# CIS7 Unit 11 Chapter 5 Notes: Cryptography and Modular Arithmetic

Cryptography has been important through the ages for sending secret military information. It has also been used by secret societies like the Freemasons.

Today, computers and the internet have made cryptography a part of all our lives. Critical information like passwords, on-line purchases, and ATM transactions all use **cryptography**. Many companies protect their industrial secrets by encryption. Companies and individuals often encrypt their email to protect themselves from third party snooping. We will introduce some simple methods of encrypting that use algebraic methods, in particular ***modular arithmetic to encrypt messages***. We refer to the original message as the plaintext and the encrypted message as the **ciphertext**.

<https://youtu.be/r4HQ8Bp-pfw>

## Simple Shift Cipher

Julius Caesar was one of the first people known to use cryptography to protect messages of military significance. It is known as **Caesar's cipher**, the **shift cipher**, **Caesar's code** or **Caesar shift**, one of the simplest and most widely known encryption techniques. It is a type of **substitution cipher** in which each ***letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet***. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on. The method is named after Julius Caesar, who used it in his private correspondence. There are also letters of his to Cicero, as well as to his intimates on private affairs, and in the latter, if he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out.

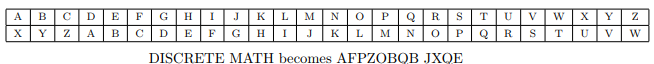
The encryption step performed by a Caesar cipher is often incorporated as part of more complex schemes, such as the **Vigenère cipher**, <https://youtu.be/-SWThsve-Hk>, and still has modern application in the **ROT13** “**rotate by 13 places**” system. As with all single-alphabet substitution ciphers, the Caesar cipher is easily broken and in modern practice offers essentially no communications security.

**Caesar ciphers** can be found today in children's toys such as **secret decoder rings**. A Caesar shift of thirteen is also performed in the ROT13 algorithm, a simple method of obfuscating text widely found on ***Usenet*** and used to obscure text (such as joke punchlines and story spoilers), but not seriously used as a method of encryption.

We call this the Caesar Cipher. ***Every letter is shifted over by three***. Using our modern alphabet, look up a plaintext letter in the top row of this table and replace that letter with the corresponding letter in the bottom row. To decrypt, look up a cipher text letter in the bottom row and replace it with the corresponding letter in the upper row.

Caesar cipher example of three places substitution "DISCRETE MATH"

Using this table, a letter is encrypted by replacing it with a letter three places further on in the alphabet or shifting it forward by three places.

The quote from Suetonius tells us that the shift was actually in the other direction, a letter was encrypted by replacing it with a letter three places further back in the alphabet or shifting it back by three places. Decrypting was then done by replacing it with a letter three places further on in the alphabet or shifting it forward by three places.

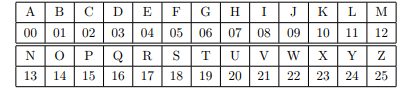
More generally, we could shift by any number from 1 to 25, for example, if we shift by 7,

DISCRETE MATH becomes KPZJYLAL THAO.

http://practicalcryptography.com/media/latex/bdd325f4306cf573601de60e4e175dfbe7acbb14-11pt.png

## Encoding

Using numbers instead of letters gives us the advantage that we can put math and computers to work to encrypt and decrypt for us. So, ***the first thing we will do is encode our plaintext***, that is, ***replace the letters with numbers by an agreed upon, public method***. There are many ways we can do this. Computers mostly use ASCII (American Standard Code for Information Interchange <http://www.lookuptables.com>) to represent characters. We will just use the numbers from 0 to 25 for the letters A to Z (or a to z).



We have added leading 0s to the single digit numbers so that all the codes are 2 digits long. If we need punctuation, we will use 26 for a space, 27 for a period and 28 for a comma.

**Encoding**, ***going from letters to numbers***, and **decoding*, going from numbers back to letters***, are different from encrypting (or enciphering) and decrypting (or deciphering). There is nothing secret about encoding and decoding.

There is nothing secret about encoding and decoding. ***MATH IS COOL becomes 12001907 0818 02141411*** if we leave the spaces or 120019072608182602141411 if we encode the spaces.

