RSA

PHONG NGUYEN

http://www.di.ens.fr/~pnguyen

WHO? SINCE 1977

- Rivest

- Shamir

- Adleman

IDEAS BEHIND RSA

- Symmetric encryption studies permutations over blocks of size a power of two (64 or 128 bits).

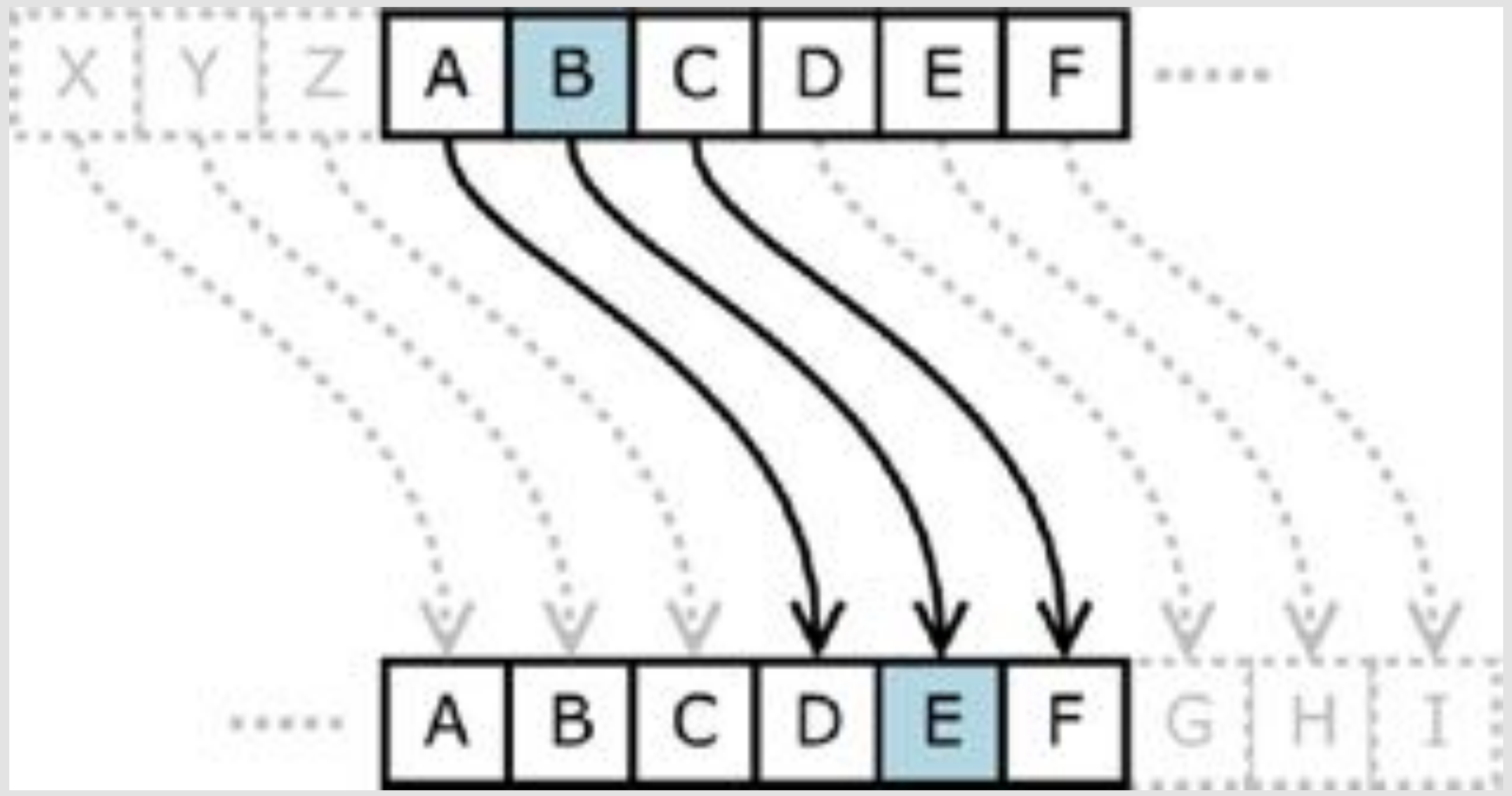
- RSA looks at permutations over the set {0,1,...,N-1} of integers modulo N=pq where p and q are two large secret primes. This set is denoted $\mathbf{Z}/N\mathbf{Z}$.

PERMUTATIONS

- A permutation is a function f mapping a set into itself:
  - f(x)=f(y) if and only if x=y
  - For every image y, there is x such that y=f(x).

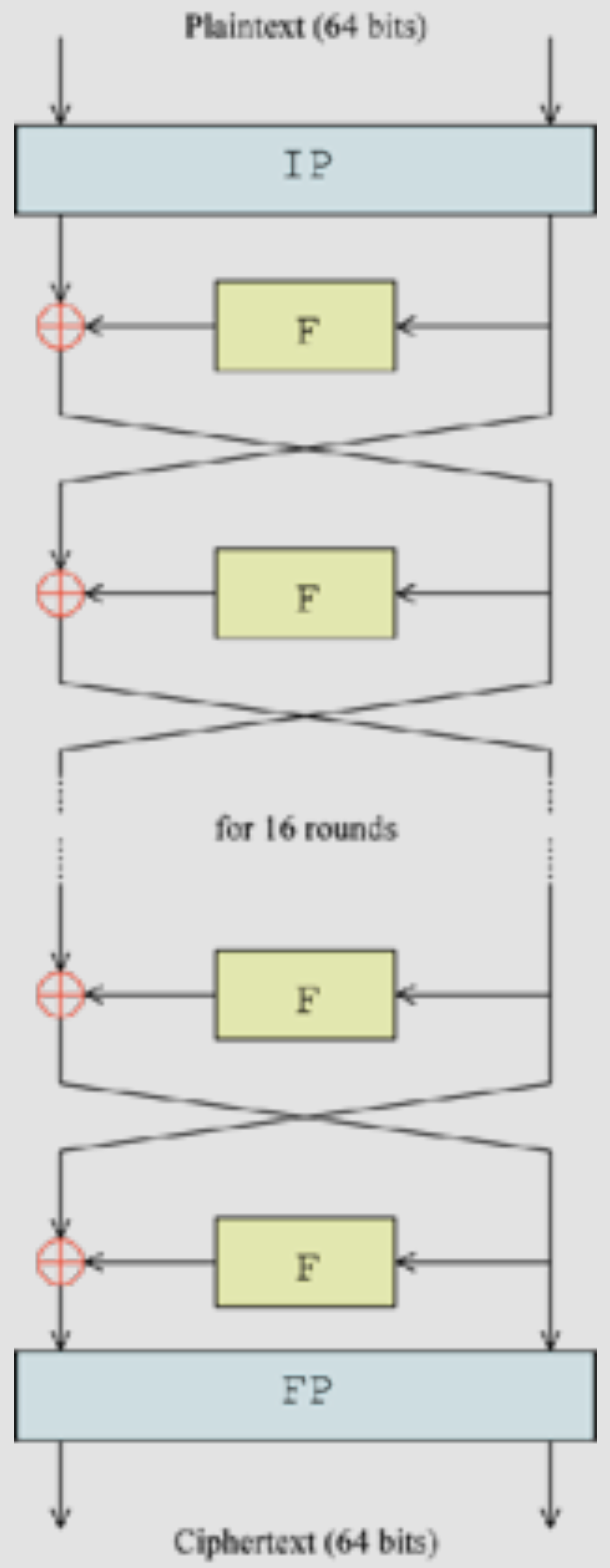
- A permutation is invertible.

- Ex: Caesar's encryption with the alphabet {A,B,C,…,Z}



PERMUTATIONS AND BLOCK CIPHERS

- A block cipher is a family of permutations
- Each 56-bit DES secret key defines a permutation over 64-bit numbers {0,1,2,…, 2^{64}-1}.



THE MATHEMATICS OF RSA

PRIME NUMBERS

- An integer $n > 1$ is prime if and only if there is no divisor d of n such that $1 < d < n$.
- Ex: 2, 3, 5, 7, 11, etc.
- Th. [Euclid]: there are infinitely many prime numbers.
- But we can count the number of prime numbers up to a bound.

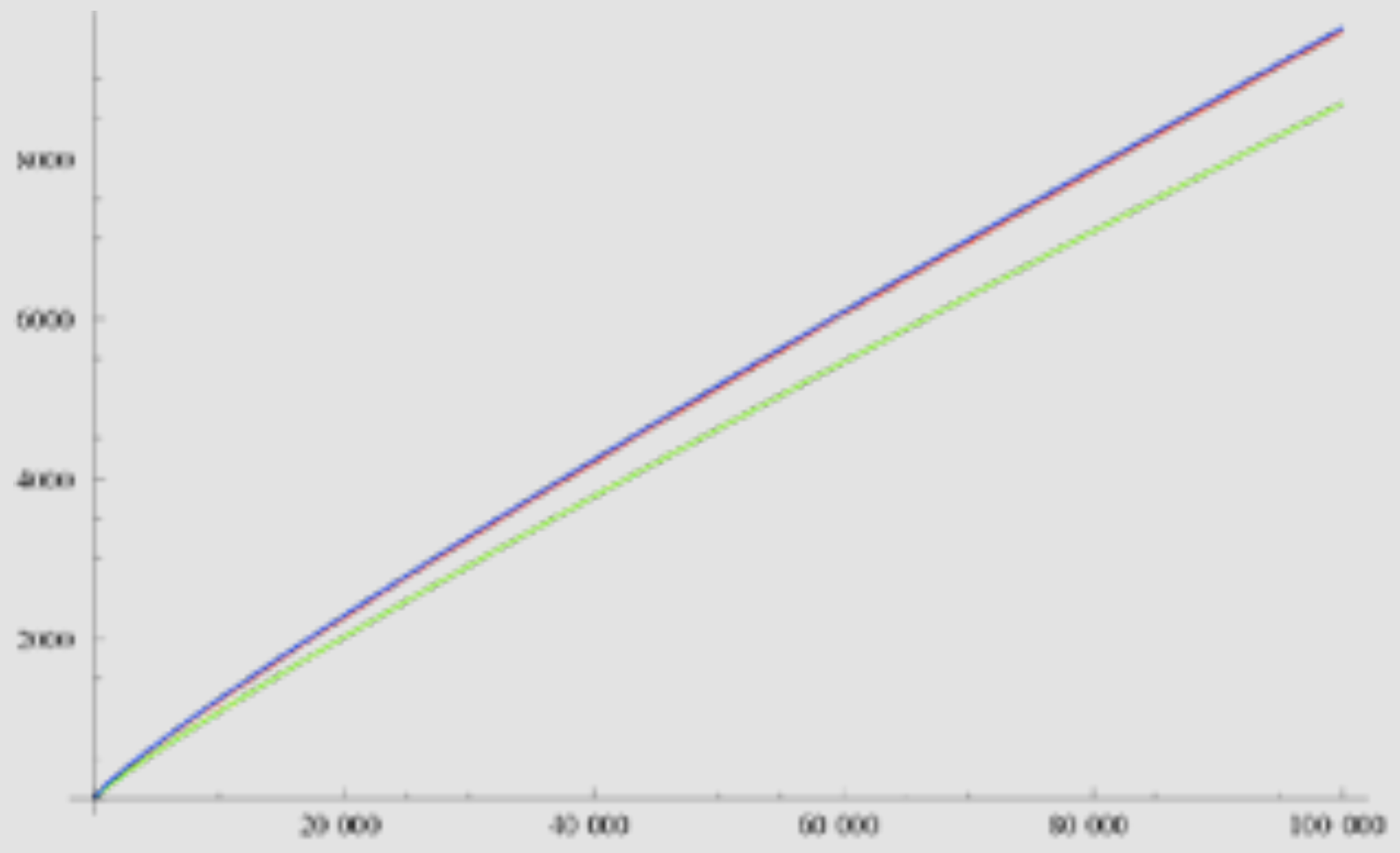
EXEMPLE

```
➤ N=1000


➤ nb = 0
➤ print("Les nombres premiers entre 1 et ",N," sont :")
➤ for i in range(2,N):
➤   premier = 1
➤   for j in range(2,i):
➤     if (i % j) == 0:
➤       premier = 0
➤   nb += premier
➤   if (premier > 0):
➤     print(i)
➤ print("Nombre de nombres premiers = ",nb)
```

COUNTING PRIME NUMBERS

- Generate large random primes by testing primality of random numbers.
- Theorem [Hadamard, de la Vallée - Poussin, 1896]: The number of prime numbers between 1 and N is asymptotically $\sim \dfrac{N}{\log N}$



Number of prime numbers

PRIMALITY TESTING

- Input: a number n
- Output: is n prime or not?
- [AKS02]: There is a deterministic poly-time algorithm to decide primality testing.
- In practice, randomized poly-time algorithms are much more efficient. They can make mistakes, but the error probability can be made arbitrarily small.

