Cryptography Class Notes

These notes follow Hoffstein Pipher and Silverman's 'An Introduction to Mathematical Cryptography', which is herein refereed to as 'the text'. This is an upper level undergraduate first class in Cryptography.

1 An Introduction to Cryptography

This class is about *Cryptography*, a subfield of *Coding Theory*. For context, Coding Theory is the study of *codes*. The main types of codes are:

i). *Secret codes* - used to transfer information so that only the intended recipient can read it.

ii). *Compression codes* - used to transfer information efficiently.

iii). *Error correcting codes* - used to correctly transfer information that might get corrupted in transfer.

'Cryptography' is usually taken to be the study of secret codes, and this is what we will study. 'Coding theory' can refer to everything, or to the study of the last two types of codes.

1.1 Simple Substitution Cipher

Example 1.1. Say Alice wanted to send a message to Bob, but didn't want Chuck to be able to read it.

Tomorrow at Lunch

She could replace every letter with the letter to the left of it on the keyboard, and instead send

Ypzpttpe sy Aimvj

Chuck can't read her message, but Bob knows he has to replace every letter with the letter to the left of it on the keyboard, and can convert it back to the original message.

What Alice and Bob have used is called a *simple substitution cipher*. It is an example of a scheme called a *cryptosystem*. The elements of a cryptosystem are, informally:

- The message 'Tomorrow at Lunch' is the *plaintext*.
- The message 'Ypzpttpe sy Aimvj' is the *ciphertext*.
- The algorithm for converting between ciphertext and the plaintext is called the *key* or *cipher*.
- The process of converting the plaintext to the ciphertext is called *encryption*.
- The process of converting the ciphertext to the plaintext using the key is called *decryption*.
- The process of converting the ciphertext to the plaintext without the key is called *cryptanalysis*, or *cracking* the cipher.

In this scheme it is required that Alice and Bob both know the key. Such cyptosystems are called *private key* cryptosystems, or *symmetric* cryptosystems.

The simple substitition cipher given above can be cracked very easily using the relative frequency of different letters, and pairs of letters, in written english. However there are other private key cryptosystems that are impossible to crack. We will spend a little time looking at private key cryptosystems. The drawback of a private key cryptosystems is that Alice and Bob must first meet to share the key, and this is not always practical.

Contrastingly, in a *public key* cryptosystem, or *asymmetric* cryptosystem, there are two keys, one for encryption, and one for decryption. In such a cryptosystem, Bob keeps the decryption key secret, but makes the encryption key public. Under this scheme, anyone, including Alice, can send Bob a message, which only Bob can read. Such ciphers can be cracked. What makes one good, is that it is difficult to crack. The majority of the course will be focused on asymmetric cryptosystems, and the mathematics behind them.

Problem 1.1. Reading Section 1.1.1 of the text if necessary, do Problem 1.4 (a) from the text

Problem 1.2. Come up with at private key cryptosystem that is impossible to crypoanalyse, and explain why it is impossible.

1.2 Divisibility and GCDs

Recall that for integers a and b, we we say that a *divides* b, and write $a \mid b$, if there is another integer q, called the *quotient*, such that $aq = b$. If a divides b, then a is a *divisor* or *factor* of b, and b is a *multiple* of a.

Where all letters involved are integers it should be simple from this definition to prove such properties i). $a|0$

ii). $0|a \Rightarrow a = 0$

iii). $1|a$

iv). $a|1 \Rightarrow a = \pm 1$

v). $a|b_1$ and $a|b_2 \Rightarrow a|c_1b_1 + c_2b_2$

vi). If $b + c = d$, and a divides two of b, c, d then it divides all of them.

With a bit more work you should be able to prove the the Division Algorithm: that for two integers a and b there is a unique pair of integers q and r with $0 \leq r < a$ such that

$$a = b \cdot q + r.$$

In this case r is called the *remainder* upon division of a by b.

The greatest common divisor $\gcd(a, b)$ of two integers a and b is the greatest integer that divides them both. From the last of the six properties given above, and the Division Algorithm, you should you know how to find the gcd of two integers using the *Euclidean Algorithm*.

We won't go into the details, but will recall the Euclidean Algorithm with an example so that we can analyse the time it takes.

