

Introduction to Cryptography*

What is cryptography? Cryptography is the mathematical study of methods that allow you to

- make “ciphers” that encode your secret messages so that only your partner can decipher them, and
- break the ciphers of your enemies so that you can read their encrypted messages.

1 A few classical cryptosystems

1.1 Substitution ciphers

Say that Alice wants to send a secret message to her friend Bob. For convenience, let’s associate the letters of the alphabet with their corresponding numbers:

A	B	C	D	E	F	G	H	I	J	K	L	M
1	2	3	4	5	6	7	8	9	10	11	12	13
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
14	15	16	17	18	19	20	21	22	23	24	25	26

Thus we can think of messages as sequences of numbers, instead of letters. (And we will ignore spaces and punctuation for simplicity.)

Recall. *Modular Arithmetic:* Given any $n \in \mathbb{N}$, \mathbb{Z}_n is the set of equivalence classes $[0], [1], \dots, [n-1]$ of the integers, with respect to the relation $a \equiv b \pmod n$. Recall that $a \equiv b \pmod n$ if and only if a and b have the same remainder when divided by n , equivalently if and only if n divides $(a - b)$.

Thus our scheme above identifies the alphabet with \mathbb{Z}_{26} .

Maybe the simplest cipher (after pig latin) is a *shift cipher*:

1. Alice and Bob agree ahead of time on a number k , their *key*
2. Alice takes her message, $m_1 m_2 \dots m_r$, and encrypts it by replacing each m_i with the sum $e_i = m_i + k$, taken modulo 26
3. Alice sends Bob the encrypted message $e_1 e_2 \dots e_r$
4. Bob decrypts the message by taking each e_i and subtracting k , modulo 26

This cipher was famously and successfully used by Julius Caesar in battle.

Exercise. *Decrypt the following message, which was encrypted using a shift cipher with $k = 5$:*

18 6 25 13 14 24 8 20 20 17

A substitution cipher generalizes this.

*These notes were written by Diane Vavrichek, and are a supplement for Math 330 at Binghamton University in Spring 2011. Some references are [HPS08] and [KL08]. Here I mainly follow [HPS08].

Recall. A function is a bijection if and only if it has an inverse.

A substitution cipher works as follows:

1. Alice and Bob agree ahead of time on a bijection $\zeta: \mathbb{Z}_{26} \rightarrow \mathbb{Z}_{26}$
2. Alice takes her message $m_1 m_2 \dots m_r$, and encrypts it by replacing each m_i with $e_i = \zeta(m_i)$
3. Alice sends Bob the encrypted message $e_1 e_2 \dots e_r$
4. Bob decrypts the message by taking each e_i and replacing it with $\zeta^{-1}(e_i) = m_i$

In this case, the function ζ (and its inverse) is the key that Alice and Bob share.

The original message $m_1 m_2 \dots m_r$ is called the *plaintext* and the encrypted message $e_1 e_2 \dots e_r$ is called the *ciphertext*.

Substitutions ciphers are considered insecure. Suppose that Eve is an eavesdropper who can see the encrypted messages that Alice sends to Bob.

For the shift cipher, if Eve knows even the smallest bit of one of the original messages, then Eve can determine k and decrypt everything that Alice sends to Bob.

For a general substitution cipher, if Eve has a large quantity of ciphertexts, then she can determine ζ using well-established statistical techniques. These techniques are based on the fact that the letters in the alphabet can be largely determined based on the frequency with which they occur in standard usage of the English language. The same type of analysis can be done with any language.

1.2 Polyalphabetic ciphers

Substitution ciphers can be improved upon by working with several different bijections, instead of just one. Given bijections $\zeta_1, \zeta_2, \dots, \zeta_s$, Alice could instead encrypt her message by using ζ_1 for the first letter, ζ_2 for the second and so on, cycling through the ζ 's and starting over again with ζ_1 at the $(s+1)^{\text{st}}$ letter, etc. This is called a *polyalphabetic cipher*.

This is equivalent to Alice breaking up her message into *blocks* of size s (and padding the last block with zeros, if need be), and having a key which is a bijection from the set of blocks of size s to itself.

