Computer Security

Modern Cryptography:
Public Key Cryptosystems

What Problems are We Solving?

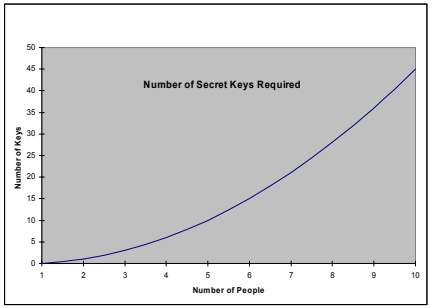
- Public key cryptography came about as a solution to two problems:
  - ***Key Distribution***
    - With private key cryptosystems, a major problem is how keys are distributed to a set of message senders and recipients.
  - ***Digital Signatures***
    - Could a method be found that would guarantee, to the satisfaction of all parties, that a digital message had been sent by a particular person?

Secret Key Distribution

- Assume that we have *n* people, each of whom needs to communicate with all the others privately.
- We therefore need a separate secret key for each pair of people.
- How many keys do we need to distribute, in total?

$$\frac{n(n-1)}{2}$$

- How does this value change, as we increase *n*?



It increases very rapidly!

Diffie/Hellman

- In 1976, Whitfield Diffie and Martin Hellman, then at Stanford University, came up with a method that addressed both problems.
  - Their method was radically different from all previous methods used in cryptography -- it is a *Public Key Cryptosytem*
    - It uses two different keys:
      - One key for encryption, and
      - A different key for decryption

Public Key Cryptosystems

- Public Key cryptography relies on the following:
  - It is computationally infeasible to determine the decryption key, given only knowledge of the cryptographic algorithm and the encryption key (and vice versa)
- In some public key algorithms (including RSA):
  - Either of the two keys can be used for encryption, with the other being used for decryption.

Public Key Cryptosystems

- The components of a public key encryption scheme are:
  - **Plaintext**, *P*
  - **An Encryption Algorithm,** *E*
  - **A Public Key,** *KU* **and a Private key,** *KR*
    - A pair of keys which have been selected so that if one is used for encryption, the other is used for decryption
  - **Ciphertext**, *C*
  - **A Decryption Algorithm,** *D*

Public Key Cryptosystems

- A public key cryptosystem can be used to provide ***confidentiality***:
  - Each user generates a pair of keys
  - Each user makes one of the two keys public; the matching key is kept private
  - If Bob wishes to send a secret message to Alice, he encrypts the message using Alice's public key

$$C = E_{KU_{Alice}}(P)$$

  - When Alice receives the message, she decrypts it using her private key. Only Alice knows her private key.

$$P = D_{KR_{Alice}}(C)$$

Public Key Cryptosystems

- However, a public key cryptosystem can also be used to provide ***authentication***:
  - Bob encrypts a message with his private key (which only he knows), and sends it to Alice:

$$C = E_{KR_{Bob}}(P)$$

  - Alice decrypts the received message using Bob's public key:

$$C = E_{KU_{Bob}}(P)$$

Digital Signatures

- In this case, Bob is said to have ***digitally signed*** the message; the message constitutes a ***digital signature***.
  - Because it is impossible to alter the message without access to Bob's private key, the message is authenticated in terms of:
    - Its ***source*** (it really came from Bob)
    - Its ***data integrity*** (it is the same message that Bob sent)

Digital Signatures

- Normally, a digital signature is not an encryption of the entire message. This approach is not terribly efficient.
- A more efficient method is to encrypt a small block of bits, called an ***authenticator***, that is a function of the document.
  - It must be infeasible to change the document without changing the authenticator
  - If the authenticator is encrypted with Bob's (the sender's) private key, it functions as a signature that verifies:
    - Origin
    - Content
    - Sequencing
- The message (in plaintext) is accompanied by its authenticator.

Digital Signatures

- Note that digital signatures provide authentication, but ***not*** confidentiality.
  - Since the public key is known to all, anyone can decrypt the message.