Example 1.2. (See Section 1.2 of the text if you forget the details of the Euclidean Algortithm.) To find $\gcd(1253, 234)$ we use the Euclidean Algorithm:

Step	a	$=$	b	$\cdot$	q	$+$	r
1	1253	=	234	·	5	+	83
2	234	=	83	·	2	+	68
3	83	=	68	·	1	+	15
4	68	=	15	·	4	+	8
5	15	=	8	·	1	+	7
6	8	=	7	·	1	+	1
7	7	=	1	·	7	+	0

So $\gcd(1253, 234) = 1$. Now to analyse the time this algorithm takes, we don't really care about how much time each step takes- this depends on the computer and such. We care about the number of steps that are necessary, in terms of the size of a and b.

We show that the algorithm takes at most $2\log_2(b) + 1$ steps. Let r_i be the remainder after the i^{th} step, and $r_0 = b$. The algorithm does at most i steps if $r_i = 0 < 1$. Observe that for $i \geq 0$ we have that $r_{i+2} < r_i/2$. Indeed $r_i = r_{i+1} \cdot q_i + r_{i+2}$, where $q_i \geq 1$ and $r_{i+1} > r_{i+2}$. This gives that $r_{2k} \leq r_0/2^k = b/2^k$.

Which is less than 1 if $b < 2^k$ or $\log_2 b < k$. So we stop after i steps if $2\log_2 b < i$ (or after $i+1$ if i is odd.) So we have no more than $2\log_2 b + 1$ steps, as needed.

As $\gcd(a, b)$ divides a and b, then it divides any linear combination $c_1a + c_2b$ of them for integers c_1 and c_2. Using this *Extended Euclidean Algorithm*, you should be able to write $\gcd(a, b)$ as a linear combination $\gcd(a, b) = c_1a + c_2b$ of a and b. So alternately we could define $\gcd(a, b)$ as the smallest positive linear combination of a and b with integer co-efficients.

Problem 1.3. Using the Extended Euclidean Algorithm (read Section 1.2 if you haven't seen it) express 1 as a linear combination of 1253 and 234.

The time complexity of this algorithm is (up to a constant) the same as for the Euclidean Algorithm.

1.3 Modular Arithmetic

You've probably seen Modular Arithmetic before, so again we skip many details. The text fills in the details if you are unfamiliar.

Fixing an integer m, we say integers a and b are *congruent mod* m, and write $a \equiv b \mod m$ if $m \mid (a - b)$.

Example 1.3.

$$-2 \equiv 5 \equiv 12 \equiv 712 \mod 7$$

One sees that $a \equiv r \mod m$ where r is the remainder upon division of a by m:

$$a = q \cdot m + r$$

So every integer is congruent modulo m to some integer in $\{0, 1, \ldots, m-1\}$. In fact reducing a to this remainder r is the canonical quotient map that takes the ring $\mathbb{Z}$ to the quotient ring $\mathbb{Z}/m\mathbb{Z} = \{0, 1, \ldots, m-1\}$ *modulo* the ideal $m\mathbb{Z}$.

As this is a homomorphism, we have for $a_1 \equiv a_2 \mod m$ and $b_1 \equiv b_2 \mod m$, then

i). $a_1b_1 \equiv a_2b_2 \mod m$,

ii). $a_1 + b_1 \equiv a_2 + b_2 \mod m$,

iii). $a_1 - b_1 \equiv a_2 - b_2 \mod m$.

(Or, because of these properties, it is a homomorphism.)

We say that b is a *multiplicative inverse* of a modulo m, if $ab \equiv 1 \mod m$. If a has a multiplicative inverse, we say it is *invertible modulo* m.

For example in $\mathbb{Z}/10\mathbb{Z}$, 7 has an inverse,

$$7 \cdot 3 = 21 \equiv 1 \mod 10,$$

but 5 does not, as all multiples of 5, $0, 5, 10, 15, 20, 25$ are congruent to 0 or 5 modulo 10.

Proposition 1.4. *Let $m \geq 1$ be an integer.*

i). If a is invertible modulo m, its inverse is unique modulo m, that is, if b and b' are inverses of a, then $b \equiv b' \mod m$.

ii). An integer a is invertible modulo m if and only if is relatively prime to *m, that is if $\gcd(a, m) = 1$,*

Proof. The first property can be shown directly, or by observing that a has a multiplicative inverse modulo m if and only if the image of a in $\mathbb{Z}/m\mathbb{Z}$ is a unit.