Unfortunately for Alice, this technique is also considered insecure. Statistical methods like those mentioned above can be used to determine what the block size is, and then decrypting Alice's message just amounts to figuring out s substitution ciphers.

Remark 1. Notice that, for polyalphabetic ciphers, blocks play the role of a large "alphabet" for Alice and Bob. By working with blocks instead of the regular alphabet, we can take the (numerical) alphabet that we are working with to be as large as we would like. In general, cryptographers work with very large alphabets.

Let p be a large prime number and let \mathbb{F}_p^* be the set of all the nonzero elements of \mathbb{Z}_p . I will give another example of a substitution cipher, for the alphabet \mathbb{F}_p^* . This example will rely on the following.

Proposition 2. All elements of \mathbb{F}_p^* have multiplicative inverses modulo p . That is, for any $a \in \mathbb{F}_p^*$, there is a unique $b \in \mathbb{F}_p^*$ such that

$$ab \equiv 1 \pmod{p}.$$

(Can you show that the conclusion of this proposition does not hold if p is not prime?)

Exercise. Let $k \in \mathbb{F}_p^*$. Prove that the function $\zeta_k: \mathbb{F}_p^* \rightarrow \mathbb{F}_p^*$ that takes any $a \in \mathbb{F}_p^*$ to ka (modulo p) is a bijection.

It follows from the above exercise that any ζ_k determines a substitution cipher on \mathbb{F}_p^* .

What would happen if we did not work modulo p here? We could still encrypt by multiplying by a fixed constant, and this map would be bijective onto its image, so Bob would be able to decrypt the ciphertext.

The trouble with this is that the ciphertext would be of the form

$$(km_1)(km_2)\dots(km_r),$$

and it is computationally feasible for Eve to discover k by taking gcd's of these terms. I will say more about this below.

Working modulo p “destroys” properties like divisibility.

Some history. *Ciphers have probably been around for roughly as long as written communication has been. As mentioned above, a substitution cipher was famously used by Caesar. The statistical methods that can reliably break polyalphabetic ciphers were developed in the 19th century.*

More recently, cryptography played an important role in World War II. Perhaps you have heard of the Nazi's Enigma machine? This was a machine that implemented a very, very complicated polyalphabetic cipher, which the Nazis used for communications between their forces. Great Britain ran a top secret project that, with help from Poland, eventually was largely able to crack this cipher. This enabled Great Britain to intercept a large number of Nazi messages, which substantially helped the Allied powers to win the war. (The Japanese had a similar machine, code named Purple. Experts in the U.S. were able to reconstruct the design of the cipher by analyzing encrypted messages, which also played a role in ending the war.) The British project was kept secret until 1974.

2 Public key cryptography

A limitation of the cryptosystems that I have discussed so far is that they require Alice and Bob to have an agreed-upon key. This requires Alice and Bob to have met ahead of time to make this arrangement, or in some other way to have agreed upon a key.

Pioneering work done within the last 40 years or so has developed a theory of “public key cryptography”, where Alice and Bob can communicate securely without setting a key ahead of time. Public key cryptography is of much mathematical interest, and will be the focus of the rest of these notes. That said, private key cryptography is still important and very much in use today. One advantage is that private key cryptosystems can be much more efficient than public key systems.

Public key cryptography had its origin (in the unclassified world!) in the work of Diffie and Hellman, which was published in 1976.

2.1 The Diffie-Hellman key exchange method

The Diffie-Hellman key exchange method is a process by which Alice and Bob can arrange a shared numerical key, while only communicating in a public forum. (It might strike you at this point that this sounds a bit like magic! If it's any consolation, it did take mankind tens of thousands of years to come up with this.)

Before I discuss Diffie-Hellman key exchange, I will touch on the computational difficulty in solving a few important problems.

2.1.1 Some “easy” problems

Suppose g and h are two integers with no more than n digits each. If you wrote a computer program that would add g and h (manually! Without using the addition function that the computer already has), how many steps would the program take to finish?

If you write your program efficiently, you should be able to do this computation in $(kn + l)$ steps, for some constants k and l that do not depend on g and h . This means that addition is a function that runs in time that is on the “order” of n . It is said that addition thus has *computational complexity* $O(n)$ (read “big-O of n ”). This is considered to be very “fast”, or computationally feasible: if we wanted to add larger

and larger numbers g and h , the increase in time it would take our computer to do this would not grow very quickly.

