Cryptography
Lecture 6

Dr. Panagiotis Rizomiliotis

PUBLIC KEY MODEL

Public Key cryptography

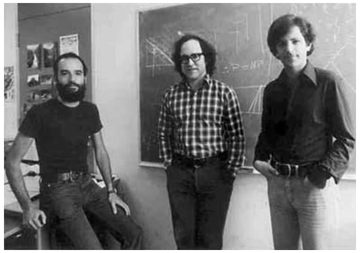
- 1976: «New Directions in Cryptography», in

IEEE Transactions on information theory by

Bailey Whitfield Diffie and Martin Hellman

Bailey Whitfield Diffie
Martin Hellman

- 1977: RSA algorithm (Rivest – Shamir – Adleman)



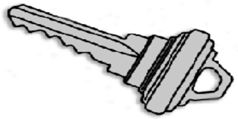
- 1970: "Non-secret encryption"

James Ellis

Government Communications Headquarters (GCHQ)

First step: generate a pair of keys

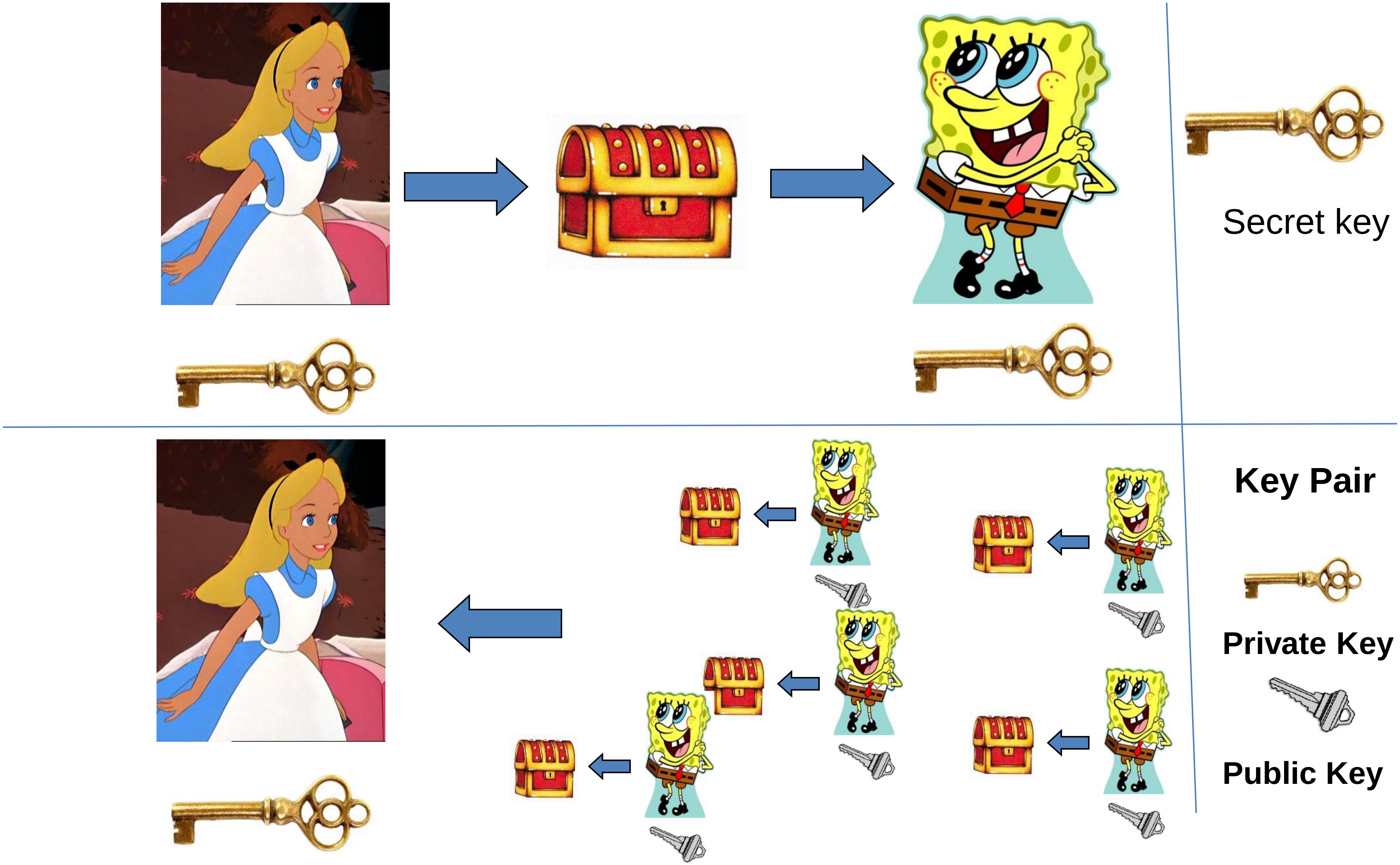
Private key



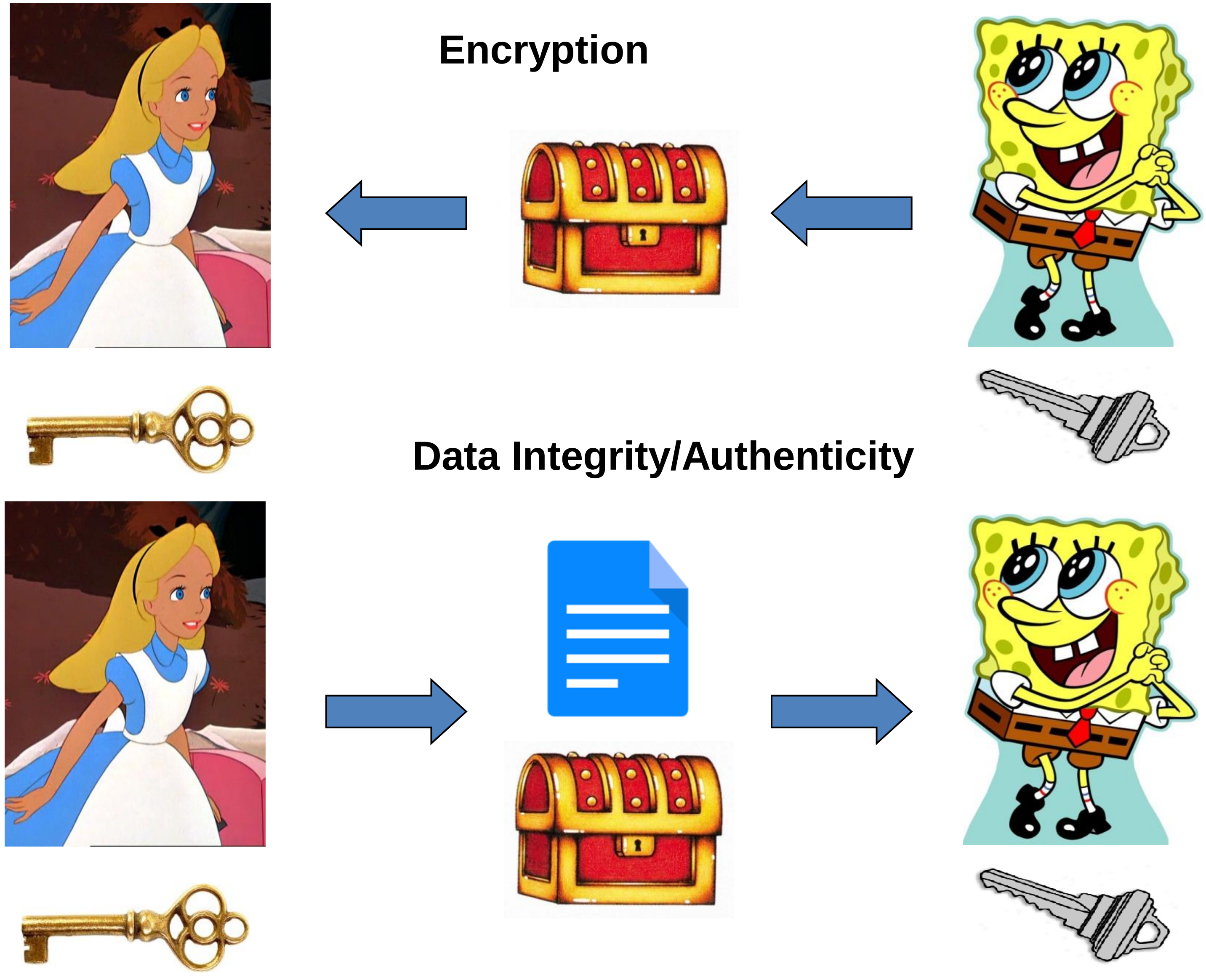
