A Basic Public Key Example

From Wikibooks, open books for an open world

The elementary working of Public Key Cryptography is best explained with an example. The working below covers the making of simple keys and the encryption and decryption of a sample of plain text. By necessity, the example is greatly simplified. In the examples it is assumed that an encrypted message is to be sent from Site A to Site B.

Contents

Basic Public Key Summary

- Public key encryption is used for internet secure links,
  such as when a browser opens a bank site or a site used with credit cards. Such addresses are prefixed by *https* as opposed to just *http*. RSA-OpenSSL is such an encryption system. An example of this secure site marking can be seen on the address of *this* page; this Wikibooks page (when the user is logged in) shows the *https* prefix or some alternative browser-specific marking that indicates that it is considered secure.
- Each site has an *encryption key* and a *decryption key* of its own, termed the *public* and *private* keys respectively. These are essentially very large numbers. These keys are always different, giving rise to the term *asymmetrical encryption*. In fact whenever we say *key* we mean a pair of numbers comprising the key; a key number to use in the raising of powers and another number that is the modulus of the arithmetic to be used for the work. For those unfamiliar with modular arithmetic, refer to the Wikibooks page **High School Mathematics Extensions/Primes/Modular Arithmetic** for a good, yet simple description.
- Public keys are openly available for anybody to see, but private keys are not. The receiving site makes his *public key* available to the message sender, or by his making use of public directories. For each message transmission the sender uses this key to make the code. In this way only the private key of the recipient will decrypt it. A worked example has been provided in the text below, and the basic process can be seen in Figure 1.
- In fact, the two keys used for public key encryption form a *reversible function*. You could *encrypt* with the private key and *decrypt* with the public key if the system designers had otherwise intended it. Of course, for less public use the public keys could just as easily be treated as secret also. This reversible nature of the key pair is useful in testing digital certificates, where the issuer encrypts a certificate with his *private key* so that the recipient can then use his freely available *public key* to test it's authenticity.
- The numbers used are made deliberately very large, and this makes the task of obtaining the private key from the public key too difficult for a hacker. It involves the factorization of very large numbers, and is very time consuming.
- Such systems, although imperfect, are nonetheless useful provided that the time to break them far exceeds the period for which the data is of any interest. The estimated time to break some such codes is many thousands of years.
- Signed digital certificates help certify the identity of user sites and to deliver public keys. Browsers take steps to confirm their validity.
- The main *advantage* of the public key system is that there is a low administrative burden.

Everything needed to send a message to a site is available in a public directory or is sent openly as a part of setting up the link.
- The main *disadvantage* of public key cryptography is that is too slow for modern internet use. Because of this, the internet most often uses *symmetric encryption* for the main task; (a different method that uses a common key for both encryption and decryption); it simply uses public key methods to conceal the symmetric keys while they are being sent to the far end.
- There are several methods that hackers use to break coding:
  - The *brute force* cracking of a key refers to trying every possible combination of *private key* while testing it against the relevant cyphertext. Such testing is time consuming, because a dictionary check or human intervention is needed at each iteration to decide whether or not plain language has emerged. Also, the numbers are very large, so this method is rarely of much interest.
  - A *mathematical* attack refers to the finding of the two prime numbers, the product of which makes the modulus of the publicly available key. If these can be found then it simplifies the finding of the private key (more later); this method has the advantage that computers can be left to the task without much intervention. At the time of writing (2014) the record for breaking a key by mathematical attack is by Lenstra et al, on 12 December 2009, when an RSA-768 bit modulus (232 decimal digits) was factored using a method called the General Number Field Sieve (GNFS). The process required two years of collaboration and many hundreds of computing machines. (see: http://eprint.iacr.org/2010/006.pdf) Today most encryption keys in use are much bigger than the one that was broken, and a 1024 or 2048-bit key in an SSL certificate is still considered fairly safe against a mathematical attack. Note that the difficulty of breaking such a key increases *exponentially* with the key length.
  - The history of successful intrusions has not involved code breaking however, but the hacking of the servers for their data and private keys. Other exploits have relied on the security omissions of individuals, or defective programming. For example, a recent SSL exploit, (2000-2014?), has involved accessing data, not by code breaking, but by a programing flaw that allows the sender to download blocks of memory content from the destination's server. The intention in SSL is to allow some text from the recipient's server to be returned as proof of message receipt and successful decryption. For this purpose the sender can specify the length of text to return, usually a header, and in any case less than 64Kbits in length. The core of the flaw was that if a very short message was sent but the sender asked for a larger block to be returned than was sent, the faulty program would oblige, so returning data that included other secure material from memory. Repeating this process every few seconds allowed a hacker to accumulate a large data block. The matter is stated to have been since corrected, but is said to have been available to any who knew of it for a period of about four years.