Digital Signatures

- However, it is possible to provide both, using a double use of the public key scheme:

$$C = E_{KU_{Alice}}(E_{KR_{Bob}}(P))$$

$$P = D_{KU_{Bob}}(D_{KR_{Alice}}(C))$$

  - The first encryption provides the digital signature.
  - The second provides the confidentiality.

Public Key Cryptography Requirements

- Diffie and Hellman laid out the following requirements:
  - It is computationally easy for a party (Bob) to generate a key pair KU_{Bob} and KR_{Bob}
  - It is computationally easy for a sender (Alice), knowing the public key, KU_{Bob}, and the message to be encrypted, P, to generate the corresponding ciphertext
  - It is computationally easy for the receiver (Bob) to decrypt the resulting ciphertext using the private key, KR_{Bob}, to recover the original message
  - It is computationally infeasible for an attacker, knowing the public key, KU_{Bob}, to determine the private key, KR_{Bob}.
- One useful, but not strictly necessary, additional requirement:
  - The encryption and decryption functions can be applied in either order:

$$P = E_{KU_{Bob}}(D_{KR_{Bob}}(P)) = D_{KU_{Bob}}(E_{KR_{Bob}}(P))$$

RSA

- Diffie and Hellman demonstrated the possibility of public key cryptography, but did not immediately come up with a practical implementation.
- However, they did challenge their colleagues to come up with one.
- The challenge was answered in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman, at MIT.
  - They came up with the RSA algorithm
  - Since its inception, RSA has dominated as the most widely accepted and implemented public key algorithm.

The Mathematics of RSA

- RSA achieves its security from the difficulty of factoring large numbers.
  - The public and private keys are functions of a pair of large (on the order of *hundreds or even thousands* of binary digits) prime numbers.
  - Based on a ***One-Way Trapdoor Function***:
    - Given the publicly known values n and e, it is easy to compute

      $m^e \bmod n$

      from m, but not the other way around.
    - If you know the factorization of n, then it is easy to do the inverse calculation; the factorization of n is the trapdoor information.

The Mathematics of RSA

- To generate the two keys:
  - Choose two random large prime numbers, p and q. (For maximum security, make them of equal length).
  - Compute the product of the two: $n = pq$
  - Then, randomly choose the encryption key, e, such that e and $\phi(n)$ ($= (p - 1)(q - 1)$) * are relatively prime.
  - Use the extended Euclidean algorithm to compute the decryption key, d, the multiplicative inverse of $e \bmod \phi(n)$, such that:

$$ed = 1 \bmod (p-1)(q-1)$$

$$\text{or}: d = e^{-1} \bmod (p-1)(q-1)$$

(Note that d and n are also relatively prime.)

Remember Euler's totient function?

The Mathematics of RSA

- The pair of numbers (e, n) is the public key
- The number d is the private key
- The prime numbers p and q are no longer needed, and can be discarded (but never divulged!)

The Mathematics of RSA

- To encrypt a block of plaintext, P, the encryption formula is: $C = P^e \bmod n$
- Decryption is: $P = C^d \bmod n$

since:

$$C^d = \left(P^e\right)^d = P^{ed} = P \qquad (\text{all mod } n)$$

Why does $P^{ed} = P$?

The Mathematics of RSA

Euler's Theorem says that, for every a and n that are relatively prime:

$$a^{\phi(n)} \equiv 1 \bmod n$$

so: $m^{\phi(q)} \equiv 1 \bmod q$

and: $\left[m^{\phi(q)}\right]^{\phi(p)} = m^{\phi(n)} = 1 \bmod q$

If m is a multiple of p, then $m = cp$, and, multiplying both sides by $m = cp$:

$$m^{\phi(n)+1} = m + kcpq = m + kcn = m \bmod n$$

Similarly, for p. An alternative form of this is:

$$\begin{aligned} m^{k\phi(n)+1} &\equiv m^{k(p-1)(q-1)+1} \\ &= \left[\left(m^{\phi(n)}\right)^k \times m\right] \bmod n \\ &= \left[(1)^k \times m\right] \bmod n, \quad \text{by Euler's theorem} \\ &= m \bmod n \end{aligned}$$

