*Information Security: Principles and Practice, Second Edition*
by Mark Stamp


# Chapter 4

# Public Key Crypto

*You should not live one way in private, another in public.*
— Publilius Syrus

*Three may keep a secret, if two of them are dead.*
— Ben Franklin

## 4.1 Introduction

In this chapter, we delve into the remarkable subject of public key cryptography. Public key crypto is sometimes know as asymmetric cryptography, or two key cryptography, or even non-secret key cryptography, but we'll stick with public key cryptography.

In symmetric key cryptography, the same key is used to both encrypt and decrypt the data. In public key cryptography, one key is used to encrypt and a different key is used to decrypt and as a result, the encryption key can be made public. This eliminates one of the most vexing problems of symmetric key crypto, namely, how to securely distribute the symmetric key. Of course, there is no free lunch, so public key crypto has its own issues when it comes to dealing with keys (see the section on public key infrastructure, below). Nevertheless, public key crypto is a big "win" in many real-world applications.

Actually, public key cryptography is usually defined more broadly than the two-key encryption and decryption description given in the previous paragraph. Any system that has cryptographic application and involves some crucial information being made public is likely to be considered a public key cryptosystem. For example, one popular public key system discussed in this chapter can only be used to establish a shared symmetric, not to encrypt or decrypt anything.

Public key crypto is a relative newcomer, having been invented by cryptographers working for GCHQ (the British equivalent of NSA) in the late 1960s and early 1970s and, independently, by academic researchers shortly thereafter [191]. The government cryptographers clearly did not grasp the full potential of their discovery, and it lay dormant until the academicians pushed it into the limelight. The ultimate effect has been nothing short of a revolution in cryptography. It is amazing that public key crypto is such a recent discovery, given that humans have been using symmetric crypto for thousands of years.

In this chapter, we'll examine most of the most important and widely used public key cryptosystems. Actually, relatively few public key systems are known, and fewer still are widely used. In contrast, there exists a vast number of symmetric ciphers, and a fairly significant number of these get used in practice. Each public key system is based on a very special mathematical structure, making it extraordinarily difficult to develop new systems.[1]

A public key cryptosystem is based on a trap door one-way function. "One-way" means that the function is easy to compute in one direction but hard to compute (i.e., computationally infeasible) in the other. The "trap door" feature ensures that an attacker cannot use the public information to recover the private information. Factoring is a relevant example—it is a one-way function since it's relatively easy to, say, generate two prime numbers $p$ and $q$ and compute their product $N = pq$, but given a sufficiently large value of $N$, it is difficult to find the factors $p$ and $q$. We can also build a trap door based on factoring, but we defer a discussion of that to later in this chapter (see the section on RSA).

Recall that in symmetric key crypto, the plaintext is $P$ and the ciphertext is $C$. But in public key crypto, tradition has it that we encrypt a message $M$, although, strangely, the result is still ciphertext $C$. Below, we follow this tradition.

To do public key crypto, Bob must have a *key pair* consisting of a *public key* and a corresponding *private key*. Anyone can use Bob's public key to encrypt a message intended for Bob's eyes only, but only Bob can decrypt the message, since, by assumption only Bob has his private key.

Bob can also apply his *digital signature* to a message $M$ by "encrypting" it with his private key. Note that anybody can "decrypt" the message since this only requires Bob's public key, which is public. You might reasonably wonder what possible use this could be. In fact, it is one of the most useful features of public key crypto.

A digital signature is like a handwritten signature—only more so. Bob is the only one who can digitally sign as Bob, since he is the only one with access to his private key. While in principle only Bob can write his handwritten

[1] Public key cryptosystems definitely do not grow on trees.

signature,[2] in practice only Bob can digitally sign as Bob. Anyone with access to Bob's public key can verify Bob's digital signature, which is much more practical than hiring a handwriting expert to verify Bob's non-digital signature.

The digital version of Bob's signature has some additional advantages over the handwritten version. For one thing, a digital signature is firmly tied to the document itself, whereas a handwritten signature can be photocopied onto another document. No photocopying attack is possible with a digital signature. Even more significant is the fact that, generally speaking, it's not feasible to forge a digital signature without the private key. In the non-digital world, a forgery of Bob's signature might only be detectable by a trained expert (if at all), while a digital signature forgery can be easily and automatically detected by anyone since verification only requires Bob's public key, and everyone has access to public keys.

Next, we'll discuss in detail several public key cryptosystems. The first one that we'll consider is the knapsack cryptosystem. This is appropriate since the knapsack was one of the first practical proposed public key systems. Although the knapsack that we'll present is known to be insecure, it's relatively easy to comprehend and nicely illustrates all of the important features of such a system. After the knapsack, we discuss the gold standard of public key crypto, namely, RSA. We'll then conclude our brief tour of public key systems with a look at the Diffie-Hellman key exchange, which is also widely used in practice.

We then discuss elliptic curve cryptography, or ECC. Note that ECC is not a cryptosystem *per se*, but instead it offers a different realm in which to do the math that arises in public key systems. The advantage of ECC is that it's more efficient (in both time and space) and so it's favored in resource-constrained environments such as wireless and handheld devices. In fact, all recent U.S. government public key standards are ECC-based.

Public key cryptography is inherently more mathematical than symmetric key. So now would be a good time to review the math topics found in the Appendix. In particular, a working knowledge of elementary modular arithmetic is assumed in this chapter.

## 4.2 Knapsack

In their seminal paper [90], Diffie and Hellman conjectured that public key cryptography was possible, but they "only" offered a key exchange algorithm, not a viable system for encryption and decryption. Shortly thereafter, the Merkle-Hellman knapsack cryptosystem was proposed by—believe it or not—Merkle and Hellman. We'll meet Hellman again later, but it is worth noting that Merkle was also one of the founders of public key cryptography. He wrote

[2]What happens in practice is a different story.

a groundbreaking paper [202] that foreshadowed public key cryptography. Merkle's paper was submitted for publication at about the same time as Diffie and Hellman's paper, although it appeared much later. For some reason, Merkle's contribution usually does not receive the attention it deserves.

The Merkle-Hellman knapsack cryptosystem is based on a problem[3] that is known to be NP-complete [119]. This seems to make it an ideal candidate for a secure public key cryptosystem.

The knapsack problem can be stated as follows. Given a set of $n$ weights labeled as

$$W_0, W_1, \ldots, W_{n-1}$$

and a desired sum $S$, find $a_0, a_1 \ldots, a_{n-1}$, where each $a_i \in \{0, 1\}$, so that

$$S = a_0 W_0 + a_1 W_1 + \cdots + a_{n-1} W_{n-1},$$

provided this is possible. For example, suppose the weights are

$$85, 13, 9, 7, 47, 27, 99, 86$$

and the desired sum is $S = 172$. Then a solution to the problem exists and is given by

$$a = (a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) = (11001100)$$

since $85 + 13 + 47 + 27 = 172$.

Although the general knapsack problem is known to be NP-complete, there is a special case that can be solved in linear time. A *superincreasing knapsack* is similar to the general knapsack except that, when the weights are arranged from least to greatest, each weight is greater than sum of all previous weights. For example,

$$3, 6, 11, 25, 46, 95, 200, 411 \tag{4.1}$$

is a superincreasing knapsack. Solving a superincreasing knapsack problem is easy. Suppose we are given the set of weights in equation (4.1) and the sum $S = 309$. To solve this, we simply start with the largest weight and work toward the smallest to recover the $a_i$ in linear time. Since $S < 411$, we have $a_7 = 0$. Then since $S > 200$, we must have $a_6 = 1$, since the sum of all remaining weights is less than 200. Then we compute $S = S - 200 = 109$ and this is our new target sum. Since $S > 95$, we have $a_5 = 1$ and we compute $S = 109 - 95 = 14$. Continuing in this manner, we find $a = 10100110$, which we can easily verify solves the problem since $3 + 11 + 95 + 200 = 309$.

[3]Ironically, the knapsack cryptosystem is not based on the knapsack problem. Instead it's based on a more restricted problem, known as subset sum. Nevertheless, the cryptosystem is universally known as the knapsack. Eschewing our usual pedantic nature, we'll refer to both the cryptosystem and the underlying problem as knapsacks.

Next, we outline the steps in the procedure used to construct a knapsack cryptosystem. The process begins with a superincreasing knapsack from which we generate a public and private key pair. After listing the steps, we'll illustrate the process with a specific example.

1. Generate a superincreasing knapsack.
2. Convert the superincreasing knapsack into a general knapsack.
3. The public key is the general knapsack.
4. The private key is the superincreasing knapsack together with the conversion factors.

Below, we'll see that it's easy to encrypt using the general knapsack and, with access to the private key, it's easy to decrypt. However, without the private key, it appears that Trudy must solve an NP-complete problem—the knapsack problem—to recover the plaintext from the ciphertext.

Now we'll present a specific example. For this example, we'll follow the numbering in the steps listed above.

