Chapter 9

Public Key Cryptography and RSA

Why Public-Key Systems

- Public-key cryptography attempts to resolve two difficult problems associated with symmetric encryption
- **Key distribution**: How to share a key for symmetric encryption without having to trust a key distribution center to distribute it
- **Digital signature**: How to publicly verify that a message comes intact from the claimed sender

Three Types

- Public-key encryption
  - Sender encrypts a message with receiver's public key
  - Receiver decrypts with his private key
- Digital signature
  - Signer signs a document with his private key
  - Verifier verifies with signer's public key
- Public key-exchange
  - Two remote parties establish a session key for encryption over public channel

History

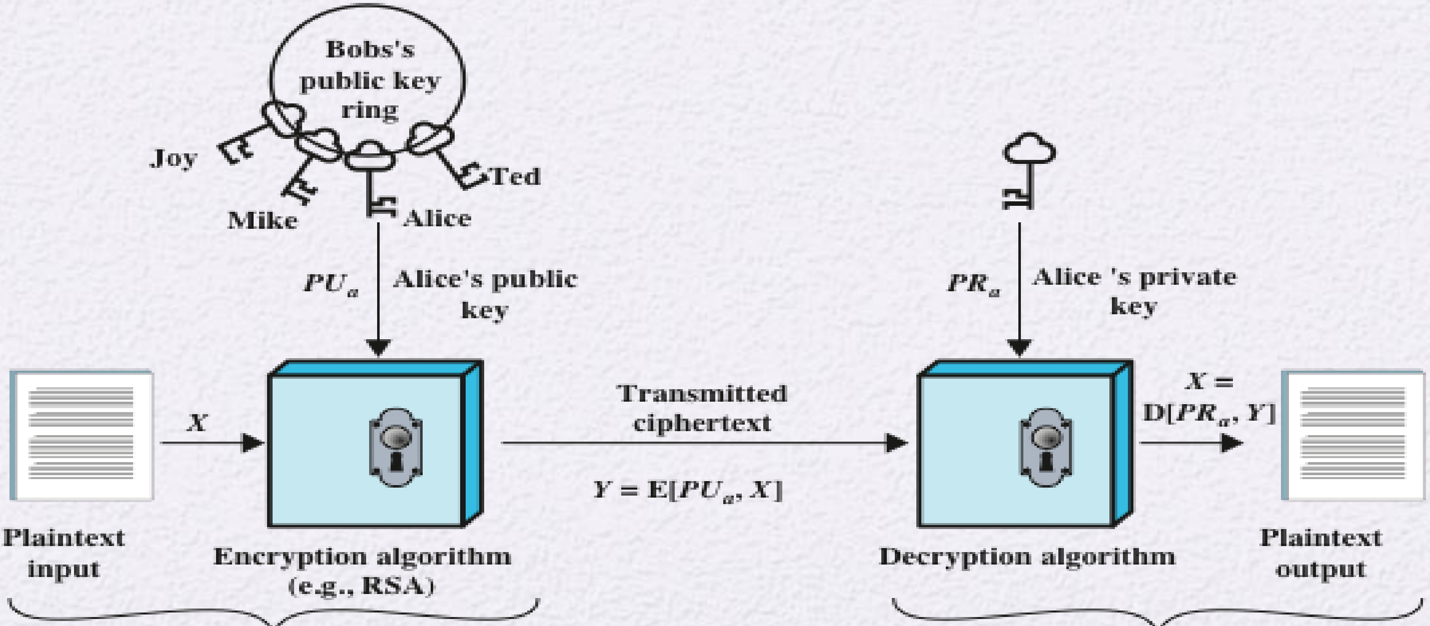
- Whitfield Diffie and Martin Hellman
  - DH-key exchange, 1976
- Ron Rivest, Adi Shamir and Leonard Adleman
  - RSA encryption, RSA digital signature, 1977
- Taher ElGamal
  - ElGamal digital signature, 1984
  - ElGamal encryption, 1985

Public-Key Encryption

- A public-key encryption scheme has six ingredients.
  - Encryption algorithm
  - Decryption algorithm
  - Public key
  - Private key
  - Plaintext
  - Ciphertext