TESTS AVEC PUISSANCE DE 2

- Si p est premier, alors 2^{p-1} vaut 1 modulo p. Pourquoi ?
- Mais si p n'est pas premier, est-il possible que 2^{p-1} vale aussi 1 modulo p ?
  - Entre 1 et 1000, les faux nombres premiers sont : [341, 561, 645]

KEY GENERATION: PRIMALITY TESTING

- Theorem [AKS2002]: PRIMES can be decided in deterministic polynomial time.
    - ➤ Input: integer N
    - ➤ Output: decide if N is prime or not

- In practice, one prefers randomized polynomial-time algorithms (e.g. Miller-Rabin), based on modular exponentiations:
    - ➤ If the answer is "composite", the input number is composite.
    - ➤ Otherwise, if the input is not prime, the probability of answering "prime" is $\leq \frac{1}{2^k}$ where k is the number of modular exponentiations.

SAGE: PRIMALITY TESTING

- **Proved primality**: is_prime()
  - is_prime(389)
    - True
  - is_prime(2000)
    - False
- **Probabilistic primality**: is_pseudoprime()
- Prime generation: next_prime(n) returns the smallest ≥n.
  - next_prime(n) returns the smallest ≥n.
  - random_prime(n) returns a random prime < n.

SAGE: PRIMALITY TESTING

- **Experimentally, what is the growth of the running time of random_prime() ?**

PRIME MYSTERIES

- Prime numbers are the most studied object in mathematics.
- There remain many mysteries surrounding prime numbers:
  - It is unknown if there are infinitely many twin primes: pairs (p,q) of prime numbers such that q-p=2.
  - How « random » are prime numbers?

COMPOSITE NUMBERS

- Th. [Factoring]: Any number $n > 1$ can be decomposed as a product of prime numbers, and this decomposition is unique up to permutation.
- Ex: 15=3x5 and 70 = 2x5x7
- A random number can be factored more or less easily, but not RSA numbers of the form N=pq where p and q are two random primes of roughly the same size.
- Sage: factor(345)

FACTORING ALGORITHMS

- ➤ Input: Composite N
- ➤ Output: A non-trivial factor of N

- The naive algorithm costs $\sqrt{N}$ poly-time operations.

- The best classical algorithm is the number field sieve:

  its heuristic run time is subexponential $(\log N)^{1.923\left(\frac{\log N}{\log\log N}\right)^{1/3}}$

  Factoring record

  - RSA numbers (2020): 829 bits = 250 digits using 2700 core-years on 2.1GHz = 2^{67} clock cycles.
  - Special numbers (2014): $2^{1199}-1$ = 361 digits

- A quantum computer can factor in polynomial time [Shor1994]

COMPARISONS

- The X-box uses an RSA-2048 number.
- VIGIK uses an RSA-1024 number.
- A random RSA-512 number can be factored on the Amazon cloud for $\leq$ 300$ and $\leq$ 2 days.

MODULAR ARITHMETIC

- Definition: Let N be an integer > 1. We write a≡b (mod N) if and only if N divides (a-b).
- The symbol ≡ behaves very much like =.
  - For +, any number is invertible: ∀a ∃b s.t. a+b ≡ 0 (mod N).
  - For x, only certain numbers are invertible: given a, ∃b s.t. ab ≡ 1 (mod N) if and only if a and N are coprime (there is no integer d > 1 such that d divides both a and N, i.e. gcd(a,N)=1.
  - Euler's φ function gives the number of invertible numbers: φ(N) is the number of integers d coprime with N such that 1≤d<N.

REMARK

- If a is invertible mod N: ∃b s.t. $ab \equiv 1 \pmod{N}$, that is, $ab=1+kN$ for some integer k, so $ab-kN=1$, which shows that a and N are coprime.
- Reciprocally, if a and N are coprime, Bézout's theorem shows that there exist integers u and v such that $au+vN=1$, which shows that a is invertible mod N.

- Etienne Bézout (1730-1783)

SAGE: EULER

- Take n=923923
- Return the number of integers in {1,…,n-1} which are coprime with n.
- Hint: gcd(a,b) outputs the gcd of a and b.

```
n=923923
s=0
for k in range(1,n):
  if gcd(k,n)==1:
    s+=1
print(s)
```

SAGE: EULER

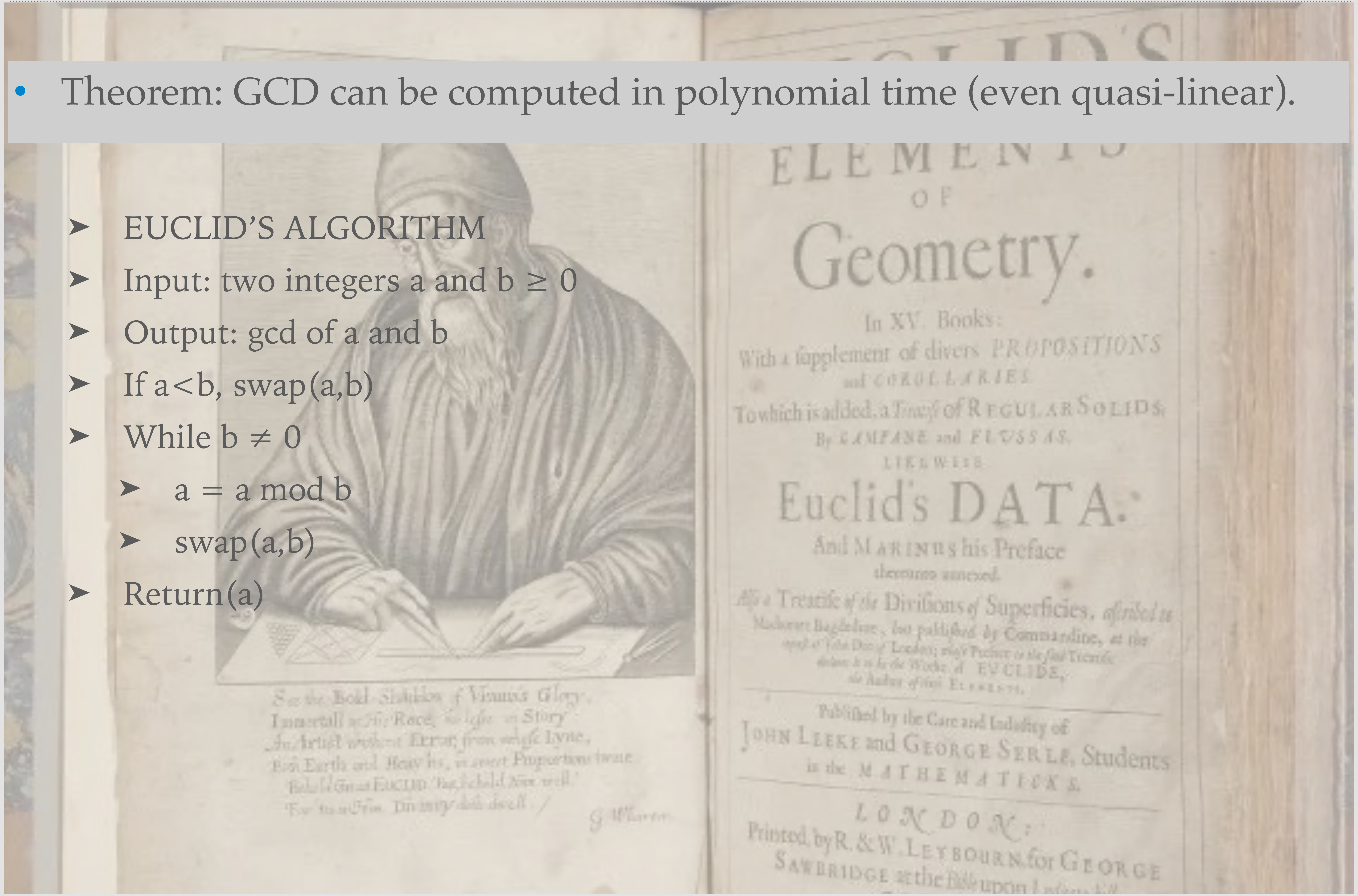
- Take n=923923

- Return euler_phi(n).

- Hint: gcd(a,b) outputs the gcd of a and b.

KEY GENERATION: GCD

- Theorem: GCD can be computed in polynomial time (even quasi-linear).



```
EUCLID'S ALGORITHM
Input: two integers a and b $\geq 0$
Output: gcd of a and b
If $a<b$, swap(a,b)
While $b \neq 0$
    $a = a \bmod b$
    swap(a,b)
Return(a)
```

ELEMENTS OF Geometry. In XV. Books: With a ſupplement of divers PROPOSITIONS and COROLLARIES. To which is added, a Treatiſe of REGULAR SOLIDS, By CAMPANE and FLUSSAS. LIKEWISE Euclid's DATA. And MARINUS his Preface thereunto annexed. Alſo a Treatiſe of the Diviſions of Superficies, aſcribed to [illegible] Publiſhed by the Care and Induſtry of JOHN LEEKE and GEORGE SERLE, Students in the MATHEMATICKS. LONDON: Printed by R. & W. LEYBOURN for GEORGE SAWBRIDGE at the Bible upon [illegible]

See the Bold Shaddow of Urania's Glory, Immortall in His Race, [illegible] Story [illegible]

RSA KEY GENERATION

RSA(1977)

- Key generation:
  - Select two large random primes p and q.
  - Let $N=pq$ then $\varphi(N)=(p-1)(q-1)$.
  - Select integers e and d s.t. $ed \equiv 1 \mod \varphi(N)$.
- Secret key: the secret exponent d.
- Public key: the public exponent e and N.

CLASSICAL EXPONENT GENERATION

- Choose a random number d coprime with $\varphi(N)$ such that $1 < d < \varphi(N)$.
- Let e be the inverse of d modulo $\varphi(N)$: $ed \equiv 1 \pmod{\varphi(N)}$.
  - (e,N) is the public key.
  - d is the private key.

SMALL EXPONENT RSA

- A popular key generation selects instead a small e such as e=3 or e=65537:

  check your certificates in your browser.

  - Choose two large random prime numbers p and q of the same bit-length until e is coprime with $\varphi(N)=(p-1)(q-1)$.
  - It selects d as the inverse of e modulo $\varphi(N)$: $ed \equiv 1 \pmod{\varphi(N)}$.

REAL-WORLD STATISTICS

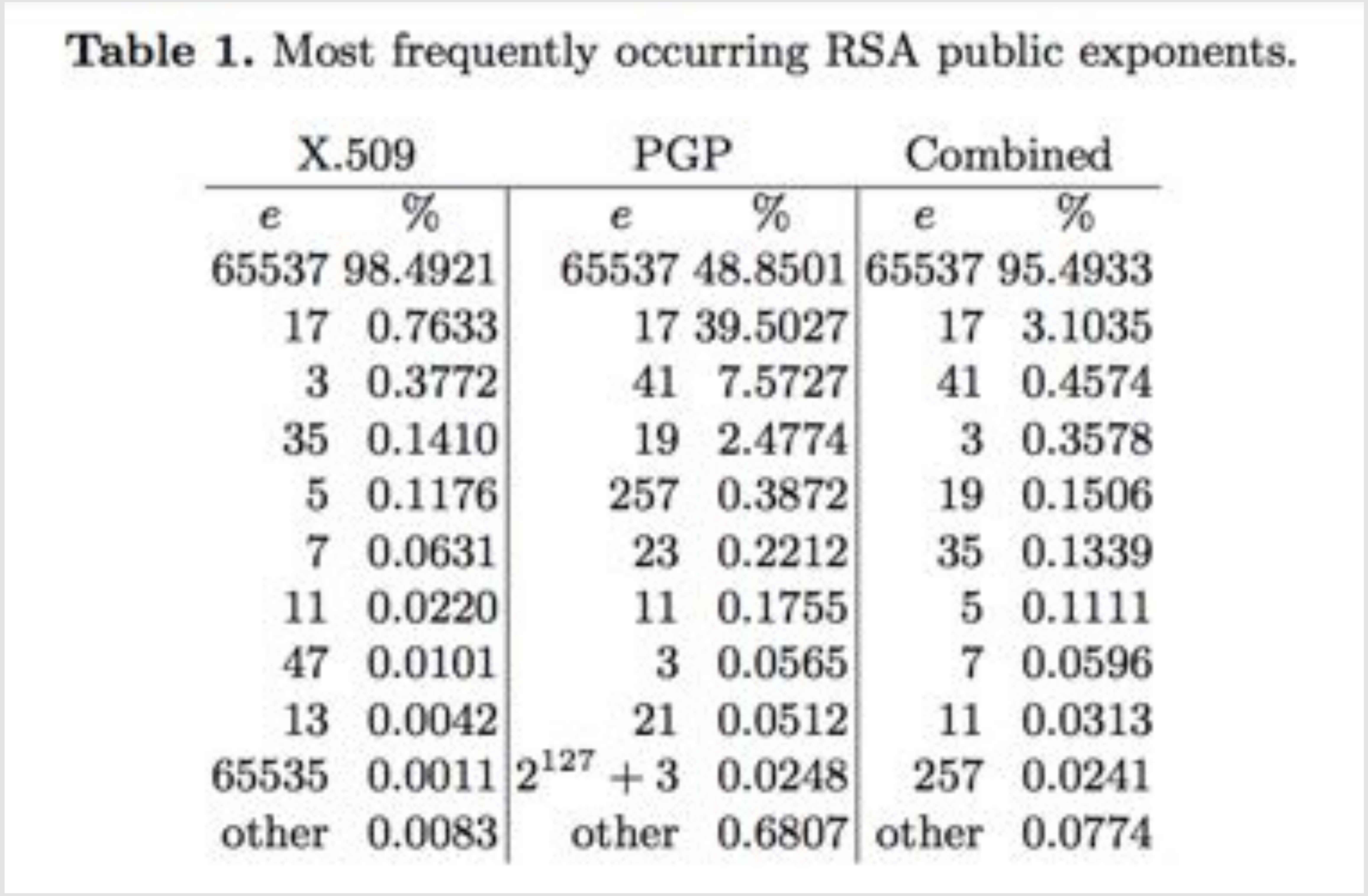


Table 1. Most frequently occurring RSA public exponents.

