Chapter 9

Public Key Cryptography and RSA

Why public-key systems

- Attempt 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 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

Misconceptions

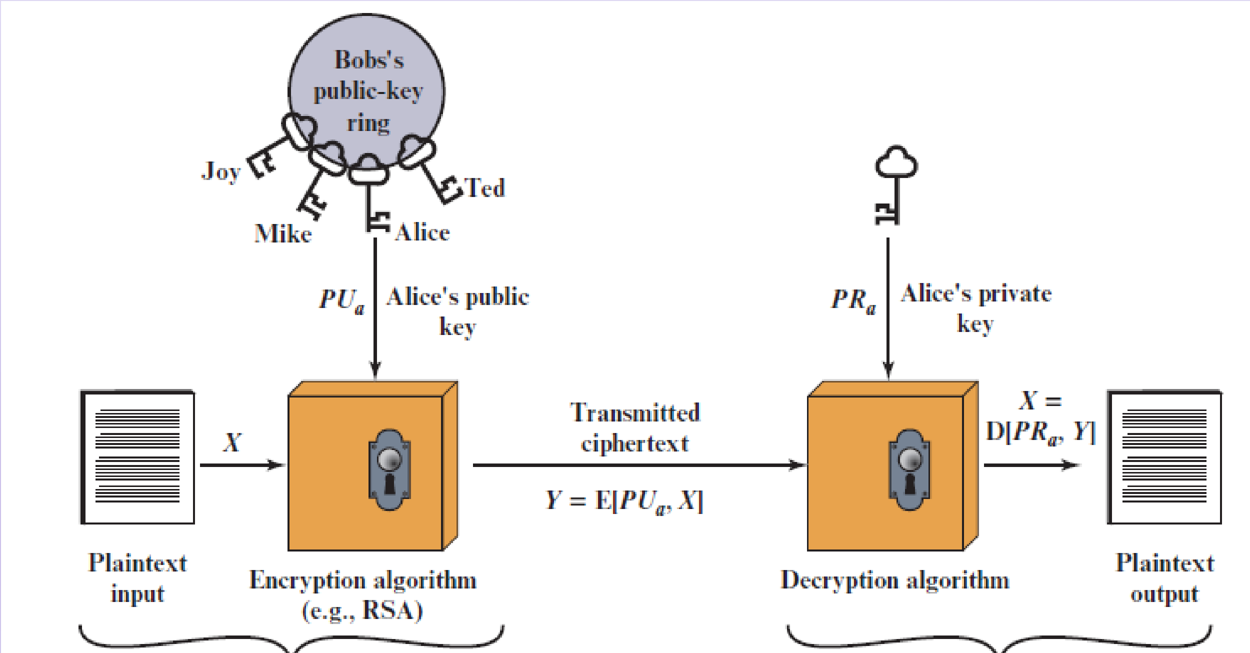
- Public-key encryption is more secure than symmetric encryption
- Public-key encryption is a general-purpose technique that has made symmetric encryption obsolete
- Key distribution is trivial when using public-key encryption

