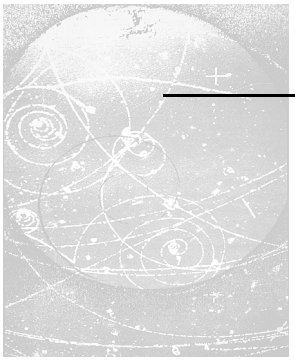
PART ONE: OVERVIEW



CHAPTER

1

Cryptographic History and Techniques

Since the beginning of time people have kept secrets. Probably from the beginning of your memory you have done the same. It's a natural human desire. People have always had, and always will have, some secrets that they either want to keep to themselves or share with only a privileged few. The easiest secret to keep is one that you will tell to no one. The more people you wish to share a secret with, and the more public the forum in which you will communicate your secret, the harder it is to keep your secret a secret.

Secrets in History

In antiquity it was easier to keep a secret because the ability to read was a privilege known to a select few. The number of people who could read a written secret was very limited. Merely by restricting access to the written word, a secret could be retained. The security of such a scheme is obviously limited.

As the ability to read became more prevalent the need to keep secrets from those with the ability to read became more necessary. This need manifested itself most notably in war. While those doing the actual fighting were most likely illiterate, the ones who waged the war were not and each side, no doubt, employed soldiers who could read and speak the language of their enemies. Military communications in the battlefield were probably the genesis of cryptography.

Early attempts at cryptography were simplistic. It is rumored that Caesar used a rudimentary cipher to obfuscate his messages. Those with whom he wished to share a secret were told how to reconstruct the original message. This cipher, *The Caesar Cipher*, was a simple substitution cipher: Every letter in the alphabet was replaced by the letter three places away modulus the length of the alphabet. In other words, the letter A became D, B became E, X became A, Y became B, Z became C, etc. It's a simple cipher to decode but *li brx grq'w nqrz krz lw'v qrw reylrxv*!—in other words, if you don't know how it's not obvious! Another variant of this is the ROT-13 cipher. Each letter is rotated 13 places.

Simple substitution ciphers are not very good since each occurrence of a letter is replaced by the same letter. Analysis of a language will result in the probability of letters following other letters—notice the occurrence of the letter *r* in the above "ciphertext." It's probably a vowel—and this information can be used to determine the substitution offset.

Confidentiality was not the only concern in antiquity. Authentication was another. When few could write, a signature would probably suffice. As the knowledge of reading and writing became more prevalent, wax seals bearing the unique mark of the "signer" were used to authenticate letters, documents, and edicts. The rise of industry brought the capability to make such a seal to more people and the seal ceased being unique. In effect, it became trivial to forge a seal.

Jumping to modern times, ciphers, and their cryptanalysis, have a very notable place in history. Prior to the United States' involvement in World War II, the United States Army was able to crack a code used by the Japanese government. This capability allowed the United States to be

forewarned about the attack on Pearl Harbor. This knowledge was not put to good use, though, and the United States suffered great losses as a result of this "surprise" attack. During the same war the German government used an encryption device called Enigma to encipher its communications. This device used a set of rotors (Enigma machines had 5 but only 3 were used for any given communication) that contained the letters of the alphabet and could be independently set. Each letter of input text was transformed into a seemingly random character of output. Seemingly random, because the permutations of transposition were astronomical. The cracking of the Enigma machine was an incredible feat started by the Polish and finished by the British and the story behind the cryptanalysis of Enigma is large enough to be its own book. In fact, several books have been written on the subject.

Communication technology has grown steadily from the days of Caesar to modern times. From papyrus paper to telegram, telex, telephone, FAX, and e-mail, the ability to communicate has been made easier and more ubiquitous. At the same time, the ability to keep such communications secret has remained something of a black art known only to a few—generally governments and military organizations.

The security of each method of communication is dependent on the medium over which the communication is made. The more open the medium the greater the possibility of the message falling into the hands of those for whom it was not intended. Modern day methods of communication are open and public. A telephone call or FAX transmission goes across a shared, public, circuit-switched phone network. An e-mail is transmitted across a shared, public, packet-switched network. An entity in the network between communications endpoints could easily intercept the message. Retention of a secret transmitted using modern methods of communication requires some sort of cryptographic technique to prevent any of these eavesdroppers from learning the secret.

At its base modern cryptography relies on a secret known by the intended recipient(s) of the message. Typically the method of encipherment, the algorithm, is known but the "key" to unlock the secret is not. There are certain cryptosystems that are based upon a secret algorithm—so-called "security through obscurity"—but typically people are reluctant to use an algorithm which is not open to public scrutiny (the debate over the Clipper Chip is a prime example of this).

The problem, then, is to ensure the secrecy of the key—that it is obtainable only by those to whom it should be known. Modern cryptography provides for this.