X.509		PGP		Combined	
e	%	e	%	e	%
65537	98.4921	65537	48.8501	65537	95.4933
17	0.7633	17	39.5027	17	3.1035
3	0.3772	41	7.5727	41	0.4574
35	0.1410	19	2.4774	3	0.3578
5	0.1176	257	0.3872	19	0.1506
7	0.0631	23	0.2212	35	0.1339
11	0.0220	11	0.1755	5	0.1111
47	0.0101	3	0.0565	7	0.0596
13	0.0042	21	0.0512	11	0.0313
65535	0.0011	$2^{127}+3$	0.0248	257	0.0241
other	0.0083	other	0.6807	other	0.0774

INVERSE MOD N

- The multiplicative inverse can be found efficiently by Euclid’s extended algorithm.
  - Euclid’s algorithm computes the gcd of two numbers a and N.
  - Euclid’s extended algorithm computes two integers (x,y) such that ax+Ny = gcd(a,N).
- Sage: 1/Mod(a,N) returns the inverse of a mod N, if it exists.

EXAMPLE

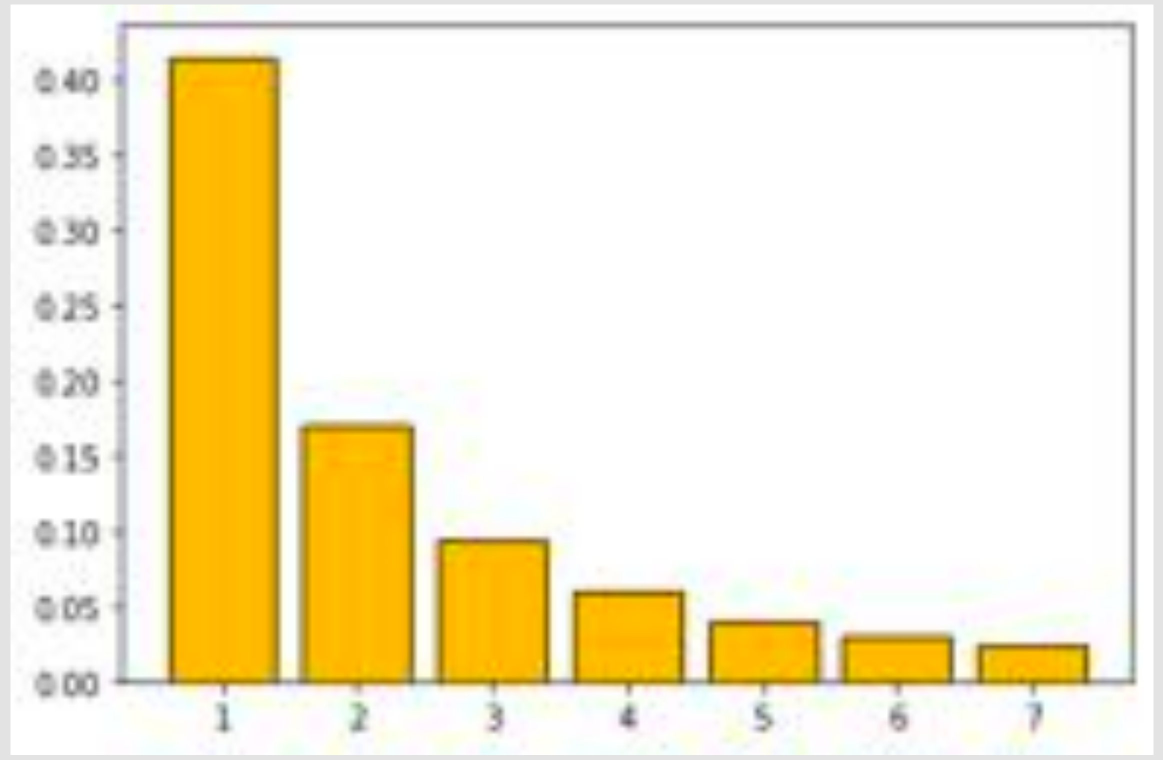
- print(1/Mod(3,31))
- print(Mod(3,31)*21)
- print(Mod(3,31)*Mod(21,31))
- print((3*21) % 31)

KEY GENERATION: MODULAR INVERSION

- Theorem: a modular inverse can be computed in polynomial time.

  - EUCLID’S EXTENDED ALGORITHM
  - Input: two integers a and $b \geq 0$
  - Output: s and t such that $as+bt = \gcd(a,b)$

```
➤ (old_r, r) := (a, b)
➤ (old_s, s) := (1, 0)
➤ (old_t, t) := (0, 1)
➤ While r ≠ 0 do
    ➤ quotient := old_r div r
    ➤ (old_r, r) := (r, old_r − quotient × r)
    ➤ (old_s, s) := (s, old_s − quotient × s)
    ➤ (old_t, t) := (t, old_t − quotient × t)
➤ Return(old_s,old_t)
```



Distribution of the quotients

SAGE

- p = random_prime(2^32)
- print("p = " ,p)
- q = random_prime(2^32)
- print("q = ",q)
- N = p * q
- phi = (p - 1)*(q - 1)
- print("N = ",N)
- print("phi(N) = ",phi)

```
e = ZZ.random_element(phi)
while gcd(e, phi) != 1:
    print(e," pas premier avec ",phi)
    e = ZZ.random_element(phi)
print("public exponent e = ",e)
```

EXECUTION

```
p =  3417738307
q =  1257596279
N =  4298134977478959653
phi(N) =  4298134972803625068
2982437488721560582  pas premier avec  4298134972803625068
2649492066977634754  pas premier avec  4298134972803625068
2575418202069102272  pas premier avec  4298134972803625068
3781171049397656138  pas premier avec  4298134972803625068
2558671472141409508  pas premier avec  4298134972803625068
1647037979566040966  pas premier avec  4298134972803625068
1557021267884204216  pas premier avec  4298134972803625068
public exponent e =  1046549980555063013
secret exponent d =  1210684402134783929
d*e mod phi =  1
```

SAGE

- d = ZZ(1/Mod(e,phi))
- print("secret exponent d = « ,d)
- print("d*e mod phi = ",mod(d * e, phi))
- print("clef publique = ",N,e," ; clef secrete = " ,d)

QUESTIONS

- If you know $N=pq$ and $\varphi(N)=(p-1)(q-1)$, can you recover p and q efficiently?
- For Small Exponent RSA we chose the public exponent e to be small, can we instead choose the secret exponent d to be small?

SMALL SECRET ATTACKS

Michael J. Wiener

- Theorem [Wiener1989]: If $q<p<2q$ and $1 \leq d \leq N^{1/4}/3$, one can recover p and q in polynomial time from (N,e).

- [BonehDurfee1999]: There is a heuristic (lattice) attack recovering p and q in polynomial time from (N,e) if $d \leq N^{0.292\ldots}$

Dan Boneh

Glenn Durfee

KEY RECOVERY

- Th: given (N,e,d), one can recover p and q efficiently.
- Finding the RSA secret key is as hard as factoring N.
- Yet, breaking RSA might be easier than factoring.

KEYSIZE

- Typically, N is 2048-bit, so p and q are 1024-bit.

ENCRYPTION

EXPONENTATION AND PERMUTATION

- **Theorem**: Let e be an integer > 0, and N=pq be an RSA modulus. The function $m \longmapsto m^e \bmod N$ is a permutation of $\mathbf{Z}/N\mathbf{Z}$ if and only if e is coprime with $\varphi(N)=(p-1)(q-1)$. In such a case, its inverse permutation is $c \longmapsto c^d \bmod N$ where d is any integer such that $ed \equiv 1 \bmod \varphi(N)$.

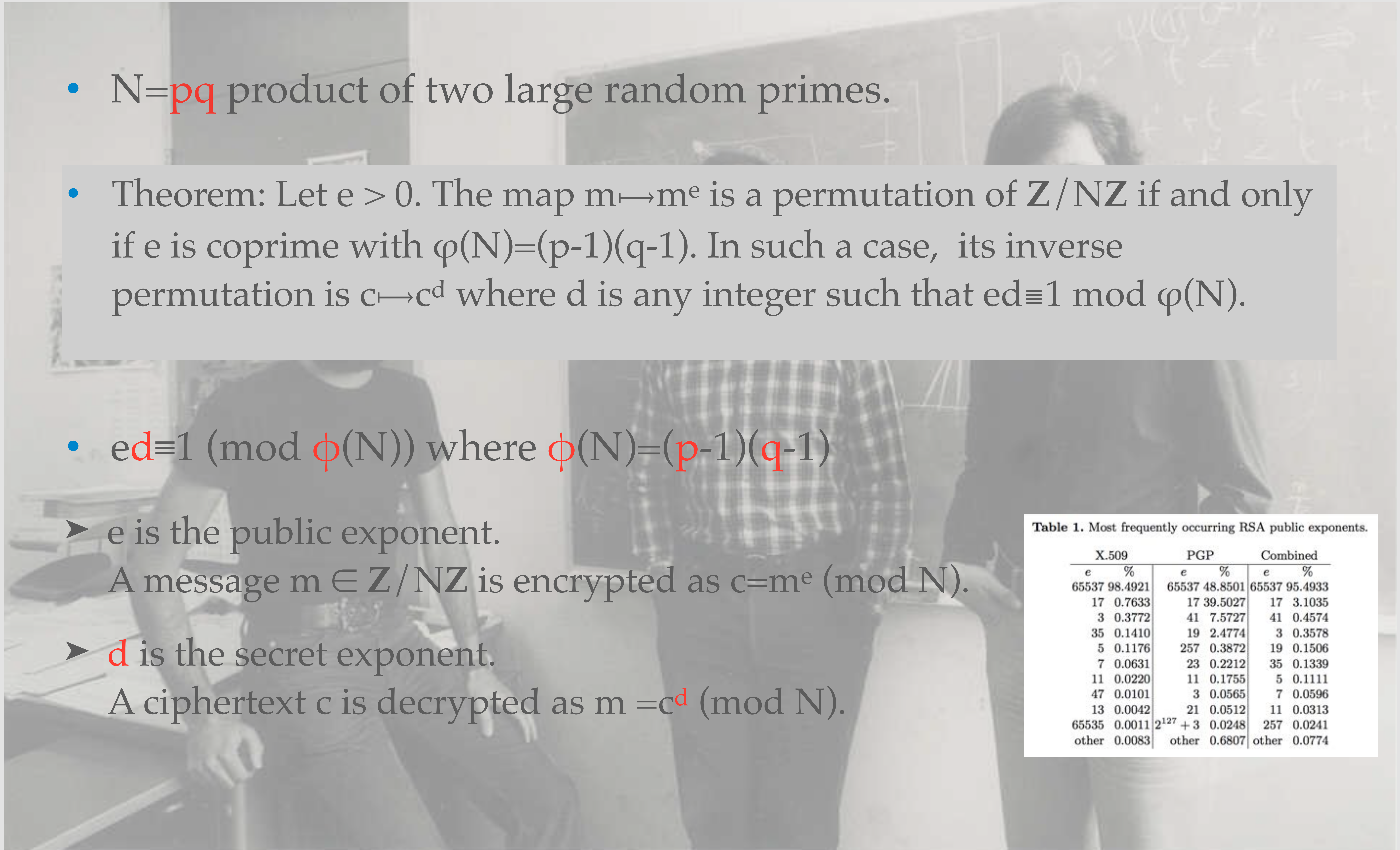
WHY?

- **Fermat's little theorem**: if p is prime then $m^p \equiv m \pmod{p}$ for any integer m.
- **Chinese remainders theorem**: if $p \neq q$, then $a \equiv b \pmod{pq}$ if and only if $a \equiv b \pmod{p}$ and $a \equiv b \pmod{q}$.

ENCRYPTION/DECRYPTION

- The message space is $\mathbf{Z}/N\mathbf{Z}$.
- A message $m \in \mathbf{Z}/N\mathbf{Z}$ is encrypted as $c=m^e \pmod N$.
- The ciphertext c is decrypted as $m = c^d \pmod N$.

THE RSA THEOREM (1978)



- $N=pq$ product of two large random primes.

- Theorem: Let $e > 0$. The map $m \longmapsto m^e$ is a permutation of $\mathbf{Z}/N\mathbf{Z}$ if and only if e is coprime with $\varphi(N)=(p-1)(q-1)$. In such a case, its inverse permutation is $c \longmapsto c^d$ where d is any integer such that $ed \equiv 1 \bmod \varphi(N)$.