Public-key encryption

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

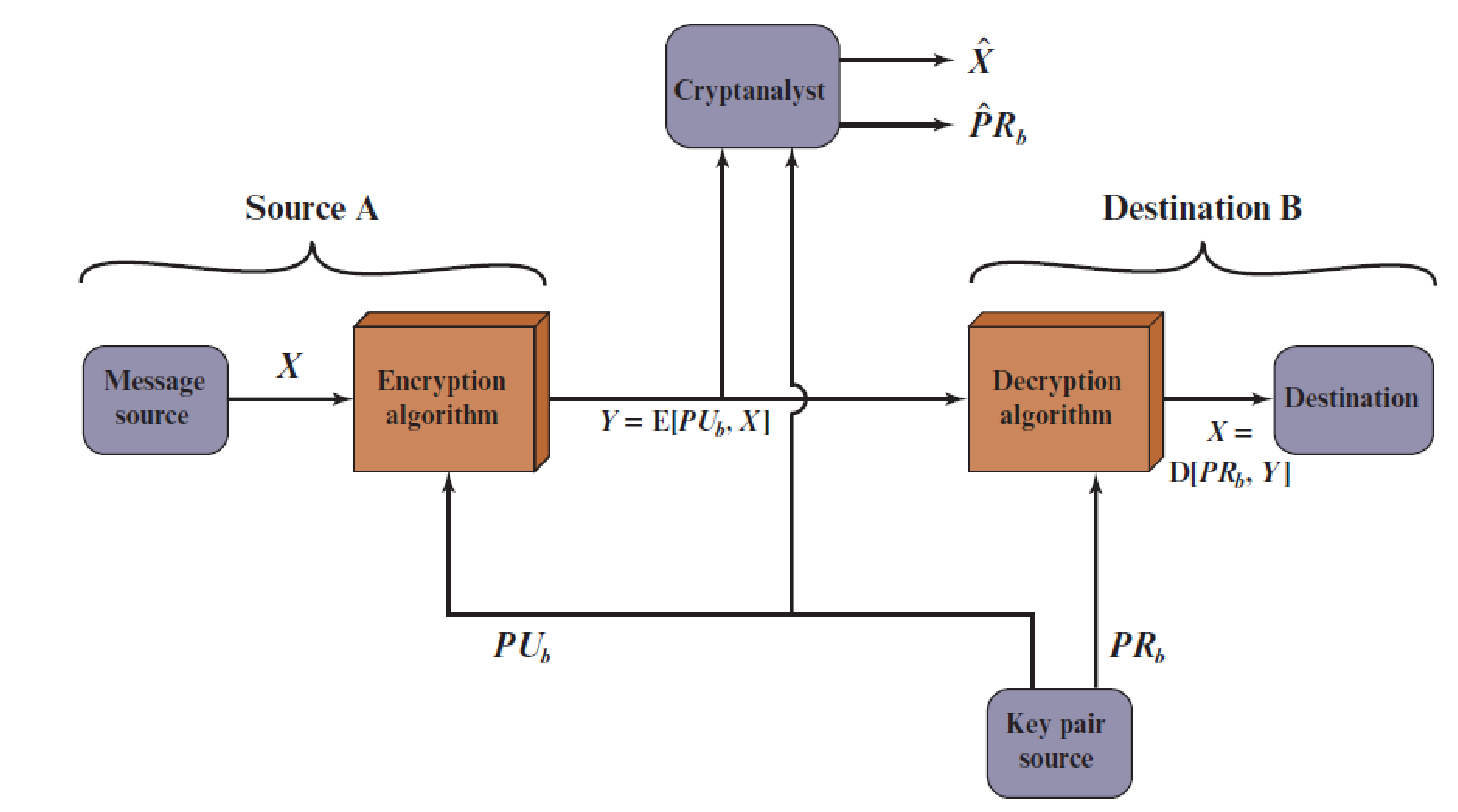
Public-key encryption: two keys

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



(a) Encryption with public key

Public-key encryption: security model



PK encryption: computing requirements

- Computationally easy
  - A user A generates his key pair: PU_A, PR_A
  - A sender computes a ciphertext $C = E(PU_A, M)$
  - The receiver A computes $M = D(PR_A, C)$
- Computationally infeasible
  - An adversary computes PR_A from PU_A
  - An adversary compute M from C and PU_A