1. For this example, we'll choose the superincreasing knapsack

$$(2, 3, 7, 14, 30, 57, 120, 251).$$

2. To convert the superincreasing knapsack into a general knapsack, we must choose a multiplier $m$ and a modulus $n$ so that $m$ and $n$ are relatively prime and $n$ is greater than the sum of all elements in the superincreasing knapsack. For this example, we select the multiplier $m = 41$ and the modulus $n = 491$. Then the general knapsack is computed from the superincreasing knapsack by modular multiplication:

$$\begin{aligned}
2m &= 2 \cdot 41 = 82 \bmod 491 \\
3m &= 3 \cdot 41 = 123 \bmod 491 \\
7m &= 7 \cdot 41 = 287 \bmod 491 \\
14m &= 14 \cdot 41 = 83 \bmod 491 \\
30m &= 30 \cdot 41 = 248 \bmod 491 \\
57m &= 57 \cdot 41 = 373 \bmod 491 \\
120m &= 120 \cdot 41 = 10 \bmod 491 \\
251m &= 251 \cdot 41 = 471 \bmod 491
\end{aligned}$$

The resulting knapsack is $(82, 123, 287, 83, 248, 373, 10, 471)$. Note that this knapsack does indeed appear to be a general knapsack.[4]

[4] Appearances can be deceiving...

3. The public key is the general knapsack,

$$\text{Public key: } (82, 123, 287, 83, 248, 373, 10, 471).$$

4. The private key is the superincreasing knapsack together with the multiplicative inverse of the conversion factor, i.e., $m^{-1} \bmod n$. For this example, we have

$$\text{Private key: } (2, 3, 7, 14, 30, 57, 120, 251) \text{ and } 41^{-1} \bmod 491 = 12.$$

Suppose Bob's public and private key pair are those given in step 3 and step 4, respectively. Suppose that Alice wants to encrypt the message (in binary) $M = 10010110$ for Bob. Then she uses the 1 bits in her message to select the elements of the general knapsack that are summed to give the ciphertext. In this case, Alice computes

$$C = 82 + 83 + 373 + 10 = 548.$$

To decrypt this ciphertext, Bob uses his private key to find

$$C \cdot m^{-1} \bmod n = 548 \cdot 12 \bmod 491 = 193.$$

Bob then solves the superincreasing knapsack for 193. Since Bob has the private key, this is an easy (linear time) problem from which Bob recovers the message in binary $M = 10010110$ or, in decimal, $M = 150$.

Note that in this example, we have

$$548 = 82 + 83 + 373 + 10$$

and it follows that

$$\begin{aligned} 548m^{-1} &= 82m^{-1} + 83m^{-1} + 373m^{-1} + 10m^{-1} \\ &= 2mm^{-1} + 14mm^{-1} + 57mm^{-1} + 120mm^{-1} \\ &= 2 + 14 + 57 + 120 \\ &= 193 \bmod 491. \end{aligned}$$

This example shows that multiplying by $m^{-1}$ transforms the ciphertext—which lives in the realm of the general knapsack—into the superincreasing realm, where it's easy for Bob to solve for the weights. Proving that the decryption formula works in general is equally straightforward.

Without the private key, attacker Trudy can break a message if she can find a subset of the elements of the public key that sum to the ciphertext value $C$. In the example above, Trudy must find a subset of the knapsack $(82, 123, 287, 83, 248, 373, 10, 471)$ that sums precisely to 548. This appears to be a general knapsack problem, which is known to be a very difficult problem.

The trapdoor in the knapsack cryptosystem occurs when we convert the superincreasing knapsack into the general knapsack using modular arithmetic, since the conversion factors are unavailable to an attacker. The one-way feature lies in the fact that it is easy to encrypt with the general knapsack, but it's apparently hard to decrypt without the private key. Of course, with the private key, we can convert the problem into a superincreasing knapsack problem that is easy to solve.

The knapsack appears to be just what the doctor ordered. First, it is fairly easy to construct a public and private key pair. And given the public key, it is easy to encrypt, while knowledge of the private key makes it easy to decrypt. Finally, without the private key it appears that Trudy will be forced to solve an NP-complete problem.

Alas, this clever knapsack cryptosystem is insecure. It was broken by Shamir (who else?) in 1983 using an Apple II computer [265]. The attack relies on a technique known as lattice reduction that we discuss in detail in Chapter 6. The bottom line is that the "general knapsack" that is derived from the superincreasing knapsack is not really a general knapsack—in fact, it's a very special and highly structured case of the knapsack. The lattice reduction attack is able to take advantage of this structure to easily recover the plaintext (with a high probability).

Much research has been done on the knapsack problem since the demise of the Merkle-Hellman knapsack. Today, there are knapsack variants that appear to be secure, but people are reluctant to use them since the name "knapsack" is forever tainted. For more information on knapsack cryptosystems, see [88, 179, 222].

## 4.3 RSA

Like any worthwhile public key cryptosystem, RSA is named after its putative inventors, Rivest, Shamir, and Adleman. We've met Rivest and Shamir previously, and we'll hear from both again. In fact, Rivest and Shamir are two of the giants of modern crypto. However, the RSA concept was actually originated by Cliff Cocks of GCHQ a few years before R, S, and A independently reinvented it [191]. This does not in any way diminish the achievement of Rivest, Shamir, and Adleman, since the GCHQ work was classified and was not even widely known within the classified crypto community—it was viewed more as a curiosity than as a practical system.[5]

If you've ever wondered why there is so much interest in factoring large numbers, it's because RSA can be broken by factoring. However, it's not known for certain that factoring is difficult in the sense that, say, the knapsack

[5]It is also worth noting that the spies seem to have never even considered the concept of a digital signature.

problem is difficult. In true cryptographic fashion, the factoring problem on which RSA rests is hard because lots of smart people have looked at it, and apparently nobody has found an efficient solution.

To generate an RSA public and private key pair, choose two large prime numbers $p$ and $q$ and form their product $N = pq$. Next, choose $e$ relatively prime to the product $(p-1)(q-1)$. Finally, find the multiplicative inverse of $e$ modulo $(p-1)(q-1)$ and denote this inverse as $d$. At this point, we have $N$, which is the product of the two primes $p$ and $q$, as well as $e$ and $d$, which satisfy $ed = 1 \bmod (p-1)(q-1)$. Now forget the factors $p$ and $q$.

The number $N$ is the *modulus*, and $e$ is the *encryption exponent* while $d$ is the *decryption exponent*. The RSA key pair consists of

$$\text{Public key: } (N, e)$$

and

$$\text{Private key: } d.$$

In RSA, encryption and decryption are accomplished via modular exponentiation. To encrypt with RSA, we treat the plaintext message $M$ as a number and raise it to the power $e$, modulo $N$, that is,

$$C = M^e \bmod N.$$

To decrypt $C$, modular exponentiation using the decryption exponent $d$ does the trick, that is,

$$M = C^d \bmod N.$$

It's probably not obvious that RSA decryption actually works—we'll prove that it does shortly. Assume for a moment that RSA does work. Now, if Trudy can factor the modulus $N$ (which is public) she will obtain $p$ and $q$. Then she can use the other public value $e$ to easily find the private value $d$ since $ed = 1 \bmod (p-1)(q-1)$ and finding modular inverses is computationally easy. In other words, factoring the modulus enables Trudy to recover the private key, which breaks RSA. However, it is not known whether factoring is the only way to break RSA.

Does RSA really work? Given $C = M^e \bmod N$, we must show that

$$M = C^d \bmod N = M^{ed} \bmod N. \tag{4.2}$$

To do so, we need the following standard result from number theory [43]:

**Euler's Theorem**: If $x$ is relatively prime to $n$ then $x^{\phi(n)} = 1 \bmod n$

Recall that $e$ and $d$ were chosen so that

$$ed = 1 \bmod (p-1)(q-1).$$

Furthermore, $N = pq$, which implies

$$\phi(N) = (p-1)(q-1)$$

(see the Appendix if you are not familiar with the $\phi$ function). These two facts together imply that

$$ed - 1 = k\phi(N)$$

for some integer $k$. We don't need to know the precise value of $k$.

Now we have all of the necessary pieces of the puzzle to verify that RSA decryption works. Observe that

$$\begin{aligned} C^d &= M^{ed} = M^{(ed-1)+1} = M \cdot M^{ed-1} \\ &= M \cdot M^{k\phi(N)} = M \cdot 1^k = M \bmod N. \end{aligned} \tag{4.3}$$

In the first line of equation (4.3) we simply added zero to the exponent and in the second line we used Euler's Theorem to eliminate the ominous-looking $M^{\phi(N)}$ term. This confirms that the RSA decryption exponent does, in fact, decrypt the ciphertext $C$. Of course, the game was rigged since $e$ and $d$ were chosen so that Euler's Theorem would make everything come out as desired in the end. That's just the way mathematicians do things.

### 4.3.1 Textbook RSA Example

Let's consider a simple RSA example. To generate, say, Alice's keypair, we'll select the two "large" primes $p = 11$ and $q = 3$. Then the modulus is $N = pq = 33$ and $(p-1)(q-1) = 20$. Next, we choose the encryption exponent $e = 3$, which is, as required, relatively prime to $(p-1)(q-1)$. We then compute the corresponding decryption exponent, which in this case is $d = 7$, since $ed = 3 \cdot 7 = 1 \bmod 20$. Now, we have

$$\text{Alice's public key: } (N, e) = (33, 3)$$

and

$$\text{Alice's private key: } d = 7.$$

As usual, Alice's public key is public but only Alice has access to her private key.

