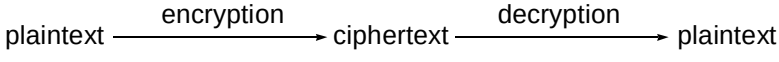
2 INTRODUCTION TO CRYPTOGRAPHY

2.1 WHAT IS CRYPTOGRAPHY?

The word *cryptography* comes from the Greek words κρυπτο (*hidden* or *secret*) and γραφη (*writing*). Oddly enough, cryptography is the art of secret writing. More generally, people think of cryptography as the art of mangling information into apparent unintelligibility in a manner allowing a secret method of unmangling. The basic service provided by cryptography is the ability to send information between participants in a way that prevents others from reading it. In this book we will concentrate on the kind of cryptography that is based on representing information as numbers and mathematically manipulating those numbers. This kind of cryptography can provide other services, such as

- integrity checking—reassuring the recipient of a message that the message has not been altered since it was generated by a legitimate source
- authentication—verifying someone's (or something's) identity

But back to the traditional use of cryptography. A message in its original form is known as **plaintext** or **cleartext**. The mangled information is known as **ciphertext**. The process for producing ciphertext from plaintext is known as **encryption**. The reverse of encryption is called **decryption**.



While cryptographers invent clever secret codes, cryptanalysts attempt to break these codes. These two disciplines constantly try to keep ahead of each other. Ultimately, the success of the cryptographers rests on the

Fundamental Tenet of Cryptography

If lots of smart people have failed to solve a problem,
then it probably won't be solved (soon).

Cryptographic systems tend to involve both an algorithm and a secret value. The secret value is known as the **key**. The reason for having a key in addition to an algorithm is that it is difficult to keep devising new algorithms that will allow reversible scrambling of information, and it is difficult to quickly explain a newly devised algorithm to the person with whom you'd like to start communicating securely. With a good cryptographic scheme it is perfectly OK to have everyone, including the bad guys (and the cryptanalysts) know the algorithm because knowledge of the algorithm without the key does not help unmangle the information.

The concept of a key is analogous to the combination for a combination lock. Although the concept of a combination lock is well known (you dial in the secret numbers in the correct sequence and the lock opens), you can't open a combination lock easily without knowing the combination.

2.1.1 Computational Difficulty

It is important for cryptographic algorithms to be reasonably efficient for the good guys to compute. The good guys are the ones with knowledge of the keys.[1] Cryptographic algorithms are not impossible to break without the key. A bad guy can simply try all possible keys until one works. The security of a cryptographic scheme depends on how much work it is for the bad guy to break it. If the best possible scheme will take 10 million years to break using all of the computers in the world, then it can be considered reasonably secure.

Going back to the combination lock example, a typical combination might consist of three numbers, each a number between 1 and 40. Let's say it takes 10 seconds to dial in a combination. That's reasonably convenient for the good guy. How much work is it for the bad guy? There are 40^3 possible combinations, which is 64000. At 10 seconds per try, it would take a week to try all combinations, though on average it would only take half that long (even though the right number is always the last one you try!).

Often a scheme can be made more secure by making the key longer. In the combination lock analogy, making the key longer would consist of requiring four numbers to be dialed in. This would make a little more work for the good guy. It might now take 13 seconds to dial in the combination. But the bad guy has 40 times as many combinations to try, at 13 seconds each, so it would take a year to try all combinations. (And if it took that long, he might want to stop to eat or sleep).

1. We're using the terms *good guys* for the cryptographers, and *bad guys* for the cryptanalysts. This is a convenient shorthand and not a moral judgment—in any given situation, which side you consider *good* or *bad* depends on your point of view.

With cryptography, computers can be used to exhaustively try keys. Computers are a lot faster than people, and they don't get tired, so thousands or millions of keys can be tried per second. Also, lots of keys can be tried in parallel if you have multiple computers, so time can be saved by spending money on more computers.

Sometimes a cryptographic algorithm has a variable-length key. It can be made more secure by increasing the length of the key. Increasing the length of the key by one bit makes the good guy's job just a little bit harder, but makes the bad guy's job up to twice as hard (because the number of possible keys doubles). Some cryptographic algorithms have a fixed-length key, but a similar algorithm with a longer key can be devised if necessary. If computers get 1000 times faster, so that the bad guy's job becomes reasonably practical, making the key 10 bits longer will make the bad guy's job as hard as it was before the advance in computer speed. However, it will be much easier for the good guys (because their computer speed increase far outweighs the increment in key length). So the faster computers get, the better life gets for the good guys.