Rise of the Internet

The popularity of the Internet has given rise to many claims on it. Everybody from browser companies to workstation vendors to router vendors lays claim to being the genesis of or the backbone of the Internet. Most agree, though, that the modern Internet was born in the late '60s under the name ARPANET. The ARPANET was a research tool for those doing work for the United States government under the direction of the Advanced Research Projects Agency (ARPA). The original contract was awarded to BBN of Cambridge, Massachusetts.

ARPANET traffic consisted of communications between universities and military and government laboratories. Researchers at disparate locations were able to exchange files and electronic messages with each other via ARPANET. As the network grew it split into two: MILNET, which was used for military use, and ARPANET (it retained the name), which continued to be used for experimental research. In the early '80s, a standard for ARPANET communications protocols, actually a suite of protocols, was specified. This was termed the TCP/IP protocol suite which eventually became just TCP/IP. It is the base of almost all network traffic today.

In 1987 the National Science Foundation (NSF) funded a network to connect the six supercomputer centers that were spread out nationwide. This network, called NSFnet, spanned the United States from San Diego, California on the west coast to Princeton, New Jersey on the east coast. The original NSFnet was over 56K leased lines, fast in those days but slow by today's standards, so NSF also solicited proposals to build a new high-speed network. The winning proposal was submitted by MCI, IBM, and MERIT (an organization which came out of a network at the University of Michigan), and the backbone of what we call the Internet was built.

Over the course of the '90s, the backbone of this network grew by the addition of different long-haul carriers providing leased line connections and local Internet Service Providers (ISPs) providing local access and short-haul connections. Today, through mutually beneficial service agreements, networks are connected with each side agreeing to carry the other's traffic on the condition that its traffic is also carried. This has created a worldwide network in which, for the price of the initial connection, access is provided to a virtually unlimited amount of resources spanning the entire globe.

Internet Security

The Internet is an ethereal thing. It can appear quite different when looked at for different purposes. For the purposes of secret-sharing, imagine the Internet as a huge town hall which is packed with people. Attempting to communicate a secret in such an environment is difficult, and the chance of others overhearing a conversation between two people increases as the distance between those two people increases. Since the Internet is truly global, no secret of any value can be communicated on it without the help of cryptography.

As the Internet grows (almost exponentially in recent years), its utility increases. Messages can be sent cheaply and reliably and communication is the lifeblood of business. For a company to engage in electronic commerce—the sale of goods and services over the Internet—security is a must. Sensitive information such as credit card numbers must be protected and a business must be able to authenticate each and every sale. In addition, businesses can use the Internet to inexpensively connect disparate offices. Interoffice electronic mail and even phone calls can be routed over the Internet. Because sensitive corporate information would most likely be transmitted over these links, the need for security should be obvious.

But, Internet security concerns are not solely business'. Each and every person has a need and a right to privacy, and when someone goes on-line, the expectation of privacy does not disappear. As consumer electronics become more and more Internet-aware, the need for security grows. When our phones and VCRs become accessible over the Internet, we won't want pranksters or hackers to steal our phone line or randomly turn our VCRs on and off.

Privacy is not just confidentiality, though; it also includes anonymity. People must be comfortable in cyberspace and an often ignored component of that is the ability for an individual to remain anonymous. What we read, where we go, to whom we talk, for whom we vote, and what we buy is not information that most people traditionally publicize, and if people are required to disclose information in cyberspace that they would not normally disclose in real life, they will be reluctant to engage in Internet activity.

Thankfully, cryptography can address these concerns.

Cryptographic Building Blocks

Every system that is established can be hacked or attacked. Each different hack or attack represents a distinct threat against the system. For every threat a threat analysis is done to determine the viability of that threat and what damage can be done if that threat is acted upon. Depending on the threat analysis countermeasures are taken such that the cost of launching the attack is greater than the expected gain from the attack.

Cryptographic tools represent such countermeasures. There is no single cryptographic tool. There are various techniques for encrypting messages, for securely exchanging keys, for maintaining the integrity of messages, and for guaranteeing authenticity of a message. These tools can be thought of as building blocks to construct protection against attack.

A single cryptographic building block solves a particular problem—how to authenticate bulk data, how to establish a shared secret—and they can be combined to build a cryptosystem to protect against threats. The cryptosystem must be stronger than the threat against it.

Generally, the strength of a cryptosystem is measured in its complexity. If 2^{32} separate operations are required to break a cryptosystem then the complexity of a particular system is 2^{32}. That's a lot of operations, but if each operation can be performed by a modern computer in hundredths or thousandths of a second, the system might not be strong enough to protect against the threat. Because of this the term *computationally secure* is used to express the security of a modern cryptosystem.

When building a cryptosystem it is necessary to ensure that the component building blocks are used properly and together maintain the necessary strength. For instance, if the strength of the building block used to establish a shared secret is 2^{90} but the strength of the building block used to encrypt the data is only 2^{40} the cryptosystem would be 2^{40}, and that is not computationally secure using modern computers.

