RSA

and

Number Theory

What is public key cryptography? Why is there a need?

- Asymmetric vs. Symmetric
- Problems solved by public key
  - Shared secret not needed
  - Authentication
- Trapdoor one-way function
  - Factoring integers
  - Discrete logs
- Slow, power hungry

Public Key Cryptographic Use

- Secure RPC
- SSL
- Cisco encrypting routers

Public Key Cryptosystem Security

- can never provide unconditional security
- Try all possible plaintexts since public key is known
- When you mach with the ciphertext ➔ corresponding plaintext is known

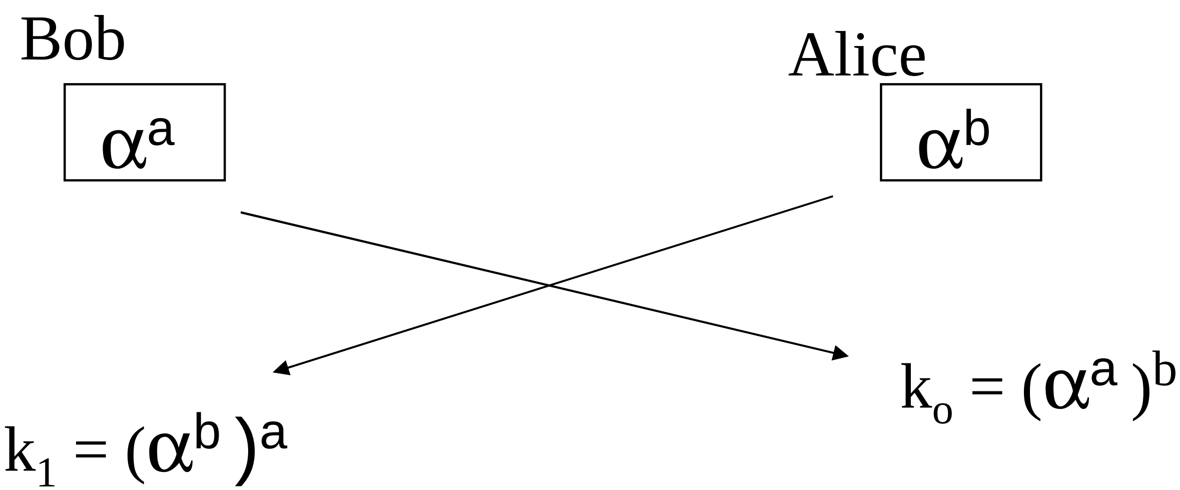
Where did public key cryptography come from?

- Diffie and Hellman
  - Credited with invention (circa 1976)
  - One year later, RSA is invented
  - April 2002, ACM communications
- 1973 James Ellis (British Gov’t)
  - “The possibility of non-secret encryption”
  - NSA claims

Key distribution

- Alice and Bob need to talk
- Insecure channel of communication
- First, set up our field that our numbers will operate within:
  - p, a large prime (sets up something called our field)
  - α is called a primitive root of Fp

Alice and Bob obtain a private key using public keys



So, $k_1 = k_o$, and a secret key is shared between Alice and Bob.

What does the adversary know, and what can he do?

- Knows α^a, α^b, α, and p
- So we want to find the key, k
  - $k = \alpha^{ab}$
  - This is believed to be hard.
- If one knows how to compute discrete logs efficiently, then one can break this scheme (and other schemes based on public key cryptography)

trapdoor one-way function

- one-way function
  - easy to compute but hard to invert
  - Example:
    - *Given:* $31 = 2^b \bmod 127$, *Find b??* (DL problem)

- trapdoor function
  - Is one-way function but easy to invert with extra secret knowledge or private info (knowledge of a certain trapdoor)

Overview

- RSA
  - Rivest, Shamir, Adleman, 1977
- Z_n
  - Modular operations (the expensive part)
  - A sender looks up the public key of the receiver, and encrypts the message with that key
  - The receiver decrypts the message with his private key
  - Although, public key is public information, private key is secret but related to the public key in a special way

Overview of Public Key Cryptosystem (PKC)

