Plan of Talk : Week 5

- From Last week
  - Security of Public Key Cryptography
  - Chosen Cipher text Attack
- RSA Signature
- Issues with Textbook version of RSA Signature
- Tomorrow: Hash Functions and Practical Signature Schemes

Summary:Hard problems on which RSA is based.

- 1. Integer Factorization problem: Given a large positive integer n, find its prime factorization. (Every number n can be expressed as

  $p1^{e1}p2^{e2} \ldots pk^{ek}$, where the pi's are distinct primes and each ei > 1). In particular, if a number n is constructed as a product of two large primes, it is difficult to factor n.
- 2. RSA problem: Given a positive integer n that is a product of two distinct odd primes p and q,(n=pq) and a positive integer e such that gcd(e,(p-1)(q-1)) = 1,and an integer c, find an integer m such that

  $m^e = c \pmod{n}$.

Security of Cryptosystems

- Almost all modern cryptosystems are based on more than one hard problems in mathematics (eg. Discrete logarithms, factorization, RSA problem etc).
- In fact there are no theoretical proofs available stating that these problems are hard.
- On the other hand, there are many instances where the so called hard problems are easy to perform.
- We should ensure that the practical implementation do not use such pathological cases. Hence, we have to address security against any known vulnerability of these hard problems.
- Such attacks based on specific vulnerability of instances of hard mathematical algorithms can be considered as Mathematical attacks. We look for active attacks next.

Security Notions for public key Cryptosystems: Active attacks

- The security of a cryptosystem is defined with respect to the attacks it can withstand.
- The attacker will not be given private or secret information of the cryptographic key whose public cryptosystem he is attacking.
- There are three types of active attacks:
  - **Chosen-plaintext attack(CPA)**
    - Encryption box is available to the attacker before the attack.
  - **Chosen-ciphertext attack(CCA)**
    - Decryption box is available to the attacker before the attack.
  - **Adaptive Chosen-ciphertext attack(CCA2)**
    - Decryption box is available to the attacker except for the challenged ciphertext.

Active attacks: Cont.

- **Chosen-plaintext attack(CPA)** Here the attacker can obtain cipher texts corresponding any chosen plain texts. The goal is to weaken the crypto system with the obtained plaintext-ciphertext pairs.
- **Chosen-ciphertext attack(CCA)** Here attacker can obtain plaintexts corresponding any chosen ciphertexts. This means the attacker gets decryption assistance for any chosen ciphertext. The goal for the attacker is to obtain any part of the plaintext after the decryption assistance is terminated.

Active attacks: Cont.

- **Adaptive Chosen-ciphertext** attack(CCA2) The attacker is challenged with a given ciphertext to decrypt or obtain any part of the plaintext. For this, he/She is provided decryption assistance forever for all chosen cipher texts except for the challenged ciphertext.
- The above notions are not impracticable and hence cryptosystems should be made secure against such attacks.

Chosen Ciphertext Attack

- The basic RSA algorithm is vulnerable to a chosen ciphertext attack (CCA).
- In this scenario, the adversary gets decryption of a number of ciphertexts of his choice.
- Adversary will then be given a challenge cipher text for which he has to produce the decryption (without having access to the private key).

