Preface

Hardly a day passes without a headline announcing that a data breach has exposed the private details of millions of customers, or that ransomware has taken the entirety of a local government's information technology infrastructure hostage, or that a *supply-chain* attack has infected thousands of computers and caused untold damage.

Reports of such occurrences are so common we have become inured to them. If we are lucky, we count ourselves among the unaffected and rebury our heads in the sand. If we are conscientious, we realize that next time we might not be so lucky; that eventually, inevitably, it is our personal data that will be leaked and possibly put to nefarious use.

Many experts liken security on the internet to that of the *wild west*. Considering that most (if not *all*) the devices we cannot fathom living without are connected to the internet—this includes not just laptops and smartphones, but also the explosion of *IoT* devices such as home assistants, refrigerators, doorbells and even automobiles—the *wild west* is far too tame a comparison; if not in terms of lethality, then certainly in terms of the number of potential threats. And although seldom lethal, the consequences of victimization can be painful; say, for example, if your identity is stolen and used to take out loans in your good name.

Given this bleak picture, one might wonder how it is we have managed to make it this far without being robbed penniless. How is it, for example, that we are able to transfer money from savings to checking using our bank's website, make purchases on amazon.com with our credit cards, transmit payment via *Zelle* or *Venmo* to our cleaning service; all over the internet with nary a glitch?

The answer, in a word, is *cryptography*. Cryptography is the fortress that stands between order and chaos in our irrevocably interconnected world. Cryptography protects not just individuals from hackers who would drain their bank accounts, but also civilization against takedowns of the power grid, or takeovers of the nuclear arsenal, by hostile state actors.

Why are the vast majority of us unaware of, or at best indifferent to, this miracle of cryptography? Because nobody pays attention to stuff that works, and therefore there's little incentive for the news and other media to cover it. Recent advances in quantum computing may change this picture. You can bet the instant a quantum computer breaks the classical encryption that secures our bank accounts, power grids and nuclear arsenals, people will start paying attention.

Cryptography is as ancient as the pyramids of Giza. Translated from the Greek, it means *secret writing*. For centuries cryptography was concerned entirely with the encryption of messages into coded ciphers, in order to keep them secret from the prying eyes (or ears) of adversaries. And for centuries it was confined largely to the domain of kings, generals and other state-level actors. But with the advent of the digital age, sometime around the middle of the 20th century, cryptography began to spill into the public domain—our digital world demanded it. Today cryptography affords security and privacy protections to every individual who uses a computer, smartphone or internet-connected device.

Cryptography is a branch of *cryptology* (in Greek, the *study* of secrets), which includes not just code-*making* (the stuff of cryptography) but also code-*breaking*. The code-breaking branch of cryptology is

called *cryptanalysis*. One might be tempted to think cryptanalysis is the province of spies and hackers, and only the *good guys* do cryptography.

This is only partially true. Cryptographers (code-makers) rely on cryptanalysts (code-breakers) to analyze the quality of the ciphers they design; they do this by trying to break the ciphers. If the cryptanalysts can break the ciphers, then the cryptographers have to go back to the drawing board to fix them before they are deployed in sensitive applications. This is a good thing if you are one of the good guys.

Cryptography is itself subdivided into two main branches. These are *secret-key* cryptography and *public-key* cryptography. The former is what most people think of when they hear the words *cryptography*, *encryption* or *cipher*. Indeed, secret-key cryptography was the standard of encryption for nearly four-thousand years, starting with the hieroglyphs carved into cave walls by the ancient Egyptians.

Public-key cryptography is a much more recent, and arguably more interesting, invention. It emerged with the advent of the digital computer, and later the internet; technologies that demanded encryption at a scale much larger than traditional, secret-key ciphers could provide. Without the improbable invention of public-key cryptography in the late 1970s, the internet as we know it would not exist; there would be no ecommerce, no online banking, and no *Facebook* or *Google*, either (at least not as we know and love them today).

It is the subject of public-key cryptography that this short book is about. I hope you find it as fascinating to read as I did to write.

Introduction

Early in the fall semester of 1974, an undergraduate at the University of California, Berkeley was required to submit two proposals for his term project in CS-244, a course on computer security offered by the department of Computer Science. In the first, the student proposed to explore the largely picked-over subject area of data compression. In the second he was considerably more ambitious—he proposed to solve a flaw common to all secret ciphers; a problem that had vexed cryptographers for over four-thousand years, and due to which no small number of kings and generals had (literally) lost their heads.

The student, a senior by the name of Ralph C. Merkle, was only lukewarm about the project concerning data compression. Devoting just two sentences to it in the proposal, he wrote, "*At this point, I must confess, that I am not entirely thrilled by the prospect of engaging in this project, and will expand on it only if prodded.*" The other proposal, the one that would fix secret ciphers, occupied seven type-written pages.

Unfortunately for Merkle, his professor, a distinguished member of the Computer Science faculty named Lance Hoffman, was not impressed. In the margin of Merkle's proposal he scribbled, "*Project 2 [data compression] looks more reasonable, maybe because your description of Project 1 [secret ciphers] is muddled terribly.*"

Undeterred, Merkle submitted a second, much simplified proposal for his preferred topic, this time condensing it to just over a single page. Again, Hoffman scribbled, "*It's getting there, but not good enough yet,*" and then, in a flair of professorial snobbery, "*All bloat,*" referring to all but the final paragraph of the revised proposal.

Most college seniors in Merkle's position would have caved, wont as they are late in their undergraduate careers simply to drag themselves over the finish line. But not only was Merkle sufficiently attached to his idea not to abandon it, he dropped Hoffman's course in order to pursue it.

The following summer, having completed the project on his own, Merkle submitted it for review to the editor of the *Communications of the Association for Computing Machinery*, a prestigious journal whose audience consisted of academics and professionals at the forefront of the field.

The response, solicited by the journal's editor from an "*experienced cryptography expert*," was disappointing. Among other things, it offered, "*I am sorry to have to inform you that the paper is not in the mainstream of present cryptography thinking and I would not recommend that it be published in the Communications of the ACM*." As if to pile on, the editor herself remarked that she too was bothered, particularly "*by the fact that there are no references to the literature.*"

In the editor's remark Merkle took some encouragement. The idea was Merkle's, after all—the reason there were no references is because none existed! Moreover, the criticisms confirmed to Merkle not only that his idea was a novel one, but one for which he would receive sole credit…if only, somehow, he could make his critics see the light.

Merkle continued to badger the ACM until finally, in April 1978, nearly four years after first floating his idea to professor Hoffman, it published his paper.[1]

Recalling the years-long effort to overthrow his skeptics, Merkle quoted the renowned biologist Peter Medawar, who said, "*The human mind treats a new idea the same way the body treats a strange protein; it rejects it.*"

Alas, publication did not earn Merkle enduring recognition. Today, few outside the field of cryptography remember his name. To their great credit, Whitfield Diffie and Martin Hellman, widely considered to be the fathers of public-key cryptography, graciously cite Merkle in their seminal treatise on the subject.

[1] Merkle, R. C. (April 1978). Secure Communications over Insecure Channels. *Communications of the ACM*. **21** (4): 294-299.

Chapter 1

1.1 Keys

The need to communicate in secret is as old as communication itself.[2] The time-honored approach is to scramble a message before sending it, which the receiver unscrambles on receipt. Without knowledge of the method by which the message is scrambled, unauthorized parties who intercept the message cannot unscramble it. The method of scrambling, and unscrambling, is generally referred to as the *key*.

The key can be a mechanical device, a number, a puzzle—anything possessed by both sender and receiver that enables the sender to encipher (or *encrypt*), and the receiver to decipher (or *decrypt*), a message.

Let's consider the example of a *substitution* cipher; this will provide a sense of how encryption works, while at the same time introducing some terminology.

Suppose Alice wants to send the message *hello* to Bob. Meanwhile, Eve plots to intercept the message.[3] In its plain form, the message *hello* is called a *plaintext*. To keep Eve from reading the message, Alice must transform it to a *ciphertext* before sending it to Bob.

Plaintext	***h***	***e***	***l***	***l***	***o***	
Key	***s***	***e***	***c***	***r***	***e***	***t***
Ciphertext	***a***	***j***	***o***	***d***	***t***	

Figure 1. Encryption using a substitution cipher

Figure 1 illustrates this transformation, which turns the plaintext *hello* into the ciphertext *ajodt*. The transformation is done using an *encryption algorithm*. The algorithm takes Alice's plaintext (*hello*) and key (*secret*) as input, and produces the ciphertext (*ajodt*) as output.

The algorithm works as follows: it converts every letter in both the plaintext and the key to a number, such that $a = 1$, $b = 2$, $c = 3$ and so on. Then, to each converted letter of the plaintext, it adds the converted letter of the key. For example, starting with the first letter of the message h (8), it adds the first letter of the key s (19), which results in $8 + 19 = 27$. This process is repeated for each letter in the plaintext (note the t in *secret* is thrown out because the plaintext is one letter shorter than the key). Applying these rules for each letter in the plaintext until its end is reached, the algorithm outputs the ciphertext *ajodt*.