Keep in mind that breaking the cryptographic scheme is often only one way of getting what you want. For instance, a bolt cutter works no matter how many digits are in the combination.

> *You can get further with a kind word and a gun than you can with a kind word alone.* —Willy Sutton, bank robber

2.1.2 To Publish or Not to Publish

Some people believe that keeping a cryptographic algorithm as secret as possible will enhance its security. Others argue that publishing the algorithm, so that it is widely known, will enhance its security. On the one hand, it would seem that keeping the algorithm secret must be more secure—it makes for more work for the cryptanalyst to try to figure out what the algorithm is.

The argument for publishing the algorithm is that the bad guys will probably find out about it eventually anyway, so it's better to tell a lot of nonmalicious people about the algorithm so that in case there are weaknesses, a good guy will discover them rather than a bad guy. A good guy who discovers a weakness will warn people that the system has a weakness. Publication provides an enormous amount of free consulting from the academic community as cryptanalysts look for weaknesses so they can publish papers about them. A bad guy who discovers a weakness will exploit it for doing bad-guy things like embezzling money or stealing trade secrets.

It is difficult to keep the algorithm secret because if an algorithm is to be widely used, it is highly likely that determined attackers will manage to learn the algorithm by reverse engineering whatever implementation is distributed, or just because the more people who know something the more likely it is for the information to leak to the wrong places. In the past, "good" cryptosystems were not economically feasible, so keeping the algorithms secret was needed extra protection. We believe (we hope?) today's algorithms are sufficiently secure that this is not necessary.

Common practice today is for most commercial cryptosystems to be published and for military cryptosystems to be kept secret. If a commercial algorithm is unpublished today, it's probably for trade secret reasons or because this makes it easier to get export approval rather than to enhance its security. We suspect the military ciphers are unpublished mainly to keep good cryptographic methods out of the hands of the enemy rather than to keep them from cryptanalyzing our codes.

2.1.3 Secret Codes

We use the terms *secret code* and *cipher* interchangeably to mean any method of encrypting data. Some people draw a subtle distinction between these terms that we don't find useful.

The earliest documented cipher is attributed to Julius Caesar. The way the **Caesar cipher** would work if the message were in English is as follows. Substitute for each letter of the message, the letter which is 3 letters later in the alphabet (and wrap around to A from Z). Thus an A would become a D, and so forth. For instance, DOZEN would become GRCHQ. Once you figure out what's going on, it is very easy to read messages encrypted this way (unless, of course, the original message was in Greek).

A slight enhancement to the Caesar cipher was distributed as a premium with Ovaltine in the 1940s as *Captain Midnight Secret Decoder rings*. (Were this done today, Ovaltine would probably be in violation of export controls for distributing cryptographic hardware!) The variant is to pick a secret number n between 1 and 25, instead of always using 3. Substitute for each letter of the message, the letter which is n higher (and wrap around to A from Z of course). Thus if the secret number was 1, an A would become a B, and so forth. For instance HAL would become IBM. If the secret number was 25, then IBM would become HAL. Regardless of the value of n, since there are only 26 possible ns to try, it is still very easy to break this cipher if you know it's being used and you can recognize a message once it's decrypted.

The next type of cryptographic system developed is known as a **monoalphabetic cipher**, which consists of an arbitrary mapping of one letter to another letter. There are 26! possible pairings of letters, which is approximately 4×10^{26}. [Remember, n!, which reads "n factorial", means $n(n-1)(n-2)\cdots1$.] This might seem secure, because to try all possibilities, if it took 1 microsecond to try each one, would take about 10 trillion years. However, by statistical analysis of language (knowing that certain letters and letter combinations are more common than others), it turns out to be fairly easy to break. For instance, many daily newspapers have a daily cryptogram, which is a monoalphabetic cipher, and can be broken by people who enjoy that sort of thing during their subway ride to work. An example is

Cf lqr'xs xsnyctm n eqxxqgsy iqul qf wdcp eqqh, erl lqrx qgt iqul!

Computers have made much more complex cryptographic schemes both necessary and possible. Necessary because computers can try keys at a rate that would exhaust an army of clerks; and possible because computers can execute the complex algorithms quickly and without errors.

2.2 BREAKING AN ENCRYPTION SCHEME

What do we mean when we speak of a bad guy Fred *breaking* an encryption scheme? The three basic attacks are known as **ciphertext only**, **known plaintext**, and **chosen plaintext**.