One-Way Functions and Trap Doors

A good portion of public key cryptography relies upon a foundation of one-way functions and trapdoors. A one-way function is something that is easy to compute in one direction but difficult, bordering on impossible, to compute in the other direction. A trapdoor is a way to sneak back, in effect a way to cheat and return using a secret passage.

For a one-way function to be useful in cryptography it must exhibit its one way-ness with *any* input. For example, in a finite field it is easy to compute the product of numbers but difficult to factor that product.

Another example is the Discrete Logarithm Problem: with a large prime, *p*, and a generator, *g*, for a particular value *y*, find *x* where

$$g^x = y \bmod p$$

Modular exponentiation is easy, but doing a discrete logarithm to recover the exponent is hard. For any class of numbers—odd numbers, palidrome numbers, numbers divisible by 47—the problem of solving the discrete logarithm is still very hard.

There are no mathematical proofs of one-way functions but certain functions seem to have the properties that a one-way function would have and are generally referred to as such. There may be ways to factor numbers that are just as fast and easy as producing the product but no one has discovered it yet. Because of that we can put our knowledge on the difficulty in factoring to good use.

Trapdoor functions are a bit harder to explain. Modern cryptographic algorithms use them but it's hard to point to a particular one and say, "that's it!" An example of a trapdoor function is a tree with many branches. To get from a leaf to the trunk is straightforward and requires no choices. To get from the trunk back out to a particular leaf requires choosing a branch, then a subbranch, then another subbranch, et cetera, and finally choosing the leaf. The trapdoor would be a description of which branch to take.

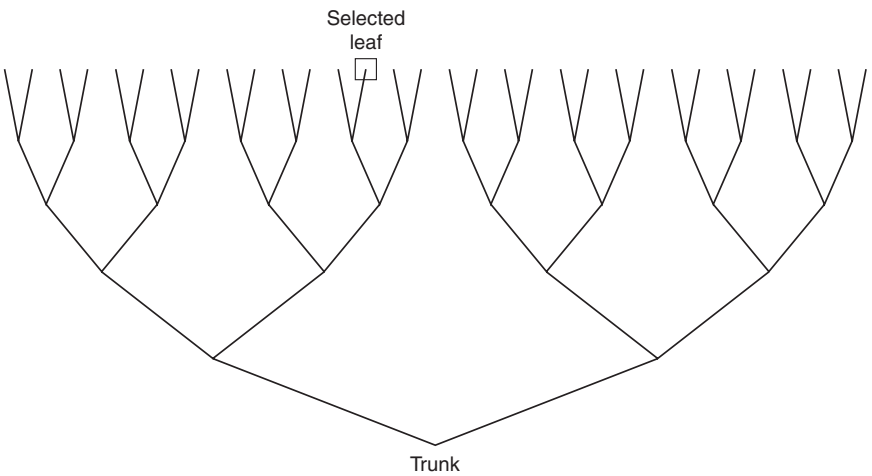


Figure 1.1 A Trap Door Function Tree

The difficulty in finding a particular leaf depends on the depth of the tree. The tree in Figure 1.1 is of depth 5 and there are therefore 2^5, or 32, leaves. The trapdoor to go from the trunk to the indicated leaf would be the "key" of LEFT-RIGHT-RIGHT-LEFT-RIGHT. It should be noted that this "trapdoor function" is wholly unsuitable for any kind of cryptographic purpose. But to illustrate the concept of a trapdoor, it is adequate.

One-Way Hash Functions

One-way hash functions are used in modern cryptosystems for authentication and integrity purposes. A one-way hash function is different than the concept of a one-way function just described. Hash functions take a variable-sized message as input, compress it, and produce a fixed-sized digest. The output of a hash function will be identical for identical input. Since the output is fixed for any length input it should be obvious that there will exist two distinct inputs, *X* and *Y*, for a hash algorithm **H**, such that **H**(*X*) equals **H**(*Y*). Such an occurrence is called a collision. One-way hash functions are designed such that finding collisions—that is, finding two random inputs that will produce identical hash digests—is difficult.

Popular hash functions in use today are: MD5 (Message Digest 5), SHA (the Secure Hash Algorithm), and RIPEMD. They all produce a different-sized digest and have different speed and collision-resistant properties, but are all used extensively today.

Use of one-way functions, which are based on a trapdoor, are much more computationally intensive than using one-way hash functions. Guaranteeing the integrity of a message using a one-way function with a trapdoor—such as a digital signature scheme—takes considerably more time than guaranteeing the integrity of the message using a hash function. There are situations, though, in which it is not possible to use a one-way hash function. In later chapters you will see how IPSec and IKE use both techniques.

Another technique used quite a bit is the simple exclusive-or (XOR) function. This is neither a one-way function, nor a trapdoor function, but is, nonetheless, a useful tool in building cryptographic systems. Remember from early math classes that the XOR of two zeros is zero, the XOR of two ones is zero and the XOR of a zero and a one (or a one and a zero) is one. XOR has a very important feature that it is commutative. Taking any data and XORing it with a key of the same size (one bit, one byte, or more) will produce an output that can be XORed with the key

again to recover the original data. It is the most simplistic "encryption" algorithm. Note, however, that knowing either input and the output it is possible to deduce the other input. This is not generally a characteristic of a real encryption algorithm and illustrates the weakness of using XOR for such a purpose.