RSA: key generation

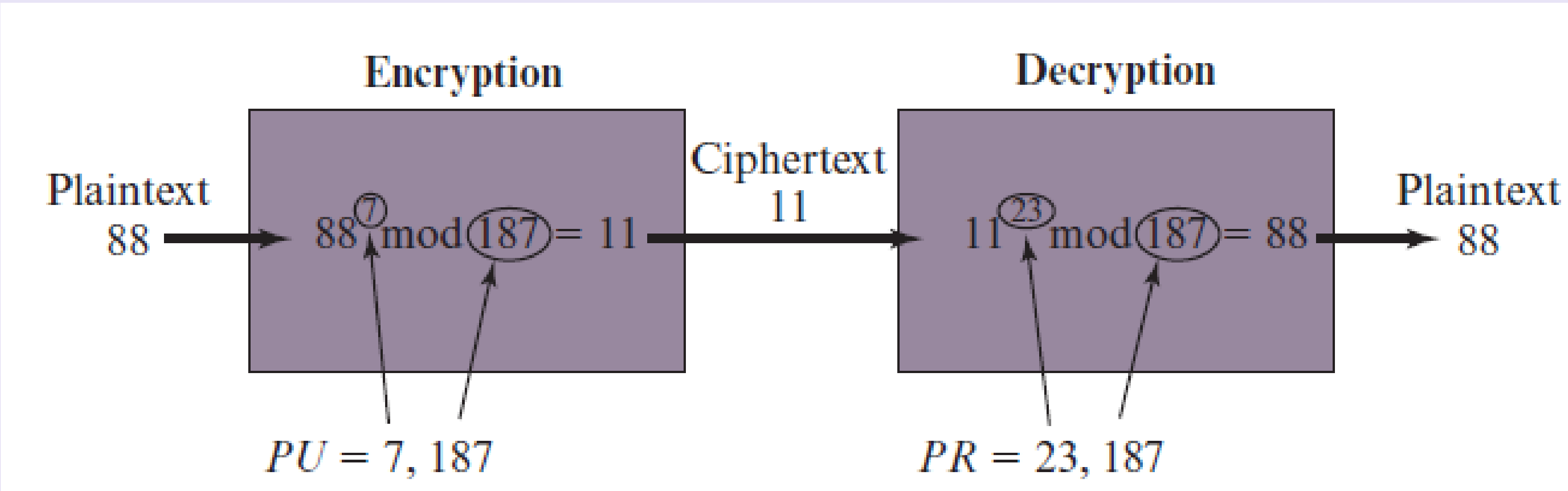
Key Generation by Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n)=(p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

RSA: encryption/decryption

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: toy example



RSA: correctness

- The operations of RSA are on group Z_n^* , where $n = pq$
- Parameters
  - $n = pq$, where p and q are large prime
  - e: $1 < e < \phi(n),\ \gcd\left(e, \phi(n)\right) = 1$
  - d: $d = e^{-1} \bmod \phi(n)$, that is, $ed = k\phi(n) + 1$
- Question:
  - for $1 \leq M \leq n - 1$, $C = M^e \bmod n$, is it indeed $C^d \bmod n = M$?

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

RSA: example

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

RSA keys: real

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

RSA: computation aspects

- Key generation
  - Pick two large random primes p and q, each, 1024-bit long.
  - Compute $\phi(n) = (p-1)(q-1)$
  - Pick e with $\gcd(e, \phi(n)) = 1$
  - Compute $d = e^{-1} \bmod \phi(n)$
- Encryption: compute $C = M^e \bmod n$
- Decryption: compute $M = C^d \bmod n$

Pick N-bit random primes

- Idea
  - pick a random N-bit number
  - Use primality test to test its primality
- Facts
  - Prime densitoy

$$\pi(x) = |\{p|p \ is \ prime, p \leq x\}|/x \ \approx 1/ln\, x$$

  - For $N = 1024$-bit, $\pi(2^{1024}) \approx 1/\ln(2^{1204}) \approx 0.00141$
- For every thousand picks of 1024-bit random numbers, the expected number of picked primes is 1.41
- The error probability of outputting a non-prime number is very low due to high success probability of primality test
- Thus, it is feasible to pick a random prime of thousand bits long

Operations on numbers

- Modular multiplication
  - Given N-bit a, b, n , compute $ab \bmod n$
  - By shift-add-mod algorithm (need carry), it takes $O(N^2)$ bit operations
- Modular exponentiation
  - Given N-bit a, b, n, compute $a^b \bmod n$
  - By square-multiply-mod algorithm, which need O(N) modular multiplications. Total time is O(N^3) bit operations.
- The above two operations are feasible theoretically. They are even faster due to our powerful CPUs