- $ed \equiv 1 \pmod{\phi(N)}$ where $\phi(N)=(p-1)(q-1)$

➤ e is the public exponent.
A message $m \in \mathbf{Z}/N\mathbf{Z}$ is encrypted as $c=m^e \pmod N$.

➤ d is the secret exponent.
A ciphertext c is decrypted as $m = c^d \pmod N$.

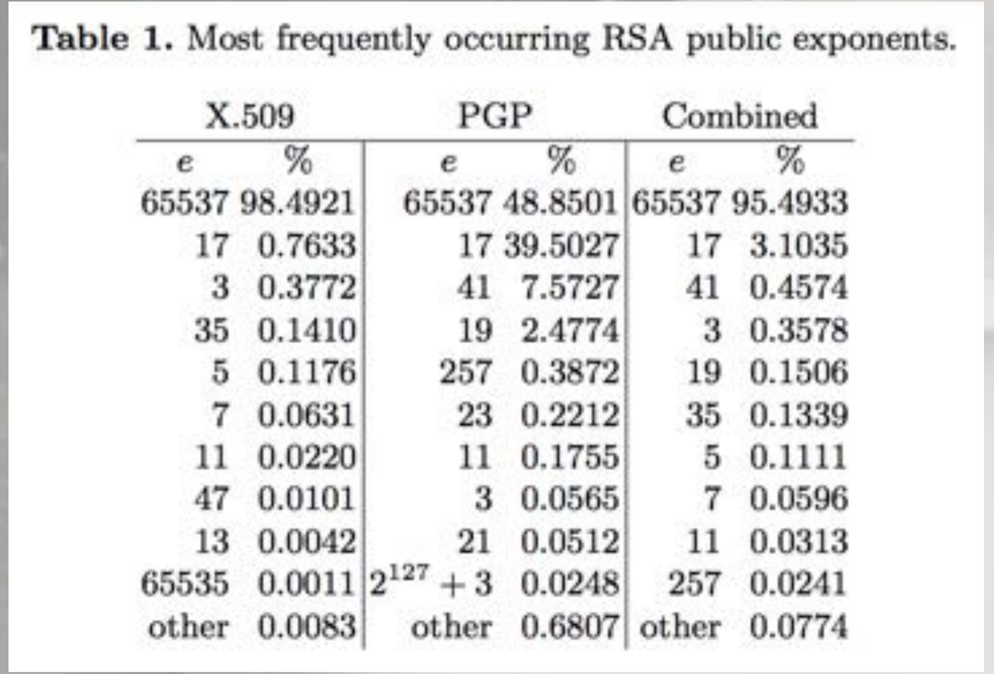


Table 1. Most frequently occurring RSA public exponents.

X.509		PGP		Combined	
e	%	e	%	e	%
65537	98.4921	65537	48.8501	65537	95.4933
17	0.7633	17	39.5027	17	3.1035
3	0.3772	41	7.5727	41	0.4574
35	0.1410	19	2.4774	3	0.3578
5	0.1176	257	0.3872	19	0.1506
7	0.0631	23	0.2212	35	0.1339
11	0.0220	11	0.1755	5	0.1111
47	0.0101	3	0.0565	7	0.0596
13	0.0042	21	0.0512	11	0.0313
65535	0.0011	$2^{127}+3$	0.0248	257	0.0241
other	0.0083	other	0.6807	other	0.0774

SAGE

- m = randrange(N)
- print("message m = ",m)
- c = power_mod(m, e, N)
- print("ciphertext c = m^e mod N = ",c)
- print("decryption(c) = c^d mod N = ",power_mod(c,d,N))

EXECUTION

```
p =  293499559
q =  2889278867
N =  848002073292519653
phi(N) =  848002070109741228
public exponent e =  610064010304828637
secret exponent d =  839899126964201069
d*e mod phi =  1
clef publique =  848002073292519653 610064010304828637  ; clef secrete =
839899126964201069
msg m =  258780777538061729
ciphertext c = m^e mod N =  262764416509344761
decryption(c) = c^d mod N =  258780777538061729
```

MODULAR EXPONENTIATION

- RSA encryption/decryption require an efficient exponentiation: there are classical algorithms.
- The main idea is:
  - $m^e = (m^{e/2})^2$ if e is even
  - $m^e = m \times (m^{e-1})$ if e is odd
- This suggests to look at the binary decomposition of the exponent.

MODULAR EXPONENTIATION

- Encryption and decryption requires an efficient modular exponentiation.
- Lemma: m^e can be computed in $2\log_2(e)$ modular multiplications.

```
SQUARE-AND-MULTIPLY
Input: $m$, $N$ and $e=e_k e_{k-1} \ldots e_0$ (in base 2)
Output: $m^e \pmod N$
$c = 1$
for $i = k$ downto $0$
    $c = c^2 \pmod N$
    if $e_i = 1$
        $c = c \times m \pmod N$
Return(c)
```

- $m^e = (m^{e/2})^2$ if e is even
- $m^e = m \times (m^{e-1})$ if e is odd

INTEGER ARITHMETIC

- How much cost a multiplication of two n-bit numbers?
  - ➤ Naïve: n^2 bit-operations
  - ➤ Karatsuba: $n^{\log_2 3} = n^{1.584\ldots}$
  - ➤ Toom-Cook: $n^{1.46\ldots}$
  - ➤ Schönhage-Strassen: $n(\log n)\log\log n$...

THE COST OF RSA (WITHOUT FFT)

	Bit-complexity
Key generation	$(\log N)^4$
Encryption with short exponent	$(\log N)^2$
Decryption	$(\log N)^3$

- Exercise: Show that decryption can be sped up with the Chinese Remaindering Theorem.

SECURITY OF RSA

SECURITY OF RSA ENCRYPTION

- Breaking RSA encryption may be easier than factoring. In fact, this basic RSA encryption is insecure:
  - it is deterministic
  - if the message is too short, it can be recovered: this is because extracting e-th roots is only hard on the average, not in every case.
- Real-life RSA transforms the message before encryption, and similarly for decryption.

SECURITY OF RSA

- Knowing $\varphi(N)=(p-1)(q-1)$ is equivalent to factoring $N=pq$.
- Knowing the secret key d is equivalent to factoring $N=pq$.

➤ Does it mean that breaking RSA is as hard as factoring?

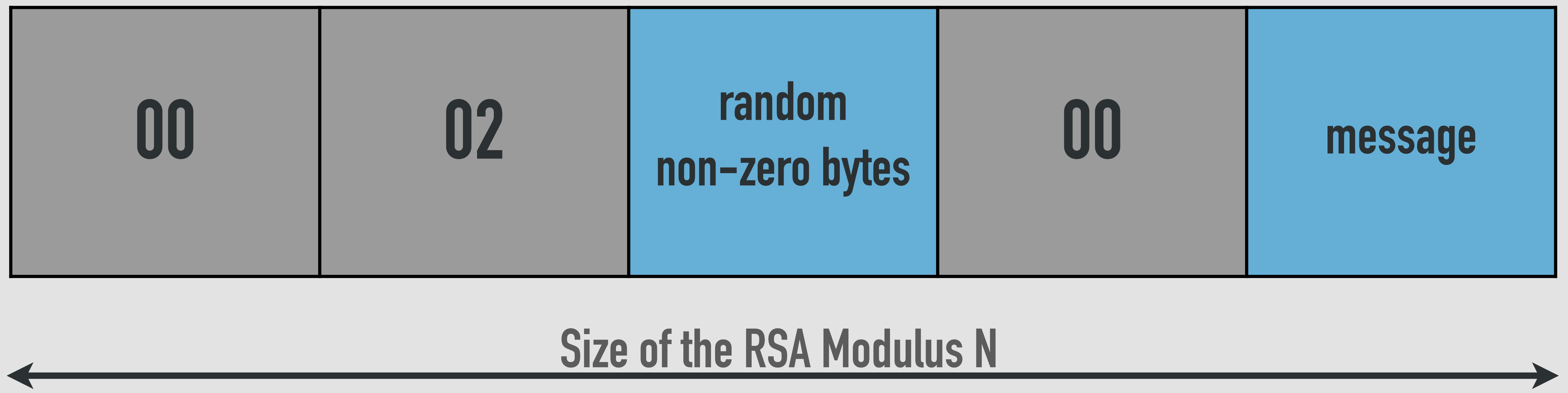
SHORT-MESSAGE ATTACK

- Assume that $e=3$, N is 2048-bit, and that we encrypt a 128-bit AES key m.
  - $c=m^e \pmod N$.
  - What is the problem?

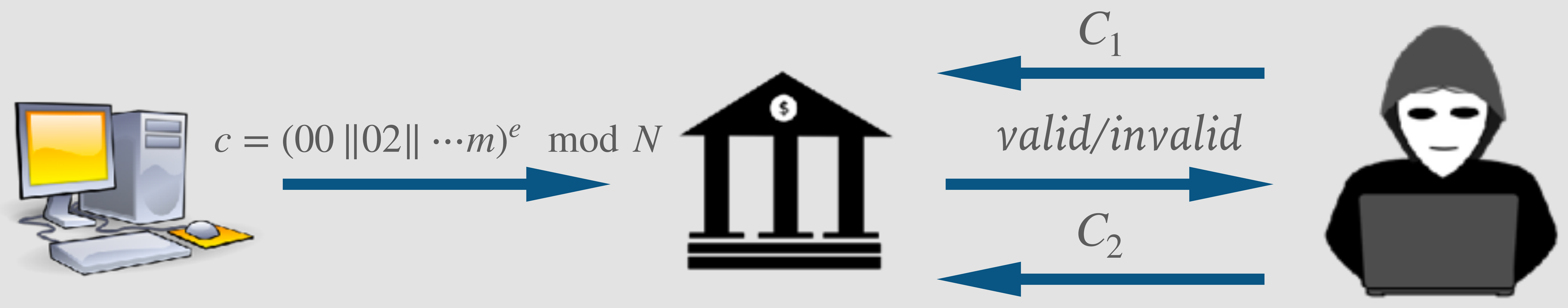
RANDOMIZING RSA

- To prevent various attacks, one preprocesses the message to make it more random, before RSA-encrypting it with the e-th power.
- After decryption with the d-th power, the preprocessing must be inverted to recover the message.
- But the preprocessing must be chosen wisely.

RSA ENCRYPTION IN SSL V3.0

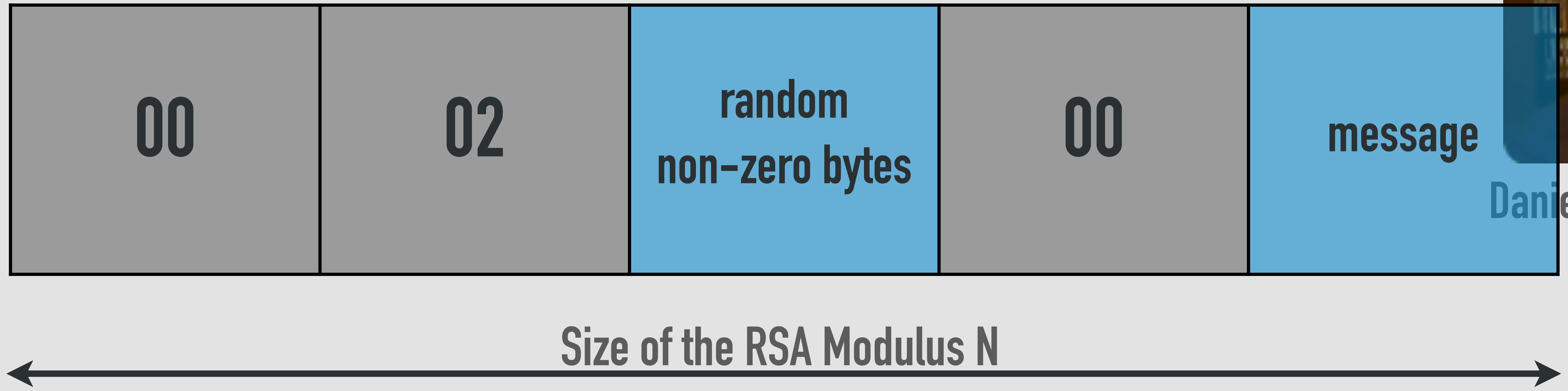


- The server checked if decrypted messages had this shape: if not, an error message was sent.



- Thus, one could know if the decryption of a ciphertext started with **00 02**.

BLEICHENBACHER'S ATTACK (1998)



Daniel Bleichenbacher

- Given a public key (N,e) and a ciphertext c=m^e (mod N).
  - Choose a random r mod N and let c′ = c r^e (mod N).
  - Ask the server if c′ is a valid ciphertext.
    - If no, pick another r..
    - If yes, we know that c'^d=mr (mod N) starts with **00 02**.
  - Repeat until enough r′s have been collected: recover m using a special algorithm.