The Mathematics of RSA

- So, to satisfy all the above math requirements:
  - We choose p, q, two prime numbers
  - We calculate $n = pq$
  - We choose e, with $\gcd(\phi(n), e) = 1; \quad 1 < e < \phi(n)$
  - We calculate $d \equiv e^{-1} \bmod \phi(n)$

and so:

$$ed = k\phi(n) + 1$$

or equivalently:

$$ed \equiv 1 \bmod \phi(n)$$

$$d \equiv e^{-1} \bmod \phi(n)$$

Exponentiating with Big Numbers

- Remember how we talked about exponentiation mod *n*?
  - The numbers we're talking about (*n*, *p*, *q*, *e*, *d*) are thousands of binary digits long, so if we tried to raise something which is thousands of binary digits to the power of something that is also thousands of binary digits long:

    "Raising a 150-digit number to a 150-digit power by this method [conventional computer arithmetic] would exhaust the capacity of all existing computers for more than the expected lifetime of the universe, and this would not be cost-effective." (Textbook, p. 154)
  - However, if you use modular reduction after each multiplication, exponentiation mod *n* remains within reasonable bounds.

Finding Large Prime Numbers

- We learned earlier now to come up with a set of small prime numbers (Sieve of Eratosthenes), but this is not a practical algorithm for very large primes.
- It turns out that there is no known practical method for absolutely determining that a number of this size is prime.
- However, there is a test for whether a number is *probably prime*; the more time we spend testing a number the more sure we are that the number is prime.

Finding Large Prime Numbers

- We can use ***Euler's Theorem:***
  - For any *a* relatively prime to *n*,

    $a^{\phi(n)} = 1 \bmod n$
  - In the case where *n* is prime, $\phi(n) = n - 1$, and the theorem takes on a different form and name: ***Fermat's Theorem:***

    If p is prime and $0 < a < p$,

    $a^{p-1} = 1 \bmod p$

Finding Large Prime Numbers

- Does $a^{n-1} = 1 \bmod n$ hold even when *n* is not prime?
- The fact that it *usually does not* can be used to provide a ***primality test***:
  - Pick a number $a < n$
  - Compute $a^{n-1} \bmod n$, and see if the result is 1
  - If it is not 1, *n* is clearly not prime.
  - If it is 1, *n* may or may not be prime, with a certain probability
    - If *n* is a randomly generated number of about a hundred digits, the probability that *n* is not prime, but $a^{n-1} \bmod n = 1$, is about 1 in 10^{13}
    - We can improve on this by trying multiple values of *a*

Finding Large Prime Numbers

- The method of choice for testing whether a number is prime is the ***Rabin-Miller test***:
  - Choose a random number, p, to test.
  - Calculate b, where b is the number of times 2 divides $p - 1$
    - That is, 2^b is the largest power of 2 that divides $p - 1$
  - Then, calculate m, such that $p = 1 + 2^b \bullet m$:
    1. Choose a random number, a, such that $a < p$
    2. Set $j = 0$ and $z = a^m \bmod p$
    3. If $z = 1$, or if $z = p - 1$, then p passes the test and may be prime
    4. If $j > 0$ and $z = 1$, then p is not prime
    5. Set $j = j + 1$. If $j < b$ and $z \neq p - 1$, then set $z = z^2 \bmod p$, and go back to step 4. If $z = p - 1$, then p passes the test and may be prime.
    6. If $j = b$ and $z \neq p - 1$, then p is not prime.

Finding Large Prime Numbers

- Here's an algorithm for generating a large prime number given by Schneier*:
  1. Generate a random n-bit number p
  2. Set the high- and low-order bits to 1
     - Ensures that the prime is odd, and of the required length
  3. Check that p is not divisible by small primes (say, less than 2000)
  4. Perform the Rabin-Miller test for some random a.
     - If a passes, generate another random a, and repeat
     - Do five such tests
  - If p fails one of the tests, generate another p and try again.