Here is a harder question: how many steps would it take you to *multiply* g and h ?

If you write an efficient algorithm, you should be able to do it in no more than $(kn^2 + l)$ steps, again for some constants k and l that do not depend on g and h , so multiplication has complexity $O(n^2)$. This is also considered very fast.

How about exponentiation? Say that e is a natural number and that we want to compute g^e . In the straightforward algorithm, one begins by computing g^2 , then multiplies this by g to get g^3 and so on, performing $(e - 1)$ multiplications.

In fact, I will only be interested in doing this operation modulo some constant, say j . This will make things easier because, as we compute higher and higher exponents of g modulo j , these values will have their number of digits bounded by the number of digits of j , and this is just a multiple of n . Thus, using that multiplication is $O(n^2)$, a straightforward algorithm to compute g^e should be $O((e - 1)n^2)$.

In fact, there is a simple way to improve this. Suppose for a moment that $e = 2^k$. Then we can compute g^e by squaring: compute g^2 , then $g^2g^2 = g^4$, $g^4g^4 = g^8$ and so on. We will compute g^e with only k multiplications, not $(e - 1)$.

If e is not a power of 2 then we can still compute g^e with a similar trick. For example, say $e = 23 = 16 + 4 + 2 + 1$. If we compute g^{16} as above, then along the way we have also found g^4 and g^2 , and so we can compute $g^{23} = g^{16}g^4g^2g$ with only three additional multiplications. In general, we can compute g^e by performing less than or equal to $2\lceil\log_2(e)\rceil$ multiplications. Thus, computing g^e is $O(\log_2(e)n^2)$. If e has no more than n digits, this operation is $O(n^3)$.

Any algorithm that can be done in $O(p(n))$ steps, where $p(n)$ is a polynomial in n , is said to have *polynomial time*. Polynomial-time algorithms are generally considered to be “fast”, meaning that they are typically computationally feasible. (If you really want to implement such an algorithm, however, you will need the constants that do not depend on g , h or e not to be gigantic!)

Another operation that we will need later is *gcd*, the greatest common divisor operation. This is also a polynomial-time function, and in fact can be made to run in $O(n^2)$.

Exercise. Write an algorithm that computes *gcd*. What is its “big- O ”?

2.1.2 A “hard” problem: the Discrete Logarithm Problem

An algorithm is of *exponential time* if it may take up to $k2^n + l$ steps, where n is the size of the inputs and k and l are constants that do not depend on n . While polynomial-time algorithms are considered computationally easy, exponential-time algorithms are considered computationally hard: as n grows, the running time of the algorithm increases dramatically. Of course, for very small values of n , an exponential-time algorithm probably is computationally feasible, but for large enough values of n it is not.

Note that if a problem is considered computationally hard (that is, exponential in time), it is often merely the case that all known solutions can take up to exponential time. It may be possible that a more efficient, even polynomial-time solution exists, but has not yet been found! An example of an exponential-time problem of this type is the well-known Discrete Logarithm Problem.

Let p be a large prime number, and recall the sets $\mathbb{Z}_p = \{[0], [1], \dots, [p - 1]\}$ and $\mathbb{F}_p^* = \{[1], \dots, [p - 1]\}$, where we can consider addition and multiplication modulo p . An element $g \in \mathbb{F}_p^*$ is called a *primitive root* for \mathbb{F}_p^* if, for each $h \in \mathbb{F}_p^*$ there is some x such that $g^x \equiv h \pmod{p}$. That is to say, g is a generator for \mathbb{F}_p^* . The Discrete Logarithm Problem is the problem of finding this x :

Definition 3. Let $g, h \in \mathbb{F}_p^*$ and suppose that g is a primitive root. The Discrete Logarithm Problem is the problem of finding some x such that

$$g^x \equiv h \pmod{p}.$$

2.1.3 Aside: problems of “medium” difficulty

There are algorithms that are “between” polynomial-time and exponential-time, meaning that they are slower than any polynomial algorithm (as $n \rightarrow \infty$) but they are faster than any exponential algorithm (also as $n \rightarrow \infty$). These algorithms are said to be *subexponential*, and are still considered not to be computationally feasible.