For the second property, **if $\gcd(a, m) = 1$ then using the Extended Euclidean algorithm we can get b and c such that $ab + mc = 1$. So**

$$1 \equiv ab + mc \equiv ab.$$

On the other hand, if $ab \equiv 1$ then $ab + cm \equiv 1$ so $\gcd(a, m)|1$. see Prop 1.13 of the text □

In light of the first property, if a is invertible modulo m, we denote its inverse by a^{-1} or $1/a$. To calculate with it, we may replace it with any inverse of a modulo m. We tend to use the one in the range $1, \ldots, m-1$ and call this the inverse.

Example:

$$4/7 = 4 \cdot 7^{-1} \equiv 4 \cdot 3 = 12 \equiv 2 \mod 10.$$

Problem 1.4. Is 2536 invertible modulto 353? If so, what is it's inverse?

The set $(\mathbb{Z}/m\mathbb{Z})^*$ of units of $\mathbb{Z}/m\mathbb{Z}$ form a group. The *order* $|(\mathbb{Z}/m\mathbb{Z})^*|$ of which will be important. It occurs enough that there is a function to describe it– *Euler's phi function:*

$$\phi(m) = ||(\mathbb{Z}/m\mathbb{Z})^*|.$$

Problem 1.5. What is $\phi(32)$?

Back to ciphers; using modular arithmetic we have a very nice description of a simple class of substitution ciphers.

Associating the i^{th} letter of the alphabet with the number $i-1$, (so $a = 0$ and $b = 1$ and, ..., $z = 25$,) we can write our message " Tomorrow at Lunch" as

$$19\,14\,12\,14\,17\,17\,14\,22 \qquad 00\,19 \qquad 11\,20\,13\,02\,07.$$

We consider this the plaintext. Now to encrypt it we pick a secret key $k \in \{0, \ldots, 25\}$ and add it to each letter, modulo 26. Taking key $k = 9$ the ciphertext is

$$03\,23\,21\,23\,01\,01\,23\,06 \qquad 09\,03 \qquad 20\,04\,22\,11\,16.$$

Now perhaps we would use the same letter to number key, and send this as

Dxvxbbxgjduewlq,

but let's just let that be overhead, and always consider our plaintext and ciphertext as numbers.

Where we encrypt by adding k, we can clearly decrypt by subtracting k. So the key k is symmetric. Clearly we could also encrypt by multiplying by the key k. Decryption would then be division by k, that is, multiplication by its inverse modulo 26. So we could only do this if $\gcd(k, 26) = 1$. What is the problem if $\gcd(k, 25) \neq 1$?

In several applications, we will be computing large powers of an integer a in $\mathbb{Z}/m\mathbb{Z}$. Let's look at how to do this quickly.

If we compute 7^{415} and then reduce it modulo 3317, then we have to do 415 multiplications of increasingly large numbers. It seems clear that we will save time and space reducing the product at each step.

Problem 1.6. Compute 7^5 modulo 11.

We can actually save more time. Observe that $415 = 2^8 + 2^7 + 2^6 + 2^2 + 2 + 1$. So

$$7^{415} = 7^{2^8+2^7+2^6+2^2+2+1} = 7^{2^8} \cdot 7^{2^7} \cdot 7^{2^6} \cdot 7^{2^2} \cdot 7^2 \cdot 7.$$

It takes us $7 \approx \log_2(415)$ multiplications to evaluate all the factors 7^i modulo 3317, and about the same to multiply them together. This is $2\log_2(415)$ operations verses the 415 operations to do it stupidly. This is called the fast powering algorithm.

1.4 Prime numbers, Unique Factorisation, and finite fields

Recall that an integer $p \geq 2$ is *prime* if and only if for positive integers d

$$d \mid p \Rightarrow d = 1 \text{ or } p.$$

I expect that you've seen and could prove the following results:

- If a prime p divides ab for integers a and b, then it divides a or b.
- There are infinitely many primes.

The first of these is infact only true of 1 and primes, and their negatives. Further, you should have seen the following. You certainly believe it.