THE ORIGINAL ATTACK

- Recall that $c=m^e \pmod N$.
- We chose random integers r_i s.t. $(mr_i) \bmod N$ starts with **00 02**.
- Instead, choose the r_i's adaptively

➤ Let r_1 be the smallest successful integer s.t. $r_1 \geq \frac{N}{00\|02\|FF\ldots FF}$

➤ Let r_2 be the smallest successful integer s.t. $r_2 > r_1$

➤ We narrow m to an intersection of intervals: see TD.

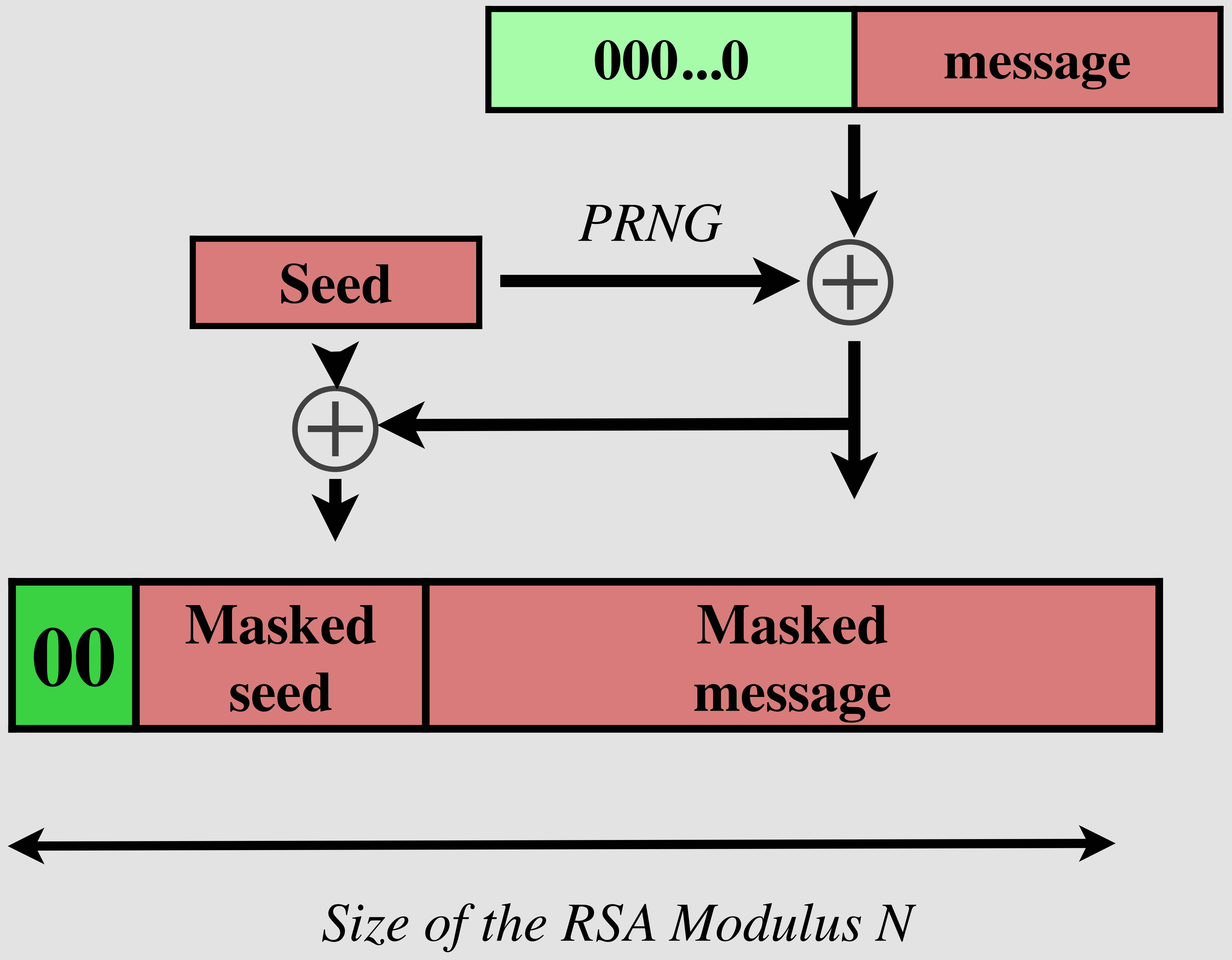
➤ The whole attack requires about a million oracle queries.

➤ In some sense, the attack is not surprising: bit-security of RSA (1996).

WHY?

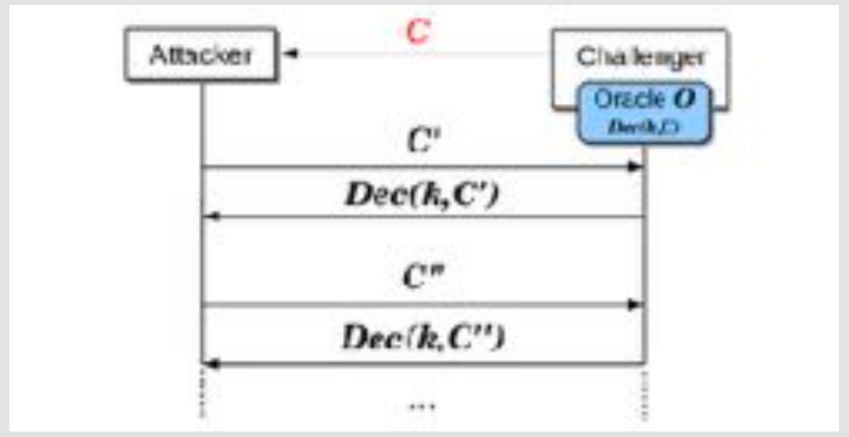
- This RSA-PKCS encryption did not satisfy the strongest security property: chosen-ciphertext security.
- Another way to say it is that it is dangerous not to have authenticated encryption. Here, anyone could submit ciphertexts to the server, including adversaries.

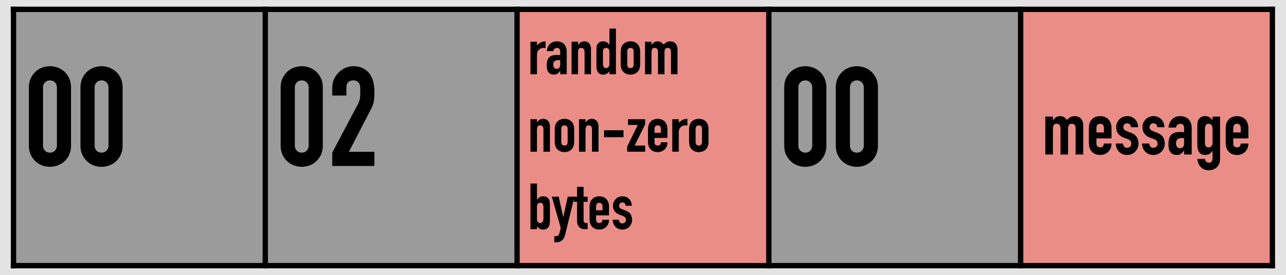
RSA-OAEP ENCRYPTION (PKCS V2.1)



SIDE-CHANNEL ATTACKS

- Bleichenbacher's chosen-ciphertext attack on SSL's RSA (1998)

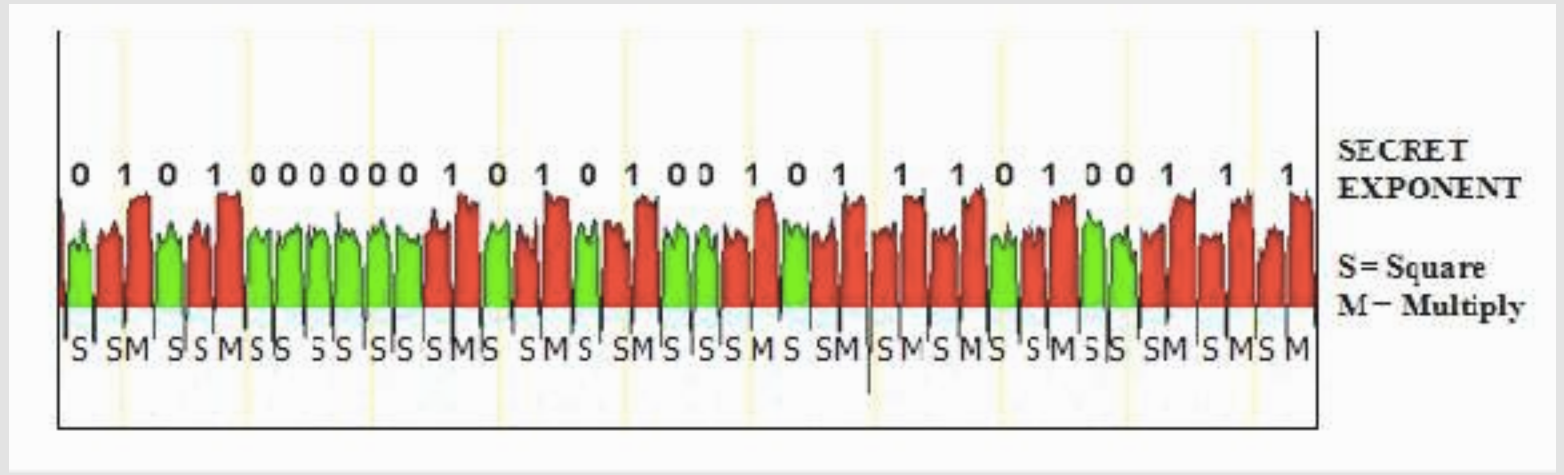




Size of the RSA Modulus N

- Timing and Power Attacks
  - ➤ Square-and-multiply depends on binary decomposition of the exponent.

```
➤ Input: m, N and $e=e_ke_{k-1}\ldots e_0$ (in base 2)
➤ $c = 1$
➤ for $i = k$ downto $0$
    ➤ $c = c^2 \pmod N$
    ➤ if $(e_i == 1)$
        ➤ $c = c \times m \pmod N$
➤ Return(c)
```



SIGNATURE

REMEMBER RSA (1977)

- Key generation:
  - Select two large random primes p and q.
  - Let $N=pq$ then $\varphi(N)=(p-1)(q-1)$.
  - Select integers e and d s.t. $ed \equiv 1 \bmod \varphi(N)$.
- Secret key: the secret exponent d.
- Public key: the public exponent e and N.

RSA SIGNATURE (1977)

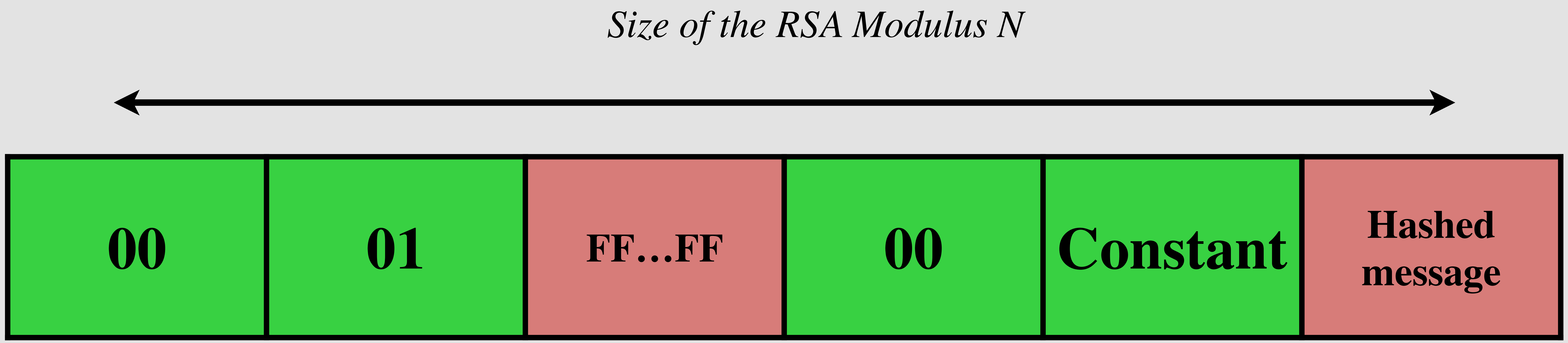
- The message space is $\mathbf{Z}/N\mathbf{Z}$.
- A message $m \in \mathbf{Z}/N\mathbf{Z}$ is signed as $s = m^d \pmod{N}$.
- The signature s is verified by checking that $m = s^e \pmod{N}$.

HASHING MESSAGES

- This basic version of RSA signatures is insecure: like for encryption, one needs to preprocess the messages before signing, using hash functions.
- This can be formalized using « provable security ».

