

Computer Security

Public Key Encryption

Public Key Algorithms

- unfortunately need some number theory
- we'll try to keep it to a minimum
- if you want more detail, see references

Public Key Algorithms

- all hash algorithms do the same things
- all secret key algorithms do the same thing
- public key algorithms more varied, in what they do and how they do it

Examples

- RSA - encryption and digital signatures
- El Gamal & DSS - digital signatures
- Diffie-Hellman - establishing secret key
- zero knowledge proof systems - authentication

Pair

- all public key algorithms have a concept of a pair of related quantities – the key pair
- one secret (private), one public
- each pair is associated with a principal

5

Modular Arithmetic

- most public key algorithms based on modular arithmetic
- modular arithmetic uses non-negative integers
 - normal arithmetic $0,1,2,3,\dots,\infty$
 - modular arithmetic $0,1,2,3,\dots,n-1$
- $x \bmod n$ - integer remainder of x when divided by n

6

Additive Inverse

- additive inverse of x is number added to x to get 0
- eg., $4+6 = 0 \bmod 10$
- note works as a cipher

7

Modular Addition Example

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

8

Modular Multiplication

- for mod 10 only 1,3,7,9 work as cipher

9

Modular Multiplication Example

.	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

10

Multiplicative Inverse

- multiplicative inverse of x is what you'd multiply x by to get 1
- written as x^{-1}
- only 1,3,7,9 have multiplicative inverses mod 10
- they are relatively prime to 10

11

Finding Inverse

- multiplicative inverse mod n is hard to find for large n by brute force
- can use Euclid's Algorithm
- given x and n it finds y (if it exists) such that

$$x \cdot y = 1 \pmod{n}$$
- only numbers relatively prime to the modulus will have a multiplicative inverse

12

How Many?

- how many numbers will have a multiplicative inverse modulo n ?
- totient function - ϕ
- $\phi(n)$ is how many numbers have a multiplicative inverse modulo n
- if n prime $\phi(n) = n-1$
- if n product of two primes p and q ,
 $\phi(n) = (p-1)(q-1)$

13

Modular Exponentiation

- $x^y \bmod n$ does not equal $x^{(y+n)} \bmod n$
- again some numbers have an exponentiative inverse

14

Modular Exponentiation

- oddly $x^y \bmod n$ is the same as $x^{(y \bmod \phi(n))} \bmod n$
where n is prime or product of distinct primes
- for example, mod 10, $\phi(n) = 4$, so i th column is the same as $(i+4)$ th column
- finally, if $y = 1 \bmod \phi(n)$ then for any x
 $x^y = x \bmod n$

15

RSA

- named after “inventors” - Rivest, Shamir and Adleman
- public key algorithm, does encryption and decryption

16

RSA Characteristics

- key length is variable
- common key length is 512 bits or 1024
 - also find 2048
- block size is variable but must be less than key length
- ciphertext length will equal key length

17

RSA Algorithm

- The following is for 512 bit keys
- first generate public and corresponding private key
- choose two large primes p and q (around 256 bits each)
- call product of p and q , n
- p and q must remain secret

18

Generating Public Key

- choose e that is relatively prime to $\phi(n)$
- remember $\phi(n)$ is $(p-1)(q-1)$
- the public key is (e,n)

19

Generating Private Key

- find d where d is multiplicative inverse of $e \pmod{\phi(n)}$
- the private key is (d,n)

20

Encryption & Decryption

- message $m (<n)$
- encrypt using public key $c = m^e \bmod n$
- decrypt using private key $m = c^d \bmod n$

21

Digital Signature

- message $m (<n)$
- signature $s = m^d \bmod n$
- verify by $m = s^e \bmod n$

22

Issues

- why does it work?
- why is it secure?
- are the operations practical?
- how do we find p and q ?

23

Why Does It Work?

