Security Issues in Cloud Computing

Modern Cryptography II – Asymmetric Cryptography

Peter Schwabe

October 21 and 28, 2011

So far we assumed that Alice and Bob both have some key, which nobody else has. How do they get this key? This was an open problem was until 1976, when Diffie and Hellman published the paper "New direction in Cryptography".

Diffie-Hellman key-exchange protocol

General description	Small example
Public: Fix a large prime number p	Fix $p = 17$
Public: Fix a number $g \in \{2, \dots, p-1\}$ called generator	Fix $g = 2$
Alice chooses $a \in \{1, \dots, p-1\}$ at random, computes $g^a \mod p$, sends this to Bob	Alices chooses $a = 7$, computes $2^7 \mod 17 = 9$, sends this to Bob
Bob chooses $b \in \{1, \dots, p-1\}$ at random, computes $g^b \mod p$, sends this to Alice	Bob chooses $b = 11$, computes $2^{11} \mod 17 = 8$, sends this to Alice
Alice computes $K = (g^b)^a \mod p$	Alice computes $K = 8^7 \mod 17 = 15$
Bob computes $K = (g^a)^b \mod p$	Bob computes $K = 9^{11} \mod 17 = 15$

A passive attacker sees $g^a \mod p$ and $g^b \mod p$. Finding the key from this information is related to two important problems:

- Computing $g^{ab} \mod p$ given $g^a \mod p$ and $g^b \mod p$ is called the *Computational Diffie-Hellman Problem (CDHP)*,
- computing a given $g^a \mod p$ is called the *Discrete-Logarithm Problem (DLP)*.

For large values of p both problems are considered hard.

Man-in-the-middle attack against Diffie-Hellman key exchange

A passive (listening) attacker needs to solve the CDHP to obtain the key shared by Alice and Bob. How about an active attacker? Assume that all messages from Alice to Bob and vice versa are routed through Oscar's computer. Oscar can read *and modify* all messages.

Alice	**Oscar**	**Bob**
Sends $g^a \mod p$		
	receives $g^a \mod p$, sends $g^o \mod p$ to Bob	
		Sends $g^b \mod p$
	receives $g^b \mod p$, sends $g^o \mod p$ to Alice	

Alice knows a, receives $g^o \mod p$, computes $k_1 = (g^o)^a \mod p$, Bob knows b, receives $g^o \mod p$, computes $k_2 = (g^o)^b$ Oscar computes k_1 and k_2 and communicates with Bob (saying "I am Alice") using k_2 and with Alice (saying "I am Bob") using k_1.

Computational issues

The main computations in Diffie-Hellman are exponentiations, computing $g^a \mod p$ from g and a (fixed base) and $(g^b)^a$ from g^b and a (variable base). If we could compute this by multiplying a times (or b times) we could solve the DLP (simply compare after each multiplication)! How do we compute the exponentiations efficiently?
Example: Consider

$$\begin{aligned} a &= 37146 \\ &= 3 \cdot 10000 + 7 \cdot 1000 + 1 \cdot 100 + 4 \cdot 10 + 6 \\ &= (((((((3 \cdot 10) + 7) \cdot 10) + 1) \cdot 10) + 4) \cdot 10) + 6 \text{ (“Horner scheme”)} \end{aligned}$$

Now compute g^a as
$$ga = (((g^3)^{10} \cdot g^7)^{10} \cdot g^1)^{10} \cdot g^4)^{10} \cdot g^6.$$

Transformation to the binary representation of a: Instead of writing $a = a_0 + 10 \cdot a_1 + 100 \cdot a_2 + \cdots + 10^n a_n$ with $a_i \in \{0, \ldots, 9\}, i = 0, \ldots, n$ write a in binary representation $a = a_0 + 2 \cdot a_1 + 4 \cdot a_2 + \cdots + 2^n a_n$ with $a_i \in \{0, 1\}, i = 0, \ldots, n$. Compute exponentation with the *square-and-multiply algorithm*:

Algorithm 1 Square-and-multiply algorithm

```
**Input:** $a = (a_n, \dots, a_0), p, g$
**Output:** $g^a \mod p$
  $r \leftarrow 1$
  for $i$ from $n$ downto 0 do
      $r \leftarrow r^2 \mod p$
      if $(a_i = 1)$ then
          $r \leftarrow r \cdot a \mod p$
      end if
  end for
  return $r$
```

Example: Compute $2^{12} \mod 17$. $12 = (1100)_2 = 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 0 \cdot 1$

$$
\begin{aligned}
& r \leftarrow 1 \\
& i = 3 : r \leftarrow r^2 \mod p (r = 1) \\
& \qquad r \leftarrow r \cdot g \mod p (r = 2) \\
& i = 2 : r \leftarrow r^2 \mod p (r = 4) \\
& \qquad r \leftarrow r \cdot g \mod p (r = 8) \\
& i = 1 : r \leftarrow r^2 \mod p (r = 64 \mod 17 = 13) \\
& i = 0 : r \leftarrow r^2 \mod p (r = 169 \mod 17 = 16) \\
& \text{return } r (r = 16)
\end{aligned}
$$

Some remarks on Diffie-Hellman

- State-of-the-art Diffie-Hellman key exchange does not use this arithmetic modulo p but arithmetic on elliptic curves. The reason is that DLP and CDHP are harder to solve, at the same security level we can use the shorter keys (e.g. 256 bits instead of 3248 bits).
- There are more efficient algorithms than square-and-multiply, they all share the idea of "scanning the exponent".
- Exponentiation g^a can typically be made much faster than the exponentiation $(g^b)^a$, because g is fixed (and g^b is not).
- Dont use the square-and-multiply algorithm with a secret exponent in any real-world application!

Public-key cryptography

The paper by Diffie and Hellman contained more than the key-exchange protocol, namely the idea of public-key cryptography:

- Use different keys to encrypt and to decrypt,
- Make encryption key public, keep the decryption key (private key) secret,
- Everybody can encrypt messages to ,e.g, Alice, but only Alice (with her private key) can decrypt,
- Obvious: It must be impossible to decrypt with the public key (in particular it must be impossible to compute the private key from the public key).

The paper only described this idea, it did not propose a way yo implement such a scheme.

The RSA cryptosystem

In 1978, Rivest, Shamir and Adleman proposed a concrete public-key cryptosystem called RSA cryptosystem.

Mathematical preliminaries:

What we used so far: addition, subtraction, multiplication modulo n. What about division or inversion?
For each integer a with $gcd(a, n) = 1$ there exists an integer b such that $a \cdot b \mod n = 1$. This integer b can be efficiently computed using the Extended Euclidean Algorithm. We call this integer b the "inverse of a modulo n".
Example: Let $n = 7, a = 2$, then $b = 4$ or let $n = 7, a = 3$, then $b = 5$.

How many integers smaller than n have an inverse modulo n? This is what the Eulerian φ-function tells us:
$\varphi(n)$ = number of integers smaller than n that have an inverse modulo n.

Example: For a prime p: $\varphi(p) = p - 1$.
For a product of two distict prime numbers p and q: $\varphi(pq) = (p - 1)(q - 1)$.

RSA key generation

- Pick two random large different prime numbers p and q,
- compute $n = p \cdot q$,
- choose $e \in \{3, \ldots, \varphi(n) - 1\}$ such that e is invertible modulo $\varphi(n)$,
- compute d, the inverse of e modulo $\varphi(n)$.

- Public key: (e, n), private key: d.
- Additional secret information: $p, q, \varphi(n)$, can be deleted.

RSA encryption

Consider a message M, which is an integer smaller than n. Compute ciphertext:

$$C = M^e \mod n$$

RSA decryption

Recover the message M from C by computing

$$M = C^d \mod n.$$

Why does RSA work? No proof here, see for example Handbook of Applied Cryptography, Section 8.3.