Ciphers

Data confidentiality is provided by encryption algorithms which convert a message (plaintext) into gibberish (ciphertext) and back again. Some encryption algorithms are symmetric—the ability to encrypt implies the ability to decrypt—while some are asymmetric—without the use of a trapdoor it is not possible to decrypt what has been encrypted. Asymmetric algorithms are treated not as two separate functions (one for encryption and one for decryption) but as a single algorithm. So, regardless of the "symmetry" of a particular algorithm, encryption algorithms are commutative.

$$plaintext = Decrypt(Encrypt(plaintext))$$

This should be most obvious because any algorithm that permanently scrambled its input would be secure but of little use.

Symmetric Ciphers

Symmetric ciphers use a single key to do both encryption and decryption. There are two types of symmetric ciphers, block ciphers and stream ciphers. Block ciphers, such as AES, CAST, and Blowfish, operate on data one block at a time, with the size of the block depending on the algorithm (AES has a 128-bit block size while both CAST and Blowfish have a 64-bit block size). Each block operation is treated as an atomic act. Stream ciphers, such as RC4, on the other hand operate on data one bit (or one byte) at a time. Appropriately seeded with a key, they will produce a stream of bits which can be XORed with the input. The encryptor and the decryptor must be syncronized to ensure that the same bit in the stream used to encrypt a particular bit of plaintext is also used to decrypt the corresponding bit of ciphertext. If the two ever get out of syncronization the plaintext will not be able to be recovered. It is this syncronization problem that makes stream ciphers inappropriate for use with IPSec. If a packet is dropped using a block cipher that will not affect the processing of subsequent packets, but if a packet is dropped using a stream cipher all

subsequent packets will be affected until the two side re-synchronize somehow.

Both types of symmetric ciphers are ideally suited for bulk encryption. Since block ciphers are used exclusively in IPSec, the reader is referred to the literature for an in-depth description of stream ciphers.

Block ciphers process data by first dividing it up into equal sized chunks. The size of each chunk is determined by the *block size* of the cipher. Since there is no guarantee that the length of the input is a multiple of the block size of a block cipher, it may be necessary to pad the input. If the block size is 64 bits and the last block of input is only 48 bits, it may be necessary to add 16 bits of padding to the block prior to performing the encryption (or decryption) operation.

The basic way to use a block cipher is in Electronic Code Book (ECB) mode. Each block of plaintext encrypts to a block of ciphertext. This causes problems though since the same block of plaintext will encrypt, with the same key, into the same block of ciphertext. Therefore it is possible to build a code book of all possible ciphertexts (using all possible keys) for a known plaintext. If we know that an IP datagram was encrypted, we know that the first 20 bytes of ciphertext represent the IP header and that certain fields of an IP header are predictable. An attacker can use that knowledge, with a code book, to determine the key.

To foil the code book attack against a block cipher it is necessary to use the block cipher in a feedback *mode*. A feedback mode chains blocks together by feeding the results of prior operations into the current operation.

Cipher Block Chaining (CBC) (Figure 1.2) mode takes the previous block of ciphertext and XORs it with the next block of plaintext prior to encryption. There is no "previous block" for the first block so this mode is jumpstarted by XORing the first block with something called an Initialization Vector (IV). The length of the IV must be the same as the block size of the cipher to ensure the entire first block is processed. The IV must have strong pseudo-random properties to ensure that identical plaintext will not produce identical ciphertext. Decryption is the opposite of encryption: Each block is decrypted and XORed with the previous block prior to decryption. The first block is decrypted and XORed with the IV. All ciphers currently defined for use in IPSec are block ciphers operating in CBC mode.