Note that, as in the case of exponential-time algorithms, most problems that are said to be subexponential have no *known* polynomial-time solutions — usually it has not been shown that no such solutions exist.

An example of a subexponential time complexity problem is the *Integer Factorization Problem*: given $N \in \mathbb{Z}$, can a prime factor of N be found? The Integer Factorization Problem is known to be subexponential, with complexity $O(2^{\sqrt{k} \log k})$.

Counterintuitively, however, if one merely wants to determine if N has any (proper) prime factors but does not need to know what they are, in other words if one wants to figure out if N is prime, then this can be done in polynomial time in the number of digits of N ! Thus it is easy to find prime numbers, a fact that will be very important in all of our methods below.

2.1.4 The Diffie-Hellman method

The *Diffie-Hellman key exchange method*, where Alice and Bob can arrange a shared secret key, while only communicating publicly, is the following.

1. Alice and Bob publicly agree on a large prime number p and an element $g \in \mathbb{F}_p^*$
2. Alice picks a secret integer a and computes $A \equiv g^a \pmod{p}$
3. Bob picks a secret integer b and computes $B \equiv g^b \pmod{p}$
4. Alice sends A to Bob and Bob computes $B' \equiv A^b \pmod{p}$
5. Bob sends B to Alice and Alice computes $A' \equiv B^a \pmod{p}$

Note that

$$A' \equiv B^a \equiv (g^b)^a = g^{ab} = (g^a)^b \equiv A^b \equiv B' \pmod{p},$$

i.e. $A' \equiv B' \pmod{p}$. This is Alice and Bob’s shared key.

Note also that this process is computationally feasible for Alice and Bob, because they only need to find p and perform exponentiations modulo p .

Can Eve the eavesdropper figure out what the key is? We should assume that Eve had access to all of the information that was exchanged between Alice and Bob, so Eve knows the values of p, g, A and B . If Eve could figure out what a or b is, i.e. if Eve could solve the Discrete Logarithm Problem, then Eve would be able to compute the key. In fact, this is the only known way for Eve to find the key.

Remark 4. *By choosing p to be a “large” prime, I mean that p should be on the order of at least around 2^{1000} . It is also best to choose g to be a prime on the order of at least $p/2$.*

2.2 The RSA cryptosystem

In 1978, the first public-key cryptosystem was invented by Rivest, Shamir and Adleman. Called the RSA cryptosystem, it (like the Diffie-Hellman method) revolutionized communication, and became the industry standard in security. In fact, whenever you enter your credit card number in a secure transaction online, a cryptosystem based on RSA is probably being used to make sure that your number is safe from any third parties.

RSA is still the most well-known public key cryptosystem, and is what we will explore in this section.

2.2.1 Some Number Theory

Before giving the RSA method, I must go over a few basic results from Number Theory. Recall our definition of the greatest common divisor:

Definition 5. Let $a, b \in \mathbb{Z}$. If $a = b = 0$ then define $\gcd(a, b) = 0$. Otherwise, define

$$\gcd(a, b) = \min\{x \in \mathbb{N} \mid x = ka + lb \text{ for some } k, l \in \mathbb{Z}\}.$$

The next result gives a necessary and sufficient condition for an integer to have a multiplicative inverse modulo N .

Proposition 6. Let $e \in \mathbb{Z}$ and $N \in \mathbb{N}$. There exists some $d \in \mathbb{Z}$ such that $de \equiv 1 \pmod{N}$ if and only if $\gcd(e, N) = 1$.

Proof. If $\gcd(e, N) = 1$ then there are integers k, l such that $1 = ke + lN$. Hence N divides $(1 - ke)$, so $ke \equiv 1 \pmod{N}$.

If there is some $d \in \mathbb{Z}$ such that $de \equiv 1 \pmod{N}$, then N divides $1 - de$, so for some $l \in \mathbb{Z}$,

$$1 - de = lN$$

$$1 = de + lN,$$

and hence $1 = \gcd(e, N)$. ■

On a computational note, finding such multiplicative inverses can be done by the same polynomial-time algorithm that computes the gcd.

