UnitelmaSapienza
Università degli Studi di Roma

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Cryptography and its Applications
Part II

Francesco Parisi Presicce

Recall Symmetric Crypto

When two parties want to communicate securely using symmetric cryptosystems:

(1) Alice and Bob agree on a cryptosystem and on a key
(2) Alice encodes plaintext using this key
(3) Alice sends resultant ciphertext to Bob
(4) Bob receives and decrypts ciphertext using key

The **key distribution problem** of secret key systems

- Must share the secret key with other party before initiating communication. Key distribution done secretly (difficult when parties are geographically distant, or don't know each other)
- If you want to communicate with n parties, you require n different keys
- Need a key for each pair of users
  - 10 users need 10*(10-1)/2=45 keys,
  - 18 users need 153 keys.

Public Key cryptosystems

- Public key cryptosystems solve the key distribution problem for secret key systems (if a reliable channel for communication of public keys can be implemented)
- Requires the **reliable** (not secret) dissemination of one public key per party and thus scales well for large systems
- Concept conceived by Diffie and Hellman in 1976 (discovered by J.Ellis (UK CESG) in 1970 but classified report)
- Rivest, Shamir and Adleman (RSA) were first to describe a public key encryption system in 1978
- Merkle and Hellman published a different solution, later in 1978
- Many proposals have been broken (including the 1978 Merkle-Hellman proposal broken by Shamir)

Serious candidates today (in public domain)

- RSA
- El Gamal
- based on Elliptic curve

A revolution of sort

- Diffie and Hellman sought to solve 2 problems:
  1. Find a better way to *distribute keys*
  2. provide for a *digital* document *signature*
- public key encryption is based on mathematical functions, not on substitution and permutation
- asymmetric -- two different keys, one kept private, one made public, generated by the principal
  - Things encrypted with the private key may only be decrypted with the corresponding public key
  - Things encrypted with the public key may only be decrypted with the corresponding private key
- Succeeded only partially (1) - NO encryption algorithm

Diffie-Hellman Key Exchange

- Proposed in 1976; first public key system (predates RSA)
- Allows a group of users to agree on a secret (session) key over an insecure channel. **Not used to encrypt messages**
- Suppose Alice and Bob want to agree on a shared key
- They agree on two large integers p and g such that $1<g<p$ (these will be shared by every member of a group)
- No prior communication between Alice and Bob needed
- Security depends on the difficulty of computing the private key privA given the public key $pubA = g^{privA} \bmod p$ (discrete logarithm problem, still hard to solve)
- Choices for g and p are critical:
  - both p and $(p-1)/2$ should be prime,
  - p large (at least 512 bits, possibly 1024 bits),
  - g is a primitive root mod p (i.e., $x^{ø(p)} = 1 \bmod p$)

Diffie-Hellman Key Exchange

- Alice chooses a random privA (private key), computes (the public key) pubA = g^{privA} mod p, and sends it to Bob
- Bob chooses a random privB (private key), computes (the public key) pubB = g^{privB} mod p, and sends it to Alice
- Alice computes k = pubB ^{privA} mod p
- Bob computes k' = pubA ^{privB} mod p
- Note that
- k= pubB ^{privA} mod p = g $^{privA\ privB}$ mod p= pubA ^{privB} mod p =k' and thus Alice and Bob now shared a session key
- If someone is listening, he/she knows p, g, pubA, and pubB, but not privA and privB

Diffie-Hellman Key Exchange

Susceptible to intruder-in-the-middle attack:

- Mal notices Alice sending Bob her public key K_{PubA} , and intercepts it.
- Mal then sends Bob his public key: K_{PubM} , claiming it is Alice’s
- Bob now sends Alice his public key. Again, this is intercepted by Mal, who substitutes K_{PubM}
- When Alice sends a message to Bob, she will use K_{PubM}
- Mal can intercept this and read it (with his own private key)
- Mal can generate an ‘appropriate’ substitute message, encode it with Bob's public key K_{PubM} , and send the new message to Bob

active malicious intruder

Notation

$C = E(KU,M)$ also $E_{KU}(M)$

$M = D(KR,C)$ also $D_{KR}(C)$

KU : Public (encryption) key, known to all

KR : Private (decryption) key, known only to B

E : Encryption Algorithm

D : Decryption Algorithm

M : Plaintext Message

C : Ciphertext Message

Two possible uses of public key

- confidentiality
  - A wants to send message to B
  - A encrypts the message with B’s public key
  - A sends the encrypted message to B
  - B decrypts the message with its private key
- authentication, or digital signature
  - A wants to send a message to B so that B is assured that A (and no one else) sent it
  - A encrypts the message with A’s private key
  - A sends the encrypted message to B
  - B decrypts the message with A’s public key
  - B then knows that only A could have sent it

Requirements for Public Key

- Computationally EASY to
  - generate a pair of keys (public KU, private KR)
  - encrypt, given the key KU and the message M
  - decrypt, given the key KR and the encrypted message C
- Computationally INFEASIBLE to
  - determine the private key KR, knowing the public key KU
  - recover the original message M, given public key KU and the ciphertext C (for the message M)
- To make this computation not feasible, key size is no smaller than 512 bits (better 1024)

RSA

- public key KU is (n,e)
- secret key KR is (n,d)
  - n is (at least) a 200 digit number

message M is represented as an integer from 0 to n-1

- $C = M^e \bmod n$
- $M = C^d \bmod n$

Why should it be the case that if M is a plaintext and C is a ciphertext and $C = M^e \bmod n$, that

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

How do we know that *there even exist* e and d such that $M^{ed} \bmod n = M$?

Modular Arithmetic

- We say that a is <u>congruent</u> <u>to</u> b <u>modulo</u> n, written $a = b \bmod n$, if $a - b = k*n$ for some integer k
- If $b < n$, b is also called the <u>residue</u> of a modulo n
- $a^{-1} = x \bmod n$ if $a*x = 1 \bmod n$
- $a^{-1} = x \bmod n$ has a unique solution if a and n are relatively prime

Examples

12 = 2 mod 5 ; 2 = 12 mod 10 ; 12 = 0 mod 6

Properties

- (a + b) mod n = ((a mod n) + (b mod n)) mod n
- (a - b) mod n = ((a mod n) - (b mod n)) mod n
- (a * b) mod n = ((a mod n) * (b mod n)) mod n
- (a * (b + c)) mod n = ((a*b) mod n) + (a*c) mod n)) mod n

We exploit these properties when we calculate $a^x \bmod n$

$a^{16} \bmod n = (((a^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 \bmod n$

Algorithm: exponentiation by repeated squaring and multiplication

- Computing M^e (mod n) takes at most $2*\log_2(e)$ multiplications and $2*\log_2(e)$ divisions
- Step 1. Let $e_k, e_{k-1}, \ldots e_1, e_0$ be binary rep. of e
- Step 2. Set the variable C to M
- Step 3. Repeat 3a and 3b for i=k-1,…,0 :
  - Step 3a. Set C to the remainder of C^2 when divided by n
  - Step 3b. If $e_i = 1$, then set C to the remainder of C*M when divided by n
- Step 4. Halt. Now C is the encrypted form of M

Theory behind RSA Cipher

Theorem (Euler and Fermat)