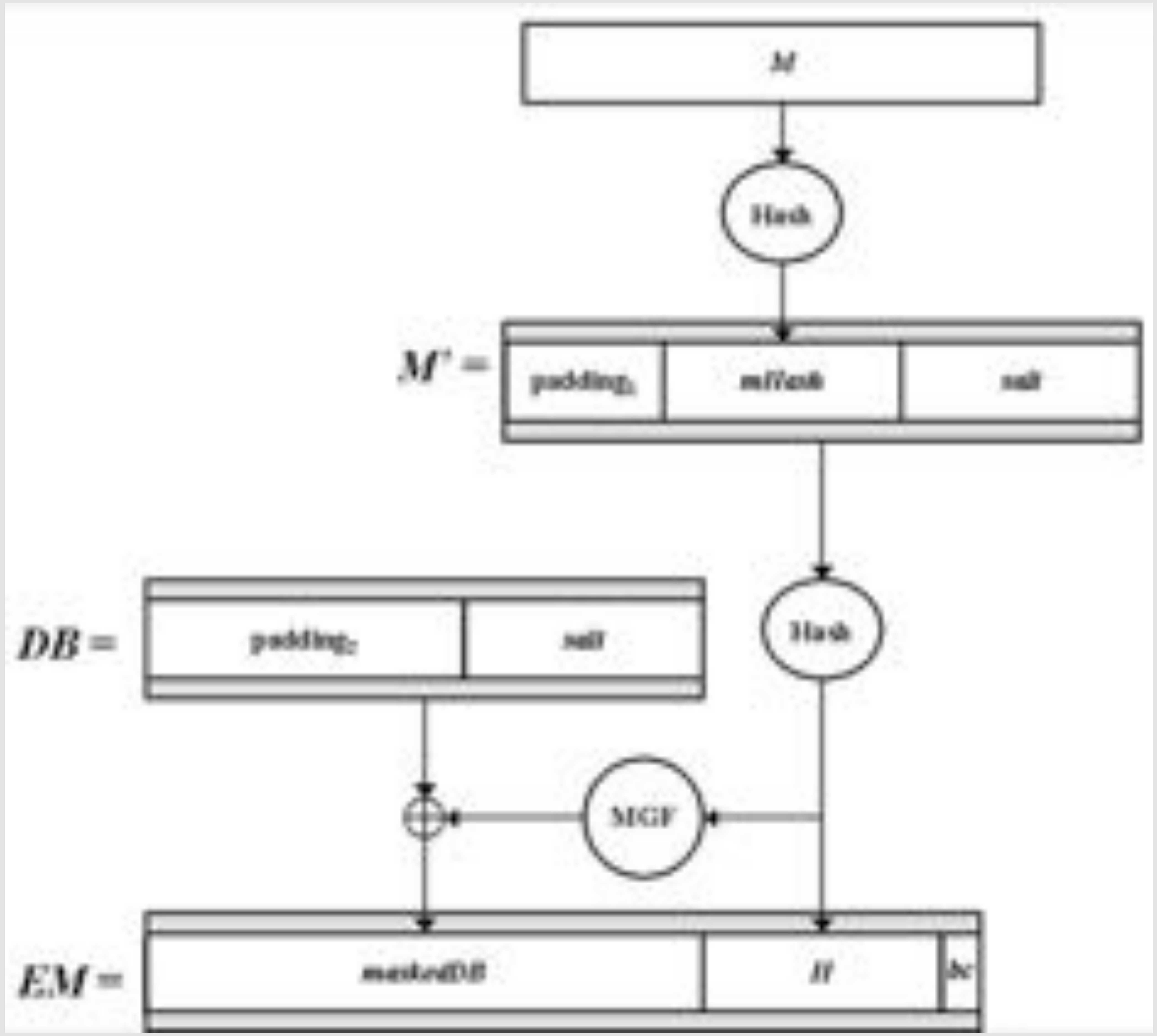
FORMER PKCS STANDARD

- The PKCS standard used to preprocess messages as follows:



- In 2006, Bleichenbacher noticed that several implementations did not fully check the shape, which allows to forge « valid » RSA signatures without knowing the secret key.

RSA-PSS STANDARD



IMPLEMENTATION: THE ZERO ATTACK

WII

- The first generation of Wii had a bug for RSA signature verification.
- The zero-signature attack allows to generate accepted signatures for "almost" every message.

WII'S RSA SIGNATURES

- The Wii uses RSA signatures to check the authenticity of DVD games.
- Each game m comes with a signature s, such that, in a simplified way:
  - SHA-1(m) $\equiv s^e \pmod{N}$.

WII'S RSA SIGNATURES

- But to check the validity of a signature, old Wiis performed the following:
  - Compute t = s^e mod N.
  - strcmp(t,SHA-1(m),up to 20 bytes)
  - If equal, then accept signature.
  - Otherwise, reject.

PROBLEM

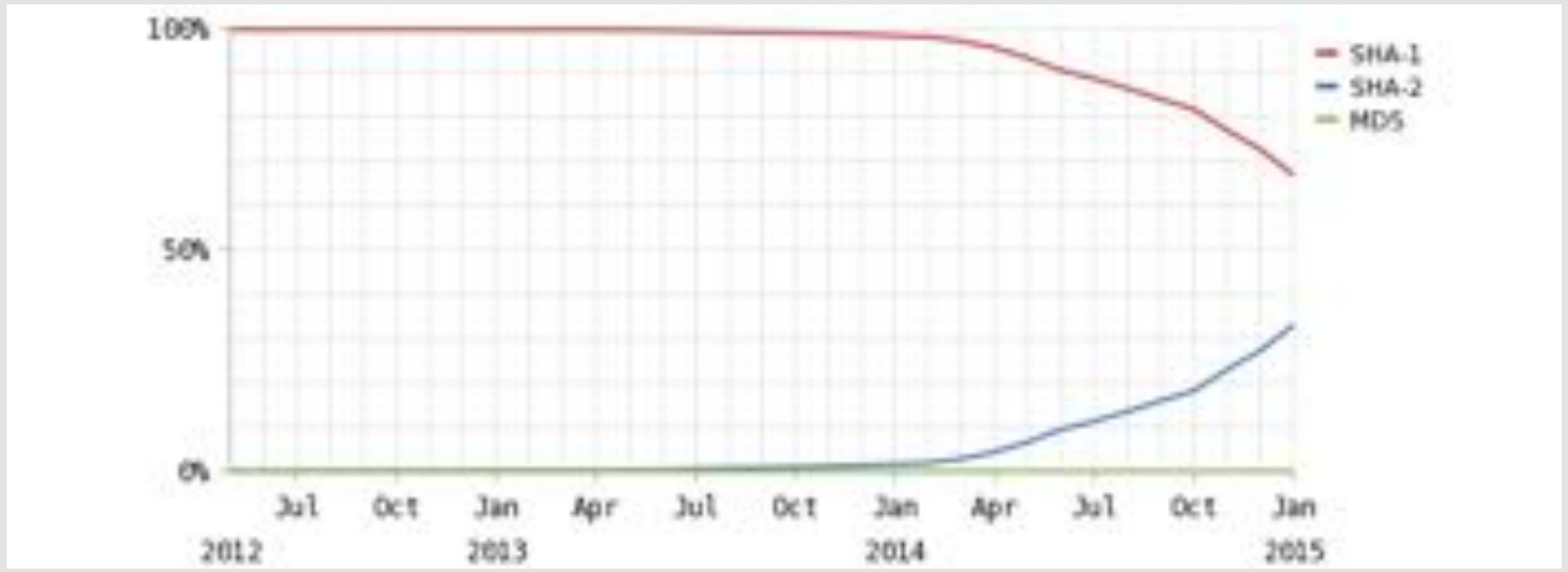
- strcmp is a string comparison:
    - it compares byte by byte until they are different, or we have reached the end of the strings, which is checked by the null character or by the size limit (20 bytes here).

BREAKING WII'S SIGNATURE

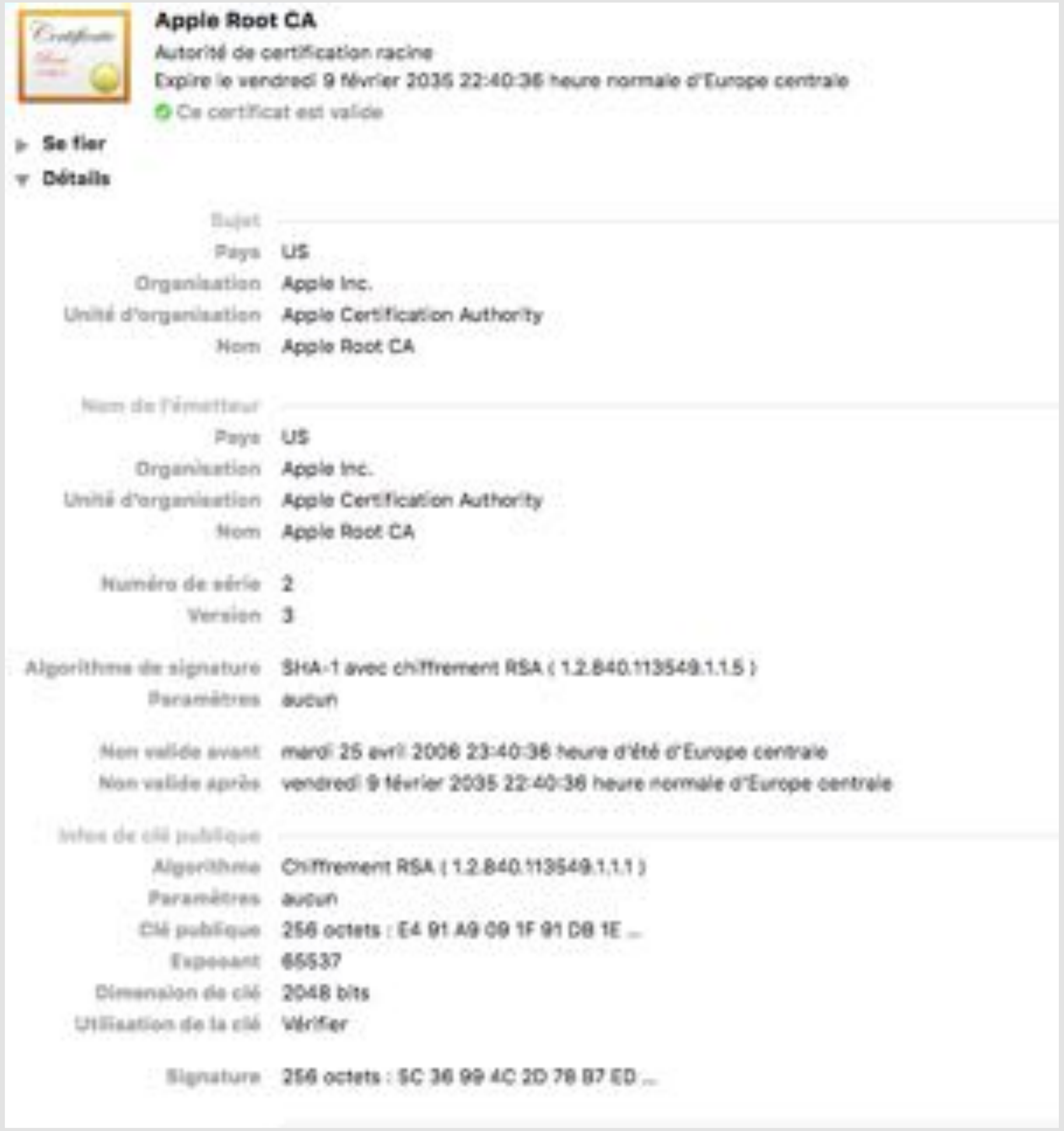
- For any message m, it is now easy to change a few bytes in m (at any position), to get a game m′ such that the zero signature s = 0 is a valid signature of m′.
  - Make sure that SHA-1(m′) starts with the 00 byte.
  - Then SHA-1(m) == (s^e mod N) as byte-strings because both are null strings!

