Bionic Buffalo Tech Note #35:

How Encryption and Digital Signatures Work

last revised Wednesday 19 May 1999

Overview

This is a very basic introduction to encryption and digital signatures. It explains how they work and gives some examples of how they are used.

Encryption scrambles or modifies a message or document so it cannot be read and understood, except by the intended recipient. A key is necessary to reverse the scrambling or modification, to make the message readable. Encryption is used for secrecy in communication.

Digital signatures are used to verify that a message or document was authored by a certain person, and that it was not altered or modified by anyone else. (The process of verifying the integrity of a document is *authentication*.)

Encryption and digital signatures can be used together, or separately.

- a message may be encrypted, but not digitally signed (only people with the key can read it, but the reader cannot be certain who actually wrote it)
- a message may be digitally signed, but not encrypted (everyone can tell who wrote it, and everyone can read it)
- a message may be encrypted first, then digitally signed (only someone with the key can read it, but anyone can tell who wrote it)
- a message may be digitally signed first, then encrypted (only someone with the key can read it, and only that same reader can be sure who sent the document)

These technologies are used nearly everywhere. They are used to safeguard network traffic among different computers, to keep e-mail private, to conduct electronic commerce, to control access to buildings and files, and to prevent playing television channels by unauthorized users.

Types of Encryption

Encryption depends on modifying or scrambling messages so a key is necessary to understand the message. As an example, suppose we take a message and change each letter of the alphabet by substituting a later letter.

- If the key is 1, then chose the next letter: A becomes B, B becomes C, C becomes D, and so on.

- If the key is 2, then chose the letter two letters later: A becomes C, B becomes D, C becomes E, and so on.

- For other keys, use the key to skip the specified number of letters.

If the original message is

```
CATS HAVE FUR
```

then the encrypted version depends on the key as follows:

```
(key = 1) DBUT IBWF GVS
(key = 2) ECVU JCXG HWT
(key = 3) FDWV KDYH IXU
...and so forth
```

To decrypt the message, the recipient uses the key to shift the letters of the alphabet backwards by the correct number.

This example is contrived and (obviously) too simple to use in real life. The key would be easy to find, just by trying 25 possible numbers. However, it illustrates two points:

- the total number of keys possible is 25; this is known as the size of the *key space*

- the same key is used to encrypt the message, as is used to decrypt the message; such a scheme is called a *symmetric* cryptosystem

In real life, the scrambling mechanisms are much more complex, and the key spaces are much larger:

- banks communicate with ATMs using a key space of 2^{56} (about 72,058,000,000,000,000) keys

- web browsers on the internet secure traffic using a key space of 2^{128} keys (a number 39 digits long)

Key spaces are normally measured in bits. The DES algorithm used by ATMs has a 56-bit key space, since there are 2^{56} possible keys. The scheme used by web browsers is considered to have a 128-bit key space.

There is a second kind of cryptosystem, which uses one key to encrypt, and a different key to decrypt. Such a system is called an *asymmetric* cryptosystem. It has the advantage that many people can use the encryption key to encrypt messages for a recipient, but only the recipient who has the decryption key can read and understand them. When the encryption key is published for anyone to use, then this is called a *public key* cryptosystem, since one of the two keys is public.

DES: AN EXAMPLE OF A SYMMETRIC CRYPTOSYSTEM

One of the most common symmetric encryption mechanisms is DES (Data Encryption Standard), which is a U S Government standard widely used in financial and many other applications. DES uses a 56-bit key to scramble and unscramble messages. Messages are encrypted or decrypted 64 bits at a time. Encryption is done in 18 steps: an initial permutation, 16 operations called *rounds*, and a final permutation.

The initial permutation is a simple rearrangement of the bits in the input.

Each of the rounds is the same, except for the inputs. The output of one round becomes the input for the next round. Each round has seven steps:

1. The 56-bit key is shifted and permuted, and 48 bits are selected.

2. The right half of the 64-bit output of the previous round is permuted and expanded, and 48 bits are selected.

3. The 48 bits from Step 1 are XORed with the 48 bits from Step 2.

4. The result of Step 3 is divided into 8 different 6-bit numbers. Each of these numbers is used to index into a table to produce 8 new 4-bit numbers. The indexing operation is not simple, and this step and the design of the tables (called *S-boxes*) are crucial to the strength of the encryption.