Now suppose Bob wants to send Alice a message $M$. Further, suppose that as a number, the message is $M = 15$. Bob looks up Alice's public key $(N, e) = (33, 3)$ and computes the ciphertext as

$$C = M^e \bmod N = 15^3 = 3375 = 9 \bmod 33,$$

which he then sends to Alice.

To decrypt the ciphertext $C = 9$, Alice uses her private key $d = 7$ to find

$$M = C^d \bmod N = 9^7 = 4{,}782{,}969 = 144{,}938 \cdot 33 + 15 = 15 \bmod 33.$$

Alice has thereby recovered the original message $M = 15$ from the ciphertext $C = 9$.

There are a couple of major problems with this textbook RSA example. For one, the "large" primes are not large—it would be trivial for Trudy to factor the modulus. In the real world, the modulus $N$ is typically at least 1024 bits, with a 2048-bit or large modulus often used.

An equally serious problem with most textbook RSA examples (ours included) is that they are subject to a forward search attack, as discussed in Chapter 2. Recall that in a forward search, Trudy can guess a possible plaintext message $M$ and encrypt it with the public key. If the result matches the ciphertext $C$, then Trudy has recovered the plaintext $M$. The way to prevent this attack (and several others) is to pad the message with random bits. For simplicity, we do not discuss padding here, but it is worth noting that several padding schemes are in common use, including the oddly named PKCS#1v1.5 [91] and Optimal Asymmetric Encryption Padding (OAEP) [226]. Any real-world RSA implementation must use a padding scheme such as one of these.

### 4.3.2 Repeated Squaring

Modular exponentiation of large numbers with large exponents is an expensive proposition. To make this more manageable (and thereby make RSA more efficient and practical), several tricks are commonly used. The most basic trick is the method of *repeated squaring* (also known as *square and multiply*).

For example, suppose we want to compute $5^{20}$. Naïvely, we would simply multiply 5 by itself 20 times and then reduce the result modulo 35, that is,

$$5^{20} = 95{,}367{,}431{,}640{,}625 = 25 \bmod 35. \tag{4.4}$$

However, this method results in an enormous value prior to the modular reduction, in spite of the fact that the final answer is restricted to the range 0 to 34.

Now suppose we want to do RSA encryption $C = M^e \bmod N$ or decryption $M = C^d \bmod N$. In a secure implementation of RSA, the modulus $N$ is at least 1024 bits. As a result, for typical values of $e$ or $d$, the numbers involved will be so large that it is impossible to compute $M^e \bmod N$ by the naïve approach in equation (4.4). Fortunately, the method of repeated squaring allows us to compute such an exponentiation without creating unmanageably large numbers at any intermediate step.

Repeated squaring works by building up the exponent one bit at a time. At each step we double the current exponent and if the binary expansion has a 1 in the corresponding position, we also add one to the exponent.

How can we double (and add one) to an exponent? Basic properties of exponentiation tell us that if we square $x^y$, we obtain $(x^y)^2 = x^{2y}$ and that $x \cdot x^y = x^{y+1}$. Consequently, we can easily double or add one to any exponent. From the basic properties of modular arithmetic (see the Appendix), we know that we can reduce any intermediate results by the modulus, and thereby avoid extremely large numbers.

An example is worth a thousand words. Consider again $5^{20}$. First, note that the exponent 20 is, in binary, 10100. The exponent 10100 can be built up one bit at a time, beginning from the high-order bit, as

$$(0, 1, 10, 101, 1010, 10100) = (0, 1, 2, 5, 10, 20).$$

As a result, the exponent 20 can be constructed by a series of steps, where each step consists of doubling and, when the next bit in the binary expansion of 20 is a 1, adding one, that is,

$$\begin{aligned} 1 &= 0 \cdot 2 + 1 \\ 2 &= 1 \cdot 2 \\ 5 &= 2 \cdot 2 + 1 \\ 10 &= 5 \cdot 2 \\ 20 &= 10 \cdot 2 \end{aligned}$$

Now to compute $5^{20}$, repeated squaring proceeds as

$$\begin{aligned} 5^1 &= (5^0)^2 \cdot 5^1 = 5 \bmod 35 \\ 5^2 &= (5^1)^2 = 5^2 = 25 \bmod 35 \\ 5^5 &= (5^2)^2 \cdot 5^1 = 25^2 \cdot 5 = 3125 = 10 \bmod 35 \\ 5^{10} &= (5^5)^2 = 10^2 = 100 = 30 \bmod 35 \\ 5^{20} &= (5^{10})^2 = 30^2 = 900 = 25 \bmod 35 \end{aligned}$$

Note that a modular reduction occurs at each step.

Although there are many steps in the repeated squaring algorithm, each step is simple, efficient, and we never have to deal with a number that is greater than the cube of the modulus. Compare this to equation (4.4), where we had to deal with an enormous intermediate value.

### 4.3.3 Speeding Up RSA

Another trick that can be employed to speed up RSA is to use the same encryption exponent $e$ for all users. As far as anyone knows, this does not

weaken RSA in any way. The decryption exponents (the private keys) of different users will be different, since different $p$, $q$, and consequently $N$ are chosen for each key pair.

Amazingly, a suitable choice for the common encryption exponent is $e = 3$. With this choice of $e$, each public key encryption only requires two multiplications. However, the private key operations remain expensive since there is no special structure for $d$. This is often acceptable since many encryptions may be done by a central server, while the decryption is effectively distributed among the clients. Of course, if the server needs to compute digital signatures, then a small $e$ does not reduce its workload. Although the math would work, it would certainly be a bad idea to choose a common value of $d$ for all users.

With an encryption exponent of $e = 3$, the following *cube root attack* is possible. If the plaintext $M$ satisfies $M < N^{1/3}$, then $C = M^e = M^3$, that is, the mod $N$ operation has no effect. As a result, an attacker can simply compute the usual cube root of $C$ to obtain $M$. In practice, this is easily avoided by padding $M$ with enough bits so that, as a number, $M > N^{1/3}$.

If multiple users all have $e = 3$ as their encryption exponent, another type of the cube root attack exists. If the same message $M$ is encrypted with three different users' public keys, yielding, say, ciphertext $C_0$, $C_1$, and $C_2$, then the Chinese Remainder Theorem [43] can be used to recover the message $M$. This is also easily avoided in practice by randomly padding each message $M$ or by including some user-specific information in each $M$, so that the messages actually differ.

Another popular common encryption exponents is $e = 2^{16}+1$. With this $e$, each encryption requires only 17 steps of the repeated squaring algorithm. An advantage of $e = 2^{16} + 1$ is that the same encrypted message must be sent to $2^{16} + 1$ users before the Chinese Remainder Theorem attack can succeed.

Next, we'll examine the Diffie-Hellman key exchange algorithm, which is a very different sort of public key algorithm. Whereas RSA relies on the difficulty of factoring, Diffie-Hellman is based on the so-called discrete log problem.

## 4.4 Diffie-Hellman

The Diffie-Hellman key exchange algorithm, or DH for short, was invented by Malcolm Williamson of GCHQ and shortly thereafter it was independently reinvented by its namesakes, Whitfield Diffie and Martin Hellman [191].

The version of DH that we discuss here is a key exchange algorithm because it can only be used to establish a shared secret. The resulting shared secret is generally used as a shared symmetric key. It's worth emphasizing that, in this book, the words "Diffie-Hellman" and "key exchange" always

go together—DH is not for encrypting or signing, but instead it allows users to establish a shared symmetric key. This is no mean feat, since this key establishment problem is one of the fundamental problems in symmetric key cryptography.

The security of DH relies on the computational difficulty of the *discrete log* problem. Suppose you are given $g$ and $x = g^k$. Then to determine $k$ you would compute the logarithm, $\log_g(x)$. Now given $g$, $p$, and $g^k \bmod p$, the problem of finding $k$ is analogous to the logarithm problem, but in a discrete setting. This discrete version of the logarithm problem is, not surprisingly, known as the discrete log problem. As far as is known, the discrete log problem is very difficult to solve, although, as with factoring, it is not known to be, say, NP-complete.

The mathematical setup for DH is relatively simple. Let $p$ be prime and let $g$ be a *generator*, which means that for any $x \in \{1, 2, \ldots, p-1\}$ there exists an exponent $n$ such that $x = g^n \bmod p$. The prime $p$ and the generator $g$ are public.

For the actual key exchange, Alice randomly selects a secret exponent $a$ and Bob randomly selects a secret exponent $b$. Alice computes $g^a \bmod p$ and sends the result to Bob, and Bob computes $g^b \bmod p$ and sends the result to Alice. Then Alice computes

$$(g^b)^a \bmod p = g^{ab} \bmod p$$

and Bob computes

$$(g^a)^b \bmod p = g^{ab} \bmod p$$

and $g^{ab} \bmod p$ is the shared secret, which is typically used as a symmetric key. The DH key exchange is illustrated in Figure 4.1.