- Integer factorization problems (RSA)
- Discrete Logarithm problems (Diffie-Helman, ElGamal)
- Elliptic Curve Cryptosystems

Algorithm family	Bit length
Integer Factorization (IF)	1024
Discrete Logarithm (DL)	1024
Elliptic curves (EC)	160
Block cipher	80

Security levels of PKCs

PKC Standards

- **IEEE P1363:** Comprehensive standard of PKC. Collection of IF, DL and EC, in particular:
  - Key establishment algorithms
  - Key transport algorithms
  - Digital Signature algorithms
- **PKCS** (Public key cryptography standard) by RSA
  - PKCS #1: RSA Cryptography Standard
  - PKCS #3: Diffie-Hellman key agreement Standard
  - PKCS #13: Elliptic Curve Cryptography Standard

PKC Standards

- ANSI Banking Standards (ANSI=American National Standards Institute)
  - Elliptic curve key agreement and transport protocols X9.63
  - Elliptic curve digital signature algorithm (ECDSA) X9.62
  - Key management using Diffie-Hellman X9.42
  - Hash algorithms for RSA X9.32-2
  - RSA signature algorithm X9.31-1
  - Hash algorithm for RSA X9.30-2
  - Digital Signature Algorithm (DSA) X9.30-1
- US Government Standards
  - Entity authentication FIPS ????
  - Digital Signature Standard (DSA) FIPS 186
  - Secure hash standard (SHA-1) FIPS 180-1

The RSA cryptosystem

- First published:
  - Scientific American, Aug. 1977.
    (after some censorship entanglements)

- Currently the "work horse" of Internet security:
  - Most Public Key Infrastructure (PKI) products.
  - SSL/TLS: Certificates and key-exchange.
  - Secure e-mail: PGP, Outlook, …

RSA

- Most popular PKC
- 1977 Invented at MIT by Rivest,Shamir,Adleman
- Based on *Integer Factorization* problem
- Each user has public and private key pair.
- Its patent expired in 2000.

RSA

- *Choose: p, q* $\in$ positive distinct *large primes*
- *Compute:* $n = p \times q$
- n = encryption/decryption modulus $\rightarrow$ computations in Z_n
- *Compute:* $\varphi(n) = (p - 1)\,(q - 1)$
- *Choose randomly:* $e \in Z_{\varphi(n)}{}^{*}$
- $\rightarrow$ $\gcd(\varphi(n), e) = 1$, (e has an inverse mod $\varphi(n)$)
- Find $d = e^{-1} = ??\ mod\ \varphi(n)$

- *Encryption:* $c = x^e \bmod n$ where $x < n$
- *Decryption:* $x = c^d \bmod n$
- *n*,*e* are made public but *p*,*q*,*d* are secret

The RSA trapdoor 1-to-1 function

- Parameters: $\begin{cases} N=pq. \quad N \approx 1024 \text{ bits.} \quad p,q \approx 512 \text{ bits.} \\ e - \text{encryption exponent.} \quad gcd(e, \varphi(N)) = 1 . \end{cases}$

- 1-to-1 function: $\mathbf{RSA(M) = M^e}$ (mod N) where $M \in Z_N^*$

- Trapdoor: **d** – decryption exponent.

  Where $e \cdot \mathbf{d} = 1 \pmod{\varphi(N)}$

- Inversion: $\mathbf{RSA(M)^d} = M^{ed} = M^{k\varphi(N)+1} = \mathbf{M} \pmod{N}$

- (n,e,t,ε)-RSA Assumption: For any t-time alg. *A*:

$$\Pr\left[A(N,e,x) = x^{1/e}\ (N) \ : \ \begin{matrix} p,q \xleftarrow{R} \text{n-bit primes,} \\ N \leftarrow pq, \quad x \xleftarrow{R} Z_N^* \end{matrix} \right] < \varepsilon$$

Example: RSA encryption & decryption

Bob

(1) chooses $p = 3$, $q = 11$

(2) $n = pq = 33$

(3) $\varphi(n) = (p-1)(q-1)=20$.