PK Encryption: Two keys

- Each person X has a pair of keys
  - Public key: PU_X
  - Private key: PR_X

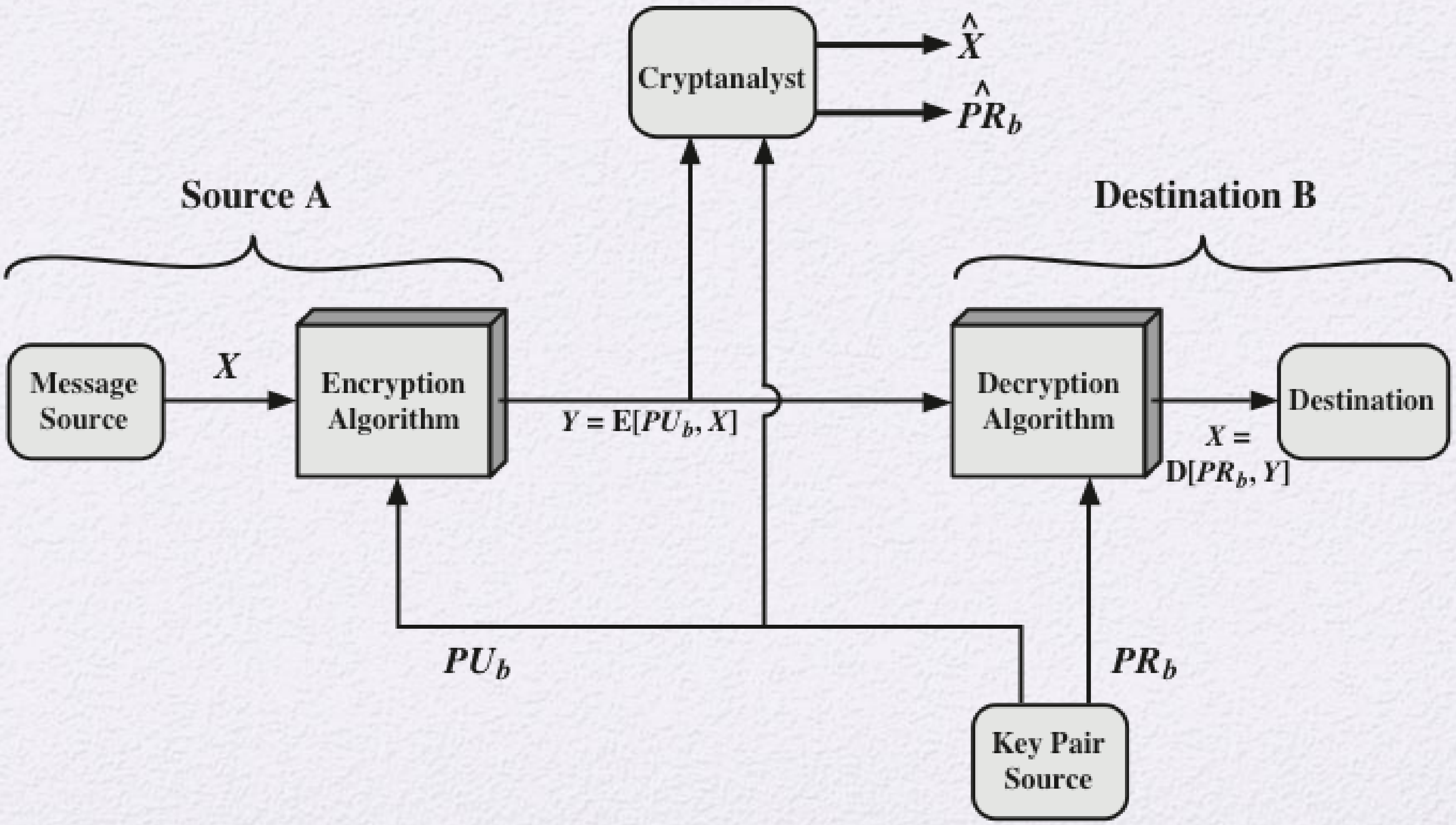


(a) Encryption with public key

Misconceptions

- Public-key encryption is more secure than symmetric encryption
- Public-key encryption is a general-purpose technique that has made symmetric encryption obsolete
- Key distribution is trivial when using public-key encryption, compared to the cumbersome handshaking involved with key distribution centers for symmetric encryption

Security Model



Security Requirements

- Computationally easy
  - For any user A, generate his key pair (public-key PU_A, private key PR_A)
  - For any sender, compute $C=E(PU_A, M)$
  - For the receiver A, compute $M=D(PR_A, C)$
- Computationally infeasible
  - For any adversary, compute PR_A from PU_A
  - For any adversary, compute M from C and PU_A

PK Theory

- A trap-door one-way function f
  - Given f and X, it is easy to compute Y = f(X)
  - Given f and Y, it is infeasible to compute
    $$X = f^{-1}(Y)$$
  - Trap-door property: there is a trap door T such that it is easy to compute $X=f^{-1}(Y, T)$
- Thus, f is the public-key and T is the private key

PK encryption: RSA

- First public-key encryption, 1977
- Invented by Rivest, Shamir and Adleman
- Math
  - Group: $(Z_n^*, \times_n)$, where n=pq, a product of two large primes
  - But, still work for $(Z_n, \times_n)$