2.2.1 Ciphertext Only

In a ciphertext only attack, Fred has seen (and presumably stored) some ciphertext that he can analyze at leisure. Typically it is not difficult for a bad guy to obtain ciphertext. (If a bad guy can't access the encrypted data, then there would have been no need to encrypt the data in the first place!)

How can Fred figure out the plaintext if all he can see is the ciphertext? One possible strategy is to search through all the keys. Fred tries the decrypt operation with each key in turn. It is essential for this attack that Fred be able to recognize when he has succeeded. For instance, if the message was English text, then it is highly unlikely that a decryption operation with an incorrect key could produce something that looked like intelligible text. Because it is important for Fred to be able to differentiate plaintext from gibberish, this attack is sometimes known as a **recognizable plaintext** attack.

It is also essential that Fred have enough ciphertext. For instance, using the example of a monoalphabetic cipher, if the only ciphertext available to Fred were XYZ, then there is not enough information. There are many possible letter substitutions that would lead to a legal three-letter English word. There is no way for Fred to know whether the plaintext corresponding to XYZ is THE or CAT or HAT. As a matter of fact, in the following sentence, any of the words could be the plaintext for XYZ:

The hot cat was sad but you may now sit and use her big red pen.

[Don't worry—we've found a lovely sanatorium for the coauthor who wrote that.

—the other coauthors]

Often it isn't necessary to search through a lot of keys. For instance, the authentication scheme Kerberos (see §10.4 *Logging Into the Network*) assigns to user Alice a DES key derived from Alice's password according to a straightforward, published algorithm. If Alice chooses her

password unwisely (say a word in the dictionary), then Fred does not need to search through all 2^{56} possible DES keys—instead he only needs to try the derived keys of the 10000 or so common English words.

A cryptographic algorithm has to be secure against a ciphertext only attack because of the accessibility of the ciphertext to cryptanalysts. But in many cases cryptanalysts can obtain additional information, so it is important to design cryptographic systems to withstand the next two attacks as well.

2.2.2 Known Plaintext

Sometimes life is easier for the attacker. Suppose Fred has somehow obtained some ⟨plaintext, ciphertext⟩ pairs. How might he have obtained these? One possibility is that secret data might not remain secret forever. For instance, the data might consist of specifying the next city to be attacked. Once the attack occurs, the plaintext to the previous day's ciphertext is now known.

With a monoalphabetic cipher, a small amount of known plaintext would be a bonanza. From it, the attacker would learn the mappings of a substantial fraction of the most common letters (every letter that was used in the plaintext Fred obtained). Some cryptographic schemes might be good enough to be secure against ciphertext only attacks but not good enough against known plaintext attacks. In these cases, it becomes important to design the systems that use such a cryptographic algorithm to minimize the possibility that a bad guy will ever be able to obtain ⟨plaintext, ciphertext⟩ pairs.

2.2.3 Chosen Plaintext

On rare occasions, life may be easier still for the attacker. In a "chosen plaintext" attack, Fred can choose any plaintext he wants, and get the system to tell him what the corresponding ciphertext is. How could such a thing be possible?

Suppose the telegraph company offered a service in which they encrypt and transmit messages for you. Suppose Fred had eavesdropped on Alice's encrypted message. Now he'd like to break the telegraph company's encryption scheme so that he can decrypt Alice's message.

He can obtain the corresponding ciphertext to any message he chooses by paying the telegraph company to send the message for him, encrypted. For instance, if Fred knew they were using a monoalphabetic cipher, he might send the message

The quick brown fox jumps over the lazy dog.

knowing that he would thereby get all the letters of the alphabet encrypted and then be able to decrypt with certainty any encrypted message.

It is possible that a cryptosystem secure against ciphertext only and known plaintext attacks might still be susceptible to chosen plaintext attacks. For instance, if Fred knows that Alice's message is either Surrender or Fight on, then no matter how wonderful an encryption scheme the telegraph company is using, all he has to do is send the two messages and see which one looks like the encrypted data he saw when Alice's message was transmitted.

A cryptosystem should resist all three sorts of attacks. That way its users don't need to worry about whether there are any opportunities for attackers to know or choose plaintext. Like wearing both a belt and suspenders, many systems that use cryptographic algorithms will also go out of their way to prevent any chance of chosen plaintext attacks.