## One-time Pad

**One-time pad (OTP),** also called **Vernam-cipher** or the perfect cipher, is a crypto algorithm where **plaintext is combined with a random key**. It is the only existing mathematically **unbreakable encryption**. Used by Special Operations teams and resistance groups during WW2, popular with intelligence agencies and their spies during the Cold War and beyond, protecting diplomatic and military message traffic around the world for many decades, the one-time pad gained a reputation as a simple yet solid encryption system with an absolute security which is unmatched by today's modern crypto algorithms.

* The **key is at least as long as the message or data** that must be encrypted.
* The **key is truly random** (not generated by a simple computer function or such).
* **Key and plaintext are calculated modulo 10 (digits), modulo 26 (letters) or modulo 2 (binary)**
* Each **key is used only once**, and both sender and receiver must destroy their key after use.
* There should **only be two copies of the key**: one for the sender and one for the receiver (some exceptions exist for multiple receivers).

See <http://users.telenet.be/d.rijmenants/en/onetimepad.htm>

Video: <https://www.khanacademy.org/computing/computer-science/cryptography/crypt/v/one-time-pad>

## Vigenere Ciphering:

This cipher was invented in 1586 by Blaise de Vigenère with a reciprocal table of ten alphabets. Vigenère's version used an agreed-upon letter of the alphabet as a primer, making the key by writing down that letter and then the rest of the message. The Vigenère Cipher was adapted as a twist on the standard Caesar cipher to reduce the effectiveness of performing frequency analysis on the ciphertext. The cipher accomplishes this using **uses a text string (for example, a word) as a key, which is then used for doing a number of alphabet shifts on the plaintext**.

Vigenere cipher algorithm: E = M+K mod 26 

Expressed mathematically, the encryption of the message at letter \*i\*, is equal to the alphabetic value of \*i\* in the plaintext plus the alphabetic value of the corresponding \*i\* in the key.

Decryption is the same process reversed, subtracting the key instead of adding to arrive back at the original, plaintext value.

Vigenere decipher algorithm: D = C - K mod 26

More popular autokeys use a tabula recta, a square with 26 copies of the alphabet, the first line starting with 'A', the next line starting with 'B' etc. Instead of a single letter, a short agreed-on keyword is used, and the key is generated by writing down the primer and then the rest of the message, as in Vigenère's version. To encrypt a plaintext, the row with the first letter of the message and the column with the first letter of the key are located. The letter in which the row and the column cross is the ciphertext letter.

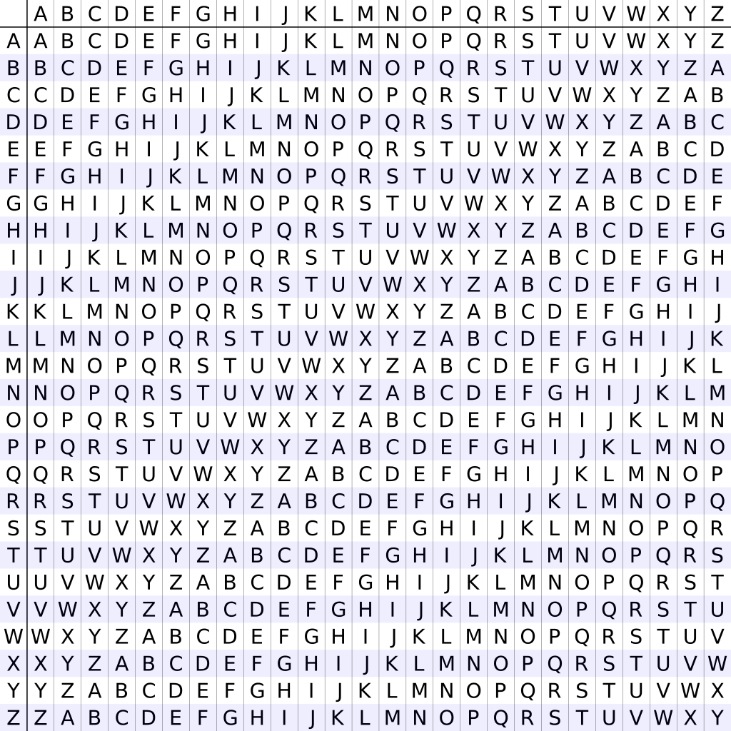
**To encrypt a letter, we write the key underneath the plaintext. We take the plaintext letter at the top and the key letter on the left. The cross section of those two letters is the ciphertext. In the first letter of our example below, the crossing between the plaintext T and key X is ciphertext Q.**