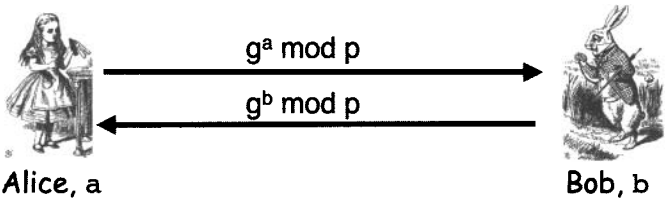



Figure 4.1: Diffie-Hellman Key Exchange

The attacker Trudy can see $g^a \bmod p$ and $g^b \bmod p$, and it seems that she is tantalizingly close to knowing the secret $g^{ab} \bmod p$. However,

$$g^a \cdot g^b = g^{a+b} \neq g^{ab} \bmod p$$

Apparently, Trudy needs to find either $a$ or $b$, which appears to require that she solve a difficult discrete log problem. Of course, if Trudy can find $a$ or $b$

or $g^{ab}$ mod $p$ by any other means, the system is broken. But, as far as is known, the only way to break DH is to solve the discrete log problem.

There is a fundamental problem with the DH algorithm—it is susceptible to a man-in-the-middle, or MiM, attack.[6] This is an active attack where Trudy places herself between Alice and Bob and captures messages from Alice to Bob and vice versa. With Trudy thusly placed, the DH exchange can be easily subverted. In the process, Trudy establishes a shared secret, say, $g^{at}$ mod $p$ with Alice, and another shared secret $g^{bt}$ mod $p$ with Bob, as illustrated in Figure 4.2. Neither Alice nor Bob has any clue that anything is amiss, yet Trudy is able to read or change any messages passing between Alice and Bob.[7]

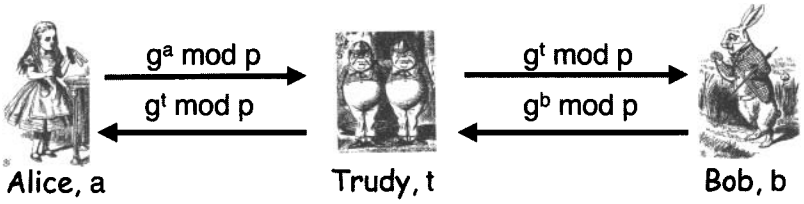



Figure 4.2: Diffie-Hellman Man-in-the-Middle Attack

The MiM attack in Figure 4.2 is a serious concern when using DH. There are several possible ways to prevent the attack, including the following:

1. Encrypt the DH exchange with a shared symmetric key.
2. Encrypt the DH exchange with public keys.
3. Sign the DH values with private keys.

At this point, you should be baffled. After all, why would we need to use DH to establish a symmetric key if we already have a shared symmetric key (as in 1) or a public key pair (as in 2 and 3)? This is an excellent question to which we'll give an excellent answer when we discuss protocols in Chapters 9 and 10.

## 4.5 Elliptic Curve Cryptography

Elliptic curves provide an alternative domain for performing the complex mathematical operations required in public key cryptography. So, for example, there is an elliptic curve version of Diffie-Hellman.

[6] Your politically incorrect author refuses to use the term "middleperson" attack.

[7] The underlying problem here is that the participants are not authenticated. In this example, Alice does not know she's talking to Bob and vice versa. It will be a few more chapters before we discuss authentication protocols.

The advantage of elliptic curve cryptography (ECC) is that fewer bits are needed to achieve the same level of security. On the down side, elliptic curve math is more complex, and consequently each mathematical operation on an elliptic curve is somewhat more expensive. Overall, elliptic curves offer a significant computational advantage over standard modular arithmetic and current U.S. government standards reflect this—all recent public key standards are ECC-based. In addition, ECC is especially important in resource-constrained environments such as handheld devices.

What is an elliptic curve? An elliptic curve $E$ is the graph of a function of the form

$$E: \quad y^2 = x^3 + ax + b,$$

together with a special point at infinity, denoted $\infty$. The graph of a typical elliptic curve appears in Figure 4.3.

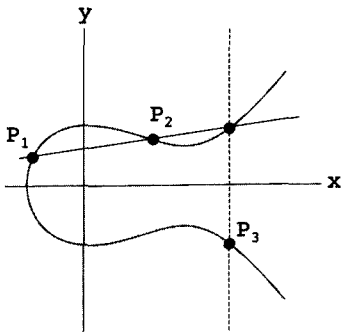



Figure 4.3: An Elliptic Curve

### 4.5.1 Elliptic Curve Math

Figure 4.3 also illustrates the method used to do arithmetic on an elliptic curve. The "sum" of two points on an elliptic curve has both a geometric and arithmetic interpretation. Geometrically, the sum of the points $P_1$ and $P_2$ is defined as follows: First, a line is drawn through the two points. This line usually intersects the curve in one other point. If so, this intersection point is reflected about the $x$ axis to obtain the point $P_3$, which is defined to be the sum, that is,

$$P_3 = P_1 + P_2.$$

This is illustrated in Figure 4.3. Also, addition is the only mathematical operation on elliptic curves that is required.

For cryptography, we want to deal with a discrete set of points. This is easily accomplished by including a "mod $p$" in the generic elliptic curve

equation, that is,

$$y^2 = x^3 + ax + b \pmod{p}.$$

For example, consider the elliptic curve

$$y^2 = x^3 + 2x + 3 \pmod{5}. \tag{4.5}$$

We can list all of the points $(x, y)$ on this curve by substituting all values for $x$ and solving for corresponding $y$ value or values. Since we are working modulo 5, we only need to consider $x = 0, 1, 2, 3, 4$. In this case, we obtain the following points:

$$\begin{aligned}
x = 0 &\Longrightarrow y^2 = 3 \Longrightarrow \text{ no solution mod } 5 \\
x = 1 &\Longrightarrow y^2 = 6 = 1 \Longrightarrow y = 1, 4 \bmod 5 \\
x = 2 &\Longrightarrow y^2 = 15 = 0 \Longrightarrow y = 0 \bmod 5 \\
x = 3 &\Longrightarrow y^2 = 36 = 1 \Longrightarrow y = 1, 4 \bmod 5 \\
x = 4 &\Longrightarrow y^2 = 75 = 0 \Longrightarrow y = 0 \bmod 5
\end{aligned}$$

That is, we find that the points on the elliptic curve in equation (4.5) are

$$(1,1)\ (1,4)\ (2,0)\ (3,1)\ (3,4)\ (4,0) \text{ and } \infty. \tag{4.6}$$

Next, we again consider the problem of adding two points on a curve. We need a more computer-friendly approach than the geometric definition discussed above. The algorithm for algebraically adding two points on an elliptic curve appears in Table 4.1.

Table 4.1: Addition on an Elliptic Curve mod $p$

**Given:** curve $E$: $y^2 = x^3 + ax + b \pmod{p}$
$P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ on $E$
**Find:** $P_3 = (x_3, y_3) = P_1 + P_2$
**Algorithm:**
$x_3 = m^2 - x_1 - x_2 \pmod{p}$
$y_3 = m(x_1 - x_3) - y_1 \pmod{p}$
where $m = \begin{cases} (y_2 - y_1) \cdot (x_2 - x_1)^{-1} \bmod p & \text{if } P_1 \neq P_2 \\ (3x_1^2 + a) \cdot (2y_1)^{-1} \bmod p & \text{if } P_1 = P_2 \end{cases}$
Special case 1: If $m = \infty$ then $P_3 = \infty$
Special case 2: $\infty + P = P$ for all $P$

Let's apply the algorithm in Table 4.1 to find the points $P_3 = (1,4) + (3,1)$ on the curve in equation (4.5). First, we compute

$$m = (1 - 4)/(3 - 1) = -3 \cdot 2^{-1} = -3 \cdot 3 = 1 \bmod 5.$$

Then

$$x_3 = 1^2 - 1 - 3 = -3 = 2 \bmod 5$$

and

$$y_3 = 1(1 - 2) - 4 = -5 = 0 \bmod 5.$$

Therefore, on the curve $y^2 = x^3+2x+3 \pmod 5$, we have $(1, 4)+(3, 1) = (2, 0)$. Note that this sum, the point $(2, 0)$, is also on the elliptic curve.

### 4.5.2 ECC Diffie-Hellman

Now that we can do addition on elliptic curves, let's consider the ECC version of the Diffie-Hellman key exchange. The public information consists of a curve and a point on the curve. We'll select the curve

$$y^2 = x^3 + 11x + b \pmod{167}, \tag{4.7}$$

leaving $b$ to be determined momentarily. Next, we select any point $(x, y)$ and determine $b$ so that this point lies on the resulting curve. In this case, we'll choose, say, $(x, y) = (2, 7)$. Then substituting $x = 2$ and $y = 7$ into equation (4.7) we find $b = 19$. The public information is

$$\text{Public: } y^2 = x^3 + 11x + 19 \pmod{167} \text{ and the point } (2, 7). \tag{4.8}$$

Alice and Bob each must randomly select their own secret multipliers.[8] Suppose Alice selects $A = 15$ and Bob selects $B = 22$. Then Alice computes

$$A(2, 7) = 15(2, 7) = (102, 88),$$

where all arithmetic is done on the curve in equation (4.8). Alice sends her computed result to Bob. Bob computes

$$B(2, 7) = 22(2, 7) = (9, 43),$$

which he sends to Alice. Now Alice multiplies the value she received from Bob by her secret multiplier $A$, that is,

$$A(9, 43) = 15(9, 43) = (131, 140).$$

Similarly, Bob computes

$$B(102, 88) = 22(102, 88) = (131, 140)$$

and Alice and Bob have established a shared secret, suitable for use as a symmetric key. Note that this elliptic curve version of Diffie-Hellman works

[8]Since we know how to do addition on an elliptic curve, we do multiplication as repeated addition.

since $AB \cdot P = BA \cdot P$, where $A$ and $B$ are multipliers and $P$ is the specified point on the curve. The security of this method rests on the fact that, although Trudy can see $A \cdot P$ and $B \cdot P$, she (apparently) must find $A$ or $B$ before she can determine the shared secret. As far as is known, this elliptic curve version of DH is as difficult to break as the regular DH. Actually, for a given number of bits, the elliptic curve version is much harder to break, which allows for the use of smaller values to obtain an equivalent level of security. Since the values are smaller, the arithmetic is more efficient.