We also have the following celebrated result of Fermat:

Fermat's Little Theorem. Let p be prime and fix any integer a . Then

$$a^{p-1} \equiv 1 \pmod{p}.$$

There are proofs of this that are not too hard. You should be able to find one in any introductory Number Theory textbook.

While the security of the Diffie-Hellman key exchange method relies on the difficulty of the Discrete Logarithm Problem, the security of RSA relies on the difficulty of taking “roots” mod N . Specifically, suppose that N, e and c are known, and consider the following equation:

$$x^e \equiv c \pmod{N}. \tag{1}$$

The following result shows that it is easy to solve such an equation for x when N is prime:

Proposition 7. Let p be prime and let $e \in \mathbb{N}$ be such that $\gcd(e, p-1) = 1$. By Proposition 6, there is some $d \in \mathbb{Z}$ such that $de \equiv 1 \pmod{p-1}$. Then $x = c^d$ is a solution to the equation

$$x^e \equiv c \pmod{p}.$$

Proof. We have that

$$(c^d)^e = c^{de} = c^{k(p-1)+1},$$

for some $k \in \mathbb{Z}$, using that $de \equiv 1 \pmod{p-1}$, so de has remainder 1 when divided by $(p-1)$.

By Fermat's Little Theorem,

$$c^{k(p-1)} = (c^{p-1})^k \equiv 1^k = 1 \pmod{p},$$

hence

$$(c^d)^e = c^{k(p-1)+1} = c \cdot c^{k(p-1)} \equiv c \cdot 1 = c \pmod{p},$$

as desired. ■

It is also computationally easy to solve (1) when N is the product of known primes and the exponent e is suitably chosen. In the case that $N = pq$ we have the following.

Proposition 8. *Let p and q be distinct primes, let $N = pq$ and let $e \in \mathbb{N}$ be such that $\gcd(e, (p-1)(q-1)) = 1$. By Proposition 6, there is some $d \in \mathbb{Z}$ such that $de \equiv 1 \pmod{(p-1)(q-1)}$. Then $x = c^d$ is a solution to the equation*

$$x^e \equiv c \pmod{N}.$$

(For a proof of this, see pp. 116-117 of [HPS08].)

However, the important point is that *there is no known computationally feasible way to solve (1) when $N = pq$ and the primes p and q are not known!* Note that if it were easy to factor N and hence to find p and q , then this would give an easy solution, so the difficulty here relies on the difficulty of the Integer Factorization Problem.

2.2.2 The RSA method

Now I will introduce the RSA cryptosystem, and you will see that the security of this system depends on the point noted above.

Bob uses the RSA method to send a secure message to Alice as follows. It will suffice to assume that the secret message is a single integer.

1. Alice chooses two secret primes p and q , sets $N = pq$, and chooses an exponent e such that $\gcd(e, (p-1)(q-1)) = 1$
2. Alice publishes N and e
3. Bob has a plaintext secret message $m \pmod{N}$, and computes $y \equiv m^e \pmod{N}$
4. Bob sends y to Alice publicly
5. Alice finds $d \in \mathbb{Z}$ such that $de \equiv 1 \pmod{(p-1)(q-1)}$, and computes $m' \equiv y^d \pmod{N}$

Note that $m' \equiv y^d \equiv (m^e)^d = m^{de} \pmod{N}$, and it follows from Proposition 8 that $m^{de} \equiv m \pmod{N}$. Thus m' , which has been computed by Alice, is Bob's plaintext message. Also note that Alice and Bob are only doing computationally feasible operations.

Eve the eavesdropper has access to N , e and y . If Eve could solve $y \equiv m^e \pmod{N}$ for m then Eve would have Bob's secret message. As I mentioned above, however, this problem has no known easy solution, if all of the numbers involved are large enough. Also if Eve could find p or q , i.e. if Eve could solve the Integer Factorization Problem, then Eve would be able to compute d and hence $y^d \equiv m \pmod{N}$, but this problem also has no known computationally feasible solution.

You can see now why it is said that the security of communication today depends on "the difficulty of factoring large numbers"!

References

- [HPS08] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *An introduction to mathematical cryptography*. Undergraduate Texts in Mathematics. Springer, New York, 2008.
- [KL08] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. Chapman & Hall/CRC Cryptography and Network Security. Chapman & Hall/CRC, Boca Raton, FL, 2008.