RSA Encryption

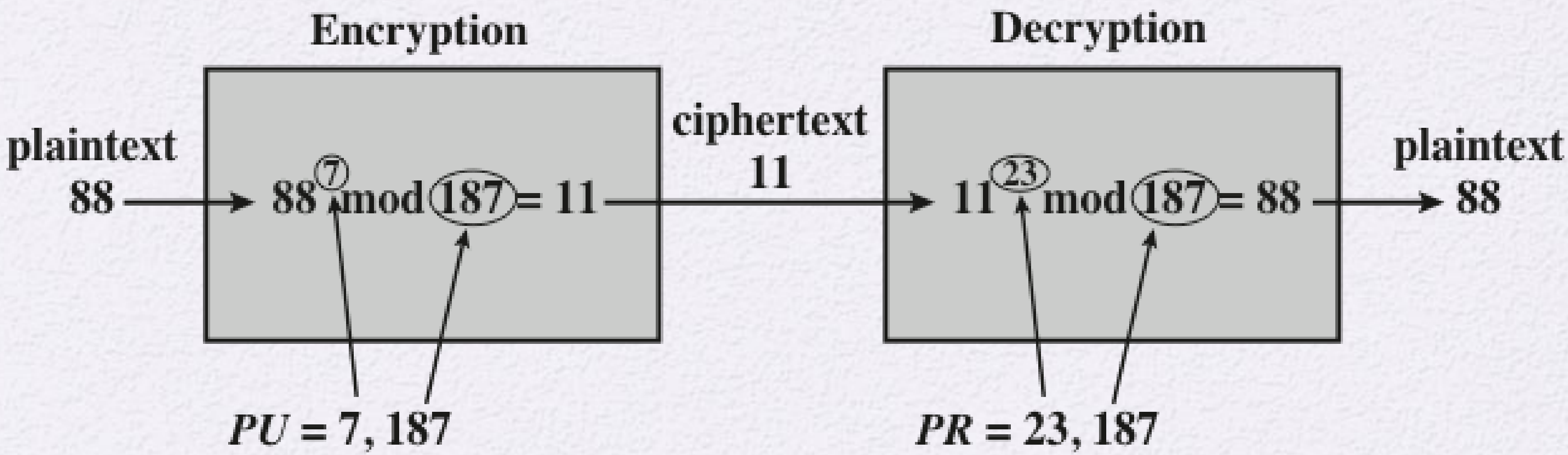
Key Generation by Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

RSA Encryption

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption by Alice with Alice's Private Key	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

RSA Encryption: Toy Example



Does it work?

- $\phi(n)=(p-1)(q-1)$, $ed = k\phi(n)+1$
- If $\gcd(M, n)=1$
  - $C^d \bmod n = M^{ed} \bmod n = M^{k\phi(n)+1} \bmod n$

    $= (M^{\phi(n)})^k \times M \bmod n = 1 \times M \bmod n = M$

    By Euler's theorem, $M^{\phi(n)} \bmod n=1$

- If $M = ap$, $0 \le a < q$
  - Let $C^d \bmod n = M^{ed} \bmod n = x$
    - We consider $r_1 = x \bmod p$ and $r_2 = x \bmod q$
      - $r_1 = x \bmod p = 0 = M \bmod p$, since $p|x$
      - $r_2 = x \bmod q = M^{k(p-1)(q-1)+1} \bmod q$
        $= (M^{q-1})^{k(p-1)} x M \bmod q = M \bmod q$
        since $\gcd(M, q)=1$, $M^{q-1} \bmod q=1$ (Fermat's little theorem)
    - By CRT, the unique solution for x is M
- If $M = bq$, $0 \le b < p$, … (similar)

Example

- $n = 11 \times 17 = 187$, $\phi(n) = (p-1)(q-1) = 160$, $e=3$, $d=107$
- $M = 12$
  - $C = 12^3 \bmod 187 = 45$
  - $D = 45^{107} \bmod 187 = 12$
- $M=22$
  - $C = 22^3 \bmod 187 = 176$
  - $M = 176^{107} \bmod 187 = 22$

Real RSA Keys

Public Modulus (hexadecimal):	e75d78949dd6e6b180d23626817ddf32a9717287ac06cebf92f77903e20d7880989c6adeda37d851 9037b54c0bde7e67422e730afc73a881861333a543d0f90706eb8c9e58cade8586c3618f89c538b0 ecf8ae81ae21e5ba4e35f3f78c334e57b8d564f042ad2bb8383c8e6604f3b5edab48fc0914ac888c 023c7e5f488d4953
Public Exponent (hexadecimal):	10001
Private Exponent (hexadecimal):	923fe89ff1224e13783de912f019f403df4e223a96c87ada68795c9ad2c2f7203ad7ed4a4fa0ab71 eb7afb7445b07030af8a1318a7ba28932f8065ce1b0f36ca414ea7fecfc4ee2589ff001579cb1635 7b5b26f3c83ee108982ef9672d28d1a119a46c3e91a893c8ced68aa54c58528e22da79f08af1f318 babe923297d61499

Efficient Computation

- Finding two large primes p and q, typically, 1024-bit long.
- Computing n=pq and $\phi(n)=(p-1)(q-1)$
- Finding e with $\gcd(e, \phi(n))=1$
- Computing the inverse $d = e^{-1} \; mod \; \phi(n)$
- Computing $C = M^e$ mod n *and* $M = C^d$ *mod* n

