intel '18:	49
Google '18:	72
Google '19:	54
IBM '19:	53
USTC China '20:	76
IBM '21:	127

Breaking RSA-2048 requires 4096 ideal qubits or 20 million real qubits

39

39

When to switch to quantum resistant cryptography? [Mosca]

Q = #years until first large quantum computer
x = #years it takes to switch (3-10 years)
y = #years data needs to be confidential (10 years)

Need to start switching in the year
2023+ Q – x – y
e.g. Q = 15, x=5, y=10: today!

For data and entity authentication: y = small
(and defense-in-depth)

40

NIST Post-Quantum Competition

(2016-2023) https://en.wikipedia.org/wiki/Post-Quantum_Cryptography_Standardization

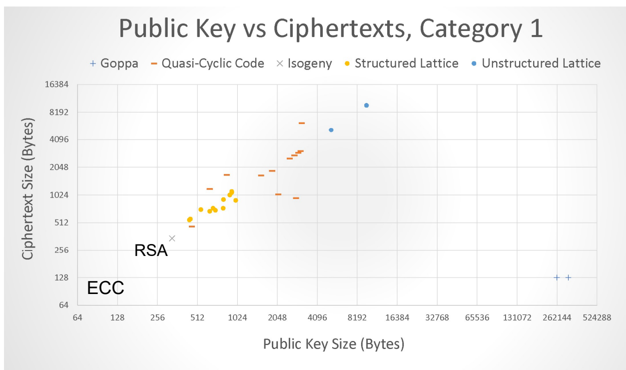
Encryption: KYBER
Digital signatures: Dilithium, Falcon, SPHINCS+ (hash-based signature)

	Signatures	Encryption/KEM	TOTAL
Lattice	4/3/2/2	24/9/3/1	28/12/5/3
Code	5/0/0/0	19/7/1/0	24/7/1/0
Multivariate	7/4/1/0	6/0/0/0	13/4/1/0
Hash	4/1/0/1	0/0/0/0	4/1/0/1
Other	3/1/0/0	10/1/0/0	13/2/0/0
TOTAL	23/9/3/3	59/17/4/1	82/26/7/4

IETF (independent of NIST): 2 hash-based signatures
RFC 8554 Leighton-Micali signatures
RFC 8391 XMSS eXtended Merkle signatures

41

41



https://csrc.nist.gov/CSRC/media/Presentations/Round-2-of-the-NIST-PQC-Competition-What-was-NIST/images-media/pqcrypto-may2019-moody.pdf

42

Encryption / KEM comparison

	Size (Bytes)		Ops/sec (Higher is better)		
	Public Key	Ciphertext	Keygen	Encaps / Encrypt	Decaps / Decrypt
Kyber-512	800	768	125,000	80,000	100,000
RSA-2048	256	256	30	150,000	1,400
ECC X25519	64	64	80,000	15,000	19,000

Disclaimer: numbers by Cloudflare, should be used with caution. These numbers vary considerably for different platforms and implementations. Should only be used as rough guideline.

Source: https://blog.cloudflare.com/nist-post-quantum-surprise/

Slide credit: Jan-Pieter D'Anvers

43