Public key

- ✓ Alice keeps the private key secret
- ✓ Reliably distributes the public key (Bob learns Alice's public key)

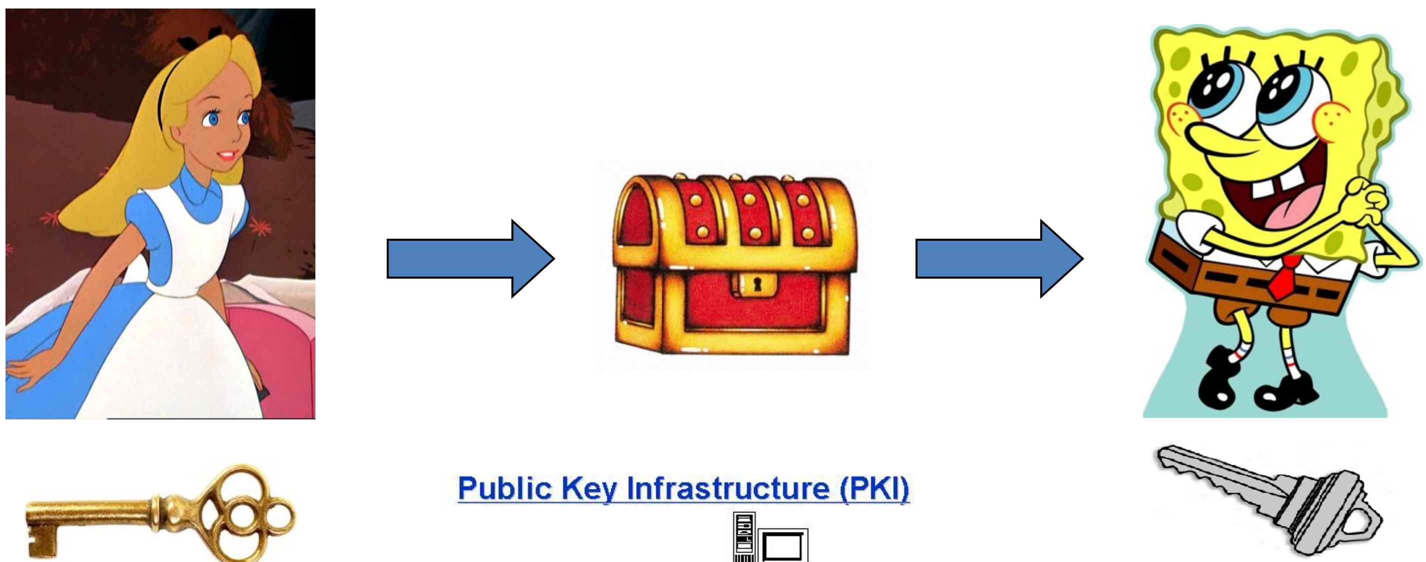
Symmetric key vs public key

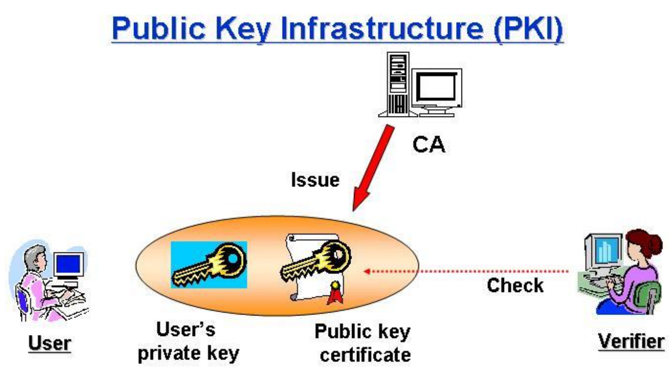


Asymmetric key (Public key)



Public key Cryptography





Public key infrastructure (PKI)

Applications of Public-Key Cryptosystems

- Digital signatures
  - ✓ data authenticity and non-repudiation
- Key agreement
  - ✓ to agree on a session key
- Encryption
  - ✓ Provides data secrecy
  - ✓ key encapsulation
- Entity Authentication
  - ✓ Zero Knowledge Proof (ZKP)