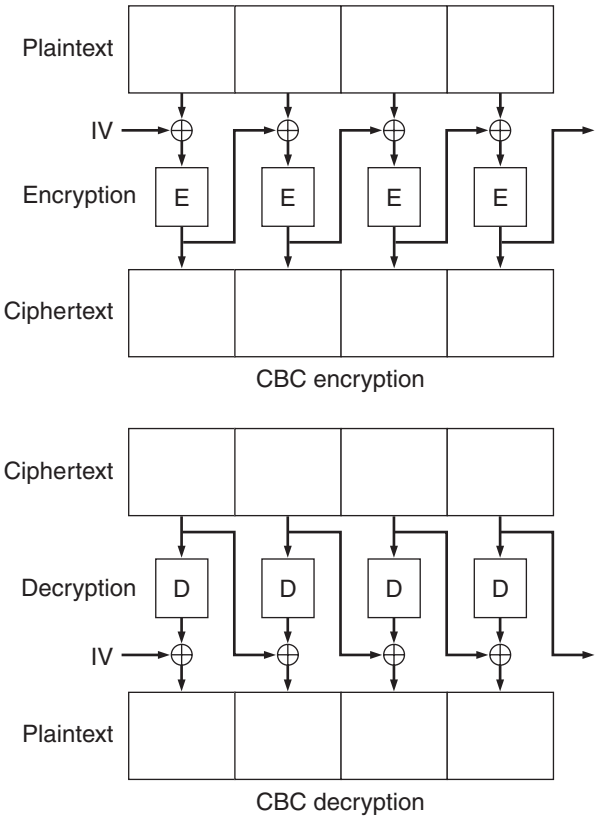


Figure 1.2 Cipher Block Chaining Mode

Other popular modes are Cipher Feedback Mode (CFB), where the previous ciphertext block is encrypted and XORed with the current plaintext block (the first block of plaintext is merely XORed with the IV), and Output Feedback Mode (OFB), which maintains a cipher state that is repeatedly encrypted and XORed with blocks of plaintext to produce ciphertext (an IV represents the initial cipher state).

Asymmetric Ciphers

Asymmetric algorithms are also known as public key algorithms. There are two keys, one public and one private. One key does the encryption, the other the decryption, and given a public key it is computationally impossible to determine the private key (as defined above, we can say that good public key algorithms are *computationally secure*). Good public key algorithms are based on one-way functions.

Public key cryptography is generally held to have been invented by Whitfield Diffie and Martin Hellman in their paper "New Directions in Cryptography," published in IEEE Transactions on Information Theory in 1976. Recently the Communications-Electronics Security Group (CESG) of the British government—the UK version of the United States' NSA— declassified some papers that showed that their cryptanalysts had actually invented the concept six years earlier. In 1970, James Ellis wrote an internal CESG report entitled "The Possibility of Secure Non-Secret Digital Encryption" which discussed an existence theorem, while Clifford Cocks and Malcolm Williamson wrote papers describing practical schemes that closely resemble the RSA and Diffie-Hellman schemes, respectively. Regardless, publication of the Diffie-Hellman paper was a seminal event whose importance is underscored by the nearly 20-year delay in release of the classified British papers. It is not beyond the realm of possibility that if "New Directions in Cryptography" had not been published, this knowledge would still be a classified secret known only to a few.

RSA The most popular public key algorithm is RSA, named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman. The security of RSA is based on the difficulty in factoring the product of two very large prime numbers. This is a one-way function: it is easy to compute the product of two large prime numbers but extremely difficult to factor the product into the original prime numbers. One of the features of RSA is that either key can be used to encrypt data that the other key can decrypt. This means that anyone can encrypt a message in your public key that you alone can decrypt. Also, you can encrypt anything with your private key that anyone with your public key can decrypt. You're probably thinking, what's the point then? But this concept is very important in nonrepudiation and digital signatures (which will be discussed shortly).

A drawback of RSA is that it is quite slow and can operate only on data up to the size of the modulus of its key. A 1024-bit RSA public key can only encrypt data that is less than or equal to that size (actually, it's 1013 bits because the definition on how to encrypt using RSA requires an encoding that consumes 11 bits). While this is a restriction similar to a symmetric block cipher, the speed of RSA makes it unsuitable for bulk data encryption. This does not mean that RSA is not useful. On the contrary, it is a de facto standard for such important techniques as key exchange and digital signature.

El-Gamal Another public key cryptosystem which is suitable for encryption is El-Gamal, named after its inventor, Taher El-Gamal. The El-Gamal cryptosystem is based on the Discrete Logarithm Problem. The main drawback of El-Gamal is that the ciphertext is twice the size of the plaintext. Given our already saturated networks, this is a large drawback. El-Gamal is quite similar to the Diffie-Hellman key exchange, which we'll discuss in detail shortly.

Authentication and Integrity

Confidentiality is necessary to keep a secret, but without authentication you have no way of knowing that the person with whom you share the secret is whom she claims to be. And with no confidence in the integrity of a received message, you don't know if it was the same message actually sent..

Authentication

Public key cryptography can be used for authentication purposes by constructing a so-called *digital signature* which has properties similar to a traditional signature. A traditional handwritten signature is difficult to forge, and is therefore difficult to repudiate. But because a handwritten signature is just more writing on a document, it is possible (although also difficult given a well-written document) for unscrupulous people to add additional text to an already signed document, giving the impression that the signer agrees to or acknowledges that text.

The Internet is a largely anonymous place and digital information can live a long time, so there are other properties we need for digital signatures in addition to those that a traditional handwritten signature affords.

A digital signature must be difficult to forge and therefore difficult to repudiate, just like a traditional signature. In addition, it must convey message integrity and must be unique. We want to prevent additional text from being added to a digitally signed file and we also want to prevent a signature from being removed from an authentic, signed document and added to other documents. These properties can all be met using public key cryptography.