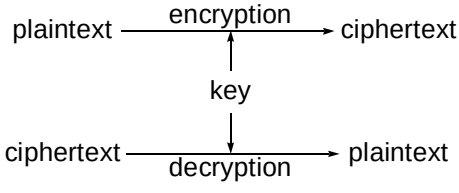
2.3 TYPES OF CRYPTOGRAPHIC FUNCTIONS

There are three kinds of cryptographic functions: hash functions, secret key functions, and public key functions. We will describe what each kind is, and what it is useful for. Public key cryptography involves the use of two keys. Secret key cryptography involves the use of one key. Hash functions involve the use of zero keys! Try to imagine what that could possibly mean, and what use it could possibly have—an algorithm everyone knows with no secret key, and yet it has uses in security.

Since secret key cryptography is probably the most intuitive, we'll describe that first.

2.4 SECRET KEY CRYPTOGRAPHY

Secret key cryptography involves the use of a single key. Given a message (called plaintext) and the key, encryption produces unintelligible data (called an IRS Publication—no! no! that was just a finger slip, we meant to say "ciphertext"), which is about the same length as the plaintext was. Decryption is the reverse of encryption, and uses the same key as encryption.



Secret key cryptography is sometimes referred to as **conventional cryptography** or **symmetric cryptography**. The Captain Midnight code and the monoalphabetic cipher are both examples of secret key algorithms, though both are easy to break. In this chapter we describe the functionality of cryptographic algorithms, but not the details of particular algorithms. In Chapter 3 *Secret Key Cryptography* we describe the details of two secret key cryptographic algorithms (DES and IDEA) in current use.

2.4.1 Security Uses of Secret Key Cryptography

The next few sections describe the types of things one might do with secret key cryptography.

2.4.2 Transmitting Over an Insecure Channel

It is often impossible to prevent eavesdropping when transmitting information. For instance, a telephone conversation can be tapped, a letter can be intercepted, and a message transmitted on a LAN can be received by unauthorized stations.

If you and I agree on a shared secret (a key), then by using secret key cryptography we can send messages to one another on a medium that can be tapped, without worrying about eavesdroppers. All we need to do is for the sender to encrypt the messages and the receiver to decrypt them using the shared secret. An eavesdropper will only see unintelligible data.

This is the classic use of cryptography.

2.4.3 Secure Storage on Insecure Media

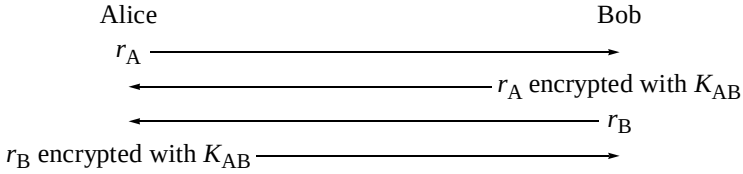
If I have information I want to preserve but which I want to assure no one else can look at, I have to be able to store the media where I am sure no one can get it. Between clever thieves and court orders, there are very few places that are truly secure, and none of these is convenient. If I invent a key and encrypt the information using the key, I can store it anywhere and it is safe so long as I can remember the key. Of course, forgetting the key makes the data irrevocably lost, so this must be used with great care.

2.4.4 Authentication

In spy movies, when two agents who don't know each other must rendezvous, they are each given a password or pass phrase that they can use to recognize one another. This has the problem

that anyone overhearing their conversation or initiating one falsely can gain information useful for replaying later and impersonating the person to whom they are talking.

The term **strong authentication** means that someone can prove knowledge of a secret without revealing it. Strong authentication is possible with cryptography. Strong authentication is particularly useful when two computers are trying to communicate over an insecure network (since few people can execute cryptographic algorithms in their heads). Suppose Alice and Bob share a key K_{AB} and they want to verify they are speaking to each other. They each pick a random number, which is known as a **challenge**. Alice picks r_A. Bob picks r_B. The value x encrypted with the key K_{AB} is known as the **response** to the challenge x.



If someone, say Fred, were impersonating Alice, he could get Bob to encrypt a value for him (though Fred wouldn't be able to tell if the person he was talking to was *really* Bob), but this information would not be useful later in impersonating Bob to the real Alice because the real Alice would pick a different challenge. If Alice and Bob complete this exchange, they have each proven to the other than they know K_{AB} without revealing it to an impostor or an eavesdropper. Note that in this particular protocol, there is the opportunity for Fred to obtain some ⟨chosen plaintext, ciphertext⟩ pairs, since he can claim to be Bob and ask Alice to encrypt a challenge for him. For this reason, it is essential that challenges be chosen from a large enough space, say 2^{64} values, so that there is no significant chance of using the same one twice.