RSA DEPLOYMENT

- The main use of signatures is certificates: a certificate allows to check the authenticity of a public key, either for encryption or signature.
- In Nov. 2015, 90% of Internet certificates used RSA signatures, and the size of the RSA modulus was 2048-bit.
- The dominant hash function for Internet certificates is now SHA-2:



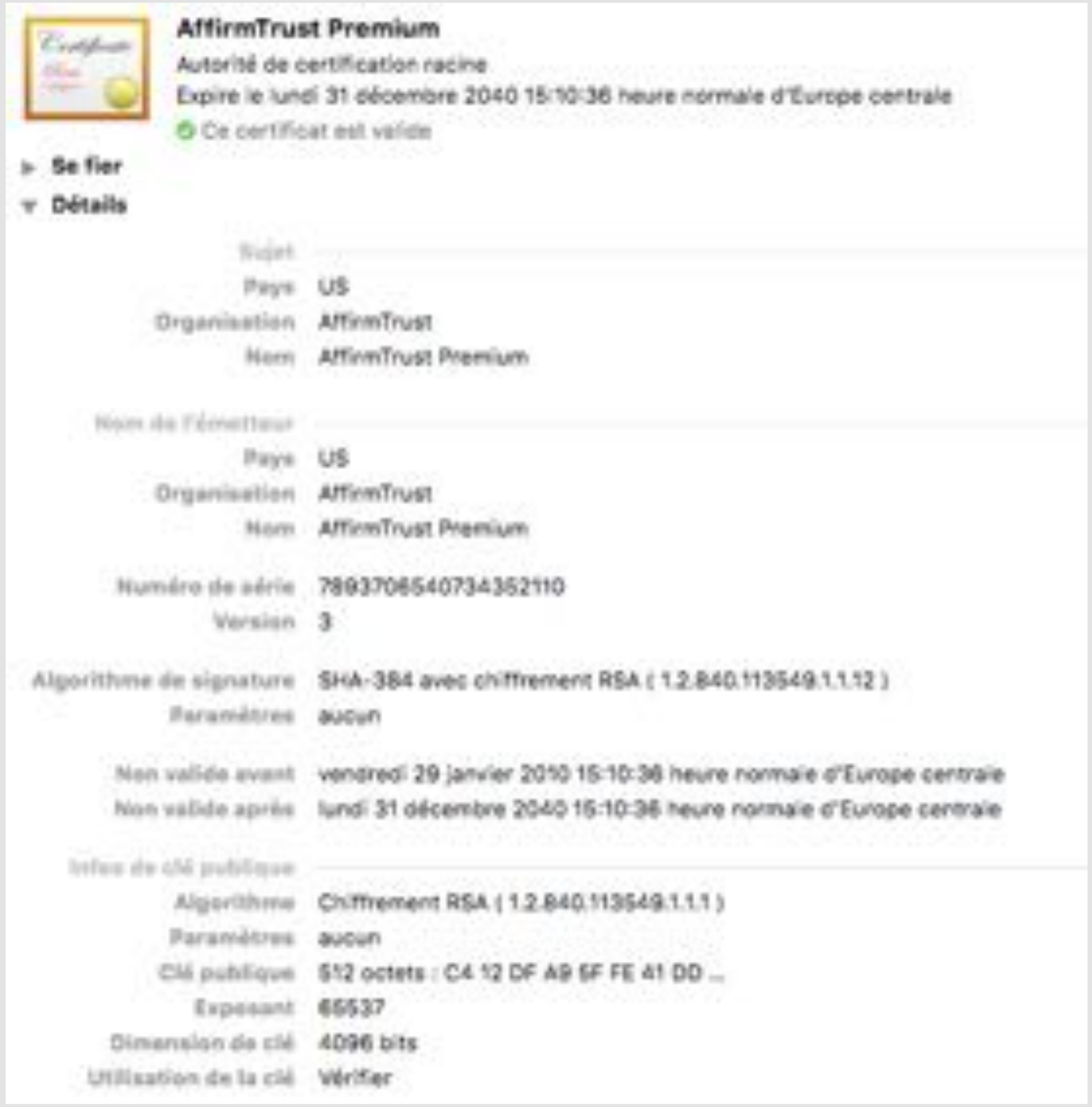
RSA CERTIFICATES



Apple Root CA
Autorité de certification racine
Expire le vendredi 9 février 2035 22:40:36 heure normale d'Europe centrale
Ce certificat est valide

▶ **Se fier**
▼ **Détails**

Sujet	
Pays	US
Organisation	Apple Inc.
Unité d'organisation	Apple Certification Authority
Nom	Apple Root CA
Nom de l'émetteur	
Pays	US
Organisation	Apple Inc.
Unité d'organisation	Apple Certification Authority
Nom	Apple Root CA
Numéro de série	2
Version	3
Algorithme de signature	SHA-1 avec chiffrement RSA (1.2.840.113549.1.1.5)
Paramètres	aucun
Non valide avant	mardi 25 avril 2006 23:40:36 heure d'été d'Europe centrale
Non valide après	vendredi 9 février 2035 22:40:36 heure normale d'Europe centrale
Infos de clé publique	
Algorithme	Chiffrement RSA (1.2.840.113549.1.1.1)
Paramètres	aucun
Clé publique	256 octets : E4 91 A9 09 1F 91 DB 1E ...
Exposant	65537
Dimension de clé	2048 bits
Utilisation de la clé	Vérifier
Signature	256 octets : 5C 36 99 4C 2D 78 B7 ED ...



AffirmTrust Premium
Autorité de certification racine
Expire le lundi 31 décembre 2040 15:10:36 heure normale d'Europe centrale
Ce certificat est valide

▶ **Se fier**
▼ **Détails**

Sujet	
Pays	US
Organisation	AffirmTrust
Nom	AffirmTrust Premium
Nom de l'émetteur	
Pays	US
Organisation	AffirmTrust
Nom	AffirmTrust Premium
Numéro de série	7893706540734352110
Version	3
Algorithme de signature	SHA-384 avec chiffrement RSA (1.2.840.113549.1.1.12)
Paramètres	aucun
Non valide avant	vendredi 29 janvier 2010 15:10:36 heure normale d'Europe centrale
Non valide après	lundi 31 décembre 2040 15:10:36 heure normale d'Europe centrale
Infos de clé publique	
Algorithme	Chiffrement RSA (1.2.840.113549.1.1.1)
Paramètres	aucun
Clé publique	512 octets : C4 12 DF A9 5F FE 41 DD ...
Exposant	65537
Dimension de clé	4096 bits
Utilisation de la clé	Vérifier

RSA KEYSIZES

- The current RSA factorization record is RSA-829.
- Some experts have estimated the security level of RSA keys, compared to symmetric keys:

Symmetric Key-length	56 bits	64 bits	80 bits	128 bits
Bit-length of the RSA modulus	512 bits	768 bits	1536 bits	6000 bits

- If we really wanted 128-bit security for RSA, we should use much bigger modulus 6000-bit than 2048-bit, the most common RSA keysize..

KEY GENERATION ISSUES

BAD RSA MODULUS

- 2022: Some Canon and Fujitsu Xerox printers used N=pq s.t. p-q is small.
  - ➤ Can be factored with Fermat's method.

- 2017: Infineon produced moduli N=pq s.t. p and q are powers of 65537 modulo the first 100 primes.

➤ Can be factored with lattice reduction techniques.

- 2012: 0.5% of TLS and 0.03% of SSH RSA keys can be factored due to shared primes.

RSA RANDOMNESS

- In 2012, two teams scanned 10 million RSA public keys on the Internet and found out severe randomness problems: they found pairs of RSA modulus (N_1,N_2) such that $\gcd(N_1,N_2)$ is non-trivial:
  - $N_1=pq_1$ and $N_2=pq_2$ and $\gcd(N_1,N_2)=p$
  - This would never have happened with correct randomness.
- In Nov. 2015, RSA secret keys were retrieved for 0.50% of TLS hosts and 0.03% of SSH hosts.
- Similarly, about 200 RSA secret keys were retrieved from Taiwan’s national « Citizen Digital Certificate » database: 2 million 1024-bit RSA keys generated by certified smart cards.ic key, either for encryption or signature.

RSA RANDOMNESS

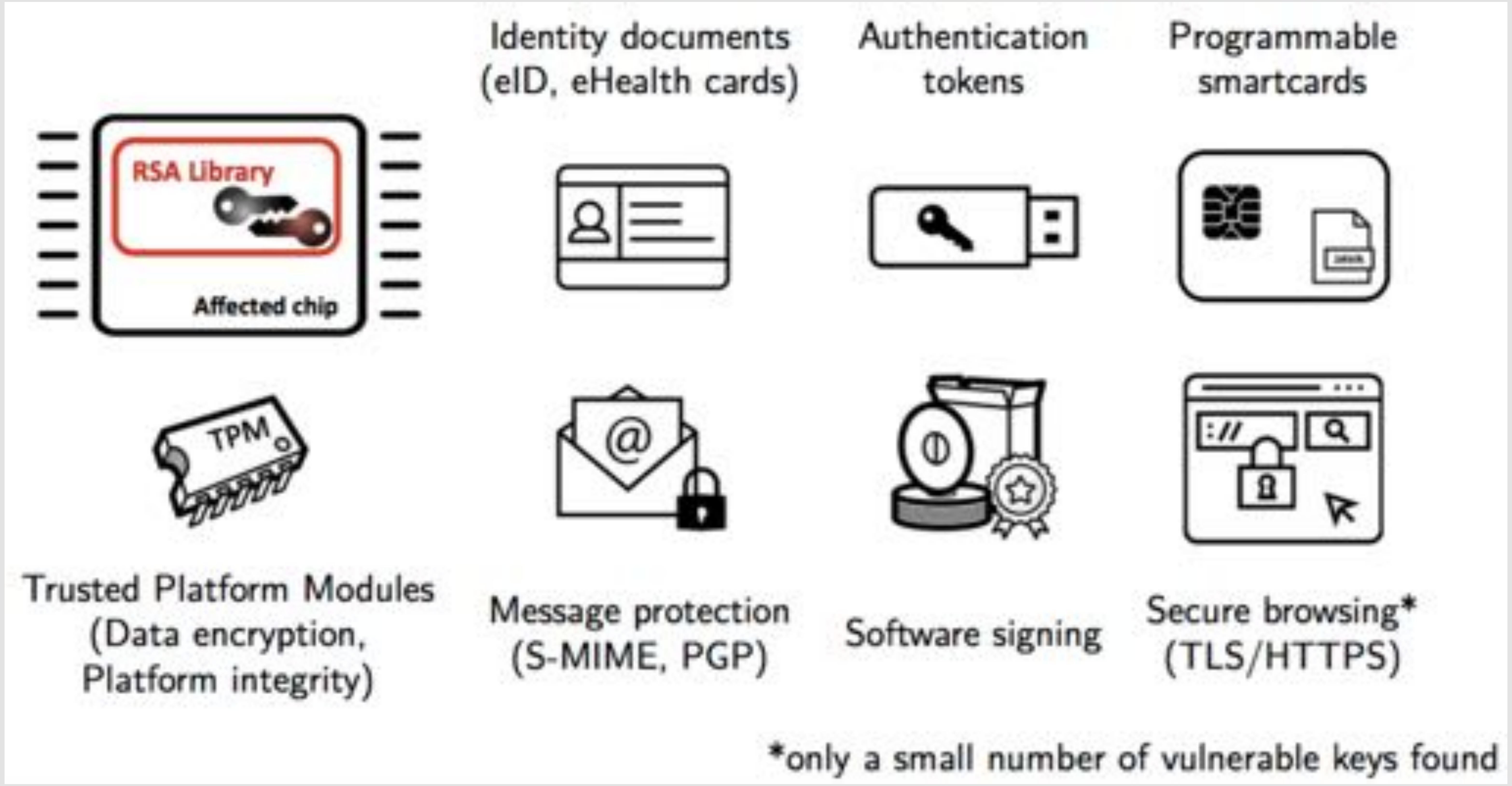
- Given k RSA modulus $N_1,\ldots,N_k$, identify the shared prime numbers.$N_1=pq_1$ and $N_2=pq_2$ and $\gcd(N_1,N_2)=p$
  - The naive solution is to perform roughly k^2 gcd calculations by considering all pairs (N_i,N_j).
  - There is a more efficient quasi-linear algorithm: both teams used it. For 11 million RSA modulus, it only took 5 hours on a single core.

ANOTHER REAL-WORLD ATTACK

- Attack on Infineon RSA keys.

- See ACM CCS '17: https://github.com/crocs-muni/roca
- The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli by Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, Vashek Matyas (Masaryk University).

IMPACT


Identity documents (eID, eHealth cards)
Authentication tokens
Programmable smartcards
RSA Library
Affected chip
TPM
Trusted Platform Modules (Data encryption, Platform integrity)
Message protection (S-MIME, PGP)
Software signing
Secure browsing* (TLS/HTTPS)
*only a small number of vulnerable keys found

- Ex: Estonia’s 750,000 ID cards.

IDENTIFYING RSA KEYS

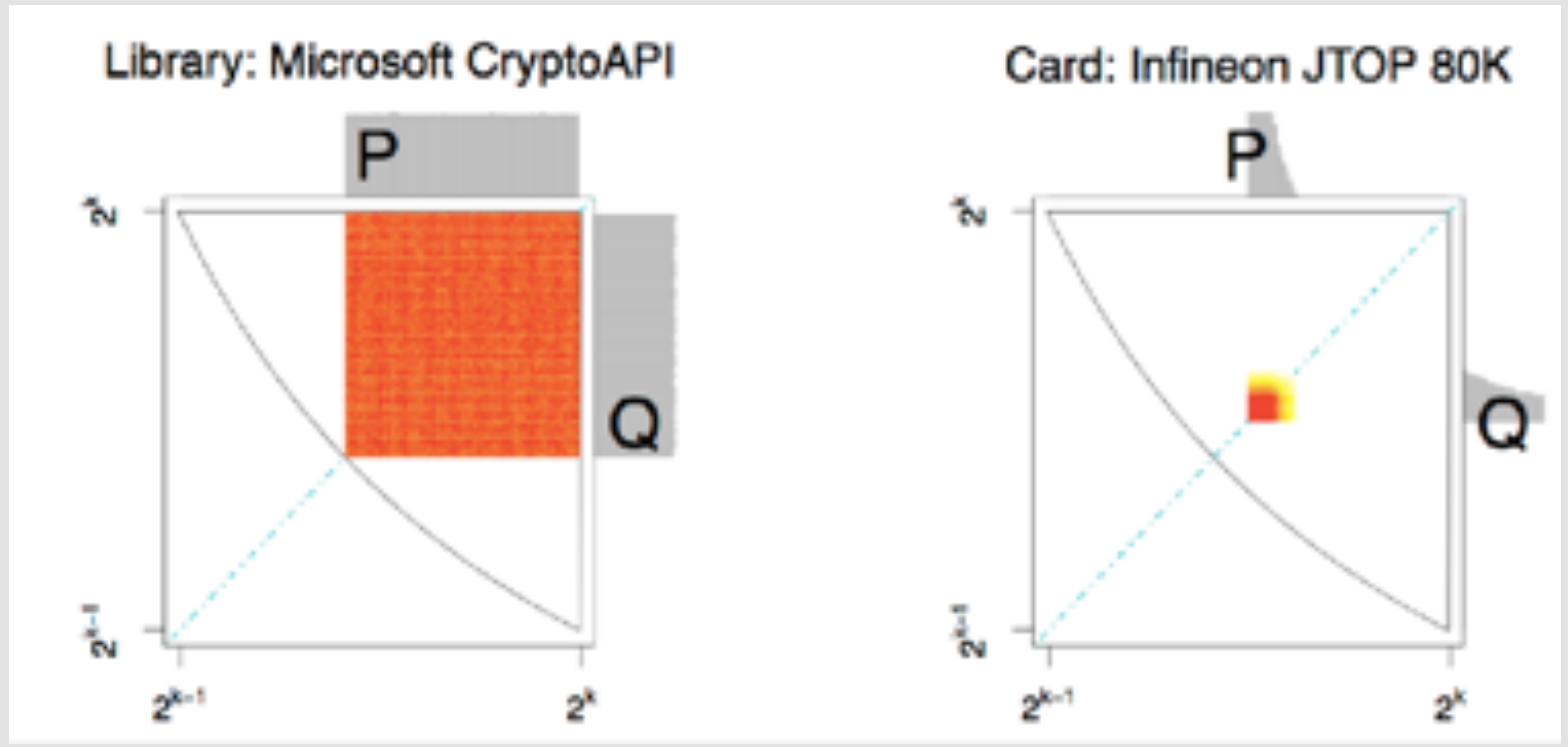
- Svenda et al. analyzed 60 millions fresh keys produced by 22 libraries and 16 smartcards from 6 manufacturers.