It is easiest to envision digital signature as encryption and verification of a digital signature as decryption. In fact, that is the way an RSA signature works. But another public key algorithm, in fact a standard for digital signatures, aptly named the Digital Signature Standard (DSS), does

not operate in that manner. The difference will be explained shortly, but for purposes of illustration it is encryption and decryption.

What the private key encrypts the public key decrypts. Provided the private key from a public/private key cryptosystem is kept secret, it can be used to construct digital signatures. By encrypting a document with a private key, anybody in possession of the corresponding public key can decrypt the document. Of course an encrypted document is hardly a signature and verification would just entail reconstruction of something that *looks* good out of the encrypted gibberish. It would also require decryption, and implicit signature verification, every time the document merely needs to be read.

A digital signature is therefore not a private-key encryption of the entire document. Digital signature techniques use one-way hash functions to reduce a document down to a digest. It is that digest that is encrypted. Remember that a hash function will produce the same digest every time it is given identical input and that the input can be of arbitrary length. Provided the hash function has strong collision-resistant properties, we can be assured that the signature is unique to the document.

The encrypted digest, the digital signature, can then be appended to an original document. Verification of the signature entails running the original document through the identical hash function to product a temporary digest and decrypting the signature to recover the original digest. If the two digests are equal, the signature is valid. This technique has all the properties we need:

1. **difficult to forge**: only the holder of the private key can generate the signature.
2. **nonrepudiable**: a signed document cannot be repudiated later due to extreme difficulty in forging.
3. **unalterable**: once signed, a document cannot be modified.
4. **nontransferable**: the signature cannot be removed and attached to another document.

It is also possible to have multiple signatures, produced from different private keys, on a single document. Each signature is generated in the same fashion by encrypting a digest of the document to be signed. These encrypted digests are merely appended, one after the other, on the end of the document.

RSA Due to its unique nature—what one key encrypts the other decrypts—RSA is well suited for digital signatures as well as for encryption. You just use a different key to do the encryption! The technique described previously is exactly what happens when using RSA with digital signatures.

There are no requirements to use any particular hash algorithm when using RSA signatures.

DSA The digital signature algorithm is similar to the El-Gamal public key scheme. Both are based on the discrete logarithm problem.

As mentioned, the Digital Signature Algorithm does not actually do encryption for signature generation and decryption for signature verification (although it does have a public and private key). Instead, the private key is used to generate two 160-bit values which represent the signature, and verification is a mathematical demonstration, using the public key, that those two values could only have been generated by the private key and the document that was signed. There is no real "decryption".

DSA requires use of SHA as a hash function for signatures. SHA is the algorithm defined in the U.S. government Federal Information Processing Standard (FIPS) for the Secure Hash Standard and was therefore selected to use for another FIPS, the Digital Signature Standard, of which DSA is the algorithm.

Message Integrity

A digital signature provides integrity on the signed document. Any modification to the document would be detected by checking the signature. One drawback of digital signatures is that they are slow and another is that the entire message must be known prior to signature generation. There is no efficient way to provide message integrity of an ongoing data stream using digital signatures.

Just as there are symmetric and asymmetric ciphers, there are symmetric and asymmetric methods of guaranteeing message integrity. Similar to symmetric ciphers, where one single key is used for both encryption and decryption, symmetric message authentication codes (MACs) use a single key for generating and verifying the authentication information. (MACs are sometimes erroneously referred to as signatures—they're not.)

Hash functions are used as MACs just as they are in digital signatures. Since the input to a hash function can be of any length, all one needs to do to generate a MAC is hash a shared secret key along with the message. The

resulting digest is attached to the message, and verification of the MAC entails hashing the shared secret key with the message to produce a temporary digest and comparing that temporary digest with the digest attached to the message. This technique is referred to as *keyed hashing*. It's important to do keyed hashing because just performing a hash on some data does not really provide any authentication. Anybody could modify the data and merely run the hash algorithm over the modified data. A hash function alone is like a checksum, a keyed hash function is a MAC.

Keyed hashing can be used to provide message authentication to a stream of data by dividing the stream into easily digestible chunks and computing a MAC on each chunk. Those MACs then become part of the stream and are used to verify the integrity of the stream as it is received. Another benefit of keyed hashing is that generation of a hash digest is much faster than generation of a digital signature.

A special kind of keyed hash is called an HMAC, and was designed by Hugo Krawczyk, Ran Canetti, and Mihir Bellare. The HMAC specification is in RFC2104 and can be utilized with any existing hash function, so SHA can become HMAC-SHA and MD5 becomes HMAC-MD5. The HMAC construction is cryptographically stronger than the underlying hashing function. There has recently been a demonstrated collision attack against MD5 (where it is possible to find two different inputs which will produce the same digest), but HMAC-MD5 is not susceptible to this attack.