(4) Chooses $e = 3$; gcd(3,20)=1

(5) Computes $d \alpha e^{-1} \bmod \varphi(n)$

$d \alpha 7$

(6) Sends (e, n) to Alice

Alice

(1) Message: $x = 4$

(2) $y \equiv x^e \bmod n \equiv 31$

(3) Sends y to Bob

(7) $x \equiv y^d \bmod n \equiv 4$

Example: RSA digital signature

Bob

(1) chooses $p = 3$, $q = 11$

(2) $n = pq = 33$

(3) $\varphi(n) = (p-1)(q-1)=20$.

(4) Chooses $e = 3$; gcd(3,20)=1

(5) Computes $d \alpha e^{-1} \bmod \varphi(n)$

$d \alpha 7$

(6) Sends (e, n) to Alice

Alice

RSA Keys generation Same as RSA encryption & decryption

(1) Message to be signed: $x = 4$

(2) $y \equiv x^e \bmod n \equiv 31$

(3) Sends x & y to Bob

(7) Compute $y^d \bmod n \equiv 4$

(8) If $x \equiv y^d \bmod n$ *(signature verified)*

RSA keys …. Example (simple)

- $p = 11$, $q = 5$ ➔ $n = 55$
- $\varphi(n) = 10 \times 4 = 40 \qquad = 2^3 \times 5$
- an integer e can be used as an encryption exponent if and only if e is not divisible by 2, 5
- We do not need to factor $\varphi(n)$ to get e
- Just verify: $\gcd(\varphi(n), e) = 1$ (Euclidean algorithm)
- *Assume:* $e = 7$ (public key)
- Extended Euclidean algorithm $\Rightarrow e^{-1} = ??\ mod\ 40$
- Secret exponent key: *23*
- other pares: $e=3,\ e^{-1}=??$ $\quad e=9,\ e^{-1}=??$ $\quad e=11,\ e^{-1}=??$ $\quad e=13,\ e^{-1}=??$ $\quad e=17,\ e^{-1}= \ ??$ $\quad e=19,\ e^{-1}= \ ??$
- $\mathbf{Z_{40}^{*}=\{1,3,7,9,11,13,17,19,21,23,27,29,31,33,37,39\}}$
- $e=3,\ e^{-1}=27$ $\quad e=13,\ e^{-1}=37$ $\quad e=17,\ e^{-1}=33$ $\quad e=e^{-1}= \{9, 11, 19, 21, 29, 31, 39\}$

RSA idea....Example

- p = 101 , q = 113 ➔ n = 11413
- $\varphi(n) = 100 \times 112 = 11200 \quad = 2^6 5^2 7$
- an integer e can be used as an encryption exponent if and only if e is not divisible by 2, 5 or 7
- We do not need to factor $\varphi(n)$ to get e
- Just verify: $\gcd(\varphi(n), e) = 1$ (Euclidean algorithm)
- *Assume:* e = 3533 (public key)
- Extended Euclidean algorithm $\Rightarrow$ $e^{-1} = 6597 \bmod 11200$
- Secret exponent key: 6597

Some notes about e, d, p, and q

- p and q must be large for security
- e, the encryption exponent, does not have to be that large ($2^{16} – 1 = 65535$ is good)
- d, the decryption exponent, needs to be sufficiently large (512 to 2048 bits)
- Having to work with such large numbers, we need to look at some other elements of RSA.

RSA: Component Operations

- Factorization
  - Believed to be difficult (security is here)
- Exponentiation
  - We need to do it fast
- Generating prime numbers
  - Mersenne Primes
  - Fermat Primes
- Testing primality
  - Fermat Test
  - Square Root test
  - Miller-Rabin test
- http://mathworld.wolfram.com/news/2002-08-07_primetest/
- http://www.cse.iitk.ac.in/primality.pdf

Some Number Theory

Factorization

- Brute force is stupid and slow
  - d = 1,2,3,4,… Does d divide n?
  - Factoring n = pq. If $p \le q$, $n \ge p^2$, so $\sqrt{n} \ge p$
  - d can go high as $\sqrt{n}$ in worst case
  - For $n \sim 10^{40}$, 10^{20} number of divisions
- Use structure of Z_n
  - p –1 method (not really used, but a good speedup)
  - Pollard’s rho method
  - Quadratic sieve, Number Field Sieve (NFS)
  - Is there a better method out there?