- arithmetic mod n , $n = pq$
- $\phi(n) = (p-1)(q-1)$
- $de = 1 \bmod \phi(n)$
- encryption is x^e
- decryption is $(x^e)^d = x^{(ed)} = x$
- similarly for signatures

24

Is It Secure?

- nobody has broken it
- depends upon the theory that factoring a big number is hard
- factoring techniques are slow
- currently factoring 512 bit number would takes 1000's of years

25

Factoring

- if you can factor n you can find p and q
- you can then find d as $de = 1 \pmod{\phi(n)}$
- we don't know that factoring is the only way to break RSA, but we don't know that any other ways do exist

26

Other Attacks

- if attacker knows all possible messages can encryt all possibilities
- then compare to ciphertext
- if in this position (probably requires there to be a small number of possible messages) should add a random number to message

27

Efficiency

- exponentiation
- finding primes
- finding an RSA key is more computationally expensive than using one

28

Exponentiation

- encryption, decryption, signing, verifying signature all involve
 - taking large number
 - raising it to large power
 - finding remainder mod n
- if done in straightforward manner expensive
- can do better (see text and references)

29

Generating RSA Keys

- need large primes
- not many large numbers are prime
- for example, for numbers of a hundred digits, about 1 in 230 are prime

30

Generating RSA Keys

- checking that a number that large is prime by trying all possible factors takes several universe lifetimes
- there is no practical way of determining that a number of this size is prime
- there are methods for testing whether a number is probably prime - see references

31

Finding d and e - Solution 1

- choose p and q
- select e at random
- test that e is relatively prime to $(p-1)(q-1)$
- if not, pick another e

32

Finding d and e - Solution 2

- first choose e
- carefully select p and q so that $(p-1)(q-1)$ is relatively prime to e

33

Other Points

- e can be fixed
- and at relatively small values at that - eg., 3 or 65537
- there are some problems with messages that are the product of relatively small primes (called smooth numbers)

34

PKCS - Public Key Cryptography Standard

- users can not be expected to cope with all possible attacks
- PKCS is a set of standards which recommends encodings

35

Threats dealt with by PKCS

- encrypting guessable messages
- signing smooth numbers
- multiple recipients of a message when $e=3$
- encrypting messages that are less than a third of the length of n when $e=3$
- signing messages where the information is in the high-order part and $e=3$

36

Diffie-Hellman

- predates RSA
- oldest public key system still in use
- less general than RSA
- does not do encryption or digital signatures
- offers better performance for what it does

37

What Does It Do?

- allows two parties to agree on a shared secret
- all messages exchanged in public
- at end only two parties know secret
- the two parties do not start with any shared secrets
- secret is usually a symmetric key

38

No Authentication

- Diffie-Hellman does not allow for authentication
- one party can not check who the other is
- you will always have a shared secret at end
- but have to trust who it is with

39

Overview

- choose a prime, p , of about 512 bits
- choose a g , which is smaller than p
- there are some restrictions on g , but not needed for overview
- p and g are publicly known

40

Overview

- each party chooses a 512 bit random number as their secret
- call these values S_a and S_b
- A computes $T_a = (g^{S_a}) \bmod p$
- B computes $T_b = (g^{S_b}) \bmod p$

41

Overview

- they exchange T_a and T_b
- now A computes $(T_b^{S_a}) \bmod p$ and B computes $(T_a^{S_b}) \bmod p$
- they will get same number as

$$T_b^{S_a} = (g^{S_b})^{S_a} = g^{(S_b \cdot S_a)} = g^{(S_a \cdot S_b)} = (g^{S_a})^{S_b} = (T_a^{S_b}) \bmod p$$

42

Discussion

- nobody can calculate $g^{(S_a \cdot S_b)}$ in reasonable amount of time
- even though they know g^{S_a} and g^{S_b} they can not calculate S_a and S_b