Public Key History

- Some algorithms/mathematical problems
  - **Diffie-Hellman,** 1976, key-exchange based on discrete logs
  - **Merkle-Hellman**, 1978, based on “knapsack problem”
  - **McEliece**, 1978, based on algebraic coding theory
  - **RSA**, 1978, based on factoring
  - **Rabin**, 1979, security can be reduced to factoring
  - **ElGamal**, 1985, based on discrete logs
  - **Blum-Goldwasser,** 1985, based on quadratic residues
  - **Elliptic curves**, 1985, discrete logs over Elliptic curves
  - **Chor-Rivest,** 1988, based on knapsack problem
  - **NTRU**, 1996, based on Lattices
  - **XTR,** 2000, based on discrete logs of a particular field

PUBLIC KEY MAIN SCHEMES

Main schemes

1. RSA and the Integer Factorization problem
2. El Gamal and the discrete logarithm problem

Factorization

- Prime Numbers
  - ➢ prime numbers only have divisors of 1 and self
  - ➢ they cannot be written as a product of other numbers
  - ➢ eg. 2,3,5,7 are prime, 4,6,8,9,10 are not

- Prime Factorisation
  - ➢ to factor a number n is to write it as a product of other numbers:
  - $n = a \times b \times c$
  - ➢ note that factoring a number is relatively hard compared to multiplying the factors together to generate the number
  - ➢ the prime factorisation of a number n is when its written as a product of primes
    - eg. $91 = 7 \times 13$; $3600 = 2^4 \times 3^2 \times 5^2$

Factorization

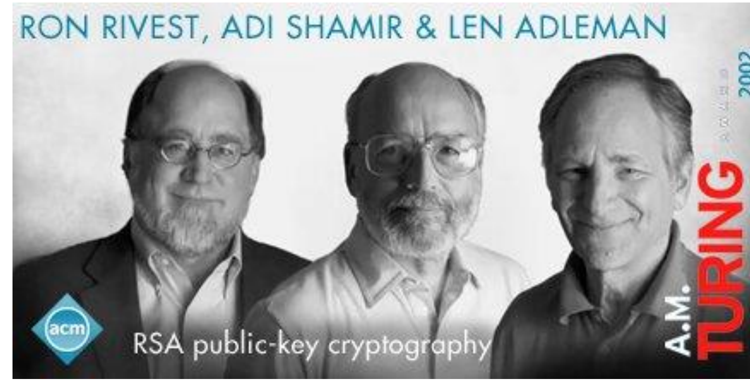
- Prime factorization is considered “hard problem”

✓ We now how to solve it
✓ We cannot do it efficiently
✓ It becomes harder as the size of the integer increases.

- Two types of factoring algorithms

➢ General purpose
➢ Special-purpose

RSA



- by Rivest, Shamir & Adleman of MIT in 1977
- security due to cost of factoring large numbers

- The RSA algorithm involves three steps:

1. key generation,
2. encryption
3. decryption

RSA (textbook)

- SetUp (key pair generation)
- Choose two distinct random prime numbers p and q.
- Compute n = p*q (n is public)
- Compute φ(n) = (p − 1)*(q − 1) (φ(n) is kept secret)
- Choose an integer e, 1 < e < φ(n) and gcd(e, φ(n)) = 1, (e is public)
  - the most commonly chosen value for *e* is $2^{16} + 1 = 65{,}537$.
  - the smallest possible value for *e* is 3
- Compute d as d e≡1 (mod φ(n)) (d is kept secret)
  - (efficiently by using the Extended Euclidean algorithm)

✓ Public key = (e, n)

✓ Private key = (d)

✓ Secret or discarded = (p, q, φ(n))

RSA Use

- Encryption
  - Let m be the plaintext, with $0 \le m < n$.
  - Compute $c = m^e \bmod n$
- Decryption
  - Let c be the ciphertext, with $0 \le c < n$.
  - Compute $m = c^d \bmod n$

RSA Example

1. SetUp (key pair generation)

  - Select primes: $p=17$ & $q=11$
  - Compute $n = pq = 17\times11=187$
  - Compute $\phi(n)=16*10=160$

  - Select e : $gcd(e,160)=1$; choose $e=7$

  - Determine d: $de=1 \bmod 160$ and $d < 160$ Value is $d=23$ since $23\times7=161= 1\times160+1$

- Publish public key KU={7,187}
- Keep secret private key KR={23,17,11}

RSA Example cont

- Given message M = 88 (nb. 88<187)

- Encryption:
  - $C = 88^7 \bmod 187 = 11$

- Decryption:
  - $M = 11^{23} \bmod 187 = 88$

IMPLEMENTATION AND SECURITY ISSUES

Modular Exponentiation

- For efficiency, modular exponentiation uses some combination of
  - Repeated squaring (or square and multiply)
  - Chinese Remainder Theorem (CRT)
  - Montgomery multiplication
  - Sliding window
  - Karatsuba multiplication

Algorithm: Square-and-Multiply(x, c, n)

Comment: compute $x^c \bmod n$, where $c = c_k c_{k-1} \ldots c_0$ in binary.

$z \leftarrow 1$

for $i \leftarrow k$ downto 0 do

$\quad z \leftarrow z^2 \bmod n$

$\quad$ if $c_i = 1$

$\quad$ then $z \leftarrow (z \times x) \bmod n$ $\Big\}$ i.e., $z \leftarrow \left(z \times x^{c_i}\right) \bmod n$

return (z)

Note: At the end of iteration i, $z = x^{c_k \ldots c_i}$.

Example: $11^{23} \bmod 187$

$$23 = 10111_b$$

$$z \leftarrow 1$$

$$z \leftarrow z^2 \cdot 11 \bmod 187 = 11 \quad \text{(square and multiply)}$$

$$z \leftarrow z^2 \bmod 187 = 121 \quad \text{(square)}$$

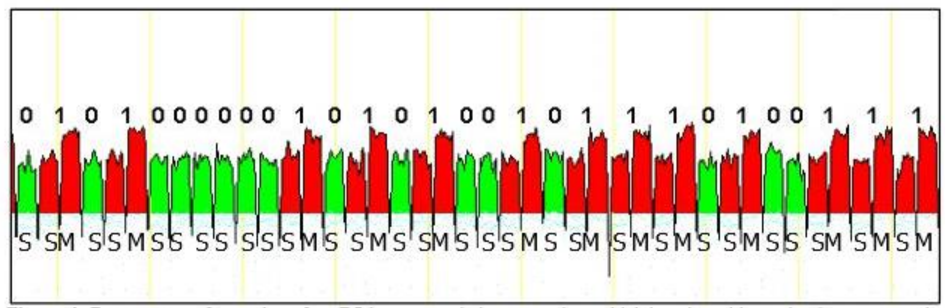
$$z \leftarrow z^2 \cdot 11 \bmod 187 = 44 \quad \text{(square and multiply)}$$

$$z \leftarrow z^2 \cdot 11 \bmod 187 = 165 \quad \text{(square and multiply)}$$

$$z \leftarrow z^2 \cdot 11 \bmod 187 = 88 \quad \text{(square and multiply)}$$

Security of Square and multiply

- Simple Power analysis (we can use for public key exponentiation)