Prime Numbers

- **prime number *p*:** $p > 1$ and divisible only by 1
- **composite number:** integer not prime

Prime Number Theorem:

- # of primes in positive integer x = $\mathbf{x / \ln x}$
- **for x=$\mathbf{10^{10}}$,** # of primes = 434,294,481

Theorem: Every positive integer is a product of primes. This factorization is unique.

- If p is a prime and it divides a product of integers $a \cdot b$
- then either $p \mid a$ or $p \mid b$.

Z_n^*

- Z_n is a ring for any positive integer n
- $b \in Z_n$
- When b^{-1} exist?
- b^{-1} exist if and only if $gcd(b, n) = 1$
- Z_n^* is a ring with elements relatively prime to n
- Z_n^* has all elements with *multiplicative inverses*
- $|Z_n^*|$ = *order* of Z_n^* = number of elements
- Z_n^* *is closed under multiplication*
  - $x , y \in Z_n^*$ *(x , y are relatively prime to n*)
  - *x.y is relatively prime to n*

Integers: $a > 0$ & $p \in$ prime

(i) (*Fermat's little theorem*) ~1600s

If $gcd(a, p) = 1$, then

$$a^p = a \ (mod\ p)$$

$$a^{p-1} = 1 \ (mod\ p)$$

(ii) (Euler's theorem) ~1700s

If $r = s \ mod\ (p - 1)$, then $a^r = a^s \ (mod\ p)$

when working modulo a prime p, exponents can be reduced modulo $p - 1$.

If $gcd(a, n)=1$, then $a^{\varphi(n)} \equiv 1 \pmod{n}$

where $\varphi(n)$ is defined as *the number of integers* $1 \leq a \leq n$ such that $gcd(a, n)=1$ and called as *Euler's φ-function.* ➔ $\varphi(p) = (p-1)$

Congruence Classes (analogy)

- Let a, b, and n be integers with $n \neq 0$. We say that
  - $\rightarrow$ $a \equiv b \pmod{n}$ (*a is congruent (equivalent) to b mod n*)
  - $\rightarrow$ if $a - b$ is a multiple of (positive or negative) n.
  - $\rightarrow$ Thus, $a = b + k \cdot n$ for some integer k (positive or negative)

Proposition: a, b, c, d, n integers with $n \neq 0$ and

$$a \equiv b \pmod{n} \text{ and } c \equiv d \pmod{n}.$$

Then

- ✓ $a + c \equiv b + d \pmod{n}$
- ✓ $a - c \equiv b - d \pmod{n}$
- ✓ $a \cdot c \equiv b \cdot d \pmod{n}$

Division in Congruence Classes

We can divide by a (mod n) when $\gcd(a, n)=1$

- Example: Solve $2x + 7 \equiv 3 \pmod{17}$
- Example: Solve $5x + 6 \equiv 13 \pmod{15}$.

Proposition: Suppose $\gcd(a, n)=1$.

- Let s and t be integers such that $a \cdot s + n \cdot t = 1$.
- Then $a \cdot s \equiv 1 \pmod{n}$
- s is called *the multiplicative inverse of a* (mod n)

Extended Euclidean algorithm is a fairly efficient method of computing multiplicative inverses in congruence classes.

principle

- $a, n, x, y \in$ integers ; $n \geq 1$ and gcd(a, n)=1.
- If $x \equiv y$ (mod $\varphi(n)$) then

$$a^x \equiv a^y \pmod{n}.$$

- i.e., mod n, $\Rightarrow$ mod $\varphi(n)$ in the exponent.

Proof: $x = y + \varphi(n) \cdot k$ from congruence relation.

- Then
- $a^x = a^{y+\varphi(n)k} \equiv a^y \cdot (a^{\varphi(n)})^k \equiv a^y \cdot (1)^k \equiv a^y \ (mod\ n)$

Example

Example 1: $2^{10} = 1024 \equiv 1 \pmod{11}$

Example 2: Compute $2^{-1} \pmod{11}$.

- $2 \cdot 2^9 = 2^{10} \equiv 1 \pmod{11} \Rightarrow 2^{-1} \equiv 2^9 \pmod{11} \equiv 6 \pmod{11}$.