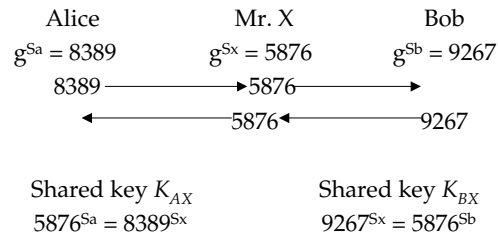
43

Bucket Brigade Attack

- if an attacker actively interposes himself between the two parties, can successfully examine all messages
- A talks to X, thinking it is B
- B talks to X, thinking it is A

44

Bucket Brigade Attack



45

Attacks

- bucket brigade attack depends upon X substituting for g^{S_A} and g^{S_B}
- Diffie-Hellman only secure against passive attacks
- needs additional mechanisms to be secure against active attacks

46

Published Numbers

- p and g can be publicly agreed
- same p and g always used
- g^{S_A} for each user published in secure location

47

Encryption with Diffie-Hellman

- what if A wants to send a message to B but B is not on-line?
- everyone first computes a public key
- public key consists of three numbers $\langle p, g, T \rangle$
- $T = (g^S \bmod p)$ where S is the person's secret (key)
- so B publishes $\langle p_b, g_b, T_b \rangle$

48

Encryption

- if A wants to send a message to B, picks S_a
 - computes $(gb^{S_a}) \bmod pb$
 - computes $K_{ab} = (Tb^{S_a}) \bmod pb$
 - sends encrypted message, with $(gb^{S_a}) \bmod pb$
 - recipient raises $(gb^{S_a}) \bmod pb$ to his own secret S_b , giving K_{ab}
 - recipient can then decrypt

49

Properties for p and g

- works with any prime p and any number g
- less secure if p and g don't have additional properties
- it is nice if $(p-1)/2$ is also prime
- such p's are called strong primes

50

Properties for p and g

- also nice if $g^x \neq 1 \bmod p$ unless $x = 0 \bmod p-1$
- if p is a strong prime this is satisfied by any $g \neq -1 \bmod p$ for which $g^{(p-1)/2} = -1 \bmod p$ (true for almost half of all mod p numbers)

51

El Gamal Signatures

- uses same sort of keys as Diffie-Hellman
- harder to understand than signing with RSA
- mathematics not easy to comprehend - we'll skip it

52

Digital Signature Standard (DSS)

- based on El Gamal
- with some optimisations
- some points which make it more convenient for signers than verifiers
- appears odd, as verification more common than signing
- done for devices with relatively low processing power, such as smart cards and mobile phones

53

DSS Algorithm

- generate p and q , which will be public
- q is a 160 bit prime
- p is a 512 bit prime
- $p = kq + 1$
- uses expensive inverse calculations

54

DSS Algorithm

- generate g , which will be public
- $g^q = 1 \pmod p$
- take any random number h
- raise it to $(p-1)/q$ to get g
- check g is not 1

55

DSS Algorithm

- choose a long term public/private key pair $\langle T, S \rangle$
- S is a random less than q
- $T = (g^S) \pmod p$

56

DSS Algorithm

- choose a per message public/private key pair $\langle T_m, S_m \rangle$
- random S_m
- $T_m = ((g^{S_m} \bmod p) \bmod q)$
- calculate $S_m^{-1} \bmod q$ (needed later)

57

DSS Algorithm

- calculate message digest dm
- DSS sister function SHS
- calculate signature $X = S_m^{-1} * (dm + S T_m) \bmod q$
- transmit message m
- per message public number T_m
- signature X

58

DSS Algorithm

- verify signature
- calculate mod q inverse of signature X^{-1}
- calculate dm
- calculate $x = dm * X^{-1} \bmod q$
- calculate $y = T_m * X^{-1} \bmod q$
- calculate $z = (g^x * T^y \bmod p) \bmod q$
- if $z = T_m$ then signature is verified

59

Per-Message Secret Key

- both DSS and El-Gamal require signer to generate a unique secret number for each message
- if the same secret number is used for two messages could expose the signer's private key