- Power trace from an RSA operation
- Uses standard square and multiply
- Square and multiply operations have visibly different power profiles
- ‘1’ relates to squaring step followed by a multiplication step
- ‘0’ in the exponent involves only a squaring step

Improving RSA's performance

- To speed up RSA decryption use

$$C^d = M \pmod{N}$$

  small private key d.
- There are several attacks:
  - 1987: Wiener showed,
    - if $d < N^{0.25}$ then RSA is insecure.
  - BD'98: if $d < N^{0.292}$ then RSA is insecure

(open: $d < N^{0.5}$)

Insecure: priv. key d can be found from (N,e).

Thus, small *d* should never be used.

RSA With Low public exponent

- To speed up RSA encryption and sig. verification

  $$C = M^e \pmod{N}$$

  use a small e.

- Minimal value: e=3 ($\gcd(e, \varphi(N)) = 1$)
- Recommended value: $e=65537=2^{16}+1$

  Encryption: 17 mod. multiplies.

- Several weak attacks. Non known on RSA-OAEP.
- Asymmetry of RSA: fast encryption (sig. verification)/ slow decryption (signature).
  - ElGamal: approx. same time for both.

RSA SECURITY

RSA Security

- 4 approaches of attacking on RSA
  - brute force key search
    - not feasible for large keys
    - actually nobody attacks on RSA in that way
  - mathematical attacks
    - based on difficulty of factorization for large numbers as we shall see in the next slide
  - side-channel attacks
    - based on running time and other implementation aspects of decryption
  - chosen-ciphertext attack
    - Some algorithmic characteristics of RSA can be exploited to get information for cryptanalysis

- https://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf

Is RSA a one-way permutation?

- To invert the RSA one-way function (without d) attacker must compute:

$$M \quad \text{from} \quad C = M^e \pmod{N}.$$

- How hard is computing e'th roots modulo N ??

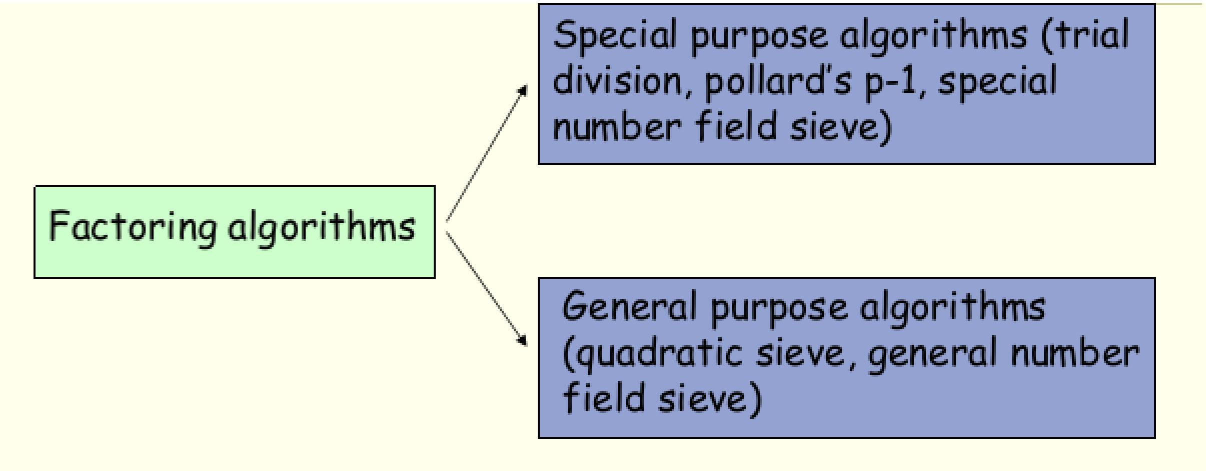
- Best known algorithm:
  - Step 1: factor N. (hard)
  - Step 2: Find e'th roots modulo p and q. (easy)

Factorization Problem

- 3 forms of mathematical attacks
  - factor $n=p*q$, hence find $\phi(n)$ and then d
  - determine $\phi(n)$ directly and find d
    - is equivalent of factoring n
  - find d directly
    - as difficult as factoring n
- So RSA cryptanalysis is focused on factorization of large n

Factoring techniques

- Most efficient
  - Generalized Number Field Sieve
  - Quadratic Sieve
  - Lattice Sieve



Reasons of improvement in Factorization

- increase in computational power
- biggest improvement comes from improved algorithm
  - “Quadratic Sieve” to “Generalized Number Field Sieve”
  - Then to “Lattice Sieve”

Implementation/side channel attacks

- Timing attack:
  - Kocher 1997
    - The time it takes to compute C^d (mod N) can expose d.
    - Systems that use repeated squaring but not CRT or Montgomery (smart cards)
  - Schindler's attack
    - Repeated squaring, CRT and Montgomery (no real systems are known)
  - Brumley-Boneh attack
    - CRT, Montgomery, sliding windows, Karatsuba (as used in openSSL)

- Power attack: (Kocher 99)
  The power consumption of a smartcard while it is computing C^d (mod N) can expose d.

- Faults attack: (BDL 97)
  A computer error during C^d (mod N) can expose d.