Example 3: $\varphi(10) = \varphi(2 \cdot 5) = (2-1) \cdot (5-1) = 4$.

- $\{1, 3, 7, 9\}$

Example 4: Compute $2^{43210} \pmod{101}$

- We know $2^{100} \equiv 1 \pmod{101} \Rightarrow$
- $2^{43210} = 2^{432 \times 100 + 10} = (2^{100})^{432} \cdot 2^{10} \equiv 2^{10} \pmod{101} \equiv 14 \bmod (101)$.

RSA idea....clarification

- $p, q \in$ positive *distinct primes*
- $n = p \times q$
- uses computations in Z_n
- $\varphi(n) = (p - 1)(q - 1)$
- $ab \equiv 1 \bmod \phi(n)$
- $ab = t\,\phi(n) + 1$
- $t \in$ *integer* > 0
- $x \in Z_n^*$
- $(x^b)^a \equiv x^{t\,\phi(n) + 1}$
- $\equiv (x^{\phi(n)})^t x \pmod n$ *See :* $x^{\varphi(n)} \equiv 1 \pmod n$
- $\equiv 1^t x \pmod n$
- $\equiv x \pmod n$
- $(x^b)^a \equiv x \pmod n$

Modular Exponentiation

$x^a \pmod{n}$

Example: $2^{1234} \bmod 789$,

- *Naïve method:* raise 2 to 1234 and then take the modulus.
- Is it practical (possible)?
- *Practical method*:
- Use binary expansion of the exponent.
- $1234 = (10011010010)_2$

Modular exponentiation example

2^{1234} mod 789 and 1234 = $(10011010010)_2$

- *1* $x = 2$
- *0* $x = 2 \cdot 2 = 4$
- *0* $x = 4 \cdot 4 = 16$
- *1* $x = 16 \cdot 16 = 256$ and $x = 256 \cdot 2 = 512$
- *1* $x = 512 \cdot 512 = 196$ and $x = 196 \cdot 2 = 392$
- *0* $x = 392 \cdot 392 = 598$
- *1* $x = 598 \cdot 598 = 187$ and $x = 187 \cdot 2 = 374$
- *0* $x = 374 \cdot 374 = 223$
- *0* $x = 223 \cdot 223 = 22$
- *1* $x = 22 \cdot 22 = 484$ and $x = 484 \cdot 2 = 179$
- *0* $x = 179 \cdot 179 = 481$

All operations are performed modulo 789

Idea Behind Fast Exponentiation

- a ^ 256 mod 7
  - Don’t do (a*a*a…*a) 256 times and mod by 7
- (a * b) mod p = (a mod p * b mod p) mod p
  - Shortcut: Look at binary representation of 256
  - $256 = 2^8$, $(((((((a^2)^2)^2)^2)^2)^2)^2)^2$ and mod 7 each time you perform a square
  - $25 = 11001 = 2^4 + 2^3 + 2^0$
    $a ^ 25 \bmod n = (a * a^8 * a^{16}) \bmod n$
    $= (a * (((a^2)^2)^2) * ((((a^2)^2)^2)^2)) \bmod n$

$(((((((a^2 \bmod n)*a) \bmod n)^2 \bmod n)^2 \bmod n)^2 \bmod n) * a) \bmod n$

Is RSA really secure??

- RSA :
  - public key: **(N,e)** Encrypt: $C = M^e \pmod N$
  - private key: **d** Decrypt: $C^d = M \pmod N$