All is not lost for Trudy. She can take some comfort in the fact that the ECC version of DH is just as susceptible to a MiM attack as any other Diffie-Hellman key exchange.

### 4.5.3 Realistic Elliptic Curve Example

To provide some idea of the magnitude of the numbers used in real-world ECC, we present a realistic example. This example appears as part of the Certicom ECCp-109 challenge problem [52], and it is discussed in Jao's excellent survey [154]. Note that the numbers are given in decimal and no commas appear within the numbers.

Let

$$p = 564538252084441556247016902735257$$
$$a = 321094768129147601892514872825668$$
$$b = 430782315140218274262276694323197$$

and consider the elliptic curve $E : y^2 = x^3 + ax + b \pmod{p}$. Let $P$ be the point

$$(973390109870590665231561339089 35, 149670372846169285760682371978898)$$

which is on $E$, and let $k = 281183840311601949668207954530684$. Then adding the point $P$ to itself $k$ times, which we denote as $kP$, we obtain the point

$$(446467696974058610576308618842 84, 522968098895785888047540374779097)$$

which is also on the curve $E$.

While these numbers are indeed large, they are downright puny in comparison to the numbers that must be used in a non-elliptic curve public key system. For example, a modest-sized RSA modulus has 1024 bits, which corresponds to more than 300 decimal digits. In contrast, the numbers in the elliptic curve example above only have about 1/10th as many digits.

There are many good sources of information on the hot topic of elliptic curve cryptography. For an accessible treatment see [251] or see [35] for more of the mathematical details.

## 4.6 Public Key Notation

Before discussing the uses of public key crypto, we need to settle on some reasonable notation. Since public key systems typically have two keys per user, adapting the notation that we used for symmetric key crypto would be awkward. In addition, a digital signature is an encryption (with the private key), but yet the same operation is a decryption when applied to ciphertext. If we're not careful, this notation thing could get complicated.

We'll adopt the following notation for public key encryption, decryption, and signing [162].

- Encrypt message $M$ with Alice's public key: $C = \{M\}_{\text{Alice}}$.
- Decrypt ciphertext $C$ with Alice's private key: $M = [C]_{\text{Alice}}$.
- The notation for Alice signing[9] message $M$ is $S = [M]_{\text{Alice}}$.

Note that curly brackets represent public key operations, square brackets are for private key operations, and the subscript tells us whose key is being used. This is somewhat awkward but, in your notationally challenged author's opinion, it is the least bad of the possibilities. Finally, since public and private key operations cancel each other out,

$$[\{M\}_{\text{Alice}}]_{\text{Alice}} = \{[M]_{\text{Alice}}\}_{\text{Alice}} = M.$$

Never forget that the public key is public and, consequently, anyone can compute $\{M\}_{\text{Alice}}$. On the other hand, the private key is private, so only Alice can compute $[C]_{\text{Alice}}$ or $[M]_{\text{Alice}}$. The implication is that anyone can encrypt a message for Alice, but only Alice can decrypt the ciphertext. In terms of signing, only Alice can sign $M$, but, since the public key is public, anyone can verify the signature. We'll have more to say about signatures and verification after we discuss hash functions in the next chapter.

## 4.7 Uses for Public Key Crypto

Anything you can do with a symmetric cipher you can do with public key crypto, only slower. This includes confidentiality, in the form of transmitting data over an insecure channel or securely storing data on an insecure media. We can also use public key crypto for integrity—a signature plays the role of a MAC in the symmetric case.

In addition, there are things that we can do with public keys that have no analog in the symmetric crypto world. Specifically, public key crypto offers

[9] Actually, this is not the correct way to digitally sign a message; see Section 5.2 of Chapter 5.

two major advantages over symmetric key crypto. The first is that, with public key crypto, we don't need to established a shared key in advance.[10] The second major advantage is that digital signatures provide integrity (see Problem 35) and non-repudiation. We look a little closer at these two topics below.

### 4.7.1 Confidentiality in the Real World

The primary advantage of symmetric key cryptography over public key is efficiency.[11] In the realm of confidentiality, the primary advantage of public key cryptography is the fact that no shared key is required.

Is it possible to get the best of both worlds? That is, can we have the efficiency of symmetric key crypto and yet not have to share keys in advance? The answer is an emphatic yes. The way to achieve this highly desirable result is with a *hybrid cryptosystem*, where public key crypto is used to establish a symmetric key and the resulting symmetric key is then used to encrypt the data. A hybrid cryptosystem is illustrated in Figure 4.4.

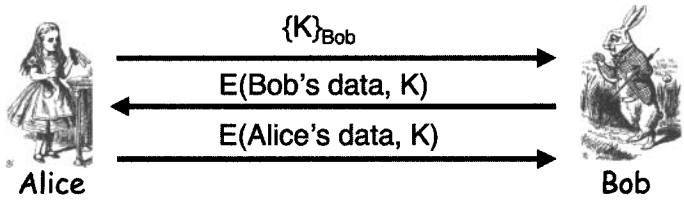



Figure 4.4: Hybrid Cryptosystem

The hybrid cryptosystem in Figure 4.4 is only for illustrative purposes. In fact, Bob has no way to know that he's talking to Alice—since anyone can do public key operations—so he would be foolish to encrypt sensitive data and send it to "Alice" following this protocol. We'll have much more to say about secure authentication and key establishment protocols in upcoming chapters. Hybrid crypto (with secure authentication) is widely used in practice today.

### 4.7.2 Signatures and Non-repudiation

As mentioned above, a digital signature can be used for integrity. Recall that a MAC is a way to provide integrity that uses a symmetric key. So, a signature is as good as a MAC when it comes to integrity. In addition, a digital

[10] Of course, we do need to get the private keys to the participants beforehand, so the key distribution problem has not been completely eliminated—it has just taken on a different form.

[11] A secondary benefit is that no public key infrastructure, or PKI, is required. We'll discuss PKI below.

signature provides *non-repudiation*, which is something that symmetric keys by their very nature cannot provide.

To understand non-repudiation, let's first consider an integrity example in the symmetric key case. Suppose Alice orders 100 shares of stock from her favorite stockbroker, Bob. To ensure the integrity of her order, Alice computes a MAC using a shared symmetric key $K_{AB}$. Now suppose that shortly after Alice places the order—but before she has paid any money to Bob—the stock loses all of its value. At this point Alice could claim that she did not place the order, that is, she could *repudiate* the transaction.

Can Bob prove that Alice placed the order? If all he has is the MAC, then he cannot. Since Bob also knows the symmetric key $K_{AB}$, he could have forged the message in which "Alice" placed the order. Note that Bob knows that Alice placed the order (since he didn't forge it), but he can't prove it in a court of law.

Now consider the same scenario, but suppose Alice uses a digital signature instead of a MAC. As with the MAC, the signature provides an integrity check. Again, suppose that the stock loses its value and Alice tries to repudiate the transaction. Can Bob prove that the order came from Alice? Yes he can, since only Alice has access to her private key.[12] Therefore, digital signatures provide integrity and non-repudiation, while a MAC can only be used for integrity. This is simply due to the fact that the symmetric key is known to both Alice and Bob, whereas Alice's private key is only known to Alice.[13] We'll have more to say about signatures and integrity in the next chapter.

### 4.7.3 Confidentiality and Non-repudiation

Suppose that Alice and Bob have public keys available and Alice wants to send a message $M$ to Bob. For confidentiality, Alice would encrypt $M$ with Bob's public key, and for integrity and non-repudiation, Alice can sign $M$ with her private key. But suppose that Alice, who is very security conscious, wants both confidentiality and non-repudiation. Then she can't just sign $M$ as that will not provide confidentiality, and she can't just encrypt $M$ as that won't provide integrity. The solution seems straightforward enough—Alice can sign and encrypt the message before sending it to Bob, that is,

$$\{[M]_{\text{Alice}}\}_{\text{Bob}}.$$

[12] Of course, we are assuming that Alice's private key has not been lost or compromised. In any case, if the keys are in the wrong hands then all bets are off.

[13] One may be the loneliest number, but when it comes to non-repudiation, two is much worse than one.

Or would it be better for Alice to encrypt $M$ first and then sign the result? That is, should Alice compute

$$[\{M\}_{\text{Bob}}]_{\text{Alice}}$$

instead? Can the order possibly matter? Is this something that only an anal-retentive cryptographer could care about?