But wait, how did the sum of the first letters, 27, become a, which is the first letter of the alphabet, not the 27^{th}? Well, to address cases in which the plaintext and key letters sum to a number greater than the size of the alphabet, the algorithm uses a technique called *modular reduction*.

[2] King to general, via trusted courier: *Attack at dawn!*
[3] The examples throughout this book feature the cast of characters ubiquitous in the literature: Alice, Bob and Eve.

It works like this: When the end of the alphabet is reached, where $z = 26$, the algorithm *wraps around* to a again. Thus $a = 27$, $b = 28$, $c = 29$ and so on. Then, if we subtract 26 (which is called the *modulus*) from each of these numbers, we get back the original numbers; $a = 1$, $b = 2$ and $c = 3$.

Ciphertext	***a***	***j***	***o***	***d***	***t***	
Key	***s***	***e***	***c***	***r***	***e***	***t***
Plaintext	***h***	***e***	***l***	***l***	***o***	

Figure 2. Decryption using a substitution cipher

On receiving the ciphertext from Alice, Bob uses the same algorithm Alice used, only he *subtracts* the letters of the key from those of the ciphertext to recover the plaintext (whereas Alice *added* the letters of the plaintext to those of the key to produce the ciphertext). Note that subtracting the first letter of the key s (19) from the first letter of the ciphertext a (1) results in -18. To recover h (8), Bob *adds* -18 to the modulus 26; that is, $-18 + 26 = 8$.

Where does this leave Eve? Well, suppose Eve intercepts the ciphertext *ajodt*. From the ciphertext alone, how is Eve to recover the plaintext *hello*? For that she needs the key. But Alice never sent the key to Bob, precluding Eve from intercepting it. This means that Eve has no way of decrypting the message. Even if she knows *how* the message was encrypted—that is, by adding together the numeric values of letters—without the key, Eve cannot decrypt the message.

As a matter of historical fact, before the 19th century the idea that an adversary would be permitted knowledge of an encryption algorithm was anathema to cryptographic orthodoxy. But that changed around 1883, when Auguste Kerckhoffs asserted that, "A cryptosystem should be secure even if everything about the system, except the key, is public knowledge". Prior to this, the efficacy of ciphers depended on the secrecy of keys *and* algorithms.

One important implication of Kerckhoffs's principle is that a cipher whose algorithm is publicly known will expose it widely to attacks, and that this is the surest way to test the efficacy of the algorithm. Indeed, nearly all cryptographic implementations in use today are fully open, and are still in use because all attempts to defeat them have thus far failed.

To formalize what we've discussed so far, we can say that for two parties to communicate securely over an *insecure* channel, each must possess a key that can be used to encipher and decipher messages transmitted over the insecure channel.[4] This key must be possessed only by the parties authorized to send and receive the messages; otherwise an eavesdropper with possession of the key will be able to read them.

The substitution cipher we just explored is an example of *symmetric-key* cryptography. It is *symmetric* because the key used by both sender (to encrypt) and receiver (to decrypt) messages is the same. But this is a book about *public-key* cryptography (which, unsurprisingly, is also known as *asymmetric-key* cryptography). We covered the symmetric variety here because it will provide useful context to the public-key concepts we are about to explore.

[4] The quintessential example of such an insecure channel is the public internet.

Chapter 2

2. 1 The Key Distribution Problem

The requirement that only the parties authorized to participate in a secure conversation share a secret key poses a problem: How is the key distributed to the authorized parties securely; that is, without it being stolen by an unauthorized party? For this you need a *secure* channel. Transmitting the key over an insecure channel is not an option, because if the channel is insecure the key can be stolen. Neither is enciphering the key before transmitting it, since you cannot decipher it without first having the key.

The most obvious and effective solution is to hand-deliver the key in advance to the party you wish to communicate with. But this is also the least efficient solution, and not at all practical in the internet age.

Suppose, for example, there are n parties that need to communicate with each other securely, and that each communication requires a separate key. The number of keys required for a group of n participants to communicate securely is given by the formula $(n^2 - n) \div 2$. For a group of 10 participants, the number of keys required is $(10^2 - 10) \div 2$, or 45; for a group of 100 the number of keys is 4950. As the number of participants increases, the number of keys grows quadratically.

2.2 A Clever Solution

In 1976, two Stanford University cryptographers proposed an elegant solution to the key distribution problem in a groundbreaking academic article titled *New Directions in Cryptography*.[5] This solution became, and remains to this day, the de facto standard for exchanging keys securely over public (i.e., insecure), digital communication channels. It is commonly known as the *Diffie-Hellman key exchange protocol*, or DH for short.

Basically, DH enables previously unacquainted parties to exchange public information over an insecure channel, and then to combine it with private information—information that each possesses independently—to compute a shared, symmetric key. Because neither the shared key, nor the private information used to compute it, are ever transmitted over the insecure channel, the shared key cannot be observed (or otherwise derived) by an eavesdropper.

2.3 Essential Diffie-Hellman

DH can be implemented by means of a number of algorithms. Most examples in the literature cite the original implementation, which employs a *multiplicative group of integers modulo* p (where p is a prime number) to demonstrate DH. This is unfortunate, because the mathematics of multiplicative groups modulo *p* are not immediately intuitive.

The graphic in *Figure 3* demonstrates DH using a much simpler algorithm: *multiplication*. Assume Alice wants to perform a secure key exchange with Bob over an insecure channel. Meanwhile, Eve observes all traffic passing between Alice and Bob; as ever, presumably for malicious purposes.

[5] Diffie, Whitfield; Hellman, Martin E. (November 1976). New Directions in Cryptography. *IEEE Transactions on Information Theory*. **22** (6): 644-654.

Step	Alice	Eve	Bob	Computations
1	2	→		
2		2	2	
3	3			
4	6	→		$3\,x\,2 = 6$
5		6	6	
6			4	
7		←	8	$4\,x\,2 = 8$
8	8	8		
9	**24**		**24**	$8\,x\,3 = \mathbf{24}$, $6\,x\,4 = \mathbf{24}$

Figure 3. Diffie-Hellman key exchange using multiplication

In steps 1 and 2, Alice selects a random integer (2) and transmits it to Bob.[6] Let's call this number the *generator*, because it will be used by Alice and Bob to generate another number; namely, by multiplying the generator by a private number each will select independently. Because the channel is insecure, Eve observes the value of the generator (2).

In steps 3, 4 and 5, Alice selects another random integer (3), multiplies it by the generator (2), and transmits the product of the multiplication (6) to Bob. Let's call this second random number Alice selects her *private* key, and the product of its multiplication by the generator her *public* key (public because it can be observed by Eve). As expected, Eve observes Alice's public key (6). But Eve does not observe Alice's private key (3), because Alice never transmits her private key to Bob.

In steps 6, 7 and 8, Bob selects a random integer (4), multiplies it by the generator Alice sent to him (2), and transmits the product of the multiplication (8) to Alice. These are Bob's private and public keys. Eve observes Bob's public key (8). Eve now knows the generator (2), Alice's public key (6) and Bob's public key (8), but she does not know Alice's or Bob's private keys (3 and 4).

The magic of DH appears in step 9. Alice multiplies Bob's public key (8) by her private key (3). The product of this multiplication is 24. Similarly, Bob multiplies Alice's public key (6) by his private key (4). The product of this multiplication is also 24. By using a combination of public and private information in this clever way, Alice and Bob have agreed that the number 24 will be the shared key with which to encrypt and decrypt messages sent between them.

Where does this leave Eve? Having only seen the value of the generator (2), Alice's public key (6) and Bob's public key (8), but neither Alice's nor Bob's private keys (3 and 4), Eve does not know by what numbers Alice's and Bob's public keys were multiplied to compute the shared encryption key (24). Note that Alice and Bob could have selected *any* private keys (besides 3 and 4, respectively) and the effect in step 9 would have been the same: they would have computed identical secret keys.

Of course, in this highly simplified implementation, Eve can easily guess Alice or Bob's private keys, and with either private key she can compute the shared key and decrypt the messages.

[6] Since computers operate on numbers—and even more specifically *integers* in cryptographic implementations—again we use integers to represent messages and keys.

Recall from *Chapter 1* that we must assume Eve knows the algorithm used by Alice and Bob to compute the shared key (i.e., *multiplication* of the generator by a private key). With this knowledge, Eve can simply divide Alice's public key (6) by the generator (2), both of which she observed, to derive Alice's private key (3).