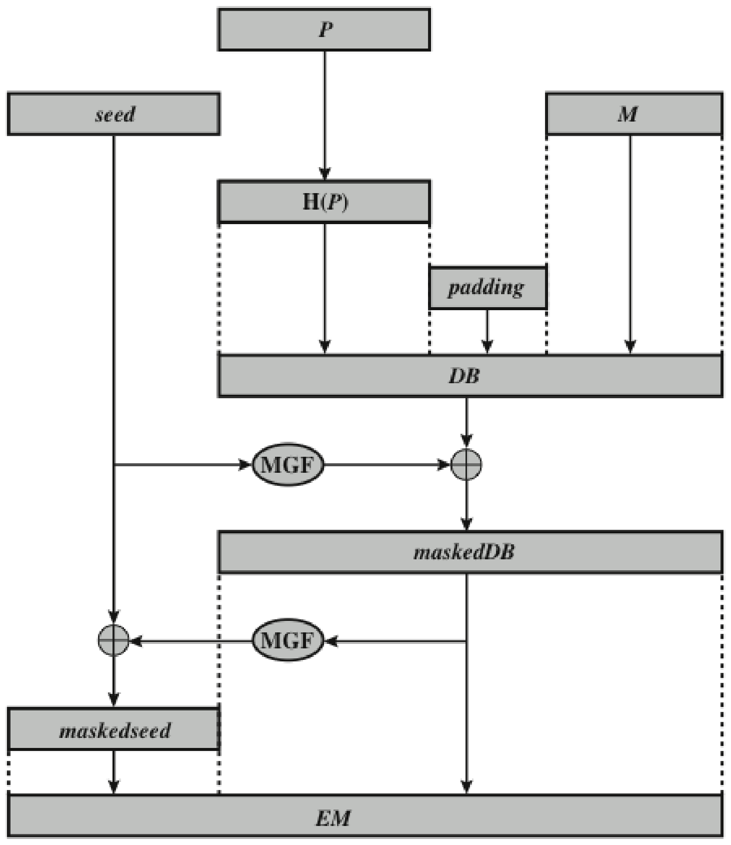
Attack

- Multiplicative property of RSA
  - E(PU,M1) x E(PU, M2) = E(PU, [M1 x M2])
- The attack procedure
  - 1. Compute X = (C x 2^e) mod n
  - 2. Submit X as a chosen ciphertext and obtain corresponding plain text Y = X^d mod n.
  - 3. Note that Y is in fact (2M) mod n
  - 4. Compute M = Inverse(2) *Y mod n

Optimal Asymmetric Encryption Padding

- To overcome the previous attack, you need somehow break the multiplicative property of the scheme.
- In practice message is introduced with a specific format, which removes the multiplicative property.

OAEP



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

Figure 9.10 Encryption Using Optimal Asymmetric Encryption Padding (OAEP)

RSA Computations

- RSA encryption and decryption involves exponentiation modulo n.
- Exponentiation(a, t, n): a^t mod n
- What is the complexity of this function?
  - Let n be a b bit binary number
  - Further assume that t is also of order b.
  - Make use of the identity:
    - [(*a* mod *n*) *x* (*b* mod *n*)*]* mod *n* =(*a x b*) mod *n*
- O(b) = O(log n).