Let's consider a couple of different scenarios, similar to those found in [77]. First, suppose that Alice and Bob are romantically involved. Alice decides to send the message

$$M = \text{"I love you"}$$

to Bob and she decides to use the sign and encrypt approach. So, Alice sends Bob the message

$$\{[M]_{\text{Alice}}\}_{\text{Bob}}.$$

Subsequently, Alice and Bob have a lovers' tiff and Bob, in an act of spite, decrypts the signed message to obtain $[M]_{\text{Alice}}$ and re-encrypts it using Charlie's public key, that is,

$$\{[M]_{\text{Alice}}\}_{\text{Charlie}}.$$

Bob then sends this message to Charlie, as illustrated in Figure 4.5. Of course, Charlie thinks that Alice is in love with him, which causes a great deal of embarrassment for both Alice and Charlie, much to Bob's delight.

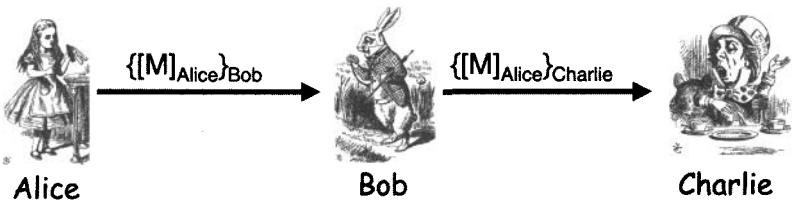



Figure 4.5: Pitfall of Sign and Encrypt

Alice, having learned her lesson from this bitter experience, vows to never sign and encrypt again. When she wants confidentiality and non-repudiation, Alice will always encrypt then sign.

Some time later, after Alice and Bob have resolved their earlier issues, Alice develops a great new theory that she wants to communicate to Bob. This time her message is [55]

$$M = \text{"Brontosauruses are thin at one end, much much thicker in the middle, then thin again at the other end"},$$

which she dutifully encrypts then signs

$$[\{M\}_{\text{Bob}}]_{\text{Alice}}$$

before sending it to Bob.

However, Charlie, who is still angry with Bob and Alice, has set himself up as a man-in-the-middle so that he is able to intercept all traffic between Alice and Bob. Charlie knows that Alice is working on a great new theory, and he also knows that Alice only encrypts important messages. Charlie suspects that this encrypted and signed message is important and somehow related to Alice's important new theory. So, Charlie uses Alice's public key to compute $\{M\}_{\text{Bob}}$, which he then signs before sending it to Bob, that is, Charlie sends

$$[\{M\}_{\text{Bob}}]_{\text{Charlie}}$$

to Bob. This scenario is illustrated in Figure 4.6.

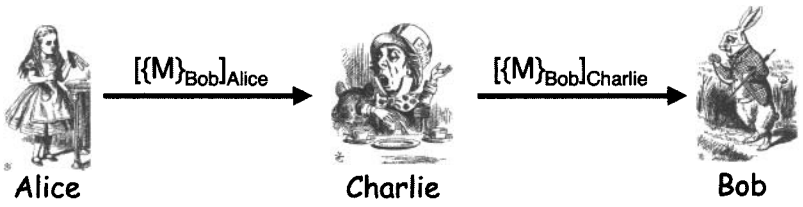



Figure 4.6: Pitfall of Encrypt and Sign

When Bob receives the message from Charlie, he assumes that this great new theory is Charlie's, and he immediately gives Charlie a promotion. When Alice learns that Charlie has taken credit for her great new theory, she swears never to encrypt and sign again.

Note that in the first scenario, Charlie assumed that $\{[M]_{\text{Alice}}\}_{\text{Charlie}}$ must have been sent from Alice to Charlie. That's not a valid assumption—Charlie's public key is public, so anyone could have done the encryption. In fact, the only thing Charlie really knows is that at some point Alice signed $M$. The problem here is that Charlie has apparently forgotten that public keys are public.

In the second scenario, Bob assumed that $[\{M\}_{\text{Bob}}]_{\text{Charlie}}$ must have originated with Charlie, which is also not a valid assumption. Again, since public keys are public, anybody could've encrypted $M$ with Bob's public key. It is true that Charlie must have signed this encrypted message, but that does not imply that Charlie actually encrypted it (or even knows what the plaintext message says).

In both of these cases, the underlying problem is that the recipient does not clearly understand the way that public key cryptography works. There are some inherent limitations to public key crypto, most of which are due to the fact that anyone can do public key operations, that is, anyone can encrypt a message and anyone can verify a signature. This fact can be a source of confusion if you are not careful.

## 4.8 Public Key Infrastructure

A *public key infrastructure*, or PKI, is the sum total of everything required to securely use public keys in the real world. It's surprisingly difficult and involved to assemble all of the necessary pieces of a PKI into a working whole. For a discussion of some of the risks inherent in PKI, see [101].

A *digital certificate* (or public key certificate or, for short, certificate) contains a user's name along with the user's public key and this is signed by a *certificate authority*, or CA. For example, Alice's certificate contains[14]

$$M = (\text{"Alice"}, \text{Alice's public key}) \quad \text{and} \quad S = [M]_{\text{CA}}.$$

To verify this certificate, Bob would compute $\{S\}_{\text{CA}}$ and verify that this matches $M$.

The CA acts as a *trusted third party*, or TTP. By signing the certificate, the CA is vouching for the fact it gave the corresponding private key to Alice. That is, the CA created a public and private key pair and it put the public key in Alice's certificate. Then the CA signed the certificate (using its private key) and it gave the private key to Alice. If you trust the CA, you believe that it actually gave the private key to Alice, and not to anyone else.

A subtle but important point here is that the CA is *not* vouching for the identity of the holder of the certificate. Certificates act as public keys and, consequently, they are public knowledge. So, for example, Trudy could send Alice's public key to Bob and claim to be Alice. Bob must not fall for this trick.

When Bob receives a certificate, he must verify the signature. If the certificate is signed by a CA that Bob trusts, then he uses that CA's public key for verification. On the other hand, if Bob does not trust the CA, then the certificate is useless to him. Anyone can create a certificate and claim to be anyone else. Bob must trust the CA and verify the signature before he can assume the certificate is valid.

But what exactly does it mean for Alice's certificate to be valid? And what useful information does this provide to Bob? Again, by signing the certificate, the CA is vouching for the fact that it gave the private key to Alice, and not to anyone else. In other words, the public key in the certificate is actually Alice's public key, in the sense that Alice—and only Alice—has the corresponding private key.

To finish beating this dead horse, after verifying the signature, Bob trusts that Alice has the corresponding private key. It's critical that Bob does not assume anything more than this. For example, Bob learns nothing about the

[14]This formula is slightly simplified. Actually, we also need to use a hash function when we sign, but we don't yet know about hash functions. We'll give the precise formula for digital signatures in the next chapter. Regardless, this simplified signature illustrates all of the important concepts related to certificates.

sender of the certificate—certificates are public information, so anyone could have sent it to Bob. In later chapters we'll discuss security protocols, where we will see how Bob can use a valid certificate to verify the identity of the sender, but that requires more than simply verifying the signature on the certificate.

In addition to the required public key, a certificate could contain just about any other information that is deemed useful to the participants. However, the more information, the more likely the certificate will become invalid. For example, it might be tempting for a corporation to include the employee's department and phone number in a certificate. But then the inevitable reorganization will invalidate the certificate.

If a CA makes a mistake, the consequences can be dire. For example, VeriSign[15] once issued a signed certificate for Microsoft to someone else [136], that is, VeriSign gave the private key to someone other than Microsoft. That "someone else" could then have acted (electronically, that is) as Microsoft. This particular error was quickly detected, and the certificate was revoked, apparently before any damage was done.

This raises an important PKI issue, namely, *certificate revocation.* Certificates are usually issued with an expiration date. But if a private key is compromised, or it is discovered that a certificate was issued in error, the certificate must be revoked immediately. Most PKI schemes require periodic distribution of certificate revocation lists, or CRLs, which are supposed to be used to filter out compromised certificates. In some situations, this could place a significant burden on users, which could to lead to mistakes and security flaws.

To summarize, any PKI must deal with the following issues:

- Key generation and management
- Certificate authorities (CAs)
- Certificate revocation

Next, we'll briefly discuss a few of the many PKI *trust models* that are used today. The basic issue is deciding who you are willing to trust as a CA. Here, we follow the terminology in [162].

Perhaps the most obvious trust model is the *monopoly model*, where one universally trusted organization is the CA for the known universe. This approach is naturally favored by whoever happens to be the biggest commercial CA at the time (currently, VeriSign). Some have suggested that the government should play the role of the monopoly CA. However, believe it or not, many people don't trust the government.

[15]Today, VeriSign is the largest commercial source for digital certificates [316].

One major drawback to the monopoly model is that it creates a big target for attack. If the monopoly CA is ever compromised, the entire PKI system fails. And if you don't trust the CA, then the system is useless for you.

The *oligarchy model* is one step away from the monopoly model. In this model, there are multiple trusted CAs. In fact, this is the approach that is used today—a Web browser might be configured with 80 or more CA certificates. A security-conscious user such as Alice is free to decide which of the CAs she is willing to trust and which she is not. On the other hand, a more typical user like Bob will trust whatever CAs are configured in the default settings on his browser.

At the opposite extreme from the monopoly model is the *anarchy model*. In this model, anyone can be a CA, and it's up to the users to decide which CAs they want to trust. In fact, this approach is used in PGP, where it goes by the name "web of trust."