  $(M \in Z_N^*)$

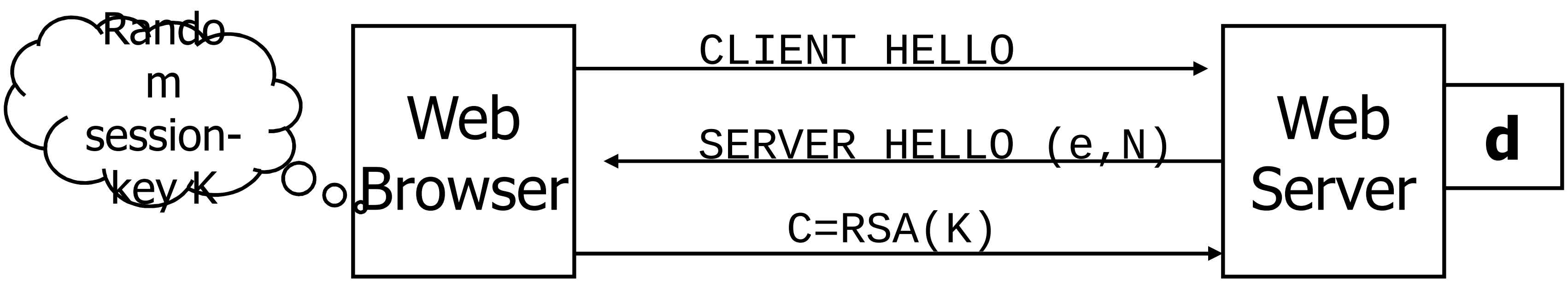
- Can RSA be an insecure cryptosystem???

  Many attacks exist.

Using RSA: What can go wrong?

- Computing $\phi(n)$ is no easier than factoring n
- From n = pq and $\phi(n) = (p-1)(q-1)$, we obtain:
  - $p^2 - (n - \phi(n) + 1)p + n = 0$
  - The roots of the above equation will be p and q
- If the decryption exponent, a is known, Bob needs to choose a new decryption exponent.
  - That isn't enough! Bob must also choose a new modulus.

A simple attack on textbook RSA

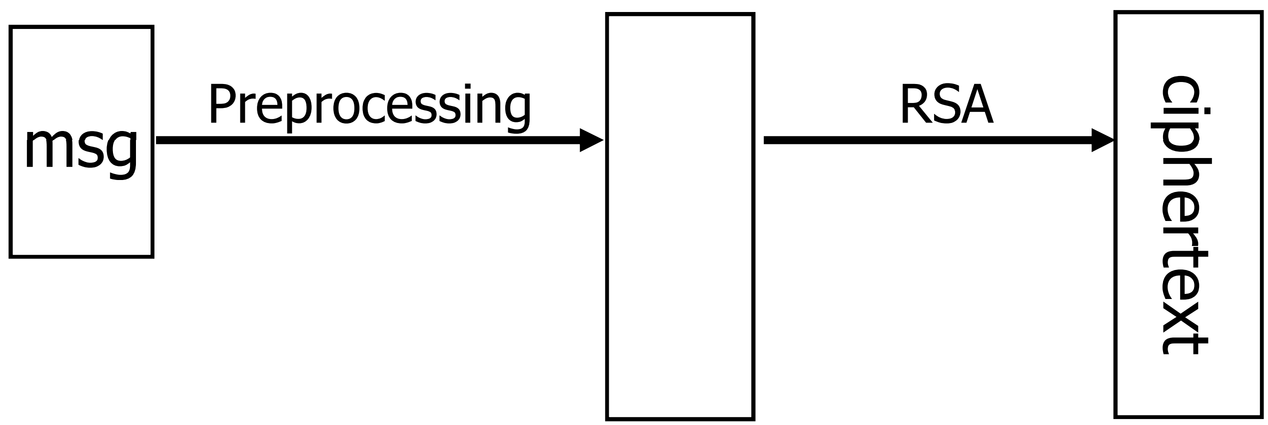


- Session-key K is 64 bits. View $K \in \{0,\ldots,2^{64}\}$
  Eavesdropper sees: $C = K^e \pmod{N}$.
- Suppose $\mathbf{K = K_1 \cdot K_2}$ where $K_1, K_2 < 2^{34}$. (prob. $\approx$20%)
  Then: $\mathbf{C/K_1^e = K_2^e} \pmod{N}$
- Build table: $C/1^e, C/2^e, C/3^e, \ldots, C/2^{34e}$. time: 2^{34}