That is the general idea of a cryptographic authentication algorithm, though this particular algorithm has a subtle problem that would prevent it from being useful in most computer-to-computer cases. (We would have preferred not bringing that up, but felt we needed to say that so as not to alarm people who already know this stuff and who would realize the protocol was not secure. Details on fixing this authentication protocol are discussed in Chapter 9 *Security Handshake Pitfalls*.)

2.4.5 Integrity Check

A secret key scheme can be used to generate a fixed-length cryptographic checksum associated with a message. This is a rather nonintuitive use of secret key technology.

What is a checksum? An ordinary (noncryptographic) checksum protects against accidental corruption of a message. The original derivation of the term *checksum* comes from the operation of

breaking a message into fixed-length blocks (for instance, 32-bit words) and adding them up. The sum is sent along with the message. The receiver similarly breaks up the message, repeats the addition, and *checks the sum*. If the message had been garbled en route, the sum will not match the sum sent and the message is rejected, unless, of course, there were two or more errors in the transmission that canceled one another. It turns out this is not terribly unlikely, given that if flaky hardware turns a bit off somewhere, it is likely to turn a corresponding bit on somewhere else. To protect against such "regular" flaws in hardware, more complex checksums called CRCs were devised. But these still only protect against faulty hardware and not an intelligent attacker. Since CRC algorithms are published, an attacker who wanted to change a message could do so, compute the CRC on the new message, and send that along.

To provide protection against malicious changes to a message, a *secret* checksum algorithm is required, such that an attacker not knowing the algorithm can't compute the right checksum for the message to be accepted as authentic. As with encryption algorithms, it's better to have a common (known) algorithm and a secret key. This is what a cryptographic checksum does. Given a key and a message, the algorithm produces a fixed-length message integrity code (MIC) that can be sent with the message.

If anyone were to modify the message, and they didn't know the key, they would have to guess a MIC and the chance of getting it right depends on the length. A typical MIC is at least 48 bits long, so the chance of getting away with a forged message is only one in 280 trillion (or about the chance of going to Las Vegas with a dime and letting it ride on red at the roulette table until you have enough to pay off the U.S. national debt).

Such message integrity codes have been in use to protect the integrity of large interbank electronic funds transfers for quite some time. The messages are not kept secret from an eavesdropper, but their integrity is ensured.

2.5 PUBLIC KEY CRYPTOGRAPHY

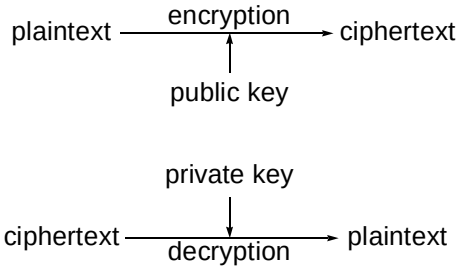
Public key cryptography is sometimes also referred to as **asymmetric cryptography**.

Public key cryptography is a relatively new field, invented in 1975 [DIFF76b] (at least that's the first published record—it is rumored that NSA or similar organizations may have discovered this technology earlier). Unlike secret key cryptography, keys are not shared. Instead, each individual has two keys: a private key that need not be revealed to anyone, and a public key that is preferably known to the entire world.

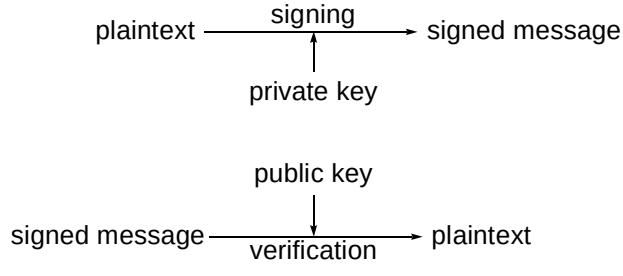
Note that we call the private key a *private key* and not a *secret key.* This convention is an attempt to make it clear in any context whether public key cryptography or secret key cryptography is being used. There are people in this world whose sole purpose in life is to try to confuse people.