Modular exponentiation

- The square-multiply-mod algorithm
  - Example, to compute $a^{131} \bmod n = a^{10000011} \bmod n$
  - Compute
    - $a_1 = a$
    - $a_2 = a_1^2 \, mod \, n = a^2 \bmod n$
    - $a_4 = a_2^2 \bmod n = \ a^4 \bmod n$
    - …
    - $a_{128} = a_{64}^2 = \ a^{128} \bmod n$
    - $a^{131} \bmod n = a_1 a_2 a_{128} \bmod n$
  - Another form is to scan from high bit to low bit of exponent b (textbook use)
- The fewer number of $1's$ in b, the fewer number of multiplications needed for $a^b \bmod n$

Find (e, d)

- Pick e, $1 < e < \phi(n)$, randomly and check $\gcd(e, \phi(n)) = 1$ by Euclidean algorithm
  - e=17 or 65537 are used often in practice.
  - $17 = 2^4 + 1 = 10001, 65537 = 2^{16} + 1 = 10000000000000001$
  - They are both prime. Very likely $\gcd(e, \phi(n)) = 1$ for random $n = pq$
  - Computation of $M^e \bmod n$ needs less time
- Compute $d = e^{-1} \bmod \phi(n)$ by the extended Euclidean algorithm of finding (x, y) for $xe + y\phi(n) = 1$. Then, $d = x \bmod \phi(n)$
- Euclidean algorithm takes $O(N)$ steps of '$x \bmod y$', which takes $O(N^2)$ bit operations. The total time complexity is $O(N^3)$ bit operations

Computation speedup

- $M^e \bmod n$
  - Pick smaller e with fewer numbers of $1's$ in e.
- $M = C^d \bmod n$, where p and q are known by key owner
  - Compute $r_1 = C^d \bmod p,\ r_2 = C^d \bmod q$
  - M is the solution of CRT equations:

$$M \bmod p = r_1, M \bmod q = r_2$$

  - If p and q are N-bit long, n is $2N$-bit long
  - Computing M directly takes $(2N)^3 = 8N^3$ bit operations
  - Computing M by the CRT method takes $2N^3 + O(N^2)$ (CRT time) – save three quarters of time

Speedup: exmple

- n=187=11x17, e=3, d=107, C = 45
- Pre-compute
  - $d_1 = 107 \text{ mod } (11 - 1) = 7,$
  - $d_2 = 107 \text{ mod } (17 - 1) = 11$
  - $p \times p^{-1} \text{mod } q = 11 \times (11^{-1} \text{ mod } 17) = 11 \times 14 = 154$
  - $q \times q^{-1} \text{ mod } p = 17 \times (17^{-1} \text{ mod } 11) = 17 \times 2 = 34$
- Compute
  - $r_1 = C^{d_1} \, mod \, 11 = 1$
  - $r_2 = C^{d_2} \, mod \, 17 = 12$
  - $M = (1 \times 34 + 12 \times 154) \text{ mod } 187 = 12$

RSA: use caution

- Two users cannot use the same n
  - User A: $(n, e_1), (n, d_1)$
  - User B: $(n, e_2), (n, d_2)$
- User B: obtaining A's public key (n, e_1)
  - Compute $k\phi(n) = e_2 d_2 - 1$
  - Compute $d_1' = e_1^{-1} \bmod k\phi(n)$
  - We can see that $d_1' \equiv d_1 \ (mod\ \phi(n))$
  - For $C = M^{e_1} \bmod n$, $C^{d_1'} \bmod n = M$

RSA: security

- It should be hard to
  - factor n
  - compute $d = e^{-1} \bmod n$ from $PU = (e, n)$
  - compute M from $PU = (e, n)$ and $C = M^e \bmod n$
- The most focused problem is to factor $n = pq$
  - The best known factorization algorithm is the general number field sieve algorithm (GNFS) with complexity:

$$e^{\left(\left(\frac{8}{3}\right)^{2/3}+o(1)\right)\cdot(\ln n)^{1/3})(\ln \ln n)^{2/3}}$$