- Most distributions of $N=pq$ and/or p were different and could be identified!

WHY?

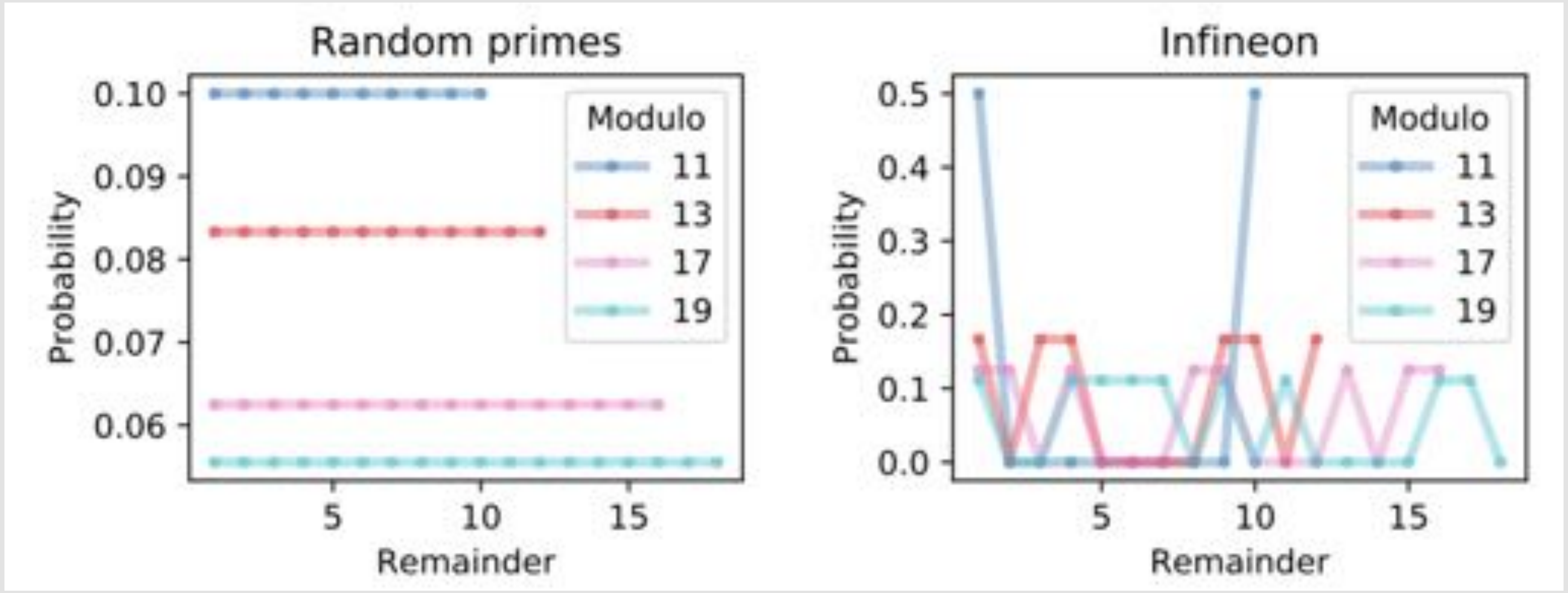
- If p and q are random primes, then (p-1)(q-1) may not be coprime with e, and N=pq will not have a fixed bit-length.

- Each manufacturer/library typically has their own distribution.



EX: INFINEON

- Infineon primes are « not random »



WHAT IS GOING ON?

- If p_i is a small prime then p mod p_i is not uniform over $\{1,\ldots, p_i-1\}$.

- It seems to be uniform over some small subgroup of $(\mathbf{Z}/p_i\mathbf{Z})^*$.

WHY?

- Typically, one generates primes as:
  - Repeat
    - Generate a large random number p
  - Until p is prime

- In practice, primality testing is a few modular exponentiations. One can increase the probability by making p not divisible by all small p_i.

GENERATION

- The subgroup of $(\mathbf{Z}/p_i\mathbf{Z})^*$ is the one generated by 65537.

- p and q are of the form:
  - $p=kM+(65537^a \bmod M)$, where M is the product of the first n primes: 2x3x5x…
  - n depends on the size of N.

- Hence, N mod M is a power of 65537, which can easily be checked.

VALUES OF M

- M is the product of the first n primes.

Bit-length(N)	Number of primes
512-960	39
992-1952	71
1984-3936	126
3968-4096	225

BREAKING INFINEON-RSA

- $p = kM + (65537^a \bmod M)$
- If one can guess the exponent a, then p mod M is known.
- From Coppersmith's 1996 work: if $M \geq N^{0.25}$, lattice attacks recover p in poly-time from N.

N	512-bit	1024-bit	2048-bit	3072-bit	4096-bit
$(\log_2 M)/(\log_2 N)$	0.43	0.46	0.47	0.32	0.48

THE TRICK

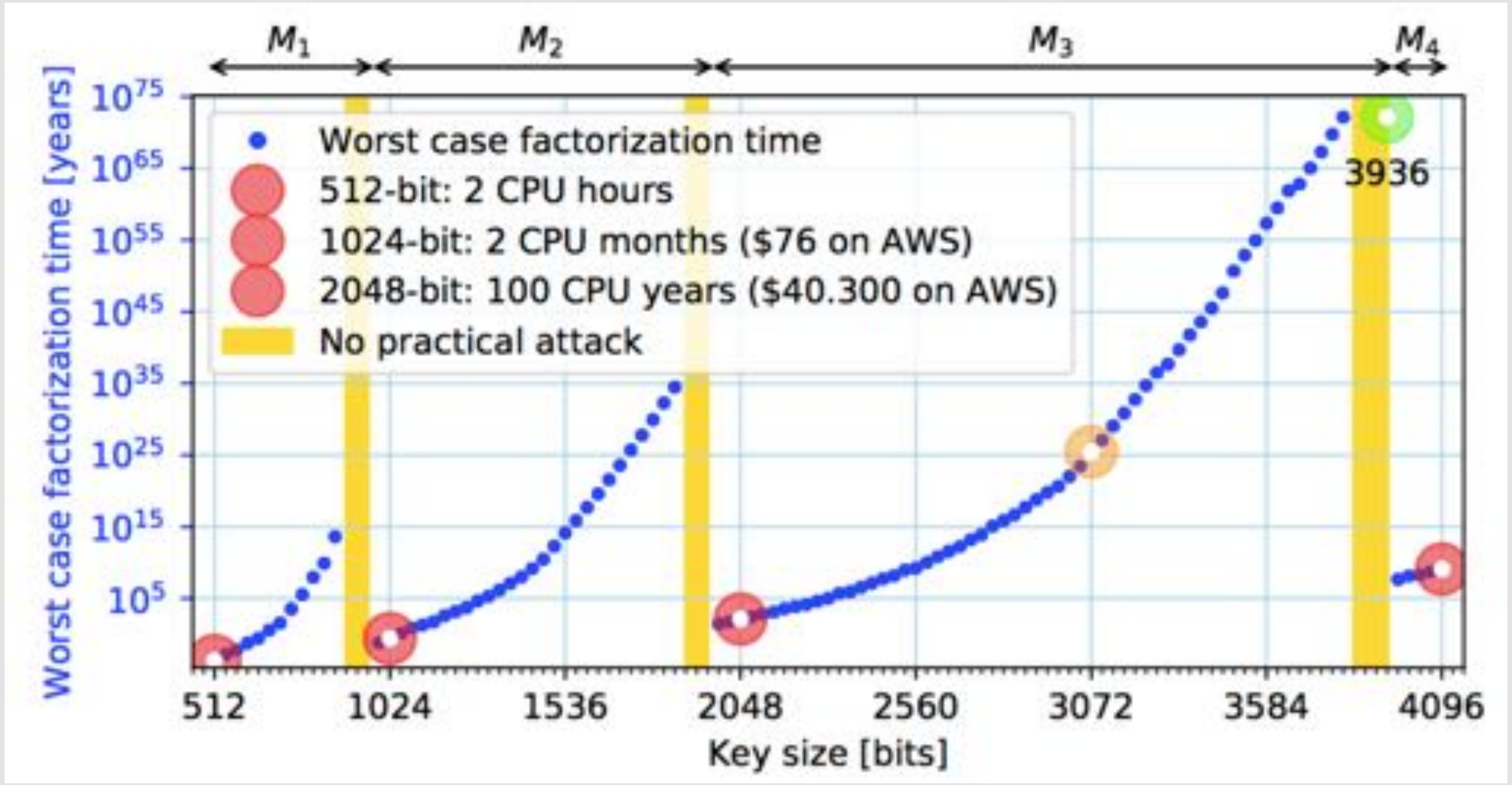
- Guessing a depends on the order of 65537 in $(\mathbf{Z}/M\mathbf{Z})^*$, which might be as big as $M \geq N^{0.4}$: exhaustive search too expensive!

- However, no need to take M: take any divisor M′ of M s.t. $M' \geq N^{1/4}$ and the order of 65537 in $(\mathbf{Z}/M'\mathbf{Z})^*$ is small.

- Ex: 20-bit order for 512-bit N, 30-bit order for 1024-bit.

IMPLEMENTATION

- Non-increasing



CONCLUSION ON RSA

- RSA is the most famous public-key cryptosystem, and still the most widespread.
- But it is not the most efficient one and perhaps not the most secure one.
- There are several alternatives.

BEYOND RSA

1999: PAILLIER'S ENCRYPTION

- Key generation
  - Public key: $N=pq$ like in RSA.
  - Secret key: d s.t. $Nd \equiv 1 \pmod{\phi(N)}$ where $\phi(N)=(p-1)(q-1)$

Pascal Paillier

- The message space is $\mathbf{Z}/N\mathbf{Z}$.
- To encrypt a message $m \in \{0,1,\ldots,N-1\}$:
  - Pick a random $r \in \{0,1,\ldots,N-1\}$
  - Output $c=(1+mN)r^N \pmod{N^2}$
- To decrypt c:
  - Compute $r = c^d \pmod{N}$
  - Compute $a=c(r^N)^{-1} \pmod{N^2}$
  - Compute $m=(a-1)/N$

1999: PAILLIER'S HOMOMORPHIC ENCRYPTION

- The message space is $\mathbf{Z}/N\mathbf{Z}$.
- To encrypt a message $m \in \{\mathbf{0,1,\ldots,N\text{-}1}\}$:
  - ➤ Pick a random $r \in \{\mathbf{0,1,\ldots,N\text{-}1}\}$
  - ➤ Output $c=(1+mN)r^N \pmod{N^2}$
- To decrypt c:
  - ➤ Compute $r = c^d \pmod{N}$
  - ➤ Compute $a=c(r^N)^{-1} \pmod{N^2}$
  - ➤ Compute $m=(a-1)/N$
- Theorem (Paillier): $E(m_1, r_1) \times E(m_2, r_2) \equiv E(m_1 + m_2 \mod N, r_1 r_2 \mod N) \mod N^2$

APPLICATION: ELECTRONIC VOTING

- A vote m_i is in $\{0,1\}$ for $1 \leq i \leq n$
- Everyone encrypts his vote: $c_i = E(m_i)$
- One computes the aggregate $C = \prod_{i=1}^{n} c_i \mod N^2$
- The authority decrypts C to disclose $\sum_{i=1}^{n} m_i$