Textbook RSA is insecure

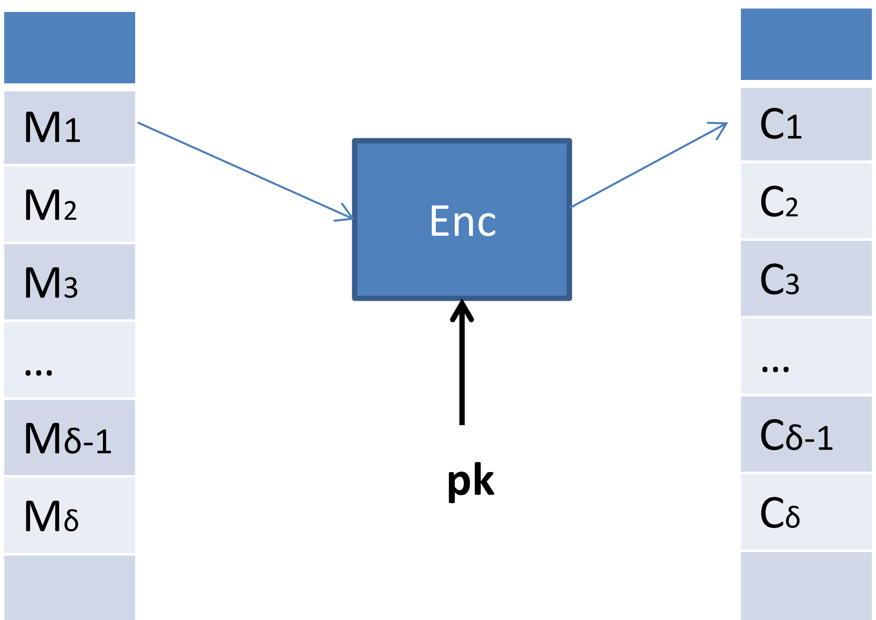
- Textbook RSA encryption:
  - public key: **(N,e)** Encrypt: $\mathbf{C = M^e}$ (mod N)
  - private key: **d** Decrypt: $\mathbf{C^d = M}$ (mod N)

- Completely insecure cryptosystem:
  - Does not satisfy basic definitions of security.
  - Many attacks exist.

- The RSA trapdoor permutation is not a cryptosystem !

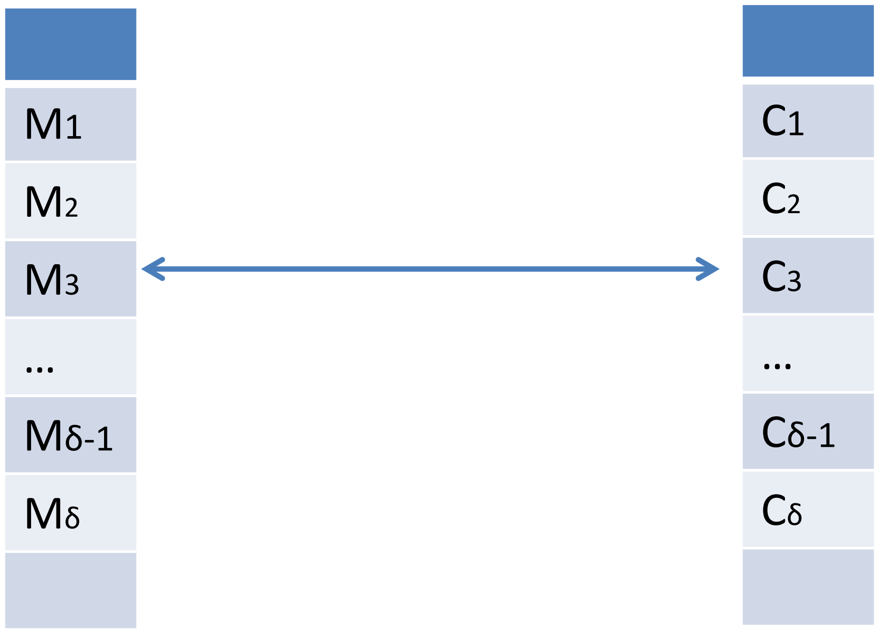
Attack 1: small message space

- If the message space is small, the attacker can encrypt all the candidate massages (offline) and store the computed ciphertexts



Attack 1: small message space

- On-line phase. For a ciphertext c (eavesdropped) the attacker finds c in the table and the corresponding message.



Attack 1: small message space

- Why it works:
  - The encryption key is known (public key)
  - It doesn’t offer semantic security
  - The attacker can repeat all actions of the message owner
- CPA doesn’t make sense
- CCA is more relevant.

Attack 2: Chosen ciphertext Attack

- The textbook RSA has multiplicative homomorphism.
- Let
  - $c1=m1^e \bmod n$
  - $c2=m2^e \bmod n$
- Thus, for
  - $c=c1*c2=m1^e*m2^e \bmod n=(m1*m2)^e \bmod n$

  i.e. c is the encryption of $m=m1*m2$, when $m1*m2<n$

Attack 2: Chosen ciphertext Attack

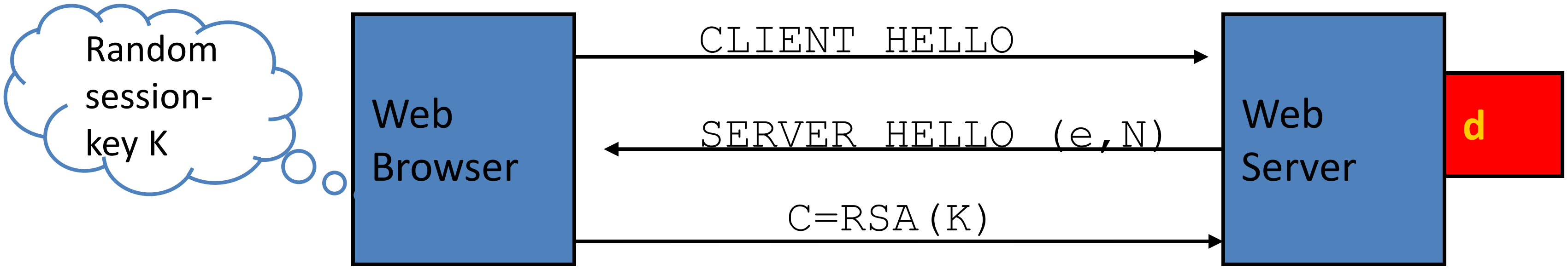
Attack scenario:

The private key owner can decrypt for us any ciphertext except a specific one (target of the attack) c_t. We want to compute the message m_t.

1. The attacker encrypts the message r = 2.
   - $c_r=2^e \bmod n$
2. The attacker computes
   - $c=c_t*c_r \bmod n$
3. The attacker asks for the decryption of c. Let m be the reply of the key owner.
4. The attacker computes m’=m/2 as m_t.

Proof: The attack works when $m_t<n/2$, i.e. when $r* m_t<n$.