Theorem 1.5 (Fundamental Theorem of Arithmetic)**.** *Let $a \geq 2$ be an integer. Then a can be written as a product of primes:*

$$a = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots p_d^{e_d},$$

where each p_i is a distinct prime, and each e_i is a positive integer. Furthermore, except for order, this representation is unique.

This representation is called the *prime factorisation* of the integer a. For a particular prime p_i, the exponent e_i is called the *order of p_i in a*, and is denoted $e_i = \mathrm{ord}_{p_i}(a)$. If a prime p does not occur in in the prime factorisation of a, then $\mathrm{ord}_p(a) = 0$.

Example 1.6. The prime factorisation of 204 is

$$2^2 \cdot 3 \cdot 17,$$

so $\mathrm{ord}_2(204) = 2$, $\mathrm{ord}_{17}(204) = 1$, and $\mathrm{ord}_7(204) = 0$.

For prime p, the numbers $1, \dots p-1$ are all relatively prime to p, and so we have the following useful fact.

Proposition 1.7. *If p is a prime then the group $(\mathbb{Z}/p\mathbb{Z})^*$ of units of $\mathbb{Z}/p\mathbb{Z}$ consists of the set*

$$\{1, 2, \dots, p-1\}$$

of non-zero elements. By Proposition 1.7, we have that for prime p, $\mathbb{Z}/p\mathbb{Z}$ is not just a ring, but a field. From field theory we know that the is a unique field of order p. This field is often denoted $\mathbb{F}_p$.

By Proposition 1.7, we have that for prime p, $\mathbb{Z}/p\mathbb{Z}$ is not just a ring, but a field. From field theory we know that the is a unique field of order p. This field is often denoted $\mathbb{F}_p$.

1.5 Powers and Primitive Roots in Finite Fields

We will use properties of $\mathbb{Z}/p\mathbb{Z} \cong \mathbb{F}_p$ that we know from field theory. Several of these properties, we can prove without much theory though, so we do that.

We have by Lagrange's theorem that ever element of a group has order dividing the order of the group: indeed the cosets of the subgroup generated by the element partition the group into equal parts. It follows that $a^{p-1} = 1$ for every element a in the multiplicitive group $(\mathbb{Z}/p\mathbb{Z})^*$, which has order $p-1$. This implies the following theorem, known as Fermat's Little Theorem; but we will prove it without applying Lagrange.

Theorem 1.8. *Let p be prime, and a be an integer. Then $a^{p-1} \equiv 0 \mod p$ if $p \mid a$ and otherwise $a^{p-1} \equiv 1 \mod p$.*

Proof. If $p|a$ then $a^{p-1} \equiv 0^{p-1} = 0$, so we may assume that $p \nmid a$. As the statement is about the image of a modulo p we may assume that a is its image

in $(\mathbb{Z}/p\mathbb{Z})$. By the assumption $p \nmid a$, we then have that a is in $(\mathbb{Z}/p\mathbb{Z})^*$. The elements

$$a, 2a, \dots (p-1)a$$

are distinct elements, for if $ia \equiv ja$ then $p \mid j - i$ which implies $i = j$. So we have

$$a^{p-1}(1 \cdot 2 \cdot \dots \cdot p-1) = a \cdot 2a \cdot \dots \cdot (p-1)a = 1 \cdot 2 \cdot \dots \cdot (p-1).$$

Thus $a^{p-1} = 1$ in $(\mathbb{Z}/p\mathbb{Z})^*$, which means that $a^{p-1} \equiv 1 \mod p$. □

Immediate corollaries of this are the fact that for a with $\gcd(a, p) = 1$, we have

- $a^p \equiv a \mod p$
- $a^{-1} \equiv a^{p-2} \mod p$

This second seems nicer than the Extended Euclidean Algorithm for computing inverses modulo p. Conceptually it is easier, but computationally it is about the same: $2\log_2 p$.

The theorem also gives a nice check that a number is prime. Given an integer n the theorem gives us that if n is prime then for any smaller a, $a^{n-1} \equiv 1 \mod n$. So if we compute $2^{n-1} \mod n$ and get anything other than 1, then n is not prime. And we didn't have to find factors. It could be that we get a 1 when n is not prime, but doing the same calculation with a couple of different bases in place of 2, we can by pretty confident about whether or not n is prime. (However, this test is not perfect: there are known composite numbers n for which any a that is relatively prime to n yields $a^{n-1} \equiv 1 \mod n$. Such numbers are called Carmichael numbers.) We will look at better prime verification algorithms later.