If p, q primes , n = p*q , and if gcd(x,n)=1 then:

$$x^{\varnothing(n)} = 1 \bmod n$$

for this choice of p and q, $\varnothing(n)=(p-1)*(q-1)$

If $e*d=1+q*\varnothing(n)$ (e and d inverses mod $\varnothing(n)$)

then

$$C^d = M^{e*d} = M^{1+q*\varnothing(n)} =$$

$$M^1 *(M^{\varnothing(n)})^q = M^1*(1)^q = M^1 \bmod n = M$$

How to find large primes

- 100-digit to 200-digit primes are recommended
- large primes can be found efficiently by generating large random odd numbers and then using probabilistic algorithms due to Solvay-Strassen or Miller-Rabin
  - Algorithms are fast: testing in time polynomial in $\log_2 n$ (the binary representation of n)
  - Algorithm could be wrong but repeating tests can reduce error arbitrarily
  - How many random integers tested until found? about 115 in average (Prime Number Theorem says that the number of primes <n tends to n/ln n for large n)

Generation of keys in RSA

- choose 2 large (100 digit) primes p and q
  - care must be exercised in choosing p and q, otherwise insecurities may result (p-1, p+1, q-1, q+1 should have large prime factors)
- compute $n = p * q$
- choose random e relatively prime to $(p-1)*(q-1)$
- compute $1<d< \phi(n)$ so that $e*d = 1 \bmod (p-1)*(q-1)$ (i.e., $d = e^{-1} \bmod (p-1)*(q-1)$ is the inverse of e) using Extended Euclidean Algorithm
  - If the factorization of n into p*q is known, this is easy to do.
- publish (n,e)
- keep (n,d) secret (and destroy p and q)
  - How hard is it to compute d given only (n,e)?
    - The security of RSA is no better than complexity of the factoring problem

RSA Keys — Example

- choose 2 large (100 digit) prime numbers p and q
  $p = 47, q = 71$
- compute $n = p * q$
  $n = p*q = 3337$
- choose e relatively prime to $(p-1)*(q-1)$
  for example $e = 79$ has no factors in common with
  $(47-1) * (71-1) = 46 * 70 = 3220$
- compute $d = e^{-1} \bmod (p-1)*(q-1) = 79^{-1} \bmod 3220 = 1019$
  (the inverse of a number modulo n can be computed using the extended Euclidean algorithm)
- publish (3337, 79)
- keep $d=1019$, $p=47$, $q=71$ secret

Example: Confidentiality

- Take $p = 7$, $q = 11$, so $n = 77$ and $\phi(n) = 60$
- Alice chooses $e = 17$, making $d = 53$
- Bob wants to send Alice secret message HELLO (07 04 11 11 14)
  - $07^{17} \bmod 77 = 28$
  - $04^{17} \bmod 77 = 16$
  - $11^{17} \bmod 77 = 44$
  - $11^{17} \bmod 77 = 44$
  - $14^{17} \bmod 77 = 42$
- Bob sends 28 16 44 44 42 received by Alice
- Alice uses private key, $d = 53$, to decrypt message:
  - $28^{53} \bmod 77 = 07$
  - $16^{53} \bmod 77 = 04$
  - $44^{53} \bmod 77 = 11$
  - $44^{53} \bmod 77 = 11$
  - $42^{53} \bmod 77 = 14$
- Alice translates message to letters to read HELLO
  - No one else could read it, as only Alice knows her private key and that is needed for decryption

Integrity/Authentication

Take p = 7, q = 11, so n = 77 and $\phi(n)$ = 60

- Alice chooses e = 17, making d = 53
- Alice wants to send Bob message HELLO (07 04 11 11 14) so Bob knows it is what Alice sent (no changes in transit, and authenticated)
  - 07^{53} mod 77 = 35
  - 04^{53} mod 77 = 09
  - 11^{53} mod 77 = 44
  - 11^{53} mod 77 = 44
  - 14^{53} mod 77 = 49
- Alice sends 35 09 44 44 49 received by Bob
- Bob uses Alice's public key, e = 17, n = 77, to decrypt message:
  - 35^{17} mod 77 = 07
  - 09^{17} mod 77 = 04
  - 44^{17} mod 77 = 11
  - 44^{17} mod 77 = 11
  - 49^{17} mod 77 = 14
- Bob translates message to letters to read HELLO
  - Alice sent it as only she knows her private key; if (enciphered) message's blocks (letters) altered in transit, would not decrypt properly