Recall as well from *Chapter 1* that the encryption algorithm used *addition* to encrypt, and *subtraction*—which is the *inverse* of addition—to decrypt messages. By using division in this example, Eve is performing the inverse of the multiplication Alice used to generate her public key. Similarly, Eve can divide Bob's public key (8) by the generator (2) to derive Bob's private key (4). The important point is that, with either Alice's or Bob's private key, Eve can compute the shared key and use it to decrypt messages between Alice and Bob.

It should not be surprising that a DH implementation that uses multiplication to generate shared keys is not secure. For an effective DH implementation, we need to make the task of guessing Alice and Bob's private keys more difficult for Eve.

2.4 Diffie-Hellman for 6th Graders

Let's look at a second example, using slightly more sophisticated math to make Eve's task more difficult.

Step	Alice	Eve	Bob	Computations
1	2	→		
2		2	2	
3	3			
4	8	←		$2^3 = 8$
5		8	8	
6			4	
7		←	16	$2^4 = 16$
8	16	16		
9	**4096**		**4096**	$16^3 = \mathbf{4096}$, $8^4 = \mathbf{4096}$

Figure 4. Diffie-Hellman key exchange using exponentiation

Instead of using multiplication to generate public keys, this time Alice and Bob use *exponentiation* (exponentiation is really just another way of saying *repeated multiplication*, as in $10^3 = 10\ x\ 10\ x\ 10 = 1000$). Except for the computations, all the steps are the same as in the previous example, so it's not necessary to repeat them here. What is different this time is (a) the algorithm used to compute the keys—exponentiation versus multiplication—and (b) the public parameters observed by Eve.

As before, Eve observes the generator (2), Alice's public key (8) and Bob's public key (16). With this information Eve must be able to compute the shared secret key (4096) to break the encryption. Because Eve knows the algorithm, she knows that Alice raised the generator (2) to the power of some exponent to compute her public key (8). To find that exponent, Eve must solve for *y* in the equation $x^y = z$, where only x and z are known. In the present example, Eve must find the value y in the equation $2^y = 8$. Solving for y in this equation is known as taking the *logarithm* of z.

Taking a logarithm is the inverse of exponentiation, just as division is the inverse of multiplication, which Eve used in the previous example to break the encryption. For small values of z (8 and 16 in the present

example) solving for y is trivial; it simply requires trying consecutive exponents until the right answer is found. For larger values of z, the complexity of Eve's task increases somewhat, but not sufficiently to thwart her attacks. Contrast this with the version of DH using multiplication, in which the complexity of Eve's task remains constant; that is, Eve must divide the value of the generator by that of the public key only once, regardless of the size of z.

With this we get closer to an effective DH implementation, but we're not quite there yet.

2.5 Diffie-Hellman for Undergraduates

Modular exponentiation brings us much closer to the realm of real-world DH.

Step	Alice	Eve	Bob	Computations
1	3			
2	7	→		
3		(3,7)	3	
4			7	
5	3			
6	6	→		$3^3 \equiv 6\ (mod\ 7)$
7		6	6	
8			4	
9		←	4	$3^4 \equiv 4\ (mod\ 7)$
10	4	4		
11	**1**		**1**	$4^3 \equiv \mathbf{1}\ (mod\ 7)$, $6^4 \equiv \mathbf{1}\ (mod\ 7)$

Figure 5. Diffie-Hellman key exchange using modular exponentiation

Modular exponentiation is the same as exponentiation, but with an additional step. This additional step is called taking a *modulus*, which is done using the *modulo* operation. The modulo operation simply finds the remainder after division of two integers. For example, if you take 32 modulo 5, then you divide 32 by 5, which gives you a quotient of 6 (because 5 goes into 32 6 times) and leaves a remainder of 2. The remainder, 2, is the answer. So, $32\ mod\ 5 = 2$. This should look very familiar, as it is the same wrap-around trick used in the substitution cipher illustrated in *Chapter 1*. The only difference here is that we are using multiplication instead of addition prior to taking the modulus.

A quick word about notation: In the ***Computations*** column of *Figure 5* now appears the strange looking triple bar symbol (≡), and not the familiar equal sign (=) seen in previous equations. This symbol means *is congruent to*, as in *3 cubed is congruent to 6 modulo 7* (in step 6 of *Figure 5*, for example). This is not an equation, but rather a *congruence relation*. The triple bar signifies that if you take the terms on both sides of the symbol modulo the number in parentheses, the terms will be equal. Staying with step 6 in *Figure 3*, $3^3 \equiv 6\ (mod\ 7)$ becomes $27 \equiv 6\ (mod\ 7)$; and since 27 divided by 7 and 6 divided by 7 both leave a remainder of 6, they are said to be congruent modulo 7. Likewise, $20 \equiv 6\ (mod\ 7)$, because 20 divided by 7 and 6 divided by 7 also both leave a remainder of 6.

If you compare the graphic in *Figure 5* with the one in *Figure 4*, which uses exponentiation only, you will find the only difference is that in *Figure 5* the modulo step is added to all the calculations.

The modulo operation requires an operand—namely, a number by which to divide in order to find a remainder. We call this operand the *modulus* (or, alternatively, the *divisor*). In this version of DH, Alice must transmit two numbers to Bob instead of one. As before, Alice transmits the generator (3), but she also transmits the modulus (7) that will be used to compute remainders. Both the generator and the modulus are observed by Eve.

The modular exponentiation of real-world DH leads to a very interesting property of the keys it generates. Compare the values of the public and shared keys in this example (6, 4 and 1) to those of the example in *Figure 4* (8, 16 and 4096). With modular exponentiation the keys are smaller; notably, they are confined to the set of positive consecutive integers 1 to 6.

Note that exponentiation of a generator g modulo p, where g is greater than 1 and less than p, and p is a prime number, guarantees the result will be in the set 1 to $p-1$; in the present example, where $g = 3$ and $p = 7$, this set contains the integers $1, 2, 3, 4, 5,$ and 6.

The graphic in *Figure 6* should clarify this. The values in the ***Power*** column are the result of raising the generator (3) to the power of the values in the ***Exponent*** column, and the values in the ***Remainder*** column are the result of performing the modulo operation on the corresponding value in the ***Power*** column. Note that all our keys (6, 4 and 1) are present in the ***Remainder*** column.

Exponent	Power	Remainder	Computations
1	3	3	$3^1 \equiv 3 \ (mod\ 7)$
2	9	2	$3^2 \equiv 2 \ (mod\ 7)$
3	27	6	$3^3 \equiv 6 \ (mod\ 7)$
4	81	4	$3^4 \equiv 4 \ (mod\ 7)$
5	243	5	$3^5 \equiv 5 \ (mod\ 7)$
6	729	1	$3^6 \equiv 1 \ (mod\ 7)$
7	2187	3	$3^7 \equiv 3 \ (mod\ 7)$
8	6561	2	$3^8 \equiv 2 \ (mod\ 7)$
9	19683	6	$3^9 \equiv 6 \ (mod\ 7)$
10	59049	4	$3^{10} \equiv 4 \ (mod\ 7)$
11	177147	5	$3^{11} \equiv 5 \ (mod\ 7)$
12	531441	1	$3^{12} \equiv 1 \ (mod\ 7)$

Figure 6. The multiplicative group of integers modulo 7

Figure 6 reveals some more interesting properties of the values in the ***Remainder*** column (recall this is the set our generated keys come from). First, if you read the values in the ***Remainder*** column from top to bottom, you find that the sequence of remainders repeats after a while (3, 2, 6, 4, 5, 1, *3*, *2*, *6*, *4*, *5*, *1*), forming what are known as *finite cyclic groups*. Second, each such group contains every integer in the set 1 to $p-1$, where p is the modulus. Finally, the members of the group are unordered, or *non-monotonic*, relative to the order of the exponents. These properties are formalized in a branch of mathematics called number theory.

To keep things simple, it will suffice to keep the following rules in mind: Given a carefully chosen generator g, and a modulus p that is a prime number, we can generate keys with the aforementioned properties.

A carefully chosen generator is one that generates the entire group of integers in the range 1 to $p-1$, where p is the prime modulus. Any generator that fulfills this property is called a *primitive root*, and the set it produces a *cyclic group*. The rules of multiplicative groups modulo p guarantee that at least one integer in the group 1 to $p-1$ is a primitive root. In the current example, 3 is a primitive root of the cyclic group of integers modulo 7. In real-world DH, the modulus should be a very large, randomly-chosen prime number. An artificially small value is used in the example to keep the math simple. It is these properties that are required for an effective DH implementation.

Recall from the previous example that Eve had to solve for the logarithm of z base x (that is, solve for y in the equation $x^y = z$) to find the shared key; a task of manageable complexity, even for very large values of z. In the finite group of integers modulo p, however, Eve's task becomes intractable given large enough values of p. This additional complexity is due to the *non-monotonicity* of values in a cyclic group. The time complexity of finding a value in an unordered set is $O(n)$, or linear; whereas in an ordered set it is $O(log\ n)$, or logarithmic.