Faster Method in the textbook

```
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 $n = 561$

Key Generation

- Each user (Alice) need to choose two primes p and q of the public modulus securely.
- Then e and d should be computed such that
  - e d = 1 mod Phi(n), Phi(n) = (p-1)(q-1).
- Primes need to be chosen from a sufficiently large set. (*n* = *pq* will be known to any potential adversary)
  - The method should be reasonably efficient
  - Preferably obtained from a true random source.

Choosing Primes: Primality Testing

[We will not be studying mathematical aspects of primality testing]

- Two methods: Probabilistic and deterministic
- Probabilistic Test:
  - 1. Pick an odd integer *m* at random (e.g., using a pseudorandom number generator-but should use truly random seed)).
  - 2. Pick an integer $a < m$ at random.
  - 3. Perform the probabilistic primality test, such as Miller-Rabin, with *a* as a parameter. If m fails the test, reject the value m and go to step 1.
- Deterministic Test
  - Complexity is polynomial, but still takes more time compared to the time taken by the probabilistic methods

Efficient Operation: Encryption

- For efficiency the RSA algorithm using the public key, a specific choice of *e* is usually made.
- The most common choice is 65537 ($2^{16} + 1$)
  - Two other popular choices are *e*=3 and *e*=17
  - Each of these choices has only two 1 bits, so the number of multiplications required to perform exponentiation is minimized
  - With a very small public key, such as *e* = 3, RSA becomes vulnerable to a simple attack

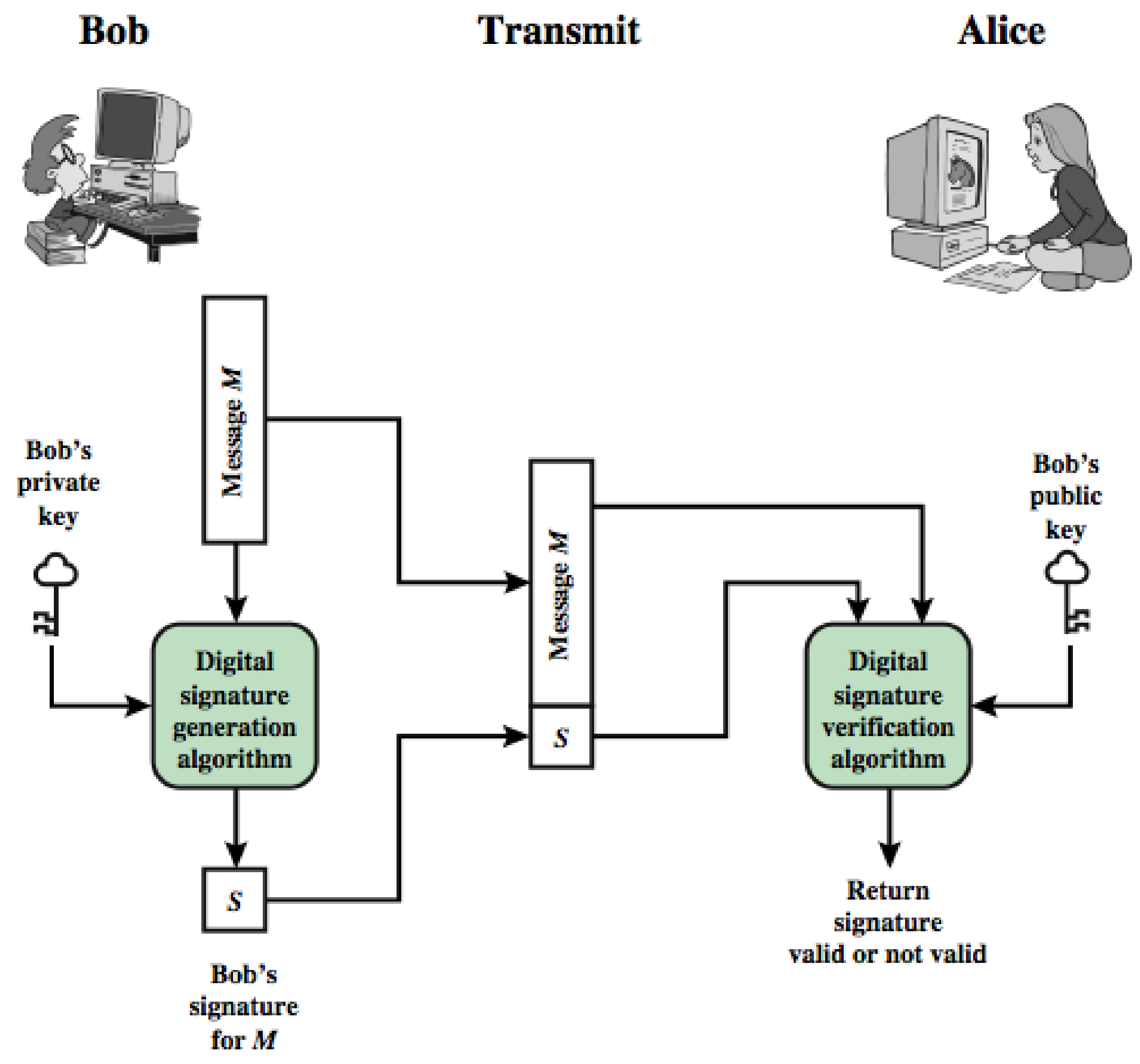
Efficient Operation: Decryption

[Please refer to the set of slides for mathematical aspects of RSA]

- Note that decryption uses exponentiation to power *d.*
  - Avoid a small value of *d;* vulnerable to a brute-force attack and to other forms of cryptanalysis

- The Chinese Remainder Theorem (CRT) method
  - The quantities d_p = *d* mod (*p* – *1)* and *d_q= d* mod (*q* – *1)* can be pre calculated
  - Compute C^d_p mod p and C^d_q mod q and then use CRT to compute M = C^d. You can show that the method is approximately four times as fast as evaluating $M = C^d \bmod n$ directly

Digital Signature Model


Bob
Transmit
Alice
Message M
Bob's private key
Digital signature generation algorithm
S
Bob's signature for M
Message M
S
Bob's public key
Digital signature verification algorithm
Return signature valid or not valid

Digital Signature Requirements

- The signature must be a bit pattern that depends on the message being signed.
- The signature must use some information unique to the sender to prevent both forgery and denial
- It must be relatively easy to produce the digital signature
- It must be relatively easy to recognize and verify the digital signature
- It must be computationally infeasible to forge a digital signature, either by constructing a new message for an existing digital signature or by constructing a fraudulent digital signature for a given message
- It must be practical to retain a copy of the digital signature in storage

Digital Signatures

- The purpose of this discussion is to understand how RSA signature works and how it is different from RSA encryption we studied earlier.
- RSA signature is a public key signature scheme
- Signature is a means for a trusted third party (Network Security Manager) to bind the identity of a user to a public key.

Digital Signatures cont.

- RSA signature is complement of RSA public encryption.