Attack 3: A simple attack on textbook RSA

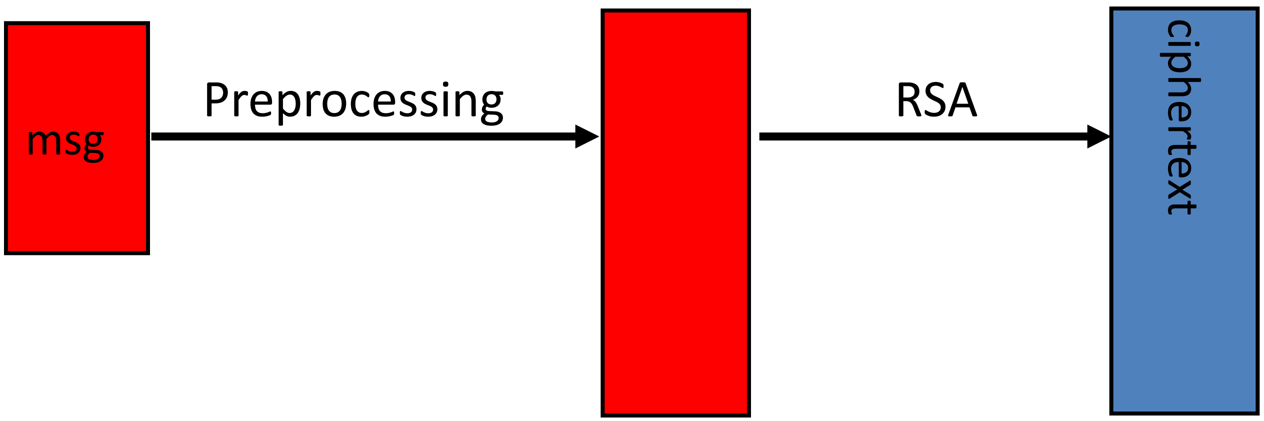


- Session-key K is 64 bits. View $K \in \{0,\ldots,2^{64}\}$
- Eavesdropper sees: $C = K^e \pmod N$.
- Suppose $\mathbf{K = K_1 \cdot K_2}$ where $K_1, K_2 < 2^{34}$. (prob. ≈20%) Then: $\mathbf{C/K_1^e = K_2^e} \pmod N$
- Build table: $C/1^e, C/2^e, C/3^e, \ldots, C/2^{34e}$. time: 2^{34}

  For $K_2 = 0,\ldots, 2^{34}$ test if K_2^e is in table. time: $2^{34} \cdot 34$
- Attack time: $\approx 2^{40} \ll 2^{64}$

Common RSA encryption

- Never use textbook RSA.
- RSA in practice:



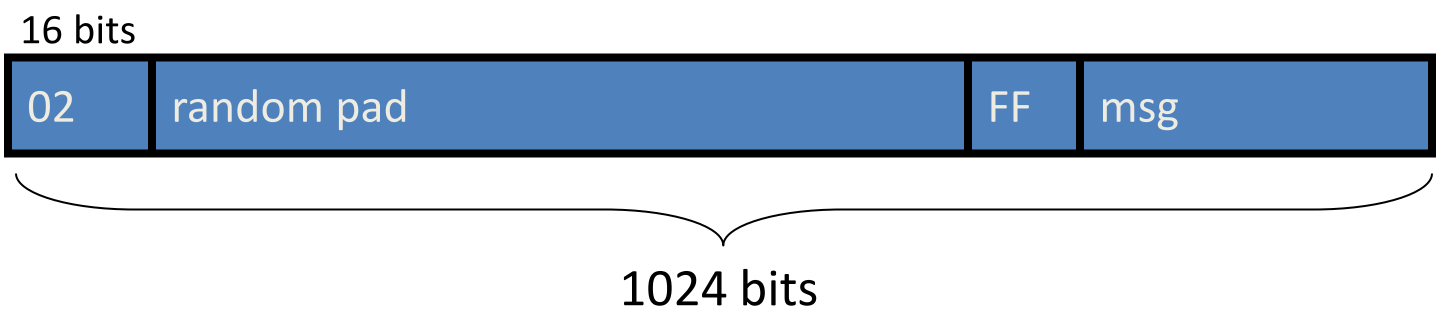
- Main question:
  - How should the preprocessing be done?
  - Can we argue about security of resulting system?

In practice

- Public key encryption schemes are rarely used to actually encrypt messages
- They are usually used to encrypt a symmetric key
- Only
  - RSA-PKCS# 1 v1.5 and
  - RSA-OAEP

can be considered as traditional public key encryption algorithms

PKCS#1 V1.5

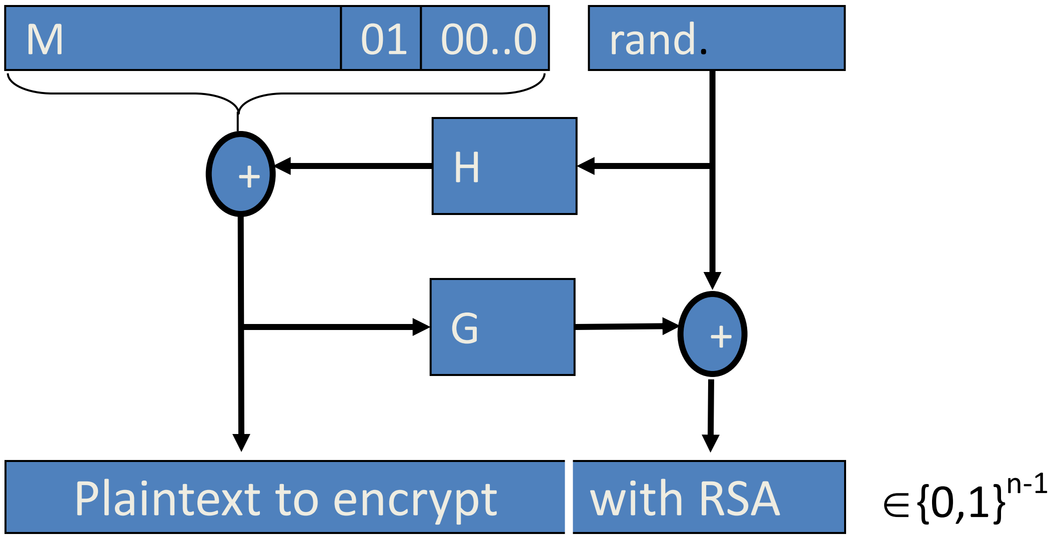


- Resulting value is RSA encrypted.
- Widely deployed in web servers and browsers. used in the SSL/TLS protocol extensively
- no modern security proof

PKCS#1 V2.0 - OAEP

- New preprocessing function: OAEP (BR94).

Check pad
on decryption.
Reject CT if invalid.



- Thm: RSA is trap-door permutation ⇒ OAEP is CCS
  when H,G are "*random oracles*".
- In practice: use SHA-1 or MD5 for H and G.