  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

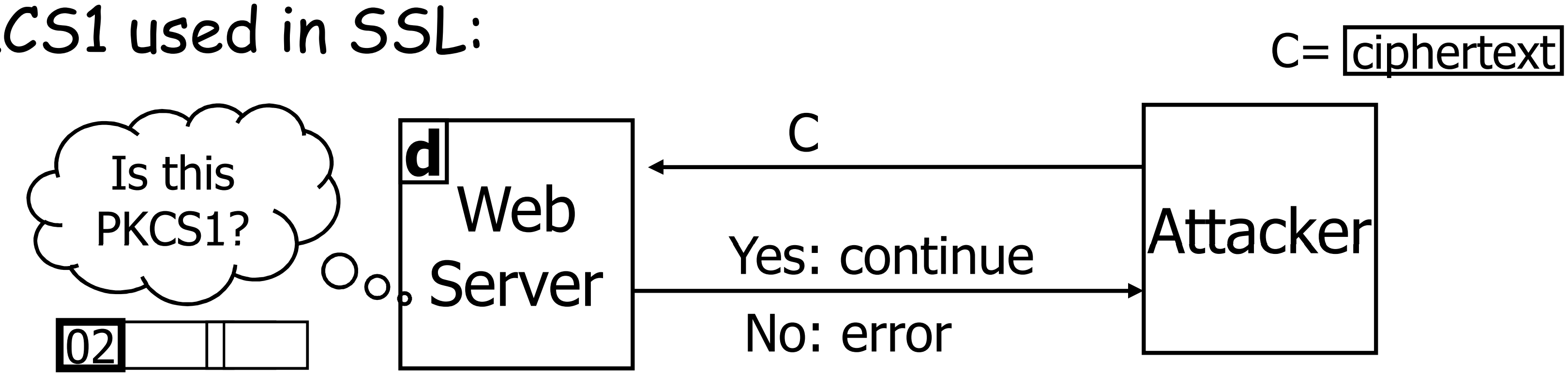
- Never use textbook RSA.
- RSA in practice:



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

Attack on PKCS1

- Bleichenbacher 98. Chosen-ciphertext attack.
- PKCS1 used in SSL:

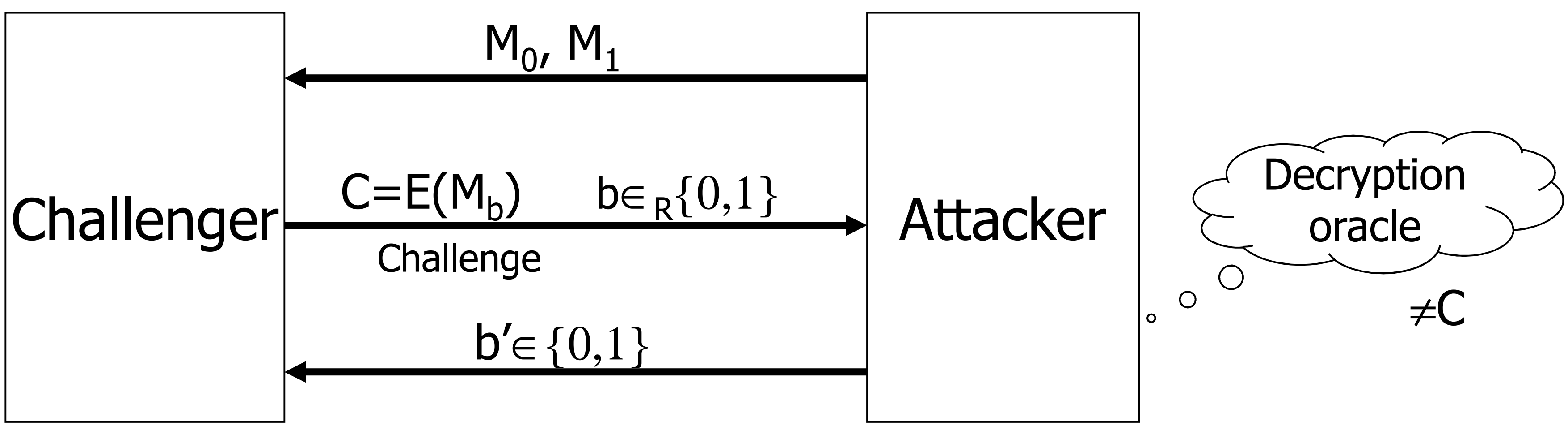


$\Rightarrow$ attacker can test if 16 MSBs of plaintext = '02'.