An HMAC is also a keyed hash but is actually a keyed hash inside a keyed hash. It uses two constant pad values—an inner pad and an outer pad—to modify the keys to the hashes. The HMAC based on hash algorithm H of message M using key K is defined as

HMAC (K, M) = H(K XOR opad, H(K XOR ipad, M))

Where the *ipad* is a 64-element array of the value 0x36 and the *opad* is a 64-element array of the value 0x5c.

All message authentication done in IPSec uses HMACs.

Key Exchanges

Symmetric ciphers and symmetric MACs both require a shared key. The security of the encryption and authentication techniques could be completely undermined by an insecure key exchange.

Diffie-Hellman

The Diffie-Hellman key exchange is the first public key cryptosystem and was the one described in the aforementioned paper "New Directions in Cryptography" by Whitfield Diffie and Martin Hellman. The Diffie-Hellman key exchange is based on the Discrete Logarithm Problem (notice how often this one-way function is used).

This key exchange is extremely important. Using the Diffie-Hellman exchange, a nonsecret, untrusted communications channel (like the Internet) can be used to securely establish a shared secret among the parties of the exchange. It is because of the Diffie-Hellman key exchange that symmetric ciphers and symmetric message integrity schemes (which both require a shared key) can be used in a scalable manner.

The usual players in describing modern cryptography are Alice and Bob and they can be used to illustrate the Diffie-Hellman exchange. All participants in a Diffie-Hellman exchange must first agree on a *group* that defines which prime, p, and generator, g, will be used. A Diffie-Hellman exchange is two-part. In the first part each side, Alice and Bob, choose a random private number (indicated by the lowercase initial of the party) and exponentiate in the group to produce a public value (uppercase initial of the party):

Alice	*Bob*
$A = g^a \bmod p$	$B = g^b \bmod p$

They exchange their public values, Alice gives A to Bob and Bob gives B to Alice, and they exponentiate again, using the other party's public value as the generator, to generate shared secret.

Alice	*Bob*
$B^a \bmod p = g^{ab} \bmod p = A^b \bmod p$	

Notice that A and B can be exchanged over an insecure network without lessening the security of the scheme. g and p do not even need to be kept secret. An eavesdropper (she's usually referred to as Eve) could know g and p a priori, intercept A and B over the insecure channel and still not be able to discover the secret! Once Alice and Bob share a secret they can use it to protect their communications. The Diffie-Hellman exchange allows an insecure channel to become secure. The importance of this cannot be overstated.

One drawback of the Diffie-Hellman exchange is that it is susceptible to a man-in-the-middle attack. In this attack, Mallory intercepts messages between Alice and Bob and fraudulently responds impersonating Bob to Alice and Alice to Bob. Alice thinks she's doing a Diffie-Hellman exchange with Bob but she's really doing with to Mallory. Similarly Bob thinks he's doing a Diffie-Hellman exchange with Alice but he's also doing it with Mallory. Alice can then send Bob secret information protected with the shared secret she thinks she shares with Bob. Mallory can decrypt it, copy it, and re-encrypt it with the secret that Bob has (which he thinks is shared with Alice). Neither Alice nor Bob detect anything out of the ordinary, except perhaps some delay in delivery due to Mallory's involvement.

The susceptibility to man-in-the-middle attack does not render the Diffie-Hellman exchange useless though, because the attack can be thwarted by having Alice and Bob digitally sign their public values. Mallory will not be able to fool Bob into signing her public value and will not be able to make Alice think that her signature is in fact Bob's. Keep this in mind when reading Chapter 7 on the Internet Key Exchange (IKE) Protocol.

RSA Key Exchange

With the RSA cryptosystem it is possible to encrypt with either the public or private key and what one key encrypts the other can decrypt. This capability can be put to use for doing a simplistic key exchange. If Alice wishes to use symmetric cryptography to protect her communications with Bob, she can choose a random number as the key, encrypt it in Bob's public key, and send it to him. Only Bob will be able to decrypt the key since he, alone, has possession of his private key.

An obvious problem with this approach is that anybody—such as Mallory— can encrypt anything in Bob's public key. Alice needs something to bind herself to this key. Once again, a digital signature can be used for such a binding. Alice can sign the key and encrypt both the key and her signature in Bob's public key. A drawback to this approach is that an RSA signature is the same as an RSA encryption: It can only be done on data that is less the size of the modulus and the result is the size of the modulus. If Alice's RSA private key is the same size as Bob's RSA public key, her signature will be too big to encrypt in a single operation.

Also, the benefit of a Diffie-Hellman exchange is that each side contributes to the resulting key, no one imposes the key on the other. For many applications this will be an important issue, for others not quite so much.

Crypto Concepts

Using the tools described above, it's possible to build a very complicated and very extensible system for network security. IPSec is an example. IPSec uses symmetric ciphers in CBC mode for encryption and HMACs for bulk data authentication. The Internet Key Exchange is basically an authenticated Diffie-Hellman exchange. One method of authentication is digital signatures, another involves HMACing a shared secret, a third involves public key encryption to authenticate a peer.

There are certain concepts that are important to IPSec that are not necessarily cryptographic tools.