Definition 1.9. For an integer a, the *order of a modulo p*, is its order in group $(\mathbb{Z}/p\mathbb{Z})^*$ that is, it is the minimum positive integer k such that $a^k \equiv 1 \mod p$.

Fermat's Little Theorem also gives us the following, (which you likely also know from group theory.)

Proposition 1.10. *Let p be a prime, and a be an integer not divisible by p. If $a^n \equiv 1 \mod p$ then the order of a modulo p divides n. In particular, it divides $p-1$.*

Proof. **Assume that $a^n \equiv 1 \mod p$. Let k be the order a modulo p. If $k \nmid n$ then $n = qp + r$ for some r with $1 \leq r < k$. So $1 \equiv a^n = a^{qp+r} = a^{qp} \cdot a^r \equiv a^r \mod p$. This contradicts $1 \leq r < k$.** See text. This is Prop 1.30. □

Now we know from field theory that any finite field has a cyclic multiplicative group. This is too much to prove here, even for finite fields $\mathbb{F}_p$ of prime order, but we will use it.

Theorem 1.11 (The Primitive Root Theorem). *Let p be a prime number. Then there exists an element $g \in \mathbb{F}_p^*$ such that*

$$\mathbb{F}_p^* = \{1, g, g^2, \dots, g^{p-2}\}.$$

The elements g of the theorem are called *generators* or *primitive roots* of $\mathbb{F}_p^*$.

Problem 1.7. Find a primitive root of $\mathbb{F}_7^*$. How many are there?

One can show that for k dividing $p-1$ there are exactly $\phi(k)$ elements in $\mathbb{F}_7^*$ having order k.

Notice that all the results of this section can easily be proved if we assume the Primitive Root Theorem.

Though $\mathbb{Z}/p\mathbb{Z}$ and $\mathbb{F}_p$ are isomorphic and the structure is clear when representing the field as $\mathbb{F}_p = \{0, 1, g, g^2, \dots, g^{p-2}\}$ for some primitive root g, what we will often use is the fact that the structure in terms of $\mathbb{Z}/p\mathbb{Z}$, is not clear.

More concretely, we can fix an isomorphism $f : \mathbb{F}_p^* \to (\mathbb{Z}/p\mathbb{Z})^*$ by choosing a primitive root $f(g)$ in $(\mathbb{Z}/p\mathbb{Z})$. The sequence $f(g), f(g^2), f(g^3), \dots, f(g^{p-1})$ list the elements of $(\mathbb{Z}/p\mathbb{Z})^*$ but mixes them up pretty good. One of the properties we will take advantage of is that there isn't any fast way to predict what integer x gives $f(g^x) = h$ for a number h in $(\mathbb{Z}/p\mathbb{Z})^*$.

Because of this we will use both $\mathbb{F}_p$ and $\mathbb{Z}/p\mathbb{Z}$. We will treat them notationally differently, but will exploit the fact that the are isomorphic.

1.6 Some History

Read this on your own if you are interested.

1.7 Symmetric and Asymmetric ciphers

We return to the subject of Cryptography, but this time with more rigour.

A *cipher* or *encryption scheme* is a tuple $(\mathcal{K}, \mathcal{M}, \mathcal{C}, e, d)$ where $\mathcal{M}$ is a space of *plaintexts* or messages or, $\mathcal{C}$ is a space of *ciphertexts*, $\mathcal{K}$ is a space of *keys*, e is function

$$e : \mathcal{K} \times \mathcal{M} \to \mathcal{C},$$

called *encryption*, and d is a function

$$d : \mathcal{K} \times \mathcal{C} \to \mathcal{K}$$

called *decryption*, such that $d(k, e(k, m)) = m$ for any key $k \in \mathcal{K}$ and message $m \in \mathcal{M}$.