- Attack: to decrypt a given ciphertext C do:
  - Pick random $r \in Z_N$. Compute $C' = r^e \cdot C = (rM)^e$.
  - Send C' to web server and use response.

Chosen ciphertext security (CCS)

- No efficient attacker can win the following game: (with non-negligible advantage)



Attacker wins if $b=b'$

Is RSA a one-way permutation?

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

  M from $C = M^e \pmod N$.

- 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)

Shortcuts?

- Must one factor N in order to compute e'th roots? Exists shortcut for breaking RSA without factoring?

- To prove no shortcut exists show a reduction:
  - Efficient algorithm for e'th roots mod N
    $\Rightarrow$ efficient algorithm for factoring N.
  - Oldest problem in public key cryptography.

- Evidence no reduction exists: (BV'98)
  - "Algebraic" reduction $\Rightarrow$ factoring is easy.
  - Unlike Diffie-Hellman (Maurer'94).

RSA With Low public exponent

- To speed up RSA encryption (and sig. verify) use a small e. $C = M^e \pmod{N}$

- 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.

- <u>Asymmetry of RSA:</u> fast enc. / slow dec.
  - ElGamal: approx. same time for both.

Implementation attacks

- Attack the implementation of RSA.
- Timing attack: (Kocher 97)
  The time it takes to compute $C^d \pmod N$
  can expose d.
- Power attack: (Kocher 99)
  The power consumption of a smartcard while
  it is computing $C^d \pmod N$ can expose d.
- Faults attack: (BDL 97)
  A computer error during $C^d \pmod N$
  can expose d.
  OpenSSL defense: check output. 5% slowdown.

DES vs. RSA

- RSA is about 1500 times slower than DES
  - Exponentiation and modulus
- Generation of numbers used in RSA can take time
- Test n against known methods of factoring
  - http://www.rsasecurity.com/rsalabs/challenges/factoring/numbers.html

Key lengths

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

NIST:

Cipher key-size	Modulus size
≤ 64 bits	512 bits.
80 bits	1024 bits
128 bits	3072 bits.
256 bits (AES)	**15360** bits

- High security $\Rightarrow$ very large moduli.

  Not necessary with Elliptic Curve Cryptography.

key length for secure RSA

- key length for secure RSA transmission is typically 1024 bits. 512 bits is now no longer considered secure.

- For more security or if you are paranoid, use 2048 or even 4096
- With the faster computers available today, the time taken to encrypt and decrypt even with a 4096-bit modulus really isn't an issue anymore.
- In practice, it is still effectively impossible for you or I to crack a message encrypted with a 512-bit key.
- An organisation like the NSA who has the latest supercomputers can probably crack it by brute force in a reasonable time, if they choose to put their resources to work on it.
- The longer your information is needed to be kept secure, the longer the key you should use.

Key Distribution

- Then hard problem for symmetric (secret) key ciphers
- Transmitting a private key on an insecure channel
  - Asymmetric system solves problem

p & q generation recommendation

- To generate the primes p and q, generate a random number of bit length b/2 where b is the required bit length of n;
- set the low bit (this ensures the number is odd) and set the *two* highest bits (this ensures that the high bit of n is also set);
- check if prime; if not, increment the number by two and check again. This is p.
- Repeat for q starting with an integer of length b-b/2.
- If p<q, swop p and q (this only matters if you intend using the CRT form of the private key).
- In the extremely unlikely event that p = q, check your random number generator.
- For greater security, instead of incrementing by 2, generate another random number each time.

e & d recommendation

- In practice, common choices for e are 3, 17 and 65537 (2^16+1).
- These are Fermat primes and are chosen because they make the modular exponentiation operation faster.
- Also, having chosen e, it is simpler to test whether gcd(e, p-1)=1 and gcd(e, q-1)=1 while generating and testing the primes.
- Values of p or q that fail this test can be rejected there and then.
- To compute the value for d, use the *Extended Euclidean Algorithm* to calculate d = e^-1 mod phi (this is known as *modular inversion*).