There is no *known*, efficient algorithm for finding logarithms in a group of integers modulo *p* (i.e., finding *y* in the congruence relation $x^y \equiv z\ (mod\ p)$ where the values of x, z and p are known). If p is large enough, the task becomes too computationally expensive to be feasible for an attacker. Formally, this is known as the *discrete logarithm problem* (or DLP), and the efficacy of DH is based, at least in part, on the difficulty of solving it.

Note that it is possible there is some other, as yet unknown (or at least unpublished), way to break DH; that is, besides solving the DLP efficiently. As long as such a possibility exists, cryptologists make a distinction between the DLP and a broader class they call the *Computational Diffie-Hellman problem*, or CDH for short.

2. 5 Diffie-Hellman for Cryptographers

There is one last problem with our implementation, which in the last incarnation mirrors precisely the implementation presented by Diffie and Hellman in *New Directions in Cryptography*. It is known as the *small subgroup* problem, and we will explore (and fix) it here.

Referring back to the example in *Figure 5*, suppose instead of choosing the number 3 for her generator, Alice had instead chosen 2. Then we have a cyclic group that looks like the one in *Figure 7*.

Exponent	Power	Remainder	Computations
1	2	2	$2^1 \equiv 2\ (mod\ 7)$
2	4	4	$2^2 \equiv 4\ (mod\ 7)$
3	8	1	$2^3 \equiv 1\ (mod\ 7)$
4	16	2	$2^4 \equiv 2\ (mod\ 7)$
5	32	4	$2^5 \equiv 4\ (mod\ 7)$
6	64	1	$2^6 \equiv 1\ (mod\ 7)$

Figure 7. The multiplicative subgroup of integers modulo 7 generated by 2

What is different this time is that the size of the cyclic group generated by 2 is now 3 (i.e., 2, 4, 1, *2, 4, 1*); not 6, as was the case when the generator was 3. This smaller group is what is known as a *subgroup* of the full group modulo 7.

Why is this a problem? Because if an attacker can replace the generator of a whole group with one that generates a subgroup, she can reduce the number of keys she must search to find the one used for encryption.

Suppose, for example, Eve had somehow managed to replace Alice's chosen generator with her own generator, 2. Eve would have halved the number of keys she must try to break Alice's and Bob's encryption. Of course, when the size of the full group is only 6, this reduction in group size is moot. But when the size of the group is a 128-bit number, its reduction to a 64-bit number is sufficient to make breaking the encryption a tractable problem.

Toward a Complete Cryptosystem

The public-key cryptosystem conceived by Diffie and Hellman in *New Directions* consists of three distinct but interrelated elements: *secure key exchange*, *encryption* and *digital signatures*. But the article only presented a workable implementation for one of them.

Beyond secure key exchange, encryption was the second component of the public-key cryptosystem envisioned, but not realized, by Diffie and Hellman in their 1976 article. For public-key encryption, they and the rest of the crypto community would have to wait another two years.

Chapter 3

RSA and Public-Key Encryption

While doing academic research at MIT in 1978, Ronald Rivest, Adi Shamir and Leonard Adleman proposed a novel solution for transforming a message from plaintext to ciphertext and back again. This alone would have been an unremarkable feat—it pretty much described every previously known cipher. What *was* remarkable is that their method worked without the upfront exchange of secret keys; the sine qua non of traditional ciphers.[7] Not only had this trick never been accomplished before, unshackling encryption from key exchange offered the potential of securing communications at otherwise impossibly massive scale.

Rivest, Shamir and Adleman published their method in an article titled *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*.[8] In it, they picked up where Diffie and Hellman left off, introducing public-key implementations for encryption and digital signatures.[9] The scheme is known simply by the initials of the surnames of its authors, or RSA. More than 40 years on, the contributions of Rivest, Shamir and Adleman form the basis of the most widely used and battle-tested public-key cryptosystems in the world.

RSA Encryption in a Nutshell

RSA encryption can be understood in terms of a very simple state-transition diagram. In this diagram, M represents a plaintext message, C represents the ciphertext of that message, and the arrow symbol (→) represents a transition from one state—or *form*, if you prefer—to another. The state transitions should be read from left to right.[10]

(1) $M \rightarrow C \rightarrow M$

Here, some plaintext message M is transformed to a ciphertext C, and then back again to its original form M.

For encryption to be useful, the second transformation, from $C \rightarrow M$, must be easy to compute for the intended receiver of the message, but *not* for an eavesdropper.[11]

To see how this requirement is enforced without a shared key, let's look at how the transformations are implemented in RSA. First, let's substitute the term C from diagram **(1)** with a new, equivalent term M^e.

(2) $M \rightarrow M^e \rightarrow M$

---

[7] Traditional ciphers were burdened by the complexity associated with the exchange of shared, secret keys; a problem that was solved by Diffie and Hellman, and that is described in detail in a previous section of this paper.
[8] Rivest, Ronald; Shamir, Adi.; Adleman Leonard. (February 1978). A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM*. **21** (2): 121-126.
[9] Digital signatures are a topic unto themselves, and are the subject of a subsequent section of this paper.
[10] Keep in mind the message M here is in fact a numeric representation of the message fit for computer consumption (e.g., ASCII or UTF8).
[11] As a public-key scheme, it is assumed that all parties—trusted or not—possess the public key required for the first transformation, from $M \rightarrow C$ but that only the receiving party possesses the private key required for the second, from $C \rightarrow M$.

Here, the term M^e denotes that we are raising the message M to the power of some positive exponent e to transform it to ciphertext.

Now let's substitute the rightmost term M in the previous diagram with the equivalent term $(M^e)^d$.[12] Again, in diagrams **(1)** and **(2)**, the leftmost M is equal to the rightmost M, the latter of which having been recovered from its ciphertext form C.

(3) $M \rightarrow M^{\mathrm{e}} \rightarrow (M^e)^d$

Here, the term $(M^e)^d$ denotes that we are raising the term M^e, which we have now substituted for the previous term C, to the power of some other positive exponent d.

If we accept that each of the preceding state-transition diagrams is equivalent, just with a substitution of terms, then by definition the terms M and $(M^e)^d$ must also be equivalent; that is, $M = (M^e)^d$.

But how can this be? If e and d are positive exponents of the base term M, then it must be the case that $M < M^e < (M^e)^d$ and, therefore, M cannot be equal to $(M^e)^d$.

In fact they cannot be equal in the infinite domain of integers. But in a *finite* domain, such as the *cyclic group* introduced in our discussion of DH, such an equivalence is very possible indeed.

To see how, let's rewrite the state transition diagram using modular arithmetic to confine the results to a finite set.

(4) $M \rightarrow M^e \ mod\ n \rightarrow (M^e)^d \ mod\ n$

It should be clear now that the leftmost term M can in fact be equal to the rightmost term $(M^e)^d$, *if* it is reduced modulo some *n* that gets us back to M.

To make the point concrete, let's plug some real values into the diagram, where the integer 5 represents the plaintext message M, 77 represents the modulus n, and the integers 7 and 43 represent the exponents e and d, respectively.

(5) $5 \rightarrow 5^7 \ mod\ 77 \rightarrow (5^7)^{43} \ mod\ 77$
(6) $5 \rightarrow 47 \rightarrow 5$

If there is any doubt the rightmost term is 5, this math can easily be verified on any calculator with a *mod* function.

This improbable round trip, from M to C back to M again, is what is known as a *one-way, trap-door* function. It is one-way because it is easy to transform from $M \rightarrow C$. But the inverse transformation, from $C \rightarrow M$, is easy only if some trap-door information is known. In this case, that trap-door information is the exponent d.

Circling back now to the requirement that the transformation from $C \rightarrow M$ be easy to compute for the message's recipient, but not for the eavesdropper, this property is enforced by confining knowledge of the exponent d—*43* in diagram featuring real values—to just the message's recipient.

[12] Note that the forms $(M^e)^d$, $M^{e\times d}$, M^{ed} are all equivalent. Moreover, due to the commutative property of multiplication, reversing the order of the exponents—as in $(M^d)^e$, $M^{d\times e}$ and M^{de}—does not affect the results.

If it weren't obvious, not just any exponents e and d, or modulus n, will do for these equivalences to hold, and it is in the derivation of these parameters that the sausage of RSA is made. But before exploring the details of how e, d and n are derived, let's apply what we've learned thus far to an example in which Bob transmits an encrypted message to Alice.

Textbook RSA

Figure 5 depicts what is referred to in the literature as *Textbook RSA*.[13] As with previous examples, the parameters are made artificially small to keep the mathematics manageable.