For fixed key k, we normally write the encryption and decryption functions

$$e_k : \mathcal{M} \to \mathcal{C} : m \mapsto e(k, m)$$

and

$$d_k : \mathcal{C} \to \mathcal{M} : c \mapsto d(k, c).$$

When we analyse an encryption scheme, we assume that the enemy, Chuck, knows the encryption scheme, but not the key.

Clearly, to have $d(k, e(k, m)) = m$, e must be injective and d must be surjective. Other properties that we want of a cipher are

- P1) $e_k(m)$ is easy to compute for $k \in \mathcal{K}$ and $m \in \mathcal{M}$.
- P2) $d_k(c)$ is easy to compute given k and c.
- P3) Given many $c_1, \dots, c_n \in \mathcal{C}$, it is difficult to compute any of the $d_k(c_i)$ without having k.
- P4) Given several pairs $(c_1, d_k(c_1)), \dots, (c_n, d_k(c_n))$, it is difficult to compute $d_k(c)$ for some $c \notin \{c_1, \dots, c_n\}$.

Having a pair $(c_i, d_k(c_i))$ is equivalent to having $(e_k(m), m)$ for some $m \in \mathcal{M}$, because we know $e_k(m)$ decodes to m.

In our earlier shift cipher example, we took a message, changed it to a string of numbers, then encrypted these numbers by adding 9.

We don't consider the *encoding* of the message into numbers as part of the encryption e. This is a easily reversible part of overhead that it is expected that everyone knows.

In our shift cipher example, we thus had $\mathcal{M} = \mathcal{C} = \mathcal{K} = \mathbb{Z}/26\mathbb{Z}$. We used the key 9, and had $e_9(m) = m + 9 \mod 26$, and $d_9(m) = m - 9 \mod 26$.

We will usually encode messages as binary strings. So with that previous example could encode

19 14 12 14 17 17 14 22   00 19   11 20 13 02 07.

into length 5 binary strings as:

10011 01110 01100 01110 10001 10001 01110 10110   00000 . . .

We didn't apply e to the whole message at once, but to each letter at a time. This is weak, in that is does not satisfy property P4. This is because, $\mathcal{M}$ is too small. Once we know the encoding of each letter we can decode the message.

Generally we will string several letters together to get an element of $\mathcal{M}$. For example, we would encode the above message as length 20 binary strings:

$$10011011100110001110100011000101110101101111100000\ldots$$

where we have encoded the space as 11111. This is also considered a part of the encoding, and not of the encryption scheme.

The length B of the strings in $\mathcal{M}$ is called the *blocksize* of the cipher. Often we have to pad to final plaintext block so that the message fits into it.

Generally $\mathcal{M}, \mathcal{C}$, and $\mathcal{K}$ are all regular length binary strings. We use B_m, B_c and B_k to refer to their respective blocklengths.

It is generally accepted that with present computing power, a cipher is safe against an exhaustive search attack if the blocklength is at least 80.

Examples of Symmetric Ciphers

Let p be a large prime ($p \approx 2^{160}$), and $\mathcal{M} = \mathcal{C} = \mathcal{K} = \mathbb{F}_p^*$. Let encryption

$$e_k(m) = k \cdot m.$$

Then

$$d_k(m) = k^{-1} \cdot m$$

is decryption.

Given k, it takes about $2\log_2(p) = 320$ steps of the Extended Euclidean Algorithm to find k^{-1}. A computer can do this in seconds, so this is easy. (We don't have to find k^{-1} for each decryption, it is part of the key that we only have to generate once. So it isn't one of the properties P1 - P4 we mentioned. But it is desirable– it is inconvenient if it takes too long to generate a key.)

Encryption and decryption are easy, so we have properties P1 and P2. Since e_k is surjective, whatever k is, some message encodes to the ciphertext c_i, so having a couple ciphertexts doesn't help us decrypt anything. So we have property P3. However we don't have property P4. As soon as you have some pair $(c, m = d_k(c))$, you know that $c = km$ so you can compute

$$c \cdot m^{-1} = km \cdot m^{-1} = l.$$

So while this satisfies P3, and is secure if Chuck only sees encrypted messages, as soon as he knows what a message encodes to, then he can crack it.

One can replace e_k with any affine shift:

$$e_{(k_1,k_2)} : m \mapsto k_1 \cdot m + k_2.$$

Such so-called affine ciphers all fail the same *chosen plaintext attack*.