Example: Both

- Alice wants to send Bob message HELLO both enciphered and authenticated (integrity-checked)
  - Alice's keys: public (17, 77); private: 53
  - Bob's keys: public: (37, 77); private: 13
- Alice enciphers HELLO (07 04 11 11 14):
  - $(07^{53} \bmod 77)^{37} \bmod 77 = 07$
  - $(04^{53} \bmod 77)^{37} \bmod 77 = 37$
  - $(11^{53} \bmod 77)^{37} \bmod 77 = 44$
  - $(11^{53} \bmod 77)^{37} \bmod 77 = 44$
  - $(14^{53} \bmod 77)^{37} \bmod 77 = 14$
- Alice sends 07 37 44 44 14

Diffusion of RSA

- Currently used in wide variety of products and platforms
  - Often found in hardware of some secure telephones and smart cards (being replaced by ecc)
  - Incorporated into many widely used protocols for internet communications, such as S/MIME, SSL/TLS, S/WAN, …
  - Built into current operating systems developed by Microsoft, Apple, SUN, …
- Since $ø(n)=(p-1)(q-1)=pq - p - q+1=n - p - n/p +1$ then $pø(n)=np - p^2 -n + p$ hence $p^2 - (n - ø(n) +1) p + n = 0$ whose solutions are p and q. So computing ø(n) is no easier than factoring
- Messages M must be integers between 1 and n-1 and relatively prime with n. Since $Pr[GCD(M,n)=1]= ø(n)/n$, if p,q have 512 bits, M and n have factors in common with probability approx $1/2^{511}$

Practical aspects of RSA

- today's computers cannot directly handle numbers larger than 32- or 64-bits
- need multiple precision arithmetic that requires libraries to handle large numbers
- RSA Key Size
  - key size should be chosen conservatively
  - cryptographers can stay ahead of (factorization) cryptanalysts by increasing the key size
  - Until 1989, factorization attacks based on "high school mathematics." Then sophisticated attacks have extended factorization to larger numbers (usually of a specific form). At present it appears that 130 digit numbers can be factored in several months using lots of idle workstations.
- For small public key (still many e=3 !), if n large then M (most messages are small) could be <n and could just take e^{th} root of M^e

“Famous” RSA Cracking Attempts

- Breaking the RSA key requires factoring or brute force...
  - RSA inventors offered $100 reward for finding a plaintext sentence enciphered via RSA-129 (129 digit modulus is approx. 429 bit binary number)
  - RSA predicted 40 quadrillion years was needed
  - 1993: Lenstra (Bellcore) and Atkins (MIT) attempted the 1977 RSA factoring challenge
  - 1600+ workstations * eight months = success !
  - A particular private key was identified that matched the public key.
  - Reward for cracking code given to Free Software Foundation (Richard Stallman)
- Blacknet Key attack
  - Muffett, Leyland, Lenstra and Gillogly managed to use enough computation power (approx. 1300 MIPS) to factor the key in 3 months.
  - Used to decrypt a publicly-available message encrypted with that key.
  - Attack done in secrecy
- RSA-640 cracked announced Nov 2005
- RSA-200 (663 bits) factored in May 2005
- RSA-704 still not factored (2010) 30K$ (only 212 decimal digits!)
- RSA-768 cracked 12 Dec 2009

Challenge no longer active

RSA Versus DES

- fastest implementations of RSA can encrypt kilobits/second
- fastest implementations of DES can encrypt megabits/second
- this 1000-fold difference in speed is likely to remain independent of technology advances
- it is often proposed that RSA be used for secure exchange of session keys for symmetric block ciphers
- the key size of DES is 64 bits (56 bits plus 8 parity bits)
- key size of RSA is selected by the user
  - many implementations choose n to be 154 digits (512 bits) so the key (n,e) is 1024 bits