They will use the term *secret key* for the private key in public key cryptography, or use the term *private key* for the secret key in secret key technology. One of the most important contributions we can make to the field is to convince people to feel strongly about using the terminology correctly—the term *secret key* refers only to the single secret number used in secret key cryptography. The term *private key* MUST be used when referring to the key in public key cryptography that must not be made public. (Yes, when we speak we sometimes accidentally say the wrong thing, but at least we feel guilty about it.)

There is something unfortunate about the terminology *public* and *private*. It is that both words begin with *p*. We will sometimes want a single letter to refer to one of the keys. The letter *p* won't do. We will use the letter *e* to refer to the public key, since the public key is used when encrypting a message. We'll use the letter *d* to refer to the private key, because the private key is used to decrypt a message. Encryption and decryption are two mathematical functions that are inverses of each other.



There is an additional thing one can do with public key technology, which is to generate a **digital signature** on a message. A digital signature is a number associated with a message, like a



checksum or the MIC (message integrity code) described in §2.4.5 *Integrity Check*. However, unlike a checksum, which can be generated by anyone, a digital signature can only be generated by someone knowing the private key. A public key signature differs from a secret key MIC because verification of a MIC requires knowledge of the same secret as was used to create it. Therefore any-

one who can verify a MIC can also generate one, and so be able to substitute a different message and corresponding MIC. In contrast, verification of the signature only requires knowledge of the public key. So Alice can sign a message by generating a signature only she can generate, and other people can verify that it is Alice's signature, but cannot forge her signature. This is called a signature because it shares with handwritten signatures the property that it is possible to be able to recognize a signature as authentic without being able to forge it.

2.5.1 Security Uses of Public Key Cryptography

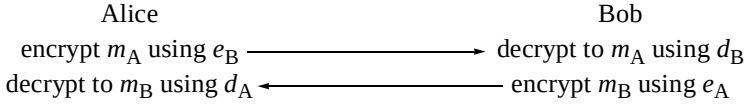
Public key cryptography can do anything secret key cryptography can do, but the known public key cryptographic algorithms are orders of magnitude slower than the best known secret key cryptographic algorithms and so are usually only used for things secret key cryptography can't do. Public key cryptography is very useful because network security based on public key technology tends to be more secure and more easily configurable. Often it is mixed with secret key technology. For example, public key cryptography might be used in the beginning of communication for authentication and to establish a temporary shared secret key, then the secret key is used to encrypt the remainder of the conversation using secret key technology.

For instance, suppose Alice wants to talk to Bob. She uses his public key to encrypt a secret key, then uses that secret key to encrypt whatever else she wants to send him. Only Bob can decrypt the secret key. He can then communicate using that secret key with whoever sent that message. Notice that given this protocol, Bob does not know that it was Alice who sent the message. This could be fixed by having Alice digitally sign the encrypted secret key using her private key.

Now we'll describe the types of things one might do with public key cryptography.

2.5.2 Transmitting Over an Insecure Channel

Suppose Alice's ⟨public key, private key⟩ pair is $\langle e_A, d_A \rangle$. Suppose Bob's key pair is $\langle e_B, d_B \rangle$. Assume Alice knows Bob's public key, and Bob knows Alice's public key. Actually, accurately learning other people's public keys is one of the biggest challenges in using public key cryptography and will be discussed in detail in §7.7.2 *Certification Authorities (CAs)*. But for now, don't worry about it.



2.5.3 Secure Storage on Insecure Media

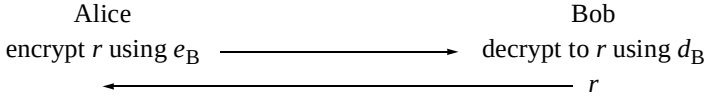
This is really the same as what one would do with secret key cryptography. You'd encrypt the data with your public key. Then nobody can decrypt it except you, since decryption will require the use of the private key. It has the advantage over encryption with secret key technology that you don't have to risk giving your private key to the machine that is going to encrypt the data for you. As with secret key technology, if you lose your private key, the data is irretrievably lost. If you are worried about that, you can encrypt an additional copy of the data under the public key of someone you trust, like your lawyer.

2.5.4 Authentication

Authentication is an area in which public key technology potentially gives a real benefit. With secret key cryptography, if Alice and Bob want to communicate, they have to share a secret. If Bob wants to be able to prove his identity to lots of entities, then with secret key technology he will need to remember lots of secret keys, one for each entity to which he would like to prove his identity. Possibly he could use the same shared secret with Alice as with Carol, but that has the disadvantage that then Carol and Alice could impersonate Bob to each other.