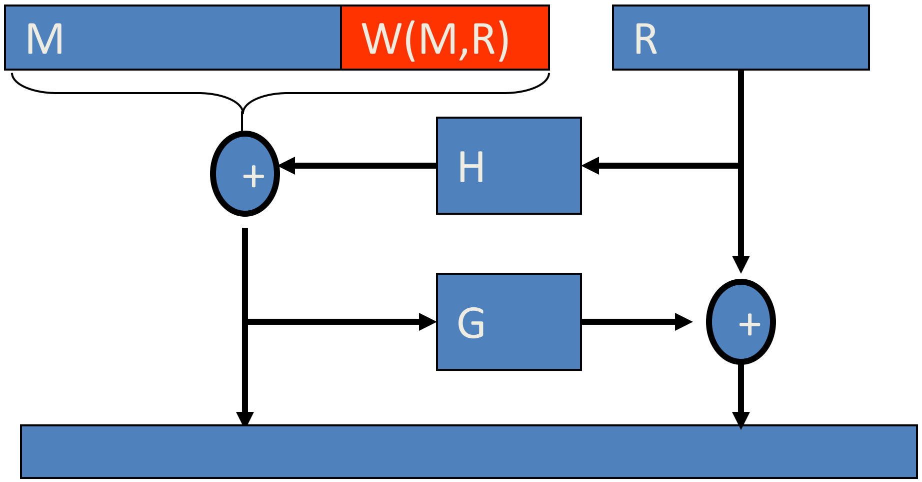
PKCS#1 V2.0 - OAEP

- The preferred method of using the RSA primitive to encrypt a *small* message
- provably secure in the random oracle model
- SHA-2/SHA-3 for future applications

OAEP Improvements

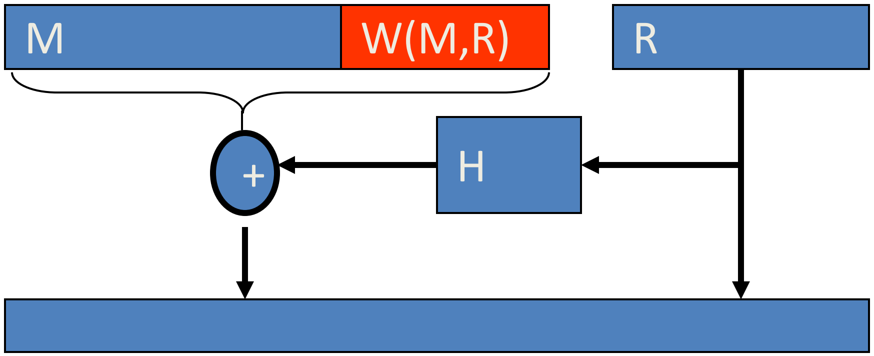
- OAEP+: (Shoup'01)

  ∀ trap-door permutation F
  F-OAEP+ is CCS when
  H,G,W are "*random oracles*".



- SAEP+: (B'01)

  RSA trap-door perm ⇒
  RSA-SAEP+ is CCS when
  H,W are "*random oracle*".



Key lengths

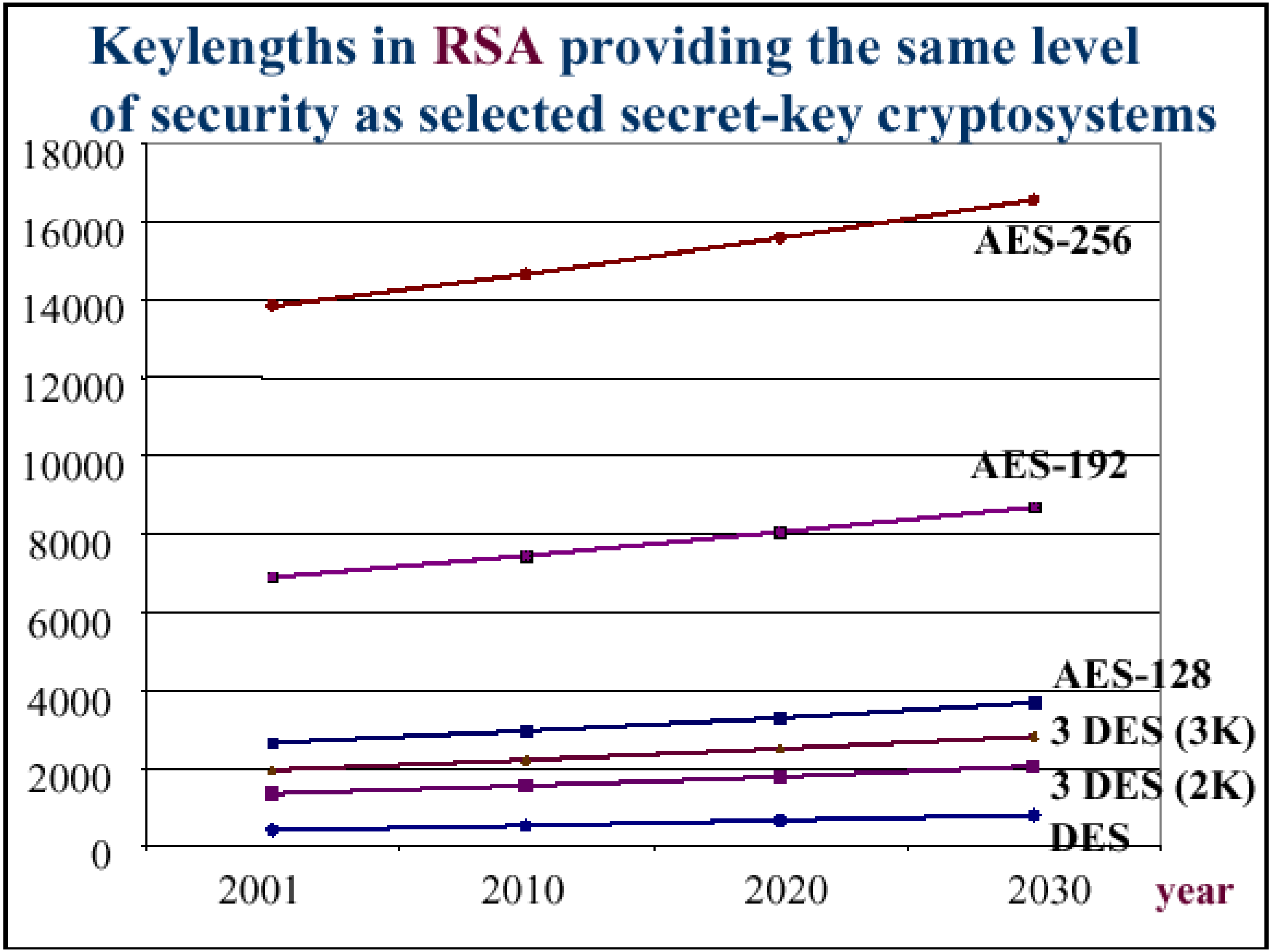
- Security of public key system should be comparable to security of block cipher.