5. The 32 bits produced by Step 4 are rearranged again. (This permutation is called the *P-box*.)

6. The left half of the 64-bit output of the previous round is XORed with the output of Step 5, to become the right half of the output of this round.

7. The right half of the 64-bit output of the previous round becomes the left half of the output of this round.

The final permutation is another rearrangement of the input bits, and is the simple inverse of the initial permutation.

Decrypting a DES message is almost the same as encrypting it, except that the selection of the 48 key bits for each round is done so that the 16 rounds see the key selections in reverse order.

Conceptually, the whole process is a complex scrambling operation, although there is an underlying mathematical rigour to the design. The reason DES encryption works is that there is no known, simple way to unscramble a message without the key. To break the code, an attacker would have to try large numbers of keys, even with known systematic attacks. It is the computational difficulty of trying this which makes DES secure. (However, see the section *Comparing Different Cryptosystems*, below.)

One characteristic of DES is typical of symmetric cryptosystems: the key size is fixed. Each symmetric algorithm almost always has a single, fixed key size.

RSA: AN EXAMPLE OF AN ASYMMETRIC CRYPTOSYSTEM

One of the most common asymmetric cryptosystems is RSA, named for the initials of the inventors (Rivest, Shamir, and Adleman). Like most asymmetric algorithms, RSA has variable key sizes, and is based on mathematical problems instead of on fixed scrambling operations. Of course, like all asymmetric algorithms, the encryption key is different from the decryption key.

RSA is based on prime numbers. Unlike DES, which is based on a complex scrambling operation, RSA depends on simple scrambling. On the other hand, for the same encryption strength, RSA uses much larger keys.

An RSA encryption key consists of a pair of numbers (N, E). To encrypt a message M, the sender computes $C = M^E \bmod N$; the result is the encrypted (ciphertext) message.

The receiver must decrypt by solving the equation $C = M^E \bmod N$, for the value of M. If this were easy, then anyone could decrypt the message and the cryptosystem would not be of much use. Therefore, it must be difficult to be useful. In fact, it must be so difficult as to be infeasible to anyone who did not have a clue about the solution.

To accomplish this, choose the values of E and N in the following way:

1. Choose two large prime numbers, P and Q.

2. Choose E, so that $E<PQ$ and $(P-1)(Q-1)$ have no common factors. (That is, they are relatively prime.)

3. Compute $D=E^{-1}(\bmod (P-1)(Q-1))$, so that $DE = 1 \pmod{(P-1)(Q-1)}$.

The number or clue D needed to solve the problem easily is known as the *decryption key* or *secret key*, since only the intended recipient knows it. The value E is known as the *encryption key* or *public key*, since anyone who needs to send a message can and must know it.

Without knowing D, if the values of P and Q are large enough, the equation $C = M^E \bmod N$ is extremely difficult to solve for M. Some implementations of RSA use values so large that all of the computers on earth, working together, could not compute the solution in less than billions of years.

Of course, someone may develop a new way to solve the equation, which does not require so much computation. However, mathematicians have been trying (for other reasons) to solve this equation for a long time, and many believe there is no shortcut to a solution.

As with other public key cryptosystems, the key length of RSA is variable. By choosing a longer key, more computation is required to solve the equation, and hence the message is more secure.

DSA: AN EXAMPLE OF A DIGITAL SIGNATURE MECHANISM

One of the most common digital signature mechanisms, the Digital Signature Algorithm (DSA) is the basis of the Digital Signature Standard (DSS), a U.S. Government document. As with other digital signature algorithms, DSA lets one person with a secret key "sign" a document, so that others with a matching public key can verify it must have been signed only by the holder of the secret key.

Digital signatures depend on *hash functions*, which are one-way computations done on a message. They are called "one-way" because there is no known way (without infeasible amounts of computation) to find a message with a given hash value. In other words, a hash value can be determined for a given message, but it is not known to be possible to construct *any* message with a given hash value. Hash functions are similar to the scrambling operations used in symmetric key encryption, except that there is no decryption key: the operation is irreversible. The result has a fixed length, which is 160 bits in the case of the Secure Hash Algorithm (SHA) used by DSA.

In practice, digital signatures are used to sign the hash values of messages, not the messages themselves. Thus it is possible to sign a message's hash value, without even knowing the content of the message. This makes it possible to have *digital notaries*, who can verify a document existed (and was signed), without the notary knowing anything about what was in the document.

The private key in DSA is a number X. It is known only to the signer.