				Parameters		
Step	**Alice**	**Eve**	**Bob**	**Public**	**Private**	**Computations**
1	7				p	
2	11				q	
3	77			n		$7\ x\ 11\ =\ 77$
4	60				$\varphi(n)$	$(7-1)(11-1)=60$
5	7	→		e		$GCD(7,60)=1$
6		(7, 77)	7	(e,n)		
7			77			
8	43				d	$7\ x\ 43 \equiv 1\ (mod\ 60)$
9			**5**		M	
10		←		C		$47 \equiv 5^7\ (mod\ 77)$
11	47	47				
12	**5**				M	$5 \equiv 47^{43}\ (mod\ 77)$

Figure 5. Encryption and decryption using textbook RSA

If you read the previous section, the last four steps (9 through 12) of *Figure 5* should look familiar. But let's restate them anyway to put them into the context of this example. Bob transforms a plaintext message M to a ciphertext message C, which he transmits to Alice; whereupon Alice transforms the ciphertext C back to the original plaintext M. Meanwhile, Eve observes the ciphertext C, but without the decryption exponent d, she cannot recover the plaintext M. To understand how Alice and Bob generated the keys—that is, the exponents and the modulus—to encrypt M and decrypt C, however, we need to back up to the beginning.

In steps 1, 2 and 3, Alice selects two random integers, p and q (7 and 11), and multiplies them to produce a modulus n (77). Alice's requirements for a suitable n are that its factors p and q both be prime numbers, and that the numeric value of the message she encrypts be smaller than n.[14]

In step 4, Alice applies *Euler's totient function* to the modulus n, denoted symbolically as $\varphi(n)$.[15] Euler's totient function tells us the number of integers in the set from 1 to n with which n is *relatively prime* (or

[13] The *textbook* model provides a good theoretical basis for understanding RSA encryption, but it lacks some features that would be required for a cryptographically secure implementation.

[14] In the real world the values of p and q should be very large, randomly chosen and distinct primes of roughly equal bit length.

[15] For now, just accept that this is part of the RSA recipe. Euler's totient function, and other details concerning the derivation of keys and moduli, will be covered in detail in the next section.

coprime). For two integers to be relatively prime, their *greatest common divisor* (abbreviated GCD) must be equal to 1.[16]

For any prime number p, the totient of p, or $\varphi(p)$, is $p - 1$. It should be clear why this is so. If p is prime, we know that the only integers that divide it evenly are 1 and p itself. Therefore, every integer in the set 1 to $p - 1$ must be relatively prime to p.

But here Alice applies Euler's totient function to the *composite* modulus n, an integer that is the product of two prime numbers, p and q.[17] The formula thus becomes the product of the totients of n's prime factors p and q, or $\varphi(n) = \varphi(p) \times \varphi(q) = (p - 1)(q - 1)$. Plugging in the values from the example we get $(7 - 1)(11 - 1) = 6\ x\ 10 = 60$.

That is, Euler's totient function tells us there are 60 integers in the set from 1 to 77 that are relatively prime to 77. Indeed, the number of pairs (a, n) in *Figure 6* is whose GCD = 1 is 60 (the table has been truncated to conserve space). As with the values of p and q, Alice must keep the totient $\varphi(n)$ secret.

a	**N**	$\boldsymbol{GCD(a, n)}$
1	77	1
2	77	1
3	77	1
4	77	1
5	77	1
6	77	1
7	**77**	**7**
8	77	1
9	77	1
10	77	1
11	**77**	**11**
12	77	1
13	77	1
14	**77**	**7**
15	77	1
…	…	…
77	**77**	**77**

Figure 6. Euler's totient function for values of 'a' relatively prime to 'n'

In step 5, Alice chooses her encryption exponent *e*. It will be helpful here (and in step 8, where Alice chooses her decryption exponent d) to refer to *Figure 7*. Alice's requirements for a suitable e are that it be greater than 1, and that it be relatively prime to $\varphi(n)$; that is, it must satisfy the equation $GCD(e, \varphi(n)) = 1$. The latter condition guarantees that e has a multiplicative inverse in the finite

[16] For example, the GCD of 4 and 9 is 1, because there is no integer greater than 1 that divides both 4 and 9 evenly. Therefore, whereas neither 4 nor 9 is *individually* prime, together they are *relatively* prime because their GCD is 1.
[17] An integer that is not prime is a *composite* integer. In addition to being composite, an integer that is the product of exactly two prime numbers is called a *semiprime* integer. Semiprime integers can only be divided evenly by 1, the two primes multiplied to produce it (its *factors*), and the product itself. In the present example these factors are 1, 7, 11 and 77.

group of integers modulo $\varphi(n)$.[18] Referring to the first and third columns in *Figure 7*, we see that 7 meets both requirements, so that is the value Alice chooses for e.

Exponents (e, d)	**Totient** $\varphi(n)$	**Coprime?** $GCD(e, \varphi(n))$	**Inverse?** $ed = 1\ (mod\ \varphi(n))$	**Computations**
1	60	1	7	$7\ x\ 1 \equiv 7\ (mod\ 60)$
2	60	2	14	$7\ x\ 2 \equiv 14\ (mod\ 60)$
3	60	3	21	$7\ x\ 3 \equiv 21\ (mod\ 60)$
4	60	4	28	$7\ x\ 4 \equiv 28\ (mod\ 60)$
5	60	5	35	$7\ x\ 5 \equiv 35\ (mod\ 60)$
6	60	6	42	$7\ x\ 6 \equiv 42\ (mod\ 60)$
7	**60**	**1**	**49**	$\mathbf{7\ x\ 7 \equiv 49\ (mod\ 60)}$
8	60	4	56	$7\ x\ 8 \equiv 56\ (mod\ 60)$
9	60	3	3	$7\ x\ 9 \equiv 3\ (mod\ 60)$
10	60	10	10	$7\ x\ 10 \equiv 10\ (mod\ 60)$
11	60	1	17	$7\ x\ 11 \equiv 17\ (mod\ 60)$
…	…	…	…	…
43	**60**	**1**	**1**	$\mathbf{7\ x\ 43 \equiv 1\ (mod\ 60)}$

Figure 7. Finding the multiplicative inverse of 7 in the group of integers modulo 60

Having selected her encryption exponent, in steps 6 and 7 Alice transmits it, along with the modulus—which together comprise her public key (e, n)—to Bob, which Bob can then use to encrypt messages to Alice.

In step 8, Alice chooses her decryption exponent *d*. The requirement for *d* is that it be the inverse of *e* in the finite group of integers modulo *φ(n)*; that is, it must satisfy the equation $ed = 1\ (mod\ \varphi(n))$. Looking at the first and fourth columns in *Figure 7*, we see that 43 is the value that meets this requirement, because $7\ x\ 43 \equiv 1\ (mod\ 60)$ (the table has again been truncated to conserve space).[19]

But where does this leave Eve? Eve has observed the value of the encryption key (e, n) and the ciphertext. But she has not observed the prime factors p and q, and therefore cannot efficiently compute the totient $\varphi(n)$. Without $\varphi(n)$, Eve cannot derive the private exponent d and thereby break the encryption.

The reason Eve cannot efficiently compute the totient $\varphi(n)$ is because there is no known, efficient way to derive the prime factors of large, composite integers. This is known as the *integer factorization*

[18] The term *multiplicative inverse* here means the value by which the encryption exponent e must be multiplied to yield $1\ mod\ n$, where n is the number of elements in (or the *order* of) the group. For example, in the real numbers, the multiplicative inverse of 3 is $1 \div 3$ (or .333), because $(3\ x\ 1) \div 3\ =\ 1$. But in an integer group the rules are different; notably, fractional numbers are not allowed. So, the inverse of an element is the factor that, when multiplied by the element, equals $1\ mod\ n$.

[19] Given the artificially small values used in the example, it is possible to demonstrate the derivation of e and d with the aid of a simple table like the one in *Figure 7*. For the much larger values used in the real world, however, we need a better way. That better way is the *Extended Euclidean Algorithm*, which provides an efficient means of finding inverses in large-order multiplicative groups.

problem. In the context of the present example, this means that given the modulus n (77), Eve must determine that its factors are p (7) and q (11) in order to break the encryption.[20]

An Informal Proof

In the previous section we learned that the encryption and decryption exponents—e and d, respectively—are inverses in the group of integers modulo $\varphi(n)$. We also learned that M and C, when raised to the power of these inverses, are reversible transformations in a different group; the group of integers modulo n. But what accounts for the relationship between these two groups; the relationship that allows us to assert that $M^{ed} \equiv M \ (mod\, n)$ for correctly chosen values of e, d and n?

For the answer we return to the mathematician Leonhard Euler, who in the year 1763 proved the following congruence relation, which holds that for any integer M that is relatively prime to n,

(1) $M^{\varphi(n)} \equiv 1 \ (mod\, n)$

This congruence, which features in its exponent another of Euler's inventions—the totient function introduced in the previous section—is known as *Euler's theorem*.