Making Site B's PUBLIC Key

A public key is available to all, and is used to encrypt messages that are being sent to the key's owner.

- Each site's computer produces two very large prime numbers, and since they are the basis of all that follows, these numbers are never revealed to any other. (*Prime numbers are those that have no factors other than themselves or unity*).
- These two numbers are multiplied together to produce the modulus used in all of that site's calculations. The main public key is also derived from these primes, and determines the exponent to which the plain language numbers will be raised.
- This public key is available in directories and from certificate authorities, so when the SENDER wants to encrypt a message by public key cryptography he can easily use the *recipient's* public key (and modulus) to do it. Each site's public key and modulus are almost certainly different.

To illustrate the point for an intending recipient, let us make a simple example with the large prime numbers replaced with very small ones.

Say the two secretly held prime numbers are:

```
p = 5  ,  q = 11
```

Then the modulus of the arithmetic that will be used is given by their product:

```
m = 5  x  11 = 55    (the modulus of the arithmetic to use)
```

The encryption key can be found as follows: First, using the two prime numbers, calculate the function:

```
   f(n)  = (p-1) x (q-1)
∵ p = 5 and q = 11
∴ f(n) = (5-1) x (11-1)
∴ f(n) =  40
```

then,
Select ANY number that is *relatively prime* to f(n) and less than it.

(*Two numbers are said to be* ***relatively prime*** *when they share no common factors other than one. This term is also referred to as* ***mutually prime****, or* ***coprime***).

```
 The possible choices become:
 3, 7, 9, 11, 13, 17, 19, 21, 23, 27, 29, 31, 33, 37, and 39.

 Say we select the public encrypt key =  7
```

The receiving site's PUBLIC key can then be safely given to the world as :

```
(7, 55) as (encryption exponent, modulus)
(In practice the numbers would be very much larger)
```

The actual size of the numbers used is very large. For example, for a 1024-bit RSA encryption, this number is the size in bits of the modulus; this is equivalent to a decimal number of about 308 digits, or 256 hex digits. The public *exponent* most often chosen has an integer value of 65537. This exponent is chosen because it produces faster encryption than some other selections; that is, because of its large zero count in the binary form (10000000000000001), it lends itself to fast processing with binary shifting methods. It is known elsewhere as Fermat number F4.

Making Site B's Private KEY

Used by Site B when decrypting messages that were sent to them, encrypted using Site B's public key.

The private key pair is used to decrypt messages, and this key will only work if the public key of the same site was used to encrypt the message. That is to say, Site B's public key is obtained from a directory, then used by Site A to encrypt a message for them. When the message gets to Site B, Site B uses its own private key for decryption.

Continuing with the simple example above, the private key of Site B is made from its public key as follows:

```
   private decrypt key = (public encrypt key)⁻¹Mod f(n)

∵ public encrypt key = 7 , and f(n) = 40

∴ (private decrypt key x 7) Mod 40 = 1

∴ private decrypt key = 23

  The Site B PRIVATE key pair is then:

  (23,55) as (decryption exponent, modulus)
```

It will have been noted by some that the same number can result for both the encrypt and decrypt exponents. This particular case must be avoided by deliberate testing since a hacker would likely test for this possibility early in the process of an attack. In the above examples, this would have been the case if 9, 11, 21, 33 or 39 were chosen for the public key instead of some other. Lest it be thought that anticipation of this error is simple, notice that even in this set that both coprimes that are themselves prime (eg; leading to: 11 * 11 = 1 mod 40), and those that are coprime but not in themselves prime (eg; 9, 21, 33, and 39), can all produce this insecure state of affairs.