NIST:

<u>Cipher key-size</u>	<u>Modulus size</u>
$\leq$ 64 bits	512 bits.
80 bits	1024 bits
128 bits	3072 bits.
256 bits (AES)	**<u>15360</u>** bits

- High security $\Rightarrow$ very large moduli.

  Not necessary with Elliptic Curve Cryptography (more details later)



Thanks to Kris Gaj for this figure

EL GAMAL

Discrete Logarithm

- Z_n*={1,2,3,…,n-1}
- Definition. Let b $\in Z_n$* . The order of b is the smallest positive integer satisfying $b^e \equiv 1 \pmod{n}$.
- Z_p* = <α>, i.e. ord(α) = p-1. when n=p=prime integer
- Example
  - Z_7* = <3> $3^1=3$, $3^2=2$, $3^3=6$, $3^4=4$, $3^5=5$, $3^6=1$
  - Z_{13}* = <2> $2^1=2$, $2^2=4$, $2^3=8$, $2^4=3$, $2^5=6$, $2^6=12$, $2^7=11$, $2^8=9$, $2^9=5$, $2^{10}=10$, $2^{11}=7$, $2^{12}=1$

Discrete Logarithm

- If g is a generator of Z_n^*, then for all y there is a unique x (mod $\phi(n)$) such that
  - ***$y = g^x \bmod n$***
- This is called the discrete logarithm of y and we use the notation
  - ***$x = \log_g(y)$***
- The discrete logarithm is conjectured to be hard as factoring.
- Example
  - $Z_{13}^* = \langle 2 \rangle$ $2^1=2$, $2^2=4$, $2^3=8$, $2^4=3$, $2^5=6$, $2^6=12$, $2^7=11$, $2^8=9$, $2^9=5$, $2^{10}=10$, $2^{11}=7$, $2^{12}=1$
  - $\text{Log}_2(5) = 9$.

ElGamal

- Invented in 1985
- Designed by Dr. Taher Elgamal
- Based on the difficulty of the discrete log
- problem
- No patents
- Digital signature and Key-exchange variants

- Works over various groups
- ✓ Z_p,
- ✓ Multiplicative group $GF(p^n)$,
- ✓ Elliptic Curves

ElGamal Public-key Cryptosystem

- SetUp (Ring of integers)

- Choose a prime number p (selected so that it is hard to solve the discrete log problem)
- All operations in the ring Z^*_p

1. Randomly select a generator g for Z^*_p
2. Randomly select an element $a \in Z^*_p$
3. Compute $\beta = g^a \bmod p$

➢ Public Key: (g, β) and the prime p (some description of the ring)

➢ Private Key: a

ElGamal Public-key Cryptosystem

- Encryption
- Encryption of the message m

- Randomly select an element $k \in Z^*_p$
- Compute the ciphertext:
- $C = (c_1, c_2)$
  $= (g^k, m * \beta^k)$
- Delete k!

- Decryption of C
- Decryption of the ciphertext $C = (c_1, c_2)$
- Compute
- $c_2 * (c_1^a)^{-1} = (m * \beta^k) * (g^{ka})^{-1} = m * \beta^k * (\beta^k)^{-1} = m$

- Randomly select an element $k \in Z^*_p$

Known k, => β^k =>c2/ β^k =m1

- Repeat k
- $C1 = (c_1, c_2)$
  $= (g^k, m1 * \beta^k)$
- $C1 = (c_1, c'_2)$
  $= (g^k, m2 * \beta^k)$
- $c_2 / c'_2 = m1/m2$

ElGamal: Example

- SetUp (Ring of integers)
- Choose a prime number p=11.
  - $g = 2$
  - $a = 8$
  - Compute $\beta = 2^8 \pmod{11} = 3$
- Public key: (2,3), Z_{11}^*
- Private key: 8

- Encryption:
- For m=7, k=4, we compute $C = (2^4, 7 * 3^4) = (5, 6)$

- Decryption:
- $6 * (5^8)^{-1} = 6 * 4^{-1} = 6 * 3 \pmod{11} = 7$

RSA vs El GAMAL

- A disadvantage of ElGamal encryption is that there is message expansion by a factor of 2. That is, the ciphertext is twice as long as the corresponding plaintext.
- El Gamal is by design probabilistic.
- RSA is more mature and has better marketing
- El Gamal can achieve much better performance.


Questions?

Fermat's Theorem

- $a^{p-1} \bmod p = 1$
  - where p is prime and gcd(a,p)=1
- also known as Fermat’s Little Theorem
- useful in public key and primality testing

Euler Totient Function $\varphi(n)$

- when doing arithmetic modulo n
- **complete set of residues** is: 0..n-1
- **reduced set of residues** is those numbers (residues) which are relatively prime to n
  - eg for n=10,
  - complete set of residues is {0,1,2,3,4,5,6,7,8,9}
  - reduced set of residues is {1,3,7,9}
- number of elements in reduced set of residues is called the **Euler Totient Function φ(n)**

Euler's Theorem

A generalisation of Fermat's Theorem

- $a^{\varphi(N)} \bmod N = 1$
  - where gcd(a,N)=1

eg.

- $a=3; n=10; \varphi(10)=4;$
- hence $3^4 = 81 = 1 \bmod 10$
- $a=2; n=11; \varphi(11)=10;$
- hence $2^{10} = 1024 = 1 \bmod 11$

Why RSA Works

- because of Euler's Theorem:
- $a^{\varphi(N)} \bmod N = 1$
  - where gcd(a,N)=1
- in RSA have:
  - N=p.q
  - $\varphi(N)=(p-1)(q-1)$
  - carefully chosen e & d to be inverses mod $\varphi(N)$
  - hence $e*d=1+k.\varphi(N)$ for some k
- hence :
  $C^d = (M^e)^d = M^{1+k.\varphi(N)} = M^1.(M^{\varphi(N)})^k = M^1.(1)^k = M^1 = M \bmod N$