Modular Exponentiation

- **a^b mod *n***
- The square-and-multiply algorithm
  - $a^{13} = a^{1101} = (((\left(1^2 \times a\right)^2 \times a)^2)^2 \times a)$
  - "mod n" is done in any intermediate

```
c ← 0; f ← 1
for i ← k downto 0
    do    c ← 2 × c
          f ← (f × f) mod n
    if    b_i = 1
          then c ← c + 1
               f ← (f × a) mod n
return f
```

Note: The integer b is expressed as a binary number $b_k b_{k-1} \ldots b_0$

Figure 9.8 Algorithm for Computing a^b mod n

i	9	8	7	6	5	4	3	2	1	0
b_i	1	0	0	0	1	1	0	0	0	0
c	1	2	4	8	17	35	70	140	280	560
f	7	49	157	526	160	241	298	166	67	1

Table 9.4 Result of the Fast Modular Exponentiation Algorithm for a^b mod n, where $a = 7$, $b = 560 = 1000110000$, and $n = 561$

Time complexity: a^b mod n

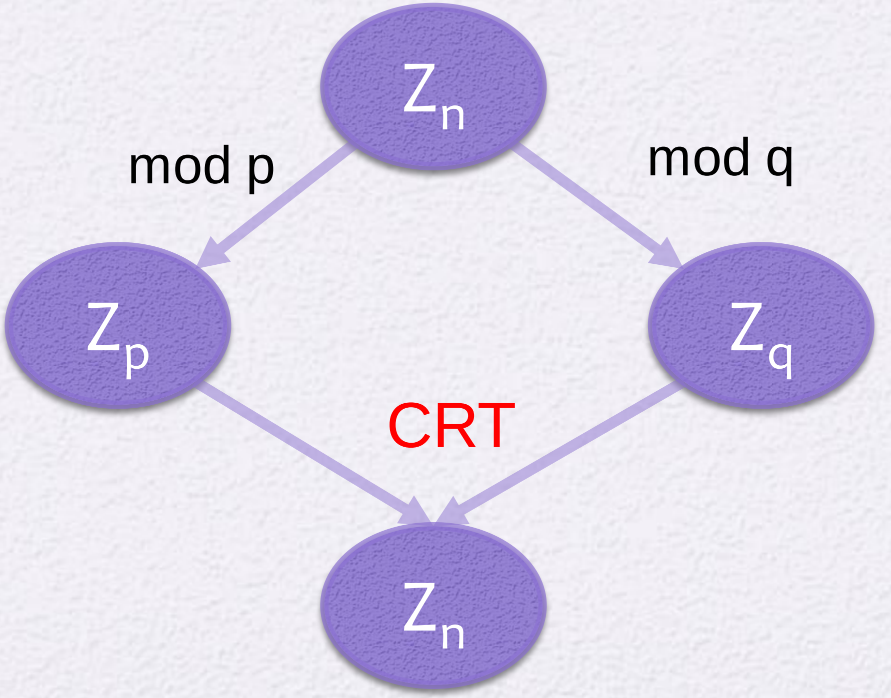
- Length (bits) of b and n = k
- Long modular multiplication: xy mod n
  - k^2 bit-operations
  - If x and y are bit-reprented, use "shift-and-XOR" algorithm
- # of long modular multiplication
  - 2k -- at most
  - 1.5 k -- on average for random b
  - k+2 -- carefully chosen b
- Total: $1.5k^3$ bit-operations on average

m^e mod n: Speedup

- The most common choices of e: $3 = 2^1 + 1$, $17 = 2^4+1$, $65537 = 2^{16} + 1$
- d should be long. Otherwise, the attacker can use the brute-force attack to search d

CRT mapping

- Isomorphism Ψ: $Z_n \rightarrow Z_p x Z_q$



CRT isomorphic mapping

- Ψ: $Z_{15} \rightarrow Z_3 x Z_5$
- Mapping:
  - $12 \rightarrow (0, 2)$
  - $7 \rightarrow (1, 2)$
- Addition: $12+7 \rightarrow (0, 2)+(1,2)=(1,4) \rightarrow 4$
- Multiplication: $12x7 \rightarrow (0,2)x(1,2)=(0,4) \rightarrow 9$

c^d mod n: speedup

- Pre-compute
  - d'= d mod $(p-1)$ and $d''=d$ mod $(q-1)$
  - $\bar{q} = q(q^{-1} \bmod p)$, $\bar{p} = p(p^{-1} \bmod q)$
- Compute
  - $C' = C \bmod p$, $M' = C'^{d'} \bmod p$.
  - $C'' = C \bmod q$, $M''=C''^{d''} \bmod q$
- Use CTR to compute M from M' and M''
  - Find x for {M'= x mod p, M''= x mod q}