Let's test Euler's theorem using a very small, conceptually manageable value for n. For the purposes of RSA, recall that n must be the product of two distinct primes, p and q. So here we choose the integer 15 for our modulus, which is the product of the two primes 3 and 5.

$\mathbf{1^8 \equiv 1 \ (mod\ 15)}$
$\mathbf{2^8 \equiv 1 \ (mod\ 15)}$
$\mathbf{3^8 \equiv 6 \ (mod\ 15)}$
$\mathbf{4^8 \equiv 1 \ (mod\ 15)}$
$\mathbf{5^8 \equiv 10 \ (mod\ 15)}$
$\mathbf{6^8 \equiv 6 \ (mod\ 15)}$
$\mathbf{7^8 \equiv 1 \ (mod\ 15)}$
$\mathbf{8^8 \equiv 1 \ (mod\ 15)}$
$\mathbf{9^8 \equiv 6 \ (mod\ 15)}$
$\mathbf{10^8 \equiv 10 \ (mod\ 15)}$
$\mathbf{11^8 \equiv 1 \ (mod\ 15)}$
$\mathbf{12^8 \equiv 6 \ (mod\ 15)}$
$\mathbf{13^8 \equiv 1 \ (mod\ 15)}$
$\mathbf{14^8 \equiv 1 \ (mod\ 15)}$
$\mathbf{15^8 \equiv 0 \ (mod\ 15)}$

Figure 8. Remainders modulo 15 for M to the power of 8

[20] Given a small, semiprime modulus n—e.g., 77 in the present example—finding its factors is trivial. For very large values of n, however, factorization is difficult. As always, small values are used in the examples to keep the math simple.

The first thing to note is that the totient of 15 is 8; or, put another way, there are 8 integers in the set 1 to 15 that are relatively prime to 15 (these are 1, 2, 4, 7, 8, 11, 13 and 14). Therefore, to see the effect of Euler's theorem, we must raise each member of the set 1 to 15 to the power of 8 modulo 15.[21]

The second thing to note is that just those values of M that are relatively prime to 15 yield 1 modulo 15 when raised to the power of 8; the rows in **bold font** in *Figure 8*. The other rows, those with values of M *not* relatively prime to 15, instead yield a multiple of one of the two prime factors.

Now, let's multiply both sides of Euler's identity by M (additions in **bold font** for clarity).

(2) $M^{\varphi(n)} \times \boldsymbol{M} \equiv 1 \times \boldsymbol{M}\ (mod\ n)$

This cumbersome form can be simplified to the following:

(3) $M^{\varphi(n)+1} \equiv M\ (mod\ n)$

On the righthand side of the identity, $1\ x\ M$ becomes simply M, because $1\ x\ M$ equals M for any M. On the lefthand side of the identity, $M^{\varphi(n)}\ x\ M$ becomes $M^{\varphi(n)+1}$. This is because $M^{\varphi(n)}\ x\ M$ is the same as multiplying M to itself one more time which, due to the rules of exponents, can be expressed by adding 1 to the value of the original exponent.

With this change, let's see what happens to the values in *Figure 8*.

$\mathbf{1^9 \equiv 1\ (mod\ 15)}$
$\mathbf{2^9 \equiv 2\ (mod\ 15)}$
$\mathbf{3^9 \equiv 3\ (mod\ 15)}$
$\mathbf{4^9 \equiv 4\ (mod\ 15)}$
$\mathbf{5^9 \equiv 5\ (mod\ 15)}$
$\mathbf{6^9 \equiv 6\ (mod\ 15)}$
$\mathbf{7^9 \equiv 7\ (mod\ 15)}$
$\mathbf{8^9 \equiv 8\ (mod\ 15)}$
$\mathbf{9^9 \equiv 9\ (mod\ 15)}$
$\mathbf{10^9 \equiv 10\ (mod\ 15)}$
$\mathbf{11^9 \equiv 11\ (mod 15)}$
$\mathbf{12^9 \equiv 12\ (mod 15)}$
$\mathbf{13^9 \equiv 13\ (mod 15)}$
$\mathbf{14^9 \equiv 14\ (mod 15)}$
$\mathbf{15^9 \equiv 15\ (mod 15)}$

Figure 9. Remainders modulo 15 for M to the power of 8 + 1

Now, after exponentiation of M by $\varphi(n) + 1$, all our remainders are again equal to M modulo n! Moreover, the equality holds for *all* values of M, not just those relatively prime to n, which recall is an exception to Euler's theorem as originally presented in equation **(1)**. By simply adding one to the exponent, the exception has been neutralized.

[21] Recall that the totient of a semiprime number is equal to the product of the totients of its two prime factors. That is, if $n = p\ x\ q$, then $\varphi(n) = \varphi(p)\ x\ \varphi(q)$, or $(p-1)(q-1)$. In the present example, if $n = 3\ x\ 5$, then $(3-1)(5-1) = 2\ x\ 4 = 8$.

A quick word about this exception is worth a paragraph of digression. That there are 7 values in the range 1 to 15 in the present example that are exceptions to Euler's theorem might cause one to think it is as much the rule as it is the exception. Therefore, shouldn't we have to prove that adding 1 to the exponent in Euler's identity neutralizes the exception, so that we know RSA will work for any M, not just one that is relatively prime to n? Indeed, where n is 15, there is a 7 in 15 chance we will choose for encryption a message M in the range $0 < M < n - 1$ that violates Euler's theorem. For astronomically large, real-world values of n, however, the number of such exceptions is infinitesimally small (it is in the proportion $(p + q) \div pq$). Indeed, for large enough values of n, the probability of selecting a message M that is a multiple of p or q is so small it is hardly worth considering. Improbable though it may have been, Rivest, Shamir and Adleman were obliged to handle this exception lest their method be judged deficient. Handle it they did, and in so doing cemented RSA as an enduring model of elegance.

With this we have the basis for effective encryption and decryption; that is, an exponent that, when applied to M modulo n, gives us back M. But so far we have only shown how to recover M using a single exponent. To complete the puzzle we need two exponents, e and d. That is, we need to break the single exponent $\varphi(n) + 1$ (9 in the present example) into two exponents; one for encryption and the other decryption. To accomplish this, we need to revisit an identity last seen in the previous section:

(4) $M^{ed} \equiv M (mod\ n)$

Recall that *e* here is our encryption exponent, and *d* our decryption exponent, and that we can encrypt a message M by raising it to the power of *e* modulo n, and then recover M by raising M^e to the power of d modulo n.

Note that, except for the exponents, identity **(4)** looks exactly the same as identity **(3)**. Thus, we can assert the following:

(5) $M^{\varphi(n)+1} \equiv M^{ed}\ (mod\ n)$

If $\varphi(n) + 1$ is the single exponent that allows us to recover M from $M^{\varphi(n)+1}\ mod\ n$ in one fell swoop, then e and d are the exponents that allow us to recover M in two steps; first via the encryption of M with $M^e mod\ n$, and second via the decryption of M^e with $M^{\mathrm{ed}} \bmod \mathrm{n}$.

For e and d, we need to find two integers such that, when they are multiplied, produce the same result as $\varphi(n) + 1$.[22] The method for finding these values is described in some detail in the previous section. But recall that for e, we choose an integer in the group 1 to $\varphi(n)$ that is relatively prime to $\varphi(n)$, and for d we find the inverse of e in the same group.

[22] Technically, we need to find an e and d that when multiplied equals $k\ x\ \varphi(n) + 1$ for some integer k. In the current example, in which we use very small values for e, *d* and n, this k happens to be 1. As such, k can be omitted from the equation without affecting the results, and doing so reduces conceptual clutter. With larger values of e, d and n, it is nevertheless important to understand that values of k greater than 1 will be necessary to balance the equation $ed = k\ x\ \varphi(n) + 1$, and moreover that for *any* value of k, the result of the operation modulo n will be unaffected. That is, adding 1 to any multiple of $\varphi(n)$ yields the same result modulo n (e.g., $2^9 \equiv 2\ (mod\ 15)$, $2^{17} \equiv 2\ (mod\ 15)$, $2^{25} \equiv 2\ (mod\ 15)$, and so on).

Let's choose the integer 3 for e, which is permissible because it is relatively prime to 8. For d, we must determine by what number to multiply e to yield 1 modulo 8. That is, we must solve for d in the following equation (which was also introduced in the previous section):

(6) $ed = 1\ (mod\ \varphi(n))$

Plugging in real values, we see that the inverse of 3 in the group of integers modulo 8 is also 3.

(7) $3\ x\ 3 = 1\ (mod\ 8)$

So for d, we must use the number 3.[23]

We now have all the ingredients necessary to encrypt a message M using a widely-known public exponent e, and to decrypt its ciphertext using a secret, privately-held exponent d.