60

Zero-Knowledge Proof Systems

- only do authentication
- allow you to prove you know a secret without revealing the secret
- RSA is a zero knowledge proof system
- there are zero-knowledge proof systems with much better performance than RSA
- they cannot do signatures or encryption

61

Example

- based on graphs (vertices and edges)
- named vertices, edges pairs of vertices
- two graphs are isomorphic if we can rename the vertices of one to get a graph identical to the other
- nobody knows how to
 - efficiently determine whether two arbitrary graphs are isomorphic
 - Or what the mapping is for two isomorphic graphs

62

Graph Example

- Alice specifies a large graph (~500 vertices)
- renames vertices to produce an isomorphic graph
- call these Graph A and Graph B

63

Graph Example

- Alice knows mapping between graphs, nobody else does
- public key is the two graphs
- her private key is mapping between two graphs

64

Graph Example

- to prove to Bob that she is Alice
- rename vertices to find a new set of graphs, G_1, G_2, \dots, G_k
- Bob asks, for each i , for Alice to show him the mapping between G_i and one of graph A or graph B
- one only, as otherwise he could then recreate base mapping

65

Graph Example

- after proof, Bob knows some mappings to A and some to B
- he could have generated these himself
- not very efficient

66

Different Example, using Squares

- Alice establishes a public key consisting of $\langle n, v \rangle$
- n is product of two large primes
- v is a number for which only Alice knows the square root mod n
- randomly choose s and square it mod n to obtain v
- s is secret key

67

Squares Example

- to prove to Bob that she is Alice
- choose k random numbers r_1, r_2, \dots, r_k
- for each r_i she sends $r_i^2 \bmod n$ to Bob

68

Squares Example

- Bob chooses a random subset of the r_i^2
- tells Alice which subset he has selected
- this is subset 1
- others are subset 2

69

Squares Example

- Alice sends $s \cdot r_i \bmod n$ for each r_i^2 of subset 1 and $r_i \bmod n$ for each r_i^2 of subset 2
- Bob squares Alice's replies mod n
- for those r_i^2 in subset 1 he checks that the square of the reply is $v \cdot r_i^2 \bmod n$
- for those r_i^2 in subset 2 he checks that the square of the reply is $r_i^2 \bmod n$

70

Why is it Secure?

- suppose Fred wants to impersonate Alice
- anyone can compute squares mod n
- Fred cannot take square roots mod n
- but can start with random r and square it
- Fred can give correct answers for subset 2 but not subset 1, as he does not know s

71

Subset 2

- why do you need subset 2?
- Bob could use the numbers supplied to him by Alice to prove to someone else that he is Alice

72

Subset 2

- need enough r_i 's (say 30)
- remember other party chooses subsets
- so if Bob tries to reuse results from Alice to prove to Carol that he is Alice
- the chance of Alice picking same set one as Bob is very low ($\sim 2^{-15}$)

73

Zero Knowledge Signatures

- any zero knowledge system can be used for signatures
- performance in terms of Bandwidth and CPU poor

74

Zero Knowledge Signatures

- assume a typical zero-knowledge system
- Alice has a secret
- she can transmit something and answer any question Bob might pose
- imposter should only be able to answer one question

75

Zero Knowledge Signatures

- for example, with graph isomorphism
- Alice's secret is mapping between graph 1 and graph 2
- Alice transmits new graph g_i
- Bob's challenges are binary values
- Alice can answer 0 (mapping from G_i to G_1) and 1 (mapping from G_i to G_2)
- imposter can only answer one

76

Zero Knowledge Signatures

- in other zero knowledge proof schemes Bob's challenge would be a larger number (eg., 16 bits)
- Alice can answer any challenge
- imposter still only one (or a few)

77

Zero Knowledge Signatures

- signature schemes are not interactive
- checker does not supply challenges
- use message digest function as Bob substitute

78

Zero Knowledge Signatures

- need sufficient challenges so attacker can not compute signature off-line
- so need large number of challenges
- enough to give only 1 in 2^{64} or so chance of guessing

79