    ➔M = x = $(M' \bar{q} + M'' \bar{p})$ mod pq

c^d mod n: speedup

- Example
  - n=187=11x17 =pxq, e=3, d=107, C = 45
- Pre-compute
  - d'= 107 mod 10 = 7, d''= 107 mod 16 = 11
  - $\bar{q}$ = 17(17^{-1} mod 11) = 17x2 = 34
  - $\bar{p}$ = 11(11^{-1} mod 17) = 11x14 = 154
- Compute
  - C'= 45 mod 11 = 1, M' = 1^7 mod 11 = 1
  - C''=45 mod 17 = 11, M''11^{11} mod 17 = 12
- Find x for {M'= x mod p, M''= x mod q}

  ➔M = x = (1x34 +12x154) mod 187 = 12

c^d mod n: speedup

- one long modular exponentiation → two half-long modular exponentiations + one CRT
- axb mod n → $O(k^2)$ bit-operations, for k-bit n.
- Without speedup
  - 1.5k multiplications = 1.5k x $O(k^2)$ = $1.5k^3$ bit-operations
- With speedup
  - 2 x 1.5k' x $O(k'^2)$ + 3 multiplications (CRT)

    = $1.5k^3/4 + 3k^2$ bit-operations

Pick a Large Prime

Algorithm PickPrime(N) -- Output an N-bit prime

1. Pick an odd N-bit integer *p* at random
2. Repeat the following for a sufficient number of times (20 times)
   - Pick an integer a at random, $1 < a < p$.
   - Perform the probabilistic primality test with *a* as a parameter – Rabin-Miller test
   - If *p* fails the test, reject the value p and go to step 1.
3. Output (p is probably prime)

Prime Density

Pick an odd integer *p* at random. p being prime is sufficiently large

- 1--100: 25 primes → density = 0.25
- 1--1000: 168 primes → density = 0.168
- 1--10000: 1209 primes → density = 0.1209
- …
- 1--2^{1024}: density $\approx \frac{1}{\ln N} = \frac{1}{\ln 2^{1024}} \approx 0.00141$
  → not too bad

RSA: Security

- It should be hard to
  - Factor n
  - Compute $d = e^{-1} \bmod \phi(n)$ from PU=(e, n)
  - Compute M from PU=(e, n) and $C=M^e \bmod n$
- Practical cautions for prime selection
  - p and q should differ in length by a few digits
  - (p-1) and (q-1) should have large factors
  - gcd(p-1, q-1) should be small
  - $d > n^{1/4}$
  - …

RSA: Security

- Two users **<u>cannot</u>** use the same n
  - $(n, e_1), (n, d_1)$
  - $(n, e_2), (n, d_2)$
- Given (n, e_1, d_1, e_2), one can compute d_2' with $d_2 \equiv d_2'$ $(\mathrm{mod}\ \phi(n))$
  - Compute $e_1 d_1 - 1 = k \cdot \phi(n)$
  - Compute $d_2' = e_2^{-1} \bmod k \cdot \phi(n)$
  - Thus, $d_2 \equiv d_2'$ $(\mathrm{mod}\ \phi(n))$

Factoring Problem

- Factor *n* into its two prime factors and compute $ø(n) = (p - 1) \times (q - 1)$. Then, compute $d = e^{-1} \ (mod\ ø(n))$
- Determine ø(n) directly without first determining *p* and *q*.
- Determine *d* directly without first determining ø(n)

Number of Decimal Digits	Number of Bits	Date Achieved
100	332	April 1991
110	365	April 1992
120	398	June 1993
129	428	April 1994
130	431	April 1996
140	465	February 1999
155	512	August 1999
160	530	April 2003
174	576	December 2003
200	663	May 2005
193	640	November 2005
232	768	December 2009

RSA Challenge, up to 2009

- The 696-bit RSA-210 was factored by Ryan Propper, 2013
- $2^{1061} - 1$ (1061 bits , 320 digits) was factored by Greg Childers, etc, 2012

G-NFS:

$$e^{\sqrt[3]{\frac{64}{9}} \times (\ln N)^{1/3})(\ln \ln N)^{2/3}}$$

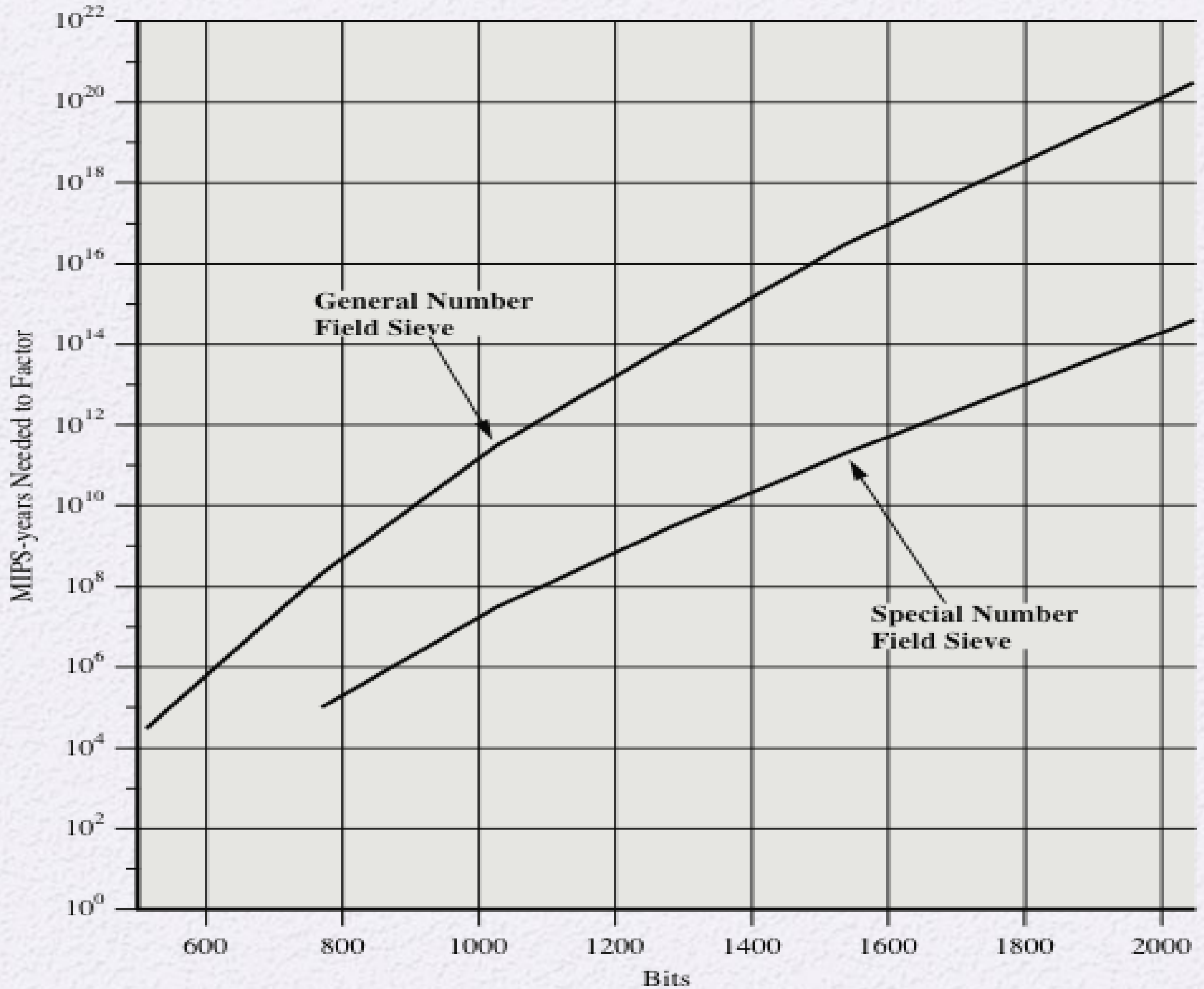


Figure 9.9 MIPS-years Needed to Factor

Quantum computing

- Schrodinger cat
  - Cat being alive and dead at the same time before observation
- Superposition
- Coherence

IBM Q System 1

2019, 20 qbits

ENIAC 1946, 170m^2, 30 tons

Quantum Factorization

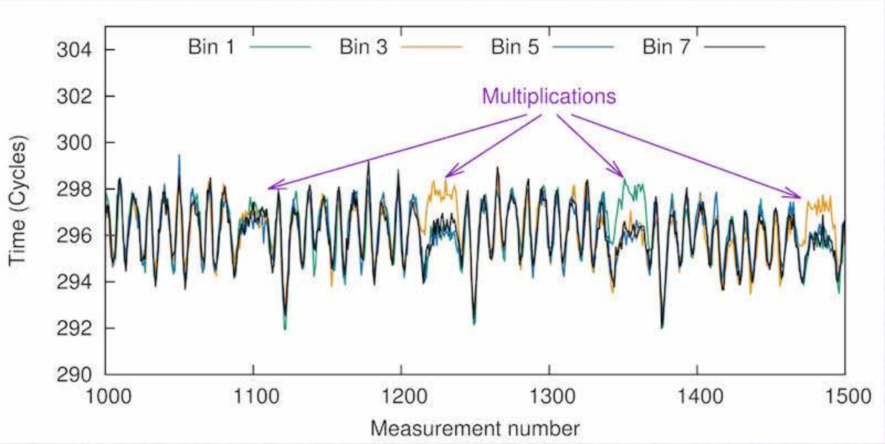
- Quantum computer: exploit quantum effect of sub-atomic particles
- Shor's quantum factoring algorithm: factoring n in poly($\log_2$ n) time
- State-of-the-art quantum computers, 2018 --
  - General purpose: ≈ 70 qbits, IBM, Google，九章
  - Special purpose (quantum annealing): 2000 qbits, D-Wave
  - Extremely high cost
- Remark: Symmetric-key encryption is still safe