To demonstrate, let's first choose a message M to encrypt. Since M must be in the set 1 to $n - 1$, let's choose the value 7. To encrypt, we raise M to the power of e modulo n:

(8) $M^e \equiv C\ (mod\ n)$

With our selected values this becomes:

(9) $7^3 \equiv 13\ (mod\ 15)$

Now let's decrypt the ciphertext C, which we do by raising C to the power of d modulo n:

(10) $C^d \equiv M\ (mod\ n)$

Recall that C^d can be rewritten as $(M^e)^d$, or alternatively M^{ed}.

Again, with our selected values, we get:

(11) $13^3 \equiv 7\ (mod\ 15)$

With the decryption operation above, we successfully recover the original message 7 from its ciphertext 13.

To summarize, we used Euler's theorem, and its properties relative to the totients of semiprime numbers as demonstrated by Rivest, Shamir and Adleman, to demonstrate the awesome power of public-key encryption in the RSA cryptosystem.

Computational Security

The public-key schemes described thus far share the property that the computations required by Alice and Bob to send each other messages securely—identifying large primes, computing greatest common divisors, computing multiplicative inverses and integer exponentiation—can be done relatively inexpensively; whereas the computations required for Eve to defeat these schemes—computing

[23] It so happens that for the small group of integers modulo 8, e and d are identical; that is 3 is the inverse of 3. For larger groups—e.g., the group modulo 60 presented in the section describing textbook RSA—this is not the case. In that group we chose the number 7 for e, whose inverse (d) is 43.

discrete logarithms and factoring large integers—cannot. It is on this asymmetry, or *one-wayness*, that the efficacy of public-key cryptosystems relies.

As such, these schemes are secure only to the extent that they are too computationally expensive to be feasible for would-be attackers. But, given unlimited time and/or computing resources, they are not unbreakable; with enough of one or the other or both, an exhaustive search of the keyspace—what is known as a *brute-force* attack—will eventually yield the correct key.

Given this mainly theoretical limitation, these schemes are therefore not *unconditionally* secure; rather, they are *computationally* secure, and may someday wither under attack from ever more powerful computers that are capable of reducing searches of the entire keyspace to a reasonable amount of time, and/or analytical techniques that have not yet been discovered.[24] In fact progress on both fronts has been made in the last half-century, and because of this the recommended key-lengths for RSA have increased several-fold over the years to keep apace of it.

RSA as an Alternative to DH

One interesting consequence of the advent of RSA was its potential to replace DH as a key-agreement mechanism. With RSA encryption, two parties now had the ability to *exchange* symmetric keys securely, rather than *negotiate* them independently in the manner specified by DH. But if RSA provided encryption, why did we need symmetric encryption keys at all anymore, never mind the ability to exchange them?

One reason is that public-key encryption is much more expensive computationally than its symmetric-key brethren, thus making it less suitable as a scheme for message encryption.[25] Another reason is that message lengths must not exceed the size of the RSA modulus, thus adding even more computational overhead (that of dividing larger messages into modulus-sized chunks).

As a result, one approach that emerged was to employ RSA keys to encrypt and exchange symmetric keys and, thus securely exchanged, the symmetric keys to bulk-encrypt message content. Such schemes are in fact in wide use, and are referred to as *hybrid* encryption schemes because they combine the advantages of both public- and symmetric-key encryption, the former for key exchange and the latter for message encryption.[26]

As to the observation that RSA is a suitable replacement for DH as a mechanism for key-exchange, it helps to keep in mind that although DH is almost always referred to as a key *exchange* protocol, strictly speaking it is a key *agreement* protocol. That is, the product of DH—a shared, symmetric key that is used by two parties in a private message exchange—is never actually exchanged, but rather negotiated independently by the parties on either side of the conversation.

---

[24] There is only one known cryptosystem that is unconditionally secure, and that is the *one-time pad*. Its key-length requirements are so cumbersome, however, as to make it impractical for all but the most sensitive applications.
[25] At current clock speeds, RSA requires 3,072-bit key lengths to achieve the same level of security as 128-bit AES, the current standard for symmetric-key encryption. The longer the keys, the larger the numbers used in the computations required to generate keys, and sign and decrypt messages.
[26] Schemes that combine DH-style key *agreement* with symmetric-key encryption are also called hybrid schemes.

A practical consequence of this distinction is that, in DH, the symmetric key is never transmitted over an insecure channel, and therefore cannot be intercepted by an eavesdropper, whereas in RSA this is not the case.

Perfect Forward Secrecy

Perfect forward secrecy is a property that guarantees that messages, once encrypted, remain so in perpetuity. Imagine that Eve intercepts, and stores, every message Alice ever sends to Bob. Then, at some future date, Eve manages to steal Bob's private key. Now Eve can use Bob's private key to decrypt everything Alice ever sent to Bob, symmetric keys and message content alike.

The loss of perfect forward secrecy is not confined to hybrid encryption schemes like the one just described; DH-style key-agreement schemes are equally vulnerable if not used properly. Any scheme in which messages—whether they contain keys or actual message content—are encrypted using *long-term*, public keys is not perfectly forward secret.

The preferred solution is for each party to generate a fresh pair of asymmetric keys each time a message is to be sent, and to use these keypairs to negotiate a fresh symmetric key to encrypt (and decrypt) a single message. Because the asymmetric keys are used only once, and then discarded, they are referred to as *ephemeral* keys. With this strategy, the motivation for an adversary to steal or crack a key is severely diminished.

Digital Signatures

Digital signature is the third and final component of the public-key cryptosystem conceived by Diffie and Hellman, and later implemented by Rivest, Shamir and Adleman. A digital signature serves three main purposes in digital communication: *message integrity*, *authentication of origin* and *non-repudiation of origin*. These properties prove that a signed message (a) has not been altered by an unauthorized party, (b) originated from its sender and (c) the sender cannot repudiate (a) or (b).[27]

In this scheme, the sender of a message *signs* it using his or her private key, and on receipt the receiver *verifies* the signed message using the signer's public key. If the signature verification fails, then either the private key used to sign the message does not correspond to the public key used to verify it, or the message was altered after it was signed. In either case, the signature is invalid and the message cannot be trusted. In short, a digital signature binds a specific message to the identity of its signer (or, more formally, the owner of the private key used to sign it).

Digital signature is the *inverse* of encryption, in the sense that in the latter messages are encrypted with a public encryption key e, and decrypted with a corresponding private decryption key d; whereas in the former messages are *signed* with a decryption key d, and *verified* with a corresponding encryption key e.

State Transition Redux

To visualize this, let's revisit the state-transition diagrams we employed to understand encryption. Here, instead of the term C occupying the middle state, we now have the term S, which stands for *signature*.

[27] Whereas hand-written signatures can be forged, and/or the documents they belong to altered, neither is possible with a properly constructed digital signature.

(1) $M \rightarrow S \rightarrow M$

As with encryption, we start with some plaintext message M, transform it to a signature S, and then back again to its original state M.

What is different this time is that the first transformation $M \rightarrow S$ needs to be difficult for an attacker, not the second $S \rightarrow M$ (with encryption, it is the second transformation $C \rightarrow M$ that must be difficult for the attacker).

Now, let's substitute the term S with its equivalent term M^d.

(2) $M \rightarrow M^d \rightarrow M$

Here, we are raising M to the power of the decryption exponent d to compute the signature. What makes this transformation difficult for an attacker is that the private exponent *d* used in the transformation is not (or at least should not be) known to anyone but the message's signer.

Next, let's substitute the rightmost M from the previous diagram with its equivalent term $\left(M^d\right)^e$.

(3) $\mathrm{M} \rightarrow M^d \rightarrow \left(M^d\right)^e$

Here, the signature M^d is raised to the power of the public encryption exponent *e* to recover the message M from the signature M^d.

Finally, let's confine the computations to the ring of integers modulo *n*.

(4) $M \rightarrow M^d \bmod n \rightarrow \left(M^d\right)^e \bmod n$

Plugging real values into the diagram yields the following transformations:

(5) $5 \rightarrow 5^{43} \bmod 77 \rightarrow (5^{43})^7 \bmod 77$
(6) $5 \rightarrow 26 \rightarrow 5$

If this all looks familiar, it should; it is the very mirror image of encryption. The difference is semantic; with encryption the public exponent *e* is used to transform a plaintext to a *ciphertext*, and with digital signature the private exponent d is used to transform a plaintext to a *signature*.[28]

In practice, encryption and digital signature can be combined to ensure the confidentiality, integrity and authenticity of a message in one fell swoop.

Resurrecting the example of encryption from *Figure 5*, assume that Bob has generated a key pair of his own and shared his public key with Alice; that is, he mimics Alice's actions in steps 1 through 8, sending his public key to Alice and keeping his private key secret. Now, Alice and Bob both possess each other's public keys, and each their own corresponding private keys. This time, however, before Bob encrypts his message M, he signs it with his private key to yield a signature S. Bob then encrypts the signature S with Alice's public key and transmits the resulting ciphertext to Alice.