**Advanced Cryptography, Second Edition*, by Bruce Schneier, Wiley

Key Management

- With public keys, we still have to worry about how the keys are distributed.
- There are a number of approaches:
  - Public announcement
    - Subject to forgery
  - Publicly available directory of public keys
    - Responsibility of some trusted entity or organization (Key Distribution Center)
    - Subject to tampering
  - Public Key Authority
    - Directory with public/private key for PKA
    - Could be a bottleneck
  - Public Key Certificates
    - Removes need to always go through a PKA to get a key
    - The PKA becomes a Certification Authority (CA)
    - CA issues certificate which contains public key and other information for a person or organization.

Diffie-Hellman Key Exchange

- Diffie and Hellman published the first public key algorithm
  - Referred to as ***Diffie-Hellman Key Exchange***
  - Used in a number of commercial products.
  - Oldest public key system still in use
  - Less general than RSA
    - It does neither encryption nor signatures
- Diffie-Hellman allows two individuals to agree on a shared private key, by exchanging public messages.

Diffie-Hellman Key Exchange

- Diffie-Hellman key exchange depends for its effectiveness on the difficulty of computing ***discrete logarithms:***
  - We define a ***generator*** or ***primitive*** or ***primitive root*** of a prime number p as one whose powers generate all the integers from 1 to p - 1. So, if a is a primitive root of p, then the numbers:

    $a \bmod p$, $a^2 \bmod p$, $a^3 \bmod p$, ... $a^{p-1} \bmod p$

    are distinct and consist of the integers from 1 through p - 1 (in some permutation)
  - For any integer b and a generator a of p, we can find a unique exponent i such that:

    $$b \equiv a^i \bmod p \quad \text{where } 0 \le i \le (p-1)$$
  - The exponent i is referred to as the ***discrete logarithm*** (or ***index***) of b for the base a, mod p. It is the inverse of modular exponentiation, and finding the discrete logarithm i, given b, is known to be a *hard problem*.

Diffie-Hellman Key Exchange

- Here is what Diffie-Hellman key exchange involves:
  - First, Alice and Bob agree on a large prime, p, and g, such that g is a generator mod p. The numbers don't have to be secret.
  - Then the protocol goes as follows:

    1) Alice chooses a random large integer x and sends Bob

    $X = g^x \bmod n$

    2) Bob chooses a random large integer y and sends Alice

    $Y = g^y \bmod n$

    3) Alice computes

    $k = Y^x \bmod n$

    4) Bob computes

    $k' = X^y \bmod n$

$$k = Y^x \bmod n = g^{xy} \bmod n$$
$$= X^y \bmod n = k'$$

Diffie-Hellman Key Exchange

- This protocol can also be extended to work with three or more people:

1) Alice chooses a random large integer x and sends Bob

$X = g^x \bmod n$

2) Bob chooses a random large integer y and sends Carol

$Y = g^y \bmod n$

3) Carol chooses a random large integer z and sends Alice

$Z = g^z \bmod n$

4) Alice sends Bob

$Z' = Z^x \bmod n$

5) Bob sends Carol

$X' = X^y \bmod n$

6) Carol sends Alice

$Y' = Y^z \bmod n$

7) Alice computes

$k = Y'^x \bmod n$

8) Bob computes

$k = Z'^y \bmod n$

9) Carol computes

$k = X'^z \bmod n$

Man-in-the-Middle Attack

- One problem with Diffie-Hellman is that there is no authentication, and so the protocol is subject to a ***man-in-the-middle attack***:
  - Alice generates g^x and sends it to "Bob"
  - Eve intercepts the message:
    - Generates g^v, and sends it to Bob in place of Alice's message
  - Bob receives g^v, generates g^y, and sends it to "Alice"
  - Eve intercepts the message:
    - Generates g^w, and sends it to Alice in place of Bob's message
  - Alice computes $k = (g^w)^x$
  - Bob computes $k' = (g^v)^y$
  - Eve computes $k = (g^x)^w$ and $k' = (g^y)^v$

Man-in-the-Middle Attack

- There are a number of techniques to defend against such an attack:
  - Each person can have a "somewhat permanent" public and secret number, instead of creating one for each message exchange. This can be considered to be a kind of *Digital Phonebook*.
  - If Alice and Bob share some kind of secret which then can use to authenticate each other, then they can use this secret to verify each other's messages indeed came from the person they expected.