Public key technology is much more convenient. Bob only needs to remember a single secret, his own private key. It is true that if Bob wants to be able to verify the identity of thousands of entities, then he will need to know thousands of public keys, but in general the entities verifying identities are computers which don't mind remembering thousands of things, whereas the entities proving their identities are often humans, which do mind remembering things.

Here's an example of how Alice can use public key cryptography for verifying Bob's identity assuming Alice knows Bob's public key. Alice chooses a random number r, encrypts it using Bob's public key e_B, and sends the result to Bob. Bob proves he knows d_B by decrypting the message and sending r back to Alice.



Another advantage of public key authentication is that Alice does not need to keep any secret information. For instance, Alice might be a computer system in which backup tapes are unencrypted and easily stolen. With secret key based authentication, if Carol stole a backup tape and read the key that Alice shares with Bob, she could then trick Bob into thinking she was Alice. In contrast, with public key based authentication, the only information on Alice's backup tapes is public key information, and that cannot be used to impersonate Bob.

In large-scale systems, like computer networks with thousands of users and services, authentication is usually done with trusted intermediaries. As we'll see in §7.7 *Trusted Intermediaries*, public key based authentication using intermediaries has several important advantages over secret key based authentication.

2.5.5 Digital Signatures

Forged in USA
engraved on a screwdriver claiming to be of brand *Craftsman*

It is often useful to prove that a message was generated by a particular individual, especially if the individual is not necessarily around to be asked about authorship of the message. This is easy with public key technology. Bob's signature for a message *m* can only be generated by someone with knowledge of Bob's private key. And the signature depends on the contents of *m*. If *m* is modified in any way, the signature no longer matches. So digital signatures provide two important functions. They prove who generated the information, and they prove that the information has not been modified in any way by anyone since the message and matching signature were generated.

An important example of a use of a signature is in electronic mail to verify that a mail message really did come from the claimed source.

Digital signatures offer an important advantage over secret key based cryptographic checksums—**non-repudiation**. Suppose Bob sells widgets and Alice routinely buys them. Alice and Bob might agree that rather than placing orders through the mail with signed purchase orders, Alice will send electronic mail messages to order widgets. To protect against someone forging orders and causing Bob to manufacture more widgets than Alice actually needs, Alice will include a message integrity code on her messages. This could be either a secret key based MIC or a public key based signature. But suppose sometime after Alice places a big order, she changes her mind (the bottom fell out of the widget market). Since there's a big penalty for canceling an order, she doesn't fess up that she's canceling, but instead denies that she ever placed the order. Bob sues. Bob knows Alice really placed the order because it was cryptographically signed. But if it was signed with a secret key algorithm, he can't prove it to anyone! Since he knows the same secret key that Alice used to sign the order, he could have forged the signature on the message himself and he can't prove to the judge that he didn't! If it was a public key signature on the other hand, he can show the signed message to the judge and the judge can verify that it was signed with Alice's key. Alice can still claim of course that someone must have stolen and misused her key (it might even be true!), but the contract between Alice and Bob could reasonably hold her responsible for damages caused by her inadequately protecting her key. Unlike secret key cryptography, where the keys are shared, you can always tell who's responsible for a signature generated with a private key.

Public key algorithms are discussed further in Chapter 5 *Public Key Algorithms*.

2.6 HASH ALGORITHMS

Hash algorithms are also known as **message digests** or **one-way transformations**.

A cryptographic hash function is a mathematical transformation that takes a message of arbitrary length (transformed into a string of bits) and computes from it a fixed-length (short) number.

We'll call the hash of a message m, $h(m)$. It has the following properties:

- For any message m, it is relatively easy to compute $h(m)$. This just means that in order to be practical it can't take a lot of processing time to compute the hash.
- Given $h(m)$, there is no way to find an m that hashes to $h(m)$ in a way that is substantially easier than going through all possible values of m and computing $h(m)$ for each one.
- Even though it's obvious that many different values of m will be transformed to the same value $h(m)$ (because there are many more possible values of m), it is *computationally infeasible* to find two values that hash to the same thing.

An example of the sort of function that might work is taking the message m, treating it as a number, adding some large constant, squaring it, and taking the middle n digits as the hash. You can see that while this would not be difficult to compute, it's not obvious how you could find a message that would produce a particular hash, or how one might find two messages with the same hash. It turns out this is not a particularly good message digest function—we'll give examples of secure message digest functions in Chapter 4 *Hashes and Message Digests*. But the basic idea of a message digest function is that the input is mangled so badly the process cannot be reversed.