El Gamal published in 1985

- Based on the Diffie-Hellman secret-public key scheme
- its security depends on the difficulty of computing discrete logarithms
- it allows secure exchange of messages at the cost of 2 exponentiations (slow)
- Expansion 2:1 in size from plaintext to ciphertext
- Same message may (should!) have different encryptions if computed at different times using different randoms
- Implemented in GnuPG
- Similar scheme used in Digital Signature Algorithm (DSA) standardized in 1993

El Gamal Key Generation

- Select a large prime p (~200 digit) to be made public
- Select value g primitive element mod p (generator of Z_p^*) also public
- Bob selects a random secret number $Priv_B$ between 2 and p-2
- Bob computes $Pub_B = g^{PrivB} \bmod p$ which is made public

El Gamal

To send a message to Bob

- To **encrypt** a message **M** into ciphertext **C**
  - Selects a random number r, $0 < r < p$
  - Computes the message key $K = Pub_B^{\,r} \bmod p$
  - Compute the ciphertext pair: $C = (c1 , c2)$
    - $c_1 = g^r \bmod p$, $c_2 = K*M \bmod p$
- To **decrypt** the message C
  - Extract the key $\mathbf{K=}c_1^{PrivB} \bmod p = g^{r*PrivB} \bmod p$
  - Extracts **M** by solving for M in the equation

$$c_2 = K * M \bmod p$$

Difficult Problems

- Discrete Logarithm Problem (DLP): Given a prime modulus p, a basis g, and a value y, find the discrete logarithm of y, i.e. an integer x so that $y = g^x \bmod p$.
- n-th Root Problem: Given integers m, n and a, find an integer b so that $a = b^n \bmod \mathrm{m}$ - the solution b is the n-th root of a modulo n.
- Factorization: Find the prime factors of an integer n.

With suitable parameters, these problems are a basis for many (modern) cryptographic algorithms. However, not all instances of these problems are difficult to solve.

Difficult Problems ?

- In 1997 P.W.Shor published a paper in which he showed how quantum computers can be used to factor integers or computed discrete logarithms in poly time
- This result makes RSA, ECC and ElGamal easily breakable using quantum computers
- The NTRU is a public-key cryptosystem whose security is based on a different type of problem, the Shortest Vector Problem (SVP) in a lattice of high dimension, for which there is no poly time algo.
- Presented first in 1996, became an IEEE standard in 2009
- Besides its believed hardness in the quantum world, it is faster in key generation, encryption and decryption than RSA and ElGamal
- Low memory use makes it suitable for mobile devices and smart cards

NTRU (example) public $(N, p, q) = (7, 3, 41)$

encrypt

- Message represented as polynomial $M(x) = -x^5 + x^3 + x^2 - x + 1$
- Choose ephemeral $r(x) = x^6 - x^5 + x - 1$
- Encrypt $C(x) = r * h + M = 31x^6 + 19x^5 + 4x^4 + 2x^3\ 40x^2 + 3x + 25 \pmod{41}$

Decrypt

- Compute $a(x) = f(x) * C(x) \bmod 41 = x^6 + 10x^5 + 33x^4 + 40x^3 + 40x^2 + x + 40 \quad \bmod 41$
- Convert to centerlift $b(x) = x^6 + 10x^5 - 8x^4 - x^3 - x^2 + x - 1$
- Recover (almost) $M(x) = fp + b \bmod 3 = 2x^5 + x^3 + x^2 + 2x + 1$
- Original message M retrieved by centerlifting mod 3

Key Points

In Classical (symmetric) cryptosystems, enciphering and deciphering use different algorithms but the same key

- Or one key is easily derived from the other
- Difficult distribution of (shared) key

In Public key (asymmetric) cryptosystems enciphering and deciphering use different keys

- Computationally infeasible to derive one (private) from the other (public)
- "Easier" distribution of (public) key