Perfect Forward Secrecy

Symmetric keys have a much shorter lifetime than asymmetric. This is due to the complexity of the algorithms. Asymmetric algorithms are based on one-way functions, symmetric algorithms are not. While both are in the same class of complexity, asymmetric algorithms are necessarily the most difficult to solve of that class. They *may* be as difficult to solve as symmetric algorithms (it's the complexity theorists debate of whether *NP* is equal to *NP-complete*) but are believed to be more difficult. Until someone proves that these two types of algorithms are of equal complexity we continue to believe that asymmetric algorithms are more complex than symmetric ones. This is a long way of explaining that certain keys have to be thrown away, and never used again, much sooner than other keys.

When a Diffie-Hellman exchange is used to generate a symmetric key (the kind of key that must be changed more frequently), both parties contribute to the result. The key is ephemeral. If that key is thrown away and replaced by a new key, which is the result of another Diffie-Hellman exchange, the two keys will have no relationship to each other. If an attacker broke a single symmetric key, he would have access to all data that was protected by that key but not to data protected by any other key. In other words, the system that uses such ephemeral, single-use, keys has *perfect forward secrecy.*

A system would not have perfect forward secrecy if there was a single secret from which all symmetric keys were derived. In that case, breaking the root key could give an attacker all keys derived from that root and therefore all data protected by all those keys.

The important issue to keep in mind regarding perfect forward secrecy is that it is not enough to just use a different key, the keys must be unique.

Perfect forward secrecy is important for some applications but not for all. There is a definite overhead associated with doing a Diffie-Hellman exchange at each rekey interval. If the data requires such security it is an appropriate price to pay, but if it doesn't, it could be excessive. So, perfect forward secrecy may not be necessary every single time. The IPSec standard key exchange, IKE, therefore has an option for perfect forward secrecy. If the parties desire it, it is possible, but not necessary.

Denial of Service

Cryptography is not free. Doing modular exponentiation or computing the product of two very large prime numbers, even decrypting and verifying the integrity of individual packets, takes both wall clock time and CPU time. If it was possible to force a computer to do unnecessary work while trying to achieve security, it might be possible to shut down that computer. Such an attack is called a *denial of service attack*.

Denial of service attacks can be launched against cryptographic systems if the system can be induced to do unnecessary work or allocate memory unnecessarily. A denial of service attack is when the attacker can cause the attackee to do more work in response to the attack than is necessary to launch the attack.

An example of such an attack would be if Alice was willing to do a Diffie-Hellman exchange and Mallory sent thousands of bogus Diffie-Hellman public values to her, all with fake return addresses. Alice could be forced to do her part for these fake exchanges. That could be quite a bit of work! It would be almost no work for Mallory, though, because it's computationally effortless to generate a string of random bits that look like a Diffie-Hellman public value. It's much more work to actually exponentiate and generate a real one.

Another denial of service attack can be launched if Alice and Bob share symmetric keys which they use to encrypt and authenticate individual IP packets. Mallory could send thousands of packets to Bob that look like they came from Alice. Since Mallory doesn't share the key the packets would be bogus, but the only way Bob could find that out is to do the work of decrypting and verifying the integrity of the packet! It's much cheaper to generate bogus packets than it is to detect that they're bogus.

Thankfully, IPSec and IKE are constructed with partial defenses against denial of service attacks. These defenses do not defeat all denial of service attacks, but merely increase the cost and complexity to launch them.

More Information

This chapter provides a brief overview of some cryptographic concepts that will be expanded on later in this book. Cryptography is a complex art, though, and it cannot be adequately explained in a short chapter like this. There are many good books that give a solid background in cryptography that you're strongly encouraged to read. A good place to start is *Cryptography and Data Security* by Dorothy Denning, and *Applied Cryptography* by Bruce Schneier.

There are important and fascinating protocols and problems that were not covered here. For instance, the zero knowledge proof: where one party proves to another that she knows some information without actually divulging the information. Another one-way function that was not discussed is the knapsack problem. Like the discrete logarithm problem, the knapsack problem can be used to construct public key cryptosystems. Other, more complicated, key exchanges also exist, like the Encrypted Key Exchange (EKE). There are even attacks against the cryptographic tools that IPSec uses, like the Birthday Attacks against hash functions. This attack takes its name from the observation that if you are in a room with only 182 other people, the chances are even that one of those persons has the same birthday as you. If there is a room of only 23 people, the chances are even that there are two people in the room that share the same birthday. This in spite of the fact that there are 365 (sometimes 366) days in the year! The birthday paradox affects hashing algorithms because it illustrates the statistical probability of finding two random inputs that will hash to the same digest—i.e., in finding a collision. If the digest from a hash algorithm is *n* bits in length, finding two distinct messages that hash to the same digest would take $O(2^{n/2})$ operations.

Cryptography is probably as old as speech but it continually evolves to solve new, interesting, and critically important problems of today and tomorrow.