Security of RSA

- We obviously need to consider chosen-plaintext attacks because everybody can encrypt. General question: Given many pairs (M_i, C_i), $i = 1, \ldots, k$, and given e, n fulfilling $C_i = M_i^e \mod n$, is it possible to find d, such that $M_i = C_i^d \mod n$, or is it possible to compute M_t, given only $C_t = M_t^e \mod n$?
- If an attacker can factor n, i.e., compute p and q, he can easily compute $\varphi(n) = (p-1)(q-1)$ and then compute the inverse of e modulo $\varphi(n)$, which is the private key d.
- It would be sufficient for an attacker to compute $\varphi(n)$ without computing p and q. This is not easier than factoring n. Assume we know $n = p \cdot q$ and $\varphi(n) = (p-1)(q-1)$. Then we can factor n as follows:

$$\begin{aligned} \varphi(n) &= (p-1)(q-1) = pq - q - p + 1 \Leftrightarrow p + q = n - \varphi(n) + 1, && (1) \\ -4n &= -4pq = (p-q)^2 - (p+q)^2 \Leftrightarrow (p-q)^2 = (p+q)^2 - 4n, && (2) \\ q &= ((p+q) - (p-q)). && (3) \end{aligned}$$

  Obtain $p + q$ from equation (1), obtain $p - q$ from equation (2) and use both in equation (3) to obtain q (and p).
- An attacker could also consider computing d without knowing $\varphi(n)$. This has been shown to be as hard as factoring n by Miller in 1975.

Summary: The currently best known way to break RSA is factoring n. It has been shown that computing the private key is as hard as factoring, but it is not proven that computing M from $M^e \mod n$ is as hard as factoring.

How hard is it to factor large numbers?

- Many numbers (those with small prime factors) can be factored relatively efficiently.
- RSA numbers (products of two large primes) are among the hardest to factor.
- RSA factoring records (factoring challenge numbers put online by RSA Security):

Name	Digits	Bits	Year	Group
RSA-100	100	330	April 1991	Lenstra et al.
RSA-130	130	430	April 1996	Lenstra et al.
RSA-576	174	576	Dec. 2003	Franke et al.
RSA-640	193	640	Nov. 2005	Franke et al.

- Most recently: In December 2009, Kleinjung, Aoki, Franke, Lenstra, Thomé, Bos, Gaudry, Kruppa, Montgomery, Osvik, te Riele, Timofeev, and Zimmermann announced to have factored RSA-768 (768 bits, 232 decimal digits). The computation consisted of 3 steps (using the so-called number-field sieve)
  - step 1: 1/2 year on 80 processors,
  - step 2: almost 2 years on many hundreds of machines (estimate: 1500 years on one 22 GHz Opteron),
  - step 3: 119 days on four clusters.

Implementation issues of RSA

- Finding large prime numbers in the key generation can be done by choosing random numbers and testing whether they are prime. This will find prime numbers efficiently because there are "many" primes and because testing for primality is efficient (e.g., using the Miller-Rabin primality test).
- Exponentiation can be done as for Diffie-Hellman.

General remarks about RSA

- Very common keysize: 1024 bits. Factoring such a number is estimated to be 1000 times harder than factoring RSA-768.
- RSA keysizes roughly corresponding to symmetric-key sizes (e.g. for an ideal block cipher), according to the recommendations by ECRYPT II from 2011:
  - 816-bit RSA – 64-bit symmetric
  - 1008-bit RSA – 72-bit symmetric
  - 1248-bit RSA – 80-bit symmetric
  - ...
  - 3248-bit RSA – 128-bit symmetric

    - 15424-bit RSA – 256-bit symmetric

- In principle, p and q are not required for decryption. Storing them alongside d allows us to speed up decryption.
- Other optimization: Use small public exponent e (e.g., $e = 3$). Have to be very careful, this can lead to weaknesses.
- Generally: Dont use plain RSA in practice, use various tweaks (for example: use a padding scheme to encrypt the same message to different ciphertexts)

General remarks about public-key cryptography

- While RSA is based on the hardness of factoring large integers, we can also construct public-key schemes based on the hardness of the DLP (e.g. ElGamal encryption)
- Again, most state-of-the-art public-key cryptography use elliptic curves.
- All the schemes mentioned here (RSA, Diffie-Hellman, ElGamal, elliptic-curve cryptosystems) will be broken by quantum computers, when they can be build.