The public key in DSA consists of four numbers:

- P is a prime number, between 512 and 1024 bits long
- Q is a 160-bit prime factor of $P-1$.
- $G = h^{(P-1)/Q}$, where $H < P-1$ and $G \bmod Q > 1$.
- $Y = G^{X} \bmod P$, which is a 160-bit number.

A signature on a document's hash value H consists of two numbers R and S:

- $R = (G^{K} \bmod P) \bmod Q$, where K is a randomly-chosen number $< Q$.

- $S = (K^{-1} (H + XR)) \bmod Q$

To verify the signature, a recipient must compute a value V from the known information:

- $W = S^{-1} \bmod Q$
- $U1 = HW \bmod Q$
- $U2 = RW \bmod Q$
- $V = ((G^{U1} Y^{U2}) \bmod P) \bmod Q$

If $V = R$, then document was signed by the person with the public key (P, Q, G, Y).

The security of DSA is based on the computational infeasibility of finding a solution for the equation $S = (K^{-1} (H + XR)) \bmod Q$, when X is not known.

COMPARING DIFFERENT CRYPTOSYSTEMS

There are hundreds of different cryptosystems and signature systems available. They are commonly compared with regard to security, computational effort, convenience, implementation quality, political considerations, and commercial factors. This Tech Note will not go into detailed comparisons, but will list some considerations that are important.

SECURITY

The first consideration in evaluating cryptosystems is security: how easy is it to read an encrypted message, or forge a digital signature?

Unfortunately, the answer cannot usually be given with certainty. It is rarely possible to prove that a certain amount of computation is needed to break a code. Advances in mathematics, and the presence of secret or hidden mechanisms in an algorithm are the most common reasons given for not being sure how secure a given system might be.

Almost all experts believe the strongest recommendation for the cryptographic strength of a system is its ability to withstand sustained, public scrutiny. The mechanisms believed to be the strongest are those, which have been public for the longest time, allowing cryptographers the opportunity to try and break them.

Don't trust a secret or proprietary system: if it hasn't withstood public study and criticism, it hasn't passed the most reliable test for security.

The mere fact that a system is in wide use does not mean it is considered secure. DES has already been broken, and DSA is considered by many to be nearly broken, in spite of the fact that these two systems are currently very popular.

The amount of security needed may also be determined by how long a message must remain secret. For example, a military battle plan might have no value after a certain period of time, or a financial transaction request might have no significance after the transaction has been completed. For these purposes, less security is required than for long-term needs, since an attacker would be required to break the code in a limited period of time and a weaker system might be sufficient.

Computational Effort

The effort required to encrypt or decrypt messages can be considerable. In many cases, this is an important consideration. Some systems are more efficient or computationally economical than others.

Many cryptosystems are only marginally different from others, and in such cases the computational effort may be the deciding factor in choosing one system over the others.

Convenience

Public key systems have a significant advantage over symmetric key systems: keys can be exchanged without prior security. Strangers can freely exchange encryption keys, since they are public and need not be kept secret. When using symmetric keys, on the other hand, any exchange must be done secretly, since someone overhearing or intercepting the exchange will be able to decrypt subsequent traffic.

Sometimes, available software already uses certain methods, so it may be more convenient to use one of these methods.

Implementation Quality

Even when an algorithm is secure, a flawed implementation may render it unsafe. The worst part is that the user may have a false sense of security, which encourages him to take unwarranted risks in using the system.

The size or reputation of the vendor is no guarantee that the algorithm will be implemented correctly. Microsoft, for example, has released erroneous implementations of encryption algorithms.

As with the security of the algorithm itself, the strongest indication that the implementation is trustworthy is the availability of the source code. If the source code has been released, then public scrutiny will presumably find errors that might exist in the code.

Political Considerations

It doesn't take long to find that encryption is a highly controversial subject, restricted in many ways by laws and regulations. There is a conflict between the need for individual liberty and economic freedom, and a desire for governments to monitor their own citizens and the military and industrial activities of competitors.

Governments have at times encouraged the use of systems, which allow the governments themselves to decrypt traffic, which is another way of saying the systems are insecure. Even large vendors such as IBM have admittedly added secret mechanisms (known as *trap doors*) into commercial software, to allow U.S. Government decryption of message traffic.

Because of the large quantity of rumour and disinformation about these matters, it is important to trust no one completely. Once again, *public scrutiny is the most significant single indicator of how trustworthy a system might be*. If the algorithm hasn't been widely studied, and the source code isn't available, don't believe it.