- Owner for the private key signs messages; Anyone with the public key can verify the signatures.
- In RSA encryption, anyone with the public key can encrypt messages and the owner of the private key can decrypt the messages.

RSA : Alice's parameters

- $N = P*Q$; P, Q Large Primes
- Choose Public key e and private key d such that $e*d \equiv 1 \bmod \phi(N)$
- Public address – [N,e]
- Private address –[d]
- Signature Generation:
- Message $0 < M < N$;
- Compute: $s = M^d \bmod N$;
- Signature—[M,s]
- Verification – if $s^e \bmod N$ =?= M

Some Consequences: Existential Forgery attack

Multiplicative property of RSA signature

$(M1 * M2)^d = M1^d * M2^d$

i.e. if s1 = Signature of M1;

s2 = Signature of M2

(s1 * s2) is the signature of (M1*M2).

This leads to a possibility of forgery of signature!

Further

- What if Message is very long?
- Also, a problem called blinding.

Blinding

- Alice's public key – [N,e]; Private key [d]
- You want to get Alice sign a message M which Alice normally may not do.
- Choose a random x – in the range [0..N-1]
- Form a blinded message -- $M_b = x^e M \mod N$
- Alice may sign $s_b = M_b^d \mod N$
- Now you can compute signature for M as
- $s = s_b/x \mod N$
- Note
- $s^e = s_b^e / x^e = (M_b)^{de} / x^e =$
  $(M_b) / x^e = x^e M / x^e = M$
- Hence s is the signature of M.

RSA Signature Modified

- Public address – [N,e] ; Private address –[d]
- Signature Generation:
- Message $0 < M < N$;
- Find $M1 = R(M)$; where R is a redundancy function,
- $1 < R(M) < N$.
- Compute: $s = (M1)^d \bmod N$;
- Signature—[M,s]
- Verification –
- Compute $s^e \bmod N = M1$
- Verify R(M) ==?==M1

RSA Signature in Practice

A practical signature scheme should take care of the two problems discussed before.

- Messages are generally long
- RSA signature scheme needs a redundancy function to avoid existential forgery attacks.
- Also repeated messages carry same signature.
- In practice, generally a suitable cryptographic hash function is applied to the message (which could be arbitrarily large); and sign the hash.

Practical RSA Signature Scheme

- Public address – [N,e] ; Private address –[d]
- Signature Generation:
- Message M of arbitrary length,
- Find h(M); where h is an hash function,
- Format the message before signing
- M1 as [h(M), identity information, random number]
- such that $0 < M1 < n$
- Compute: $s = (M1)^d \bmod N$;
- Signature—[M,s]
- Verification –
- Extract M1 by computing $s^e \bmod N = M1$
- Any formatting violations– reject the signature
- Further Verify h(M) ==?==M1

Security of RSA

- Brute force Attack: (infeasible given size of numbers)
- Attack by making use of loop holes in Key distribution.
- Mathematical attacks (Factoring and RSA problem)
    - Elementary attacks
    - Advanced Factorization methods
- Brute force Attack: (infeasible given size of numbers)
- Network attacks
    - Timing attacks (on running of decryption)

Mathematical attacks

- The RSA function is one way
- The problem
- Given n,e, $c=M^e$ (mod n),
  - Can we determine M?
  - Do we have an algorithm to find the e^{th} root of c mod n?
  - Can we find d such that de = 1 mod Φ(n) ?
- Can we factor n?

Factorization Problem

- In general the factorization is hard.
- Brute force algorithm is exponential in b, where b is number of bits in the representation of the number n to be factored.
- Complexity of the best known algorithm for factorization:

  $$\exp((c+O(1)b^{1/3} \mathrm{Log}^{2/3}(b)),$$

  for some integer $c < 2$
- It is not worth thinking of factoring.
- May be quantum computers come to our rescue; but may not in our life time!