Quantum computer: practice

- D-wave's quantum annealing
  - Factor 376289 = 571 x 659 using 94 qbits, 2018
  - Extrapolation from this result
    - Factoring 1024-bit n → ~28,000 qubits
    - Factoring 3072-bit n → ~2,500,000 qubits
- General-purpose quantum computer
  - Factor 1024-bit n
    → theoretically, 2048 quantum bits
    → practically (error correction),
    2048x100 -- 2048x10000 qbits

Timing Attacks

- A snooper can determine a private key by keeping track of the time of computing in each step, 1996
- Side-channel attack: fault-based attack, power analysis, …




```
c ← 0; f ← 1
for i  ← k downto 0
      do   c ← 2 × c
           f ← (f × f) mod n
      if   b_i = 1
           then c ←  c + 1
                f ← (f × a) mod n
return f
```

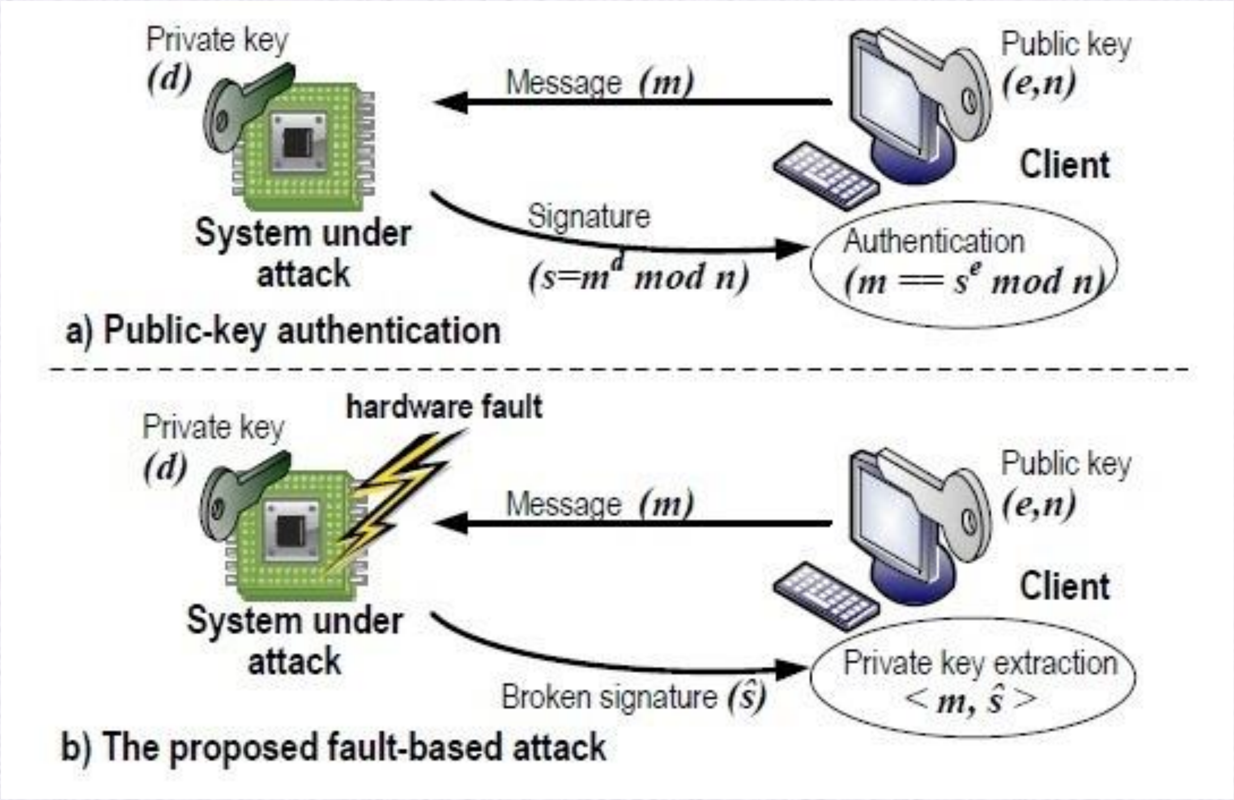
Countermeasures

- Constant exponentiation time: all exponentiations take the same amount of time before returning a result
- Random delay: add a random delay to the exponentiation algorithm to confuse the timing attack
- Blinding: multiply ciphertext by a random number before performing exponentiation

Fault-Based Attack

- An attack on a processor
  - The attack algorithm involves inducing single-bit errors and observing the results
  - Induce faults in the signature computation by reducing the power to the processor
  - The faults cause the software to produce invalid signatures which can then be analyzed by the attacker to recover the private key
- The attack does not seem serious since it requires that the attacker has physical access to the target machine

- "Fault-based attack on RSA authentication", by Andrea Pellegrini, Valeria Bertacco, Todd Austin, 2010
- OpenSSL with FPGA implementation RSA, 100 hours to obtain 1024-bit RSA signing key.