With the use of long primes, m the modulus (their product), is very much longer, but it should be apparent that an intending hacker could still obtain the private key if he were able to find the two secret primes as a starting point. Both the public key and the modulus to use with it are given to all who require it for encryption, so the burden of a mathematical attack reduces to the difficulty of factoring the modulus into these two secret primes. For the simple example shown above (m=55) this task is very simple, but for a very large number this effort is prohibitively long.

The native format in which the private key is delivered is in fact base-64. Unlike the public key string, **the layout of a practical private key string** for a 1024-bit RSA encryption contains the private key details, the public key details, and the secret numbers used in their making, as well as various other numbers and headers. The private key *exponent*, unlike the public exponent, is quite long, and is the equivalent of 256 hex digits in length. The secret primes are each 128 hex numbers in length. The decimal equivalent lengths are 308 digits for the *private* exponent (and the modulus), and 154 digits for each of the *secret numbers*.

Factoring the Modulus

One way of factoring the modulus without any other algorithm to assist, is to simply divide the modulus by a succession of increasing primes until a zero remainder results. Then the two primes would be known. However the number of primes is also very high in such a large modulus. In general the following approximation gives the number of primes in the number x as:

```
   number of primes ≅ x/(logx - 1)

   now a 64 bit space is equivalent to about 20 digits

∴ number of primes ≅ 4 * 10¹⁷

   then assuming 1 million calculations per second, (a wildly optimistic assumption for
most):

   the time to test all the primes ≅ 13,500 years
```

The example here was limited to 64 bits because the more representative figures, 128, 256, 512, 1024, and 2048-bit calculations are too big for most calculators. See **The Math Behind Estimations to Break a 2048-bit Certificate** by DigiCert for more details.

This example does not consider the use of improved algorithms for factoring, and these appear frequently in the literature. At present, (2014), the best of these is considered to be the General Number Field Sieve (GNFS), used to establish the **record in December 2009**.

Encryption with B's Public Key

Assume that the public key pair belong to a Site B. Assume also that a plain language character represented by the number '2' is to be encrypted by Site A and sent to the recipient Site B: Site A uses Site B's public key pair to do so.

```
   Assume plaintext=2
```

   cyphertext = plaintext $^{\text{public encrypt key}}$ Mod n

∵ public encrypt key =7, and modulus = 55

∴ cyphertext = 2^7 Mod 55 = 128 Mod 55

∴ cyphertext = 18

With the very small numbers used in the example the cracking of the code would be relatively simple. But for very large values of primes *p* and *q*, the burden becomes very difficult. In some cases the task would involve an unreasonable time even for a very large number of computers.

Public key encryption does not disguise the relative frequency of the characters used. This is considered a failing in such systems since it improves the chances of cracking the code. So, the plaintext characters are arranged into groups before encryption to hide their natural frequencies of use; the groups are very large, the limit being that the size of a number encrypted must be smaller than the modulus in use.

Decryption with B's Private Key

Decryption using the above specific example is acheived as follows: For the received cyphertext = 18

```
   With cyphertext=18 from previous section
```

   Plaintext = cyphertext$^{\text{private decrypt key}}$ Mod n

∵ private decrypt key =23, and modulus = 55

∴ Plaintext = 18^{23} Mod 55 = 74347713614021927913318776832 Mod 55

∴ Plaintext = 2 (You can only just confirm this with the Windows scientific calculator)

Notice that the plain language value of *2* has been recovered, which is the required result.

Retrieved from "https://en.wikibooks.org/w/index.php?title=A_Basic_Public_Key_Example&oldid=3009481"