  - The difficulty is about the same level as discrete logarithm problem

Factorization: progress up to 2013

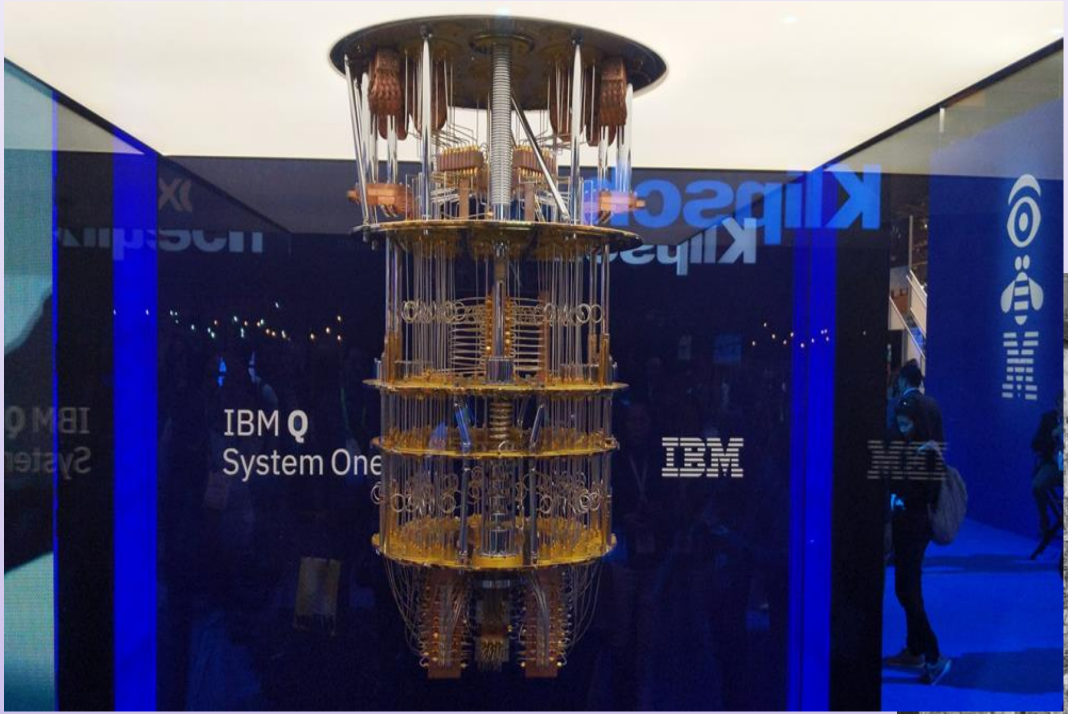
- $2^{1061} - 1$ (1061 bits , 320 digits) was factored by Greg Childers, etc, 2012
- The 696-bit RSA-210 was factored by Ryan Propper, 2013

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

Quantum computing

- ***Superposition***: simultaneous storage of bits 0 and 1 in one qbit
- ***Entanglement***: quantum computing is to entangle qbits by quantum gates
- State-of-the-art quantum computers
  - Osprey: 433 qbits, 2022, IBM
  - 九章三號 : 255 qbits, 2023
  - Quantum annealing: ≥ 2000 qbits, D-Wave
- Don’t expect to get it on you desktop anytime soon

IBM Q System 1

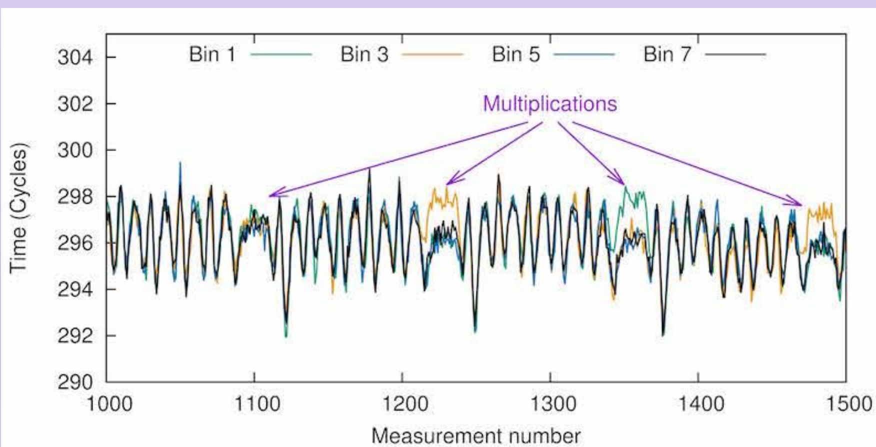


Quantum factorization

- Shor's quantum factoring algorithm
  - factoring n in poly($\log_2 n$) time, 1994
- D-wave's quantum annealing
  - Factor $376289 = 571 \times 659$ 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
  - To factor 1024-bit n, need
    - 2048 qbits theoretically
    - 2048x100 -- 2048x10000 qbits practically, by estimation
- Remark: symmetric-key encryption is still safe