Elementary attacks

- Can we use common modulus for more than one user?
- Facts:
- Knowing e and d such that

$$ed = 1 \bmod \Phi(n)$$

Is equivalent to factoring.

- Knowing n, $\Phi(n)$ is also equivalent to factoring

Common modulus

- Every user chooses same modulus n=pq set up by a trusted central authority. But each user chooses their own private and public key pairs
- User i ------(e_i,d_i)
- So using the facts in previous slide, any user can factor common modulus n and
- can find the private information of other user by using only the public information.
- Hence it is extremely important that every entity chooses its own RSA modulus n.

Broadcast Problem

A group of entities may all have a same encryption exponent but should have different modulus. Further, to improve the public encryption, let the public key be small, say e=3.

- A wishes to send a common message m to three entities with modulus n1,n2 and n3.
- The cipher text for three entities are given by
- $c1 = m^3 \pmod{n1}$
- $c2 = m^3 \pmod{n2}$
- $c3 = m^3 \pmod{n3}$.

- Then, to recover the message m solve,
- x = c1 (mod n1),
- x = c2 (mod n2),
- x = c3 (mod n3),

- You can use CRT and Then obtain an unique
- $x=m^3$ modulo n1 n2 n3
- m can then be obtained by taking the cube root of x.

RSA-PSS

- RSA Probabilistic Signature Scheme, included in the 2009 version of FIPS 186
- Latest of the RSA schemes and the one that RSA Laboratories recommends as the most secure of the RSA schemes
- For all schemes developed prior to PSS is has not been possible to develop a mathematical proof that the signature scheme is as secure as the underlying RSA encryption/decryption primitive
- The PSS approach first proposed by Bellare and Rogaway
- This approach, unlike the other RSA-based schemes, introduces a randomization process that enables the security of the method to be shown to be closely related to the security of the RSA algorithm itself
- More details on the algorithm are available in the textbook.

Timing Attacks

- Attack introduced by Paul Kocher in 1996.
- If implemented correctly, an attacker can recover private key by keeping track of how long a receiver computer takes to decipher messages.
- This is a serious attack as the previous models do not address this attack.
- The worry is that it is applicable many other crypto systems including symmetric key ciphers.

Some typical methods for Timing attacks

- Constant exponentiation time: Ensure that all exponentiations take the same amount of time before returning a result. This is a simple fix but does degrade performance.
- Random delay: Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack.
- Blinding: Multiply the ciphertext by a random number before performing exponentiation. This process prevents the attacker from knowing what ciphertext bits are being processed inside the computer and therefore prevents the bit-by-bit analysis essential to the timing attack.

Fault-Based Attack

- An attack on a processor that is generating RSA digital signatures
  - Induces 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 algorithm involves inducing single-bit errors and observing the results
- While worthy of consideration, this attack does not appear to be a serious threat to RSA
  - It requires that the attacker have physical access to the target machine and is able to directly control the input power to the processor

Summary

- Security of Public Key Algorithms
- CCA Attack on Textbook RSA
- Textbook RSA Signature Algorithm
- Blinding
- RSA Attacks