2.6.1 Password Hashing

When a user types a password, the system has to be able to determine whether the user got it right. If the system stores the passwords unencrypted, then anyone with access to the system storage or backup tapes can steal the passwords. Luckily, it is not necessary for the system to know a password in order to verify its correctness. (A proper password is like pornography. You can't tell what it is, but you know it when you see it.)

Instead of storing the password, the system can store a hash of the password. When a password is supplied, it computes the password's hash and compares it with the stored value. If they match, the password is deemed correct. If the hashed password file is obtained by an attacker, it is

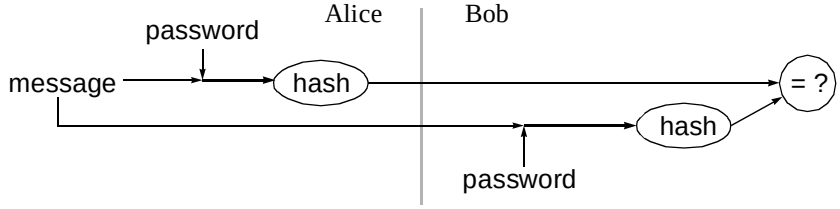
not immediately useful because the passwords can't be derived from the hashes. Historically, some systems made the password file publicly readable, an expression of confidence in the security of the hash. Even if there are no cryptographic flaws in the hash, it is possible to guess passwords and hash them to see if they match. If a user is careless and chooses a password that is guessable (say, a word that would appear in a 50000-word dictionary or book of common names), an exhaustive search would "crack" the password even if the encryption were sound. For this reason, many systems hide the hashed password list (and those that don't should).

2.6.2 Message Integrity

Cryptographic hash functions can be used to generate a MIC to protect the integrity of messages transmitted over insecure media in much the same way as secret key cryptography.

If we merely sent the message and used the hash of the message as a MIC, this would not be secure, since the hash function is well-known. The bad guy can modify the message and compute a new hash for the new message, and transmit that.

However, if Alice and Bob have agreed on a password, Alice can use a hash to generate a MIC for a message to Bob by taking the message, concatenating the password, and computing the hash of *message*|*password*. Alice then sends the hash and the message (without the password) to Bob. Bob concatenates the password to the received message and computes the hash of the result. If that matches the received hash, Bob can have confidence the message was sent by someone knowing the password. [Note: there are some cryptographic subtleties to making this actually secure; see §4.2.2 *Computing a MIC with a Hash*].



2.6.3 Message Fingerprint

If you want to know whether some large data structure (e.g. a program) has been modified from one day to the next, you could keep a copy of the data on some tamper-proof backing store and periodically compare it to the active version. With a hash function, you can save storage: you simply save the message digest of the data on the tamper-proof backing store (which because the

hash is small could be a piece of paper in a filing cabinet). If the message digest hasn't changed, you can be confident none of the data has.

A note to would-be users—if it hasn't already occurred to you, it has occurred to the bad guys—the program that computes the hash must also be independently protected for this to be secure. Otherwise the bad guys can change the file but also change the hashing program to report the checksum as though the file were unchanged!

2.6.4 Downline Load Security

It is common practice to have special-purpose devices connected to a network, like routers or printers, that do not have the nonvolatile memory to store the programs they normally run. Instead, they keep a bootstrap program smart enough to get a program from the network and run it. This scheme is called **downline load**.

Suppose you want to downline load a program and make sure it hasn't been corrupted (whether intentionally or not). If you know the proper hash of the program, you can compute the hash of the loaded program and make sure it has the proper value before running the program.

2.6.5 Digital Signature Efficiency

The best-known public key algorithms are sufficiently processor-intensive that it is desirable to compute a message digest of the message and sign that, rather than to sign the message directly. The message digest algorithms are much less processor-intensive, and the message digest is much shorter than the message.

2.7 HOMEWORK

1. Random J. Programmer discovers a much faster method of generating a 64-bit signature for a message using secret key technology. The idea is to simply encrypt the first 64 bits of the message, and use that as the signature. What's wrong with this idea?
2. What's wrong with adding up the words of a message and using the result as a hash of the message?
3. Random J. Protocol-Designer has been told to design a scheme to prevent messages from being modified by an intruder. Random J. decides to append to each message a hash of that

message. Why doesn't this solve the problem? (We know of a protocol that uses this technique in an attempt to gain security.)