Side-channel attacks

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

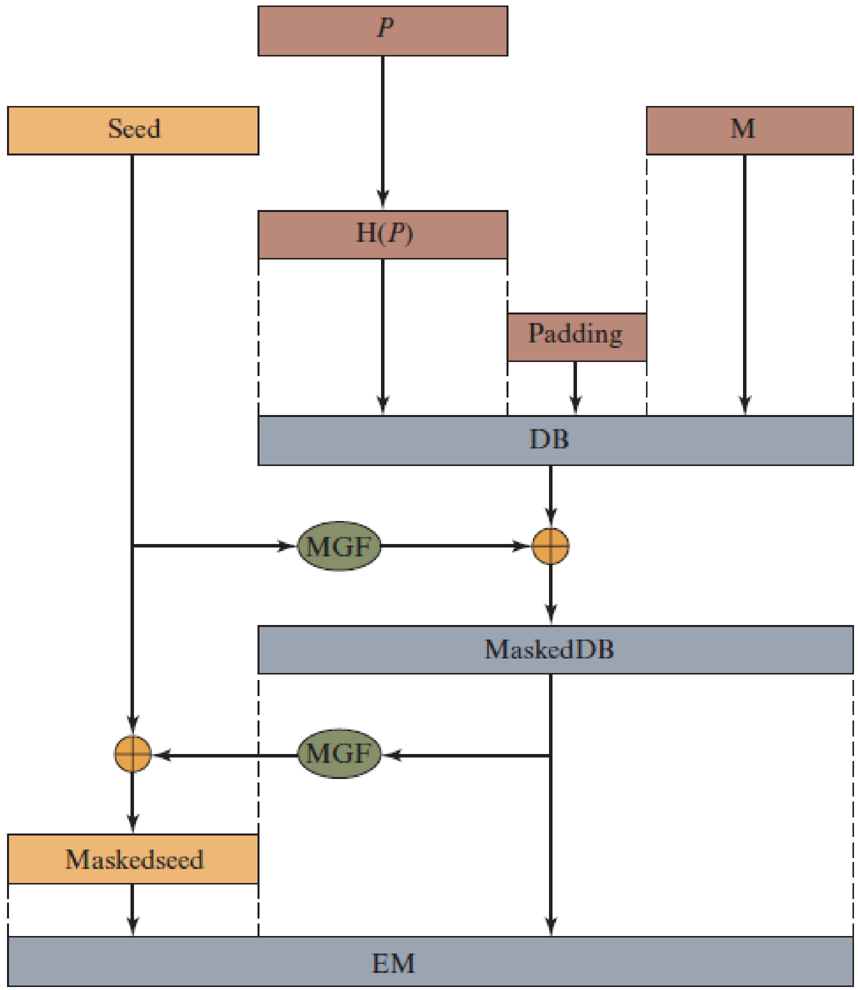



```
c ← 0; f ← 1
for i  ← k downto 0
      do   c ← 2 × c
           f ← (f × f) mod n
      if   b_i = 1
           then c ←  c + 1
                f ← (f × a) mod n
return f
```

Chosen ciphertext attack

- Given a ciphertext C, decrypt it, but allow to ask plaintext of his chosen ciphertext $C' \neq C$
- The attack
  - Compute $C_1' = C \times r^e \bmod n$, where r is randomly picked
  - Ask plaintext of C' and obtain $x = \mathrm{C'}^d \bmod n$
  - Compute $M = xr^{-1} \bmod n = C^d \bmod n$
- Countermeasure: encrypt plaintext into OAEP-form ciphertext
  - OAEP: Optimal Asymmetric Encryption Padding

OAEP: schema



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