The anarchy model can place a significant burden on users. For example, suppose you receive a certificate signed by Frank and you don't know Frank, but you do trust Bob and Bob says Alice is trustworthy and Alice vouches for Frank. Should you trust Frank? This is clearly beyond the patience of the average user, who is likely to simply trust everybody or nobody so as to avoid headaches like this.

There are many other PKI trust models, most of which try to provide reasonable flexibility while putting a minimal burden on end users. The fact that there is no generally agreed upon trust model is itself one of the major problems with PKI.

## 4.9 Summary

In this chapter, we've covered most of the most important public key crypto topics. We began with the knapsack, which has been broken, but provides a nice introductory example. We then discussed RSA and Diffie-Hellman in some detail.

We also discussed elliptic curve cryptography (ECC), which promises to play an ever-increasing role in the future. Remember that ECC is not a particular type of cryptosystem, but instead it offers another way to do the math in public key cryptography.

We then considered signing and non-repudiation, which are major benefits of public key cryptography. And we presented the idea of a hybrid cryptosystem, which is the way that public key crypto is used in the real world for confidentiality. We also discussed the critical—and often confused—topic of digital certificates. It is important to realize exactly what a certificate does and does not provide. Finally, we took a very brief look at PKI, which is often a major roadblock to the deployment of public key crypto.

This concludes our overview of public key cryptography. We will see many applications of public key crypto in later sections of the book. In particular, many of these topics will resurface when we discuss security protocols.

## 4.10 Problems

1. This problem deals with digital certificates (aka public key certificates).
   a. What information must a digital certificate contain?
   b. What additional information can a digital certificate contain?
   c. Why might it be a good idea to minimize the amount of information in a digital certificate?

2. Suppose that Bob receives Alice's digital certificate from someone claiming to be Alice.
   a. Before Bob verifies the signature on the certificate, what does he know about the identity of the sender of the certificate?
   b. How does Bob verify the signature on the certificate and what useful information does Bob gain by verifying the signature?
   c. After Bob verifies the signature on the certificate, what does he know about the identity of the sender of the certificate?

3. When encrypting, public key systems operate in a manner analogous to a block cipher in ECB mode. That is, the plaintext is chopped into blocks and each block is encrypted independently.
   a. Why is ECB mode a bad idea when encrypting with a block cipher? Why is a chaining mode, such as CBC, a much better way to use a block cipher?
   b. Why is it not necessary to perform any sort of chaining mode when using public key encryption?
   c. Could your reasoning in part b be applied to block ciphers? Why or why not?

4. Suppose Alice's RSA public key is $(e, N)$ and her private key is $d$. Alice wants to sign the message $M$, that is, she wants to compute $[M]_{\text{Alice}}$. Give the mathematical formula that she will use.

5. In equation (4.3) we proved that RSA encryption works, that is, we showed $[\{M\}_{\text{Alice}}]_{\text{Alice}} = M$. Give the analogous proof that RSA signature verification works, that is, $\{[M]_{\text{Alice}}\}_{\text{Alice}} = M$.

6. Suppose that Alice's RSA public key is $(N, e) = (33, 3)$ and her private key is $d = 7$.

   a. If Bob encrypts the message $M = 19$ using Alice's public key, what is the ciphertext $C$? Show that Alice can decrypt $C$ to obtain $M$.

   b. Let $S$ be the result when Alice digitally signs the message $M = 25$. What is $S$? If Bob receives $M$ and $S$, explain the process Bob will use to verify the signature and show that in this particular case, the signature verification succeeds.

7. Why is it a bad idea to use the same RSA key pair for both signing and decryption?

8. To speed up RSA, it is possible to choose $e = 3$ for all users. However, this creates the possibility of a cube root attack as discussed in this chapter.

   a. Explain the cube root attack and how to prevent it.

   b. For $(N, e) = (33, 3)$ and $d = 7$, show that the cube root attack works when $M = 3$ but not not when $M = 4$.

9. Recall that with the RSA public key system it is possible to choose the same encryption exponent, $e$, for all users. For the sake of efficiency, sometimes a common value of $e = 3$ is used. Assume this is the case.

   a. What is the cube root attack on RSA and when does it succeed?

   b. Give two different ways of preventing the cube root attack. Both of your proposed fixes must still provide improved efficiency over the case where a common encryption exponent $e = 3$ is not used.

10. Consider the RSA public key cryptosystem. The best generally known attack is to factor the modulus, and the best known factoring algorithm (for a sufficiently large modulus) is the number field sieve. In terms of bits, the work factor for the number field sieve is

$$f(n) = 1.9223n^{1/3}(\log_2 n)^{2/3},$$

    where $n$ is the number of bits in the number being factored. For example, since $f(390) \approx 60$, the work required to factor a 390-bit RSA modulus is roughly equivalent to the work needed for an exhaustive search to recover a 61-bit symmetric key.

    a. Graph the function $f(n)$ for $1 \leq n \leq 10{,}000$.

    b. A 1024-bit RSA modulus $N$ provides roughly the same security as a symmetric key of what length?

c. A 2048-bit RSA modulus $N$ provides roughly the same security as a symmetric key of what length?

d. What size of modulus $N$ is required to have security roughly comparable to a 256-bit symmetric key?

11. On the diagram of the Diffie-Hellman key exchange in Figure 4.1, clearly indicate which information is public and which is private.

12. Suppose Bob and Alice share a symmetric key $K$. Draw a diagram to illustrate a variant of the Diffie-Hellman key exchange between Bob and Alice that prevents the man-in-the-middle attack.

13. Consider the Diffie-Hellman key exchange protocol. Suppose that Alice sends her Diffie-Hellman value, $g^a \bmod p$, to Bob. Further, suppose that Bob wants the resulting shared secret to be a specific value $X$. Can Bob choose his Diffie-Hellman value so that, following the protocol, Alice will compute the shared secret $X$? If so, provide precise details and if not, why not?

14. Suppose that Alice and Bob share a 4-digit PIN number, $X$. To establish a shared symmetric key, Bob proposes the following protocol: Bob will generate a random key $K$ that he will encrypt using the PIN number $X$, that is, $E(K, X)$. Bob will send $E(K, X)$ to Alice, who will decrypt it using the shared PIN number $X$ to obtain $K$. Alice and Bob will then use the symmetric key $K$ to protect their subsequent conversation. However, Trudy can easily determine $K$ by a brute force attack on the PIN number $X$, so this protocol is insecure. Modify the protocol to make it more secure. Note that Alice and Bob only share the 4-digit PIN number $X$ and they do not have access to any other symmetric key or public keys. Hint: Use Diffie-Hellman.

15. A digital signature provides for data integrity and a MAC provides for data integrity. Why does a signature also provides for non-repudiation while a MAC does not?

16. A hybrid cryptosystem uses both public key and symmetric key cryptography to obtain the benefits of each.

a. Illustrate a hybrid system using Diffie-Hellman as the public key system and DES as the symmetric cipher.

b. Illustrate a hybrid system using RSA as the public key system and AES as the symmetric cipher.

17. Illustrate a man-in-the-middle attack on the ECC version of Diffie-Hellman.

18. Suppose that Alice signs the message $M$ = "I love you" and then encrypts it with Bob's public key before sending it to Bob. As discussed in the text, Bob can decrypt this to obtain the signed message and then encrypt the signed message with, say, Charlie's public key and forward the resulting ciphertext to Charlie. Could Alice prevent this "attack" by using symmetric key cryptography?

19. When Alice sends a message $M$ to Bob, she and Bob agree to use the following protocol:

    (i) Alice computes $S = [M]_{\text{Alice}}$.
    (ii) Alice sends $(M, S)$ to Bob.
    (iii) Bob computes $V = \{S\}_{\text{Alice}}$.
    (iv) Bob accepts the signature as valid provided $V = M$.

    With this protocol it's possible for Trudy to forge Alice's signature on a random "message" as follows. Trudy generates a value $R$. She then computes $N = \{R\}_{\text{Alice}}$ and sends $(N, R)$ to Bob. Following the protocol above, Bob computes $V = \{R\}_{\text{Alice}}$ and, since $V = N$, Bob accepts the signature. Bob then believes that Alice sent him the signed nonsense "message" $N$. As a result, Bob gets very annoyed with Alice.

    a. Is this attack a serious concern, or just an annoyance? Justify your answer.
    b. Suppose we modify the protocol as follows:
        (i) Alice computes $S = [F(M)]_{\text{Alice}}$.
        (ii) Alice sends $(M, S)$ to Bob.
        (iii) Bob computes $V = \{S\}_{\text{Alice}}$.
        (iv) Bob accepts the signature as valid provided $V = F(M)$.

        What conditions must the function $F$ satisfy so as to prevent this annoying attack?

20. Suppose that Bob's knapsack private key consists of $(3, 5, 10, 23)$ along with the multiplier $m^{-1} = 6$ and modulus $n = 47$.

    a. Find the plaintext given the ciphertext $C = 20$. Give your answer in binary.
    b. Find the plaintext given the ciphertext $C = 29$. Give your answer in binary.
    c. Find $m$ and the public key.