Just because the encryption program itself can be studied, is not sufficient reason to believe it is secure. An operating system, for example, or a virus, can intercept key strokes input to an application, rendering any cryptosystem unsafe. Open systems, such as FreeBSD and Linux, which are built by the user from the sources, are considered the most secure platforms.

Because of export or import restrictions, the use of a foreign implementation (which has no problems with export) or a domestic implementation (which has no problems with import) might be preferred.

Commercial Factors

Some cryptosystems are patented, and the patents are not equally valid in all jurisdictions. For example, RSA is patented in the United States, but can be used freely in other countries. On the other hand, IDEA (a symmetric key system) is patented in Switzerland, but can be used freely in the United States.

Needless to say, the use of a patented system may entail royalties or other expenses, which would not apply to a system free of such restrictions.

There is also a considerable amount of lobbying by patent holders to include their algorithms into public standards, since it increases their revenue. The owners of the RSA patent have successfully worked to incorporate the RSA algorithm into various formal standards, making its use required for compliance.

Certificates: Whom Do You Trust?

Digital signatures can assure that a document was signed by a person with a certain public key, but ultimately it may be important to know who that person is. Anyone can create a public key with common software, and say their name is X, their address is Y, and so on. How do you know if they're telling the truth?

There are two approaches to answering this question. Both involve *certificates*, which are digitally signed statements, which attest to the identity of a keyholder. The difference is in who issues the certificates.

One approach, used by PGP, allows anyone to vouch for anyone else's identity. It is up to the individual to decide whom to trust. The user must decide whom to believe, when a statement is made that a key belongs to a certain person. If someone you trust introduces someone else by vouching for the authenticity of his key, then you are more inclined to believe it than if you were introduced by a stranger. In the PGP approach, one person can sign another person's key, as a statement that the key belongs to the ostensive owner. The overall structure is called the *web of trust*.

The other approach, more favored by governments and other hierarchical entities, uses formal *certificate authorities* (or *CA*s). The root CA issues certificates of authenticity, after asking the applicant to present credentials such as driver's licenses, passports, or other such items. Usually, the CAs are organized in hierarchies; for example, a national government might operate a root CA, which accredits secondary CAs, which accredit individual users.

Technically, there is no inherent advantage to one approach over another. The choice should be based on practical - and philosophical - considerations. Certainly, it is possible to use both systems together.

Putting Them All Together

A typical scenario uses all three tools: symmetric cryptography, asymmetric cryptography, and digital signatures.

The common steps in a secure communication are:

1. Exchange public keys, so it is possible to send secure messages between the users.

2. Generate a pseudo-random session key, which can be used by symmetric cryptosystems.

3. Using asymmetric (public key) cryptography, share the secret session key.

4. Switch to symmetric cryptography, using the secret session key. The switch to symmetric cryptography is done, because the computational burden is lower than with public key cryptography. The initial public key may be used only to exchange session keys, or it may be more permanent.

5. Conduct the session using messages symmetrically encrypted with the session key.

6. If necessary, exchange digitally-signed certificates to establish each other's identity.

The certificates are exchanged after the encrypted session is established. Otherwise, an attacker might "sneak in" between the certificate exchange in the beginning of the encrypted session. The attacker could intercept the traffic, and then pretend to be one or both of the parties. (This is called a "man in the middle" attack.)

Session keys, and the above scenario, are used not only for on-line communication, but also for e-mail. For example, PGP encrypts each e-mail message using a pseudo-random session key and symmetric cryptography. Each mail message is sent along with its symmetric key, but the symmetric key is encrypted using public-key cryptography. If a message is to be read by more than one recipient, then multiple copies of the symmetric session key are included, each encrypted with the public key of one of the intended recipients.

Sometimes, a session may not need to be private (encrypted), but it must be secure against lost or inserted messages. An attacker may desire to insert or delete messages, modifying the message sequence numbers to fool the recipients.

A defense against message insertion or deletion can be done without encryption, using the hash functions described in the above section describing digital signatures. To use this defense, the parties must share a secret value, which can be exchanged beforehand or which can be exchanged at the time of the session using public key cryptography. Along with the message sequence number, each message contains a hash of the message, the sequence number and the secret value together. Even if an attacker is able to modify the sequence numbers or the message itself, he cannot correctly modify the hash values without knowing the shared, secret value. (Digital signatures can be used to achieve this same purpose, but the computational effort would be much greater.)