[28] A signature is really just another name for a ciphertext, but one that is generated using a private key rather than a public key.

On receipt of the ciphertext, Alice reverses the procedure. First, she decrypts the ciphertext using her private key to recover the signature S. She then verifies the signature using Bob's public key to recover M from S.

Again, let's use state transition diagrams to depict this graphically. In the diagrams we distinguish Alice's and Bob's exponents by adding their initials; that is, eA and dA are Alice's encryption and decryption exponents, and eB and dB are Bob's encryption and decryption exponents, respectively. Although they have been omitted from the diagrams to reduce conceptual clutter, all the operations are reduced modulo n.[29]

First, Bob signs the message M using M^{dB} and then encrypts the signature M^{dB} using $\left(M^{dB}\right)^{eA}$, which he transmits to Alice.

(1) $M \rightarrow M^{dB} \rightarrow \left(M^{dB}\right)^{eA}$

Having received the encrypted signature $\left(M^{dB}\right)^{eA}$ from Bob, Alice recovers the signature M^{dB} using $\left(\left(M^{dB}\right)^{eA}\right)^{dA}$, and then recovers M from M^{dB} using $\left(M^{dB}\right)^{eB}$.

(2) $\left(M^{dB}\right)^{eA} \rightarrow \left(\left(M^{dB}\right)^{eA}\right)^{dA} \rightarrow M^{dB} \rightarrow \left(M^{dB}\right)^{eB} \rightarrow M$

Caveats

The RSA-style digital signatures described in the previous section are not suitable for real-world applications; they prove only in concept such schemes are possible.[30]

For one thing, that a signature can be *verified* in the aforementioned scheme is a dubious assertion. Suppose Alice produces a signature at random, claiming to some third party it is a signed message from Bob. Alice can then use Bob's public key to recover a completely valid (albeit nonsensical) message from this random signature. But how is the third party to determine the signature is not real? Illegibility is not sufficient proof a message was not signed with the private key corresponding to the public key used for verification.

Here, Alice has performed what is known as an existential forgery attack. In practice, such an attack can be thwarted if the signer applies a cryptographic hash to the message, signs the hash, and sends the signed hash together with the original message to the recipient.[31] The signature-message pair is then verified by recovering the signer's hash from the signature using signer's the public key, applying the same cryptographic hash to the original message, and comparing the results of both operations; if they agree, the signature is provably valid.

Even with protection from existential forgery, digital signatures are useless if a private key is stolen—or otherwise derived in some fashion—by an attacker. Then the private key can be used to sign any message the owner did not in fact sign. Because of this, it is virtually impossible to prove—in a court of

[29] Note that Alice and Bob will each have computed their own n, as well.
[30] Not long after publication of the RSA paper, real-world implementations began to appear.
[31] A *cryptographic hash* is a one-way function without a trap-door; that is, it can be computed in one direction but not the other.

law, for example—that just because a message was signed with a private key, that it was signed by that key's owner.

The Underbelly of Public-Key Cryptography

The public-key cryptosystems of DH and RSA are at once powerful and elegant. Moreover, industrial strength versions of them are virtually ubiquitous in securing electronic commerce, online banking and all manner of sensitive communication on the internet. But a discussion of them would not be complete without pointing out a glaring weakness to which they are all susceptible. This vulnerability is known as the *man-in-the-middle* attack, or *MITM* for short.

Imagine the following familiar scenario: Bob wishes to send an encrypted message to Alice. As ever, Eve is listening. Alice computes her public-private key pair and transmits the public key to Bob. Eve, meanwhile, intercepts Alice's key, substitutes it for a public key of her own, and forwards it to Bob. Bob, *having no way of knowing that the public key he receives belongs to Eve and not Alice*, blithely encrypts the message intended for Alice using Eve's public key. Eve intercepts Bob's message and decrypts it with her private key. Eve has thus broken encryption and, what's more, she has done so without using any math.[32]

Every public-key scheme—whether the secure key agreement of DH, the encryption and digital signature of RSA, or otherwise—is vulnerable to this attack.

How can this be if the security of the internet is based on public-key schemes? The answer is a Frankenstein-like bolt-on to the public-key cryptosystem called *public-key infrastructure*, or *PKI* for short.

Public Key Infrastructure

Not wanting their legacy to be relegated to one of interesting academic research, RSA's inventors founded a company seeking to commercialize their invention not long after publishing their groundbreaking paper. Acutely aware of the threat MITM posed to their cryptosystem, the founders knew that *RSA Security Inc.*'s success would depend largely on an effective solution to it. That solution was to be PKI.

At a very high level, a PKI establishes a trust relationship between Alice, Bob and a third party known as a *certificate authority* (CA). The CA's role in the PKI is to vouch for the authenticity of public keys, which it accomplishes by binding public keys to the identities of their owners. For a PKI to be effective, the CA has to be trusted by both Alice and Bob.[33]

To make this more concrete, take the recent example involving Alice and Bob, where Eve mounted an MITM attack, but this time in the context of a PKI.

---

[32] MITM is an illustration of the *weakest-link* maxim, which in the information security setting holds that a system is only as secure as its weakest link. From the perspective of an attacker, the most rational approach to defeating a cryptosystem is to attack its weakest link. In a public-key cryptosystem, the MITM vulnerability is a weak link.

[33] PKI is not restricted to the CA model described in this section. Other solutions include *Web of Trust*, *Trust on First Use* and other variants offering varying degrees of trust and security. The CA model is interesting and relevant because it is the one used to secure the vast majority of communication on the worldwide web.

After generating her public key (but before transmitting it to Bob), Alice submits it to a CA. The CA vets Alice and, if it concludes she is trustworthy, combines her public key with some name that uniquely identifies Alice into a digital document known as a *certificate*.[34] The CA signs this certificate with its own private key, and returns the signed certificate to Alice. Now, when Bob wants to send an encrypted message to Alice, instead of sending her public key to Bob she sends her CA-signed certificate. On receipt of the certificate, Bob verifies it using the CA's public key, which is published and widely available. If the verification succeeds, Bob knows the public key contained within the certificate in fact belongs to Alice, and proceeds confidently to encrypt messages to her with it. This time, if Eve intercepts Alice's certificate and forwards Bob a phony one of her own, Bob's verification step will fail and he will know he is being attacked.

Though effective at thwarting MITM attacks, the CA-based PKI is not without weaknesses of its own; chief among them the trustworthiness (or lack thereof) of the CA. Indeed, there are hundreds of public CAs the world over, the vast majority of which have sterling reputations. However, CAs themselves are not impervious to attack, and in some cases have been compromised.[35]

Going Forward

For all their elegance and power, cryptosystems based on the public-key implementations described in this paper are not without flaws. This is due in no small part to the complexity of the PKIs we trust to enforce good behavior in a hostile environment. Moreover, techniques based on multiplicative groups have become somewhat dated in the half-century since their introduction, and increasingly find themselves replaced by more modern and powerful implementations.[36]

Nevertheless, public-key schemes based on the difficulty of solving the discrete logarithm and integer factorization problems are the best we have in the internet age. Indeed, they are responsible for securing the vast majority of sensitive communications on the internet today, and achieve this objective with remarkable success.

The advent of cryptocurrencies, in particular the distributed blockchains on which they are based, offers tantalizing prospects for the establishment of trust that today is centralized in a handful of global certificate authorities. A blockchain-based PKI would decentralize trust, spreading it across a distributed network of synchronized databases, instead of concentrating it in the hands of a vulnerable few. Progress on this front has been halting, not least because its efficacy depends on an alignment of incentives that many would-be applications of a distributed blockchain, such as identity authentication, lack.[37]

---

[34] The official specification for public-key certificates is defined by the International Telecommunications Union's (ITU) *X.509* standard.

[35] In some cases in spectacular fashion. See *DigiNotar* for the quintessential case study.

[36] Elliptic curve cryptography (ECC) provides a novel implementation of the discrete log problem. ECC emerged as a countermeasure both to more effective attacks on traditional implementations, and more powerful computers to run them on. ECC achieves security levels equivalent to those of RSA with much shorter key lengths.

[37] This alignment of incentives, known as a *consensus algorithm*, works remarkably well in the realm of cryptocurrencies, where curators of blockchains are compensated for being honest with remunerative tokens (e.g., bitcoin).

Meanwhile, quantum computing looms menacingly on the horizon, and threatens the existence of public-key cryptography as we know it. Whereas even today's most powerful computers cannot reverse the one-way functions of the classical public-key cryptography described in this paper—at least not sufficiently fast to make the effort worthwhile—a fit-for-purpose quantum computer could break them in minutes. There is little doubt that well-funded, state-level actors are attempting to build quantum computers with such capabilities today. The invention of such a computer would render classical public-key cryptography instantly obsolete, and enable its inventors to break the cryptosystems on which the world's security-critical computing infrastructure largely relies.