21. Suppose that for the knapsack cryptosystem, the superincreasing knapsack is $(3, 5, 12, 23)$ with $n = 47$ and $m = 6$.

a. Give the public and private keys.

b. Encrypt the message $M = 1110$ (given in binary). Give your result in decimal.

22. Consider the knapsack cryptosystem. Suppose the public key consists of $(18, 30, 7, 26)$ and $n = 47$.

a. Find the private key, assuming $m = 6$.

b. Encrypt the message $M = 1101$ (given in binary). Give your result in decimal.

23. Prove that for the knapsack cryptosystem, it is always possible to decrypt the ciphertext in linear time, provided that you know the private key.

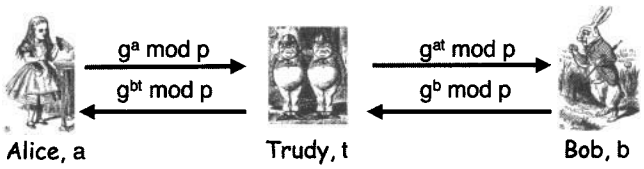
24. For the knapsack example given in the text, the ciphertext was not reduced modulo $n$.

a. Show that for the specific example given in this chapter, the knapsack also works if the ciphertext is reduced modulo $n$.

b. Show that this is always the case, that is, show that it makes no difference to the recipient whether the ciphertext was reduced modulo $n$ or not.

c. Is either case (reducing the ciphertext modulo $n$ or not) preferable from Trudy's perspective?

25. The man-in-the-middle attack on Diffie-Hellman is illustrated in Figure 4.2. Suppose that Trudy wants to establish a single Diffie-Hellman value, $g^{abt} \bmod p$, that she, Alice, and Bob all share. Does the attack illustrated below succeed? Justify your answer.




26. This problem deals with Diffie-Hellman.

a. Why is $g = 1$ not an allowable choice for $g$?

b. Why is $g = p - 1$ not an allowable choice for $g$?

27. In RSA, a common encryption exponent of $e = 3$ or $e = 2^{16} + 1$ is sometimes used. The RSA math will also works if we use a common decryption exponent of, say, $d = 3$. Why would it be a bad idea to use $d = 3$ as a common decryption exponent? Can you find a secure common decryption exponent $d$? Explain.

28. If Trudy can factor the modulus $N$, then she can break the RSA public key cryptosystem. The complexity class for the factorization problem is not known. Suppose that someone proves that integer factorization is a "really hard problem," in the sense that it belongs to a class of (apparently) intractable problems. What would be the practical importance of such a discovery?

29. In the RSA cryptosystem, it is possible that $M = C$, that is, the plaintext and the ciphertext may be identical.

    a. Is this a security concern in practice?

    b. For modulus $N = 3127$ and encryption exponent $e = 17$, find at least one message $M$ that encrypts to itself.

30. Suppose that Bob uses the following variant of RSA. He first chooses $N$, then he finds two encryption exponents $e_0$ and $e_1$ and the corresponding decryption exponents $d_0$ and $d_1$. He asks Alice to encrypt her message $M$ to him by first computing $C_0 = M^{e_0} \bmod N$, then encrypting $C_0$ to obtain the ciphertext, $C_1 = C_0^{e_1} \bmod N$. Alice then sends $C_1$ to Bob. Does this double encryption increase the security as compared to a single RSA encryption? Why or why not?

31. Alice receives a single ciphertext $C$ from Bob, which was encrypted using Alice's RSA public key. Let $M$ be the corresponding plaintext. Alice challenges Trudy to recover $M$ under the following rules. Alice sends $C$ to Trudy, and Alice agrees to decrypt one ciphertext that was encrypted with Alice's public key, provided that it is not $C$, and give the resulting plaintext to Trudy. Is it possible for Trudy to recover $M$?

32. Suppose that you are given the following RSA public keys, which are of the form $(e, N)$.

| User name | Public key |
|---|---|
| Alice | (3, 5356488760553659) |
| Bob | (3, 8021928613673473) |
| Charlie | (3, 56086910298885139) |

    You also know that Dave has encrypted the same message $M$ (without padding) using each of these public keys, where the message, which

contains only uppercase and lowercase English letters, is encoded with the method[16] used at [144]. Suppose that Dave's ciphertext messages are the following:

| Recipient | Ciphertext |
|---|---|
| Alice | 4324345136725864 |
| Bob | 2102800715763550 |
| Charlie | 46223668621385973 |

a. Use the Chinese Remainder Theorem to find $M$.

b. Are there other feasible ways to find $M$?

33. As mentioned in this chapter, "textbook" RSA is subject to a forward search attack. An easy way to prevent this attack is to pad the plaintext with random bits before encrypting. This problem shows that there is another RSA issue that is also prevented by padding the plaintext. Suppose that Alice's RSA public key is $(N, e)$ and her private key is $d$. Bob encrypts the message $M$ (without padding) using Alice's public key to obtain the ciphertext $C = M^e \bmod N$. Bob sends $C$ to Alice and, as usual, Trudy intercepts $C$.

    a. Suppose that Alice will decrypt one message of Trudy's choosing, provided that it is not $C$. Show that Trudy can easily determine $M$. Hint: Trudy chooses $r$ and asks Alice to decrypt the ciphertext $C' = Cr^e \bmod N$.

    b. Why is this "attack" prevented by padding the message?

34. Suppose that Trudy obtains two RSA ciphertext messages, both of which were encrypted with Alice's public key, that is, $C_0 = M_0^e \bmod N$ and $C_1 = M_1^e \bmod N$. Trudy does not know Alice's private key or either plaintext message.

    a. Show that Trudy can easily determine $(M_0 \cdot M_1)^e \bmod N$.

    b. Can Trudy also determine $(M_0 + M_1)^e \bmod N$?

    c. Due to the property in part a, RSA is said to be *homomorphic* with respect to multiplication. Recently, a fully homomorphic encryption scheme has been demonstrated, that is, the multiplicative homomorphic property (part a) and the additive homomorphic property (part b) both hold [67]. Discuss some significant potential uses for a practical fully homomorphic encryption scheme.

[16]Note that at [144], letters are encoded in the following nonstandard way: Each lowercase letter is converted to its uppercase ASCII equivalent, and uppercase letters are converted to (decimal) according to A = 33, B = 34, ..., Z = 58.

35. This problem deals with digital signatures.

    a. How and why does a digital signature provide integrity?

    b. How and why does a digital signature provide non-repudiation?

36. In the context of cryptography,

    a. Define non-repudiation.

    b. Give an example—different from the one given in this chapter—where non-repudiation is critical.

37. A digital signature or a MAC can be used to provide a cryptographic integrity check.

    a. Suppose that Alice and Bob want to use a cryptographic integrity check. Which would you recommend that they use, a MAC or a digital signature? Why?

    b. Suppose that Alice and Bob require a cryptographic integrity check and they also require non-repudiation. Which would you recommend that Alice and Bob use, a MAC or a digital signature? Why?

38. Alice wants to be "extra secure," so she proposes to Bob that they compute a MAC, then digitally sign the MAC.

    a. Does Alice's method provide a cryptographic integrity check? Why or why not?

    b. Does Alice's method provide for non-repudiation? Why or why not?

    c. Is Alice's method a good idea? Why or why not?

39. In this chapter, we showed that you can prevent a forward search attack on a public key cryptosystem by padding with random bits.

    a. Why would we like to minimize the amount of random padding?

    b. How many bits of random padding are needed? Justify your answer.

    c. Other than padding, is there another simple and practical method for preventing a forward search attack?

40. Consider the elliptic curve

$$E: \quad y^2 = x^3 + 7x + b \pmod{11}.$$

    a. Determine $b$ so that the point $P = (4, 5)$ is on the curve $E$.

    b. Using the $b$ found in part a, list all points on $E$.

c. Using the $b$ found in part a, find the sum $(4,5)+(5,4)$ on $E$.

d. Using the $b$ found in part a, find the point $3(4,5)$.

41. Consider the elliptic curve

$$E: \quad y^2 = x^3 + 11x + 19 \pmod{167}.$$

a. Verify that the point $P = (2,7)$ is on $E$.

b. Suppose this $E$ and $P = (2,7)$ are used in an ECC Diffie-Hellman key exchange, where Alice chooses the secret value $A = 12$ and Bob chooses the secret value $B = 31$. What value does Alice send to Bob? What does Bob send to Alice? What is the shared secret?

42. The Elgamal digital signature scheme employs a public key consisting of the triple $(y,p,g)$ and a private key $x$, where these numbers satisfy

$$y = g^x \bmod p. \tag{4.9}$$

To sign a message $M$, choose a random number $k$ such that $k$ has no factor in common with $p-1$ and compute

$$a = g^k \bmod p.$$

Then find a value $s$ that satisfies

$$M = xa + ks \bmod (p-1)$$

which is easy to do using the Euclidean Algorithm. The signature is verified provided that

$$y^a a^s = g^M \bmod p. \tag{4.10}$$

a. Select values $(y,p,g)$ and $x$ that satisfy equation (4.9). Choose a message $M$, compute the signature, and verify that equation (4.10) holds.

b. Prove that the math in Elgamal works, that is, prove that equation (4.10) always holds for appropriately chosen values. Hint: Use Fermat's Little Theorem, which states that if $p$ is prime and $p$ does not divide $z$, then $z^{p-1} = 1 \bmod p$.