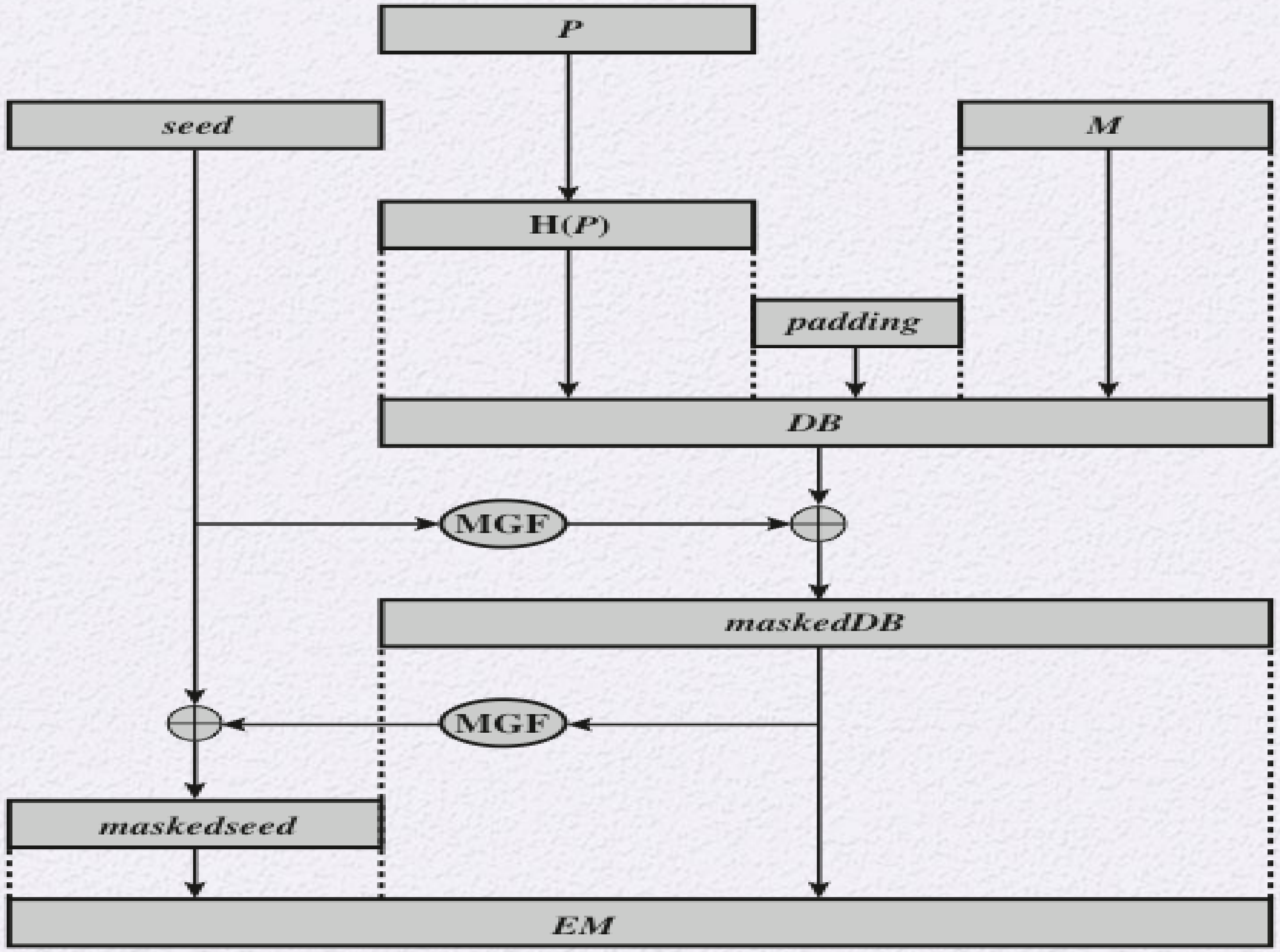


Chosen Ciphertext Attack

- CCA: given a target C, allow the adversary to ask the plaintext of a ciphertext $C' \neq C$
- The attack
  - Compute $C' = C \cdot r^e \bmod n$
  - Ask to decrypt $C' \neq C$ and obtain $M' = C'^d \bmod n$
  - Compute $M = (M' / r) \bmod n$
- To counter such attacks, RSA Security Inc. recommends modifying the plaintext using a procedure known as *optimal asymmetric encryption padding* (OAEP)

RSA: OAEP padding mode

- OAEP: Optimal Asymmetric Encryption Padding
- Used for defending the CCA1 and CCA2 attacks
- Provable security



P = encoding parameters
M = message to be encoded
H = hash function
DB = data block
MGF = mask generating function
EM = encoded message

Summary

- Public-key cryptosystems
- Applications for public-key cryptosystems
- Requirements for public-key cryptography
- Public-key cryptanalysis
- The RSA algorithm
  - Description of the algorithm
  - Computational aspects
  - Security of RSA