Encryption with Diffie-Hellman

- Another disadvantage with Diffie-Hellman is that, in order for Alice and Bob to communicate, they first have to have an active exchange, where they both have to be present.
  - We can solve this as follows:
    - Alice computes a personal public key, consisting of (p_A, g_A, T_A), where:

      $T_A = g_A^{S_A} \bmod p$

      for her private key S_A.
    - Bob does likewise, to produce (p_B, g_B, T_B)
    - These public keys are displayed in a reliable public place.
    - If Alice wants to send Bob an encrypted message, she picks a random number S_A, computes $g_B^{S_A} \bmod p_B$, and then computes:

      $K_{AB} = T_B^{S_A} \bmod p_B$
    - She uses K_{AB} to encrypt the message using any secret key cipher
    - Bob eventually computes the same expressions with A and B reversed, to calculate the value of K_{AB}, which allows him to decrypt the message

Diffie-Hellman and Safe Primes

- Diffie-Hellman works with any prime p and any number g
- However, it it less secure if p and g don't have additional mathematical properties
  - It turns out that things work better if $(p - 1)/2$ is also a prime.
    - Such a prime is called a ***Safe Prime***
  - It's also better if :

    $g \neq -1 \bmod p$, for which $g^{(p-1)/2} = -1 \bmod p$

    (true for almost half of all mod p numbers)

Digital Signature Standard (DSS)

- NIST has proposed an algorithm for digital signatures:
  - Based on ElGamal, an alternative signature scheme, which is harder to understand than RSA.
- NIST published DSS in 1991, as a proposed standard for digital signatures.
  - Generated lots of debate; still continues
  - Why ElGamal, and not RSA?
  - Mandated 512-bit (p) /160-bit (q) moduli ?
  - If an attacker breaks DSS for (p, q, g), breaks all keys for that triple. With RSA, the attacker would only have broken a single key.
  - Trapdoor primes require you have to trust the source
  - Performance vs RSA
  - Secret numbers required (problematic)
  - Patent issues

Elliptic Curve Cryptography (ECC)

- There are known subexponential (but superpolynomial) algorithms for breaking RSA and Diffie-Hellman
- Mathematicians do not (yet) have subexponential algorithms for breaking ciphers based on the mathematics of ***Elliptic Curves***.
  - As a result, it is believed that Elliptic Curve Cryptography (ECC) is more secure for a given key size than other forms of cryptography.
  - For some cryptographic schemes, it is possible to replace modular multiplication with elliptic curve multiplication directly:
    - ECC Diffie-Hellman
    - ECC ElGamal
    - etc.

Elliptic Curve Cryptography (ECC)

- An Elliptic Curve is a set of points in a coordinate plane which satisfy an equation of the form:

$$y^2 + axy + by = x^3 + cx^2 + dx + e$$

- We need some mathematical operation on 2 points in the set which will always produce a point also in the set.
  - Called "multiplication"
  - Must be associative (to allow repeated squaring for exponentiation)
    $(g^x)^y = g^{xy} = (g^y)^x$
  - Determining discrete logs must also be hard to do:
    - Knowing g and g^x, it is disproportionately difficult to compute x.

Elliptic Curve Cryptography (ECC)

- In ECC, there are at least two types of arithmetic that satisfy these requirements:
  - $\mathbf{Z}_p$ arithmetic (modular arithmetic with a large prime p as the modulus)
  - GF(2^n) arithmetic, which can be done with shifts and XORs. (Modular arithmetic of polynomials with coefficients mod 2)
- ECC can be faster, because it is computationally no more difficult, and because it can use shorter keys to accomplish the same level of security.

Summary

- We've covered a number of topics:
  - Key Distribution
  - RSA
  - Diffie-Hellman
  - Authentication
  - Digital Signatures and Certificates
  - Elliptic Curve Cryptography