**Example:**

**Plaintext: T H I S I S S E C R E T**

**OTP-Key : X V H E U W N O P G D Z**

**Ciphertext: Q C P W C O F S R X H S**



An **autokey cipher (also known as the autoclave cipher**) is a cipher that **incorporates the message (the plaintext) into the key**. The key is generated from the message in some automated fashion, sometimes by selecting certain letters from the text or, more commonly, by adding a short primer key to the front of the message.

There are two forms of autokey cipher: key-autokey and text-autokey ciphers. A key-autokey cipher uses previous members of the keystream to determine the next element in the keystream. A text-autokey uses the previous message text to determine the next element in the keystream.

In modern cryptography, self-synchronizing stream ciphers are autokey ciphers.

## Simple Substitution Cipher

A simple substitution cipher is a cryptographic system in which letters (or their codes), are arbitrarily transposed or replaced with other letters (or their codes). The Caesar Cipher and general Shift Cipher are both simple substitution ciphers. Each letter is replaced by another letter. We will study some simple substitution ciphers that can be generated by using the mod or modulo function.

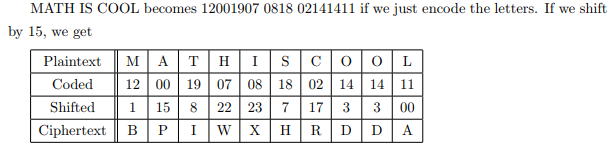
Once we have encoded the letters A, ..., Z, a general shift cipher with shift k can be described by:

**n → (n + k) mod 26.**

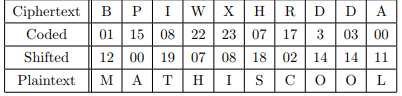
or by

**n → (n + k) mod 29.**

if we encode and encipher space, “.” and “,” as well as the letters A · · · Z. If we want our encrypted message to look like letters, possibly with punctuation, we decode the shifted codes to get our ciphertext. Here’s an example:



If we receive the message “BPIWXHRDDA” and know that the shift key is 15, we just reverse the procedure above to decrypt our message, code the letters, ***shift by −15 which is the same as +11 mod 26***, decode the result.

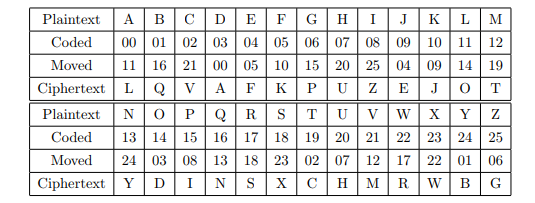


## Linear Ciphers

We can create somewhat more complex simple substitution ciphers by using linear functions along with mod instead of just adding a constant and then using mod. Let’s work again with just the 26 letters. In general, we choose two constants m and k then generate a linear cipher is given by

**a → (m · a + k) mod 26.**

Let’s look at an example with m = 5 and k = 11



***Two different letters always have two different cipher letters*** so an enciphered message should be decipherable. No letter is enciphered by itself so the message won’t be trivial to read. Given the table, it is pretty easy to decipher a message.

Can you decipher this message? QHKKBXOLBXMLTISFX

There are a few questions we should think about when we make a simple linear cipher.

1. What values of **m** and **k** make good linear ciphers if the alphabet has 26 characters?
2. What if the alphabet has 29 characters, e.g. with space, “.” and “,” included?
3. What if the alphabet has 128 ASCII characters?
4. Can we say anything in general for an alphabet of n characters?
5. Can the person receiving our message decipher it without reconstructing the table, i.e. with just knowing n, m, and k? This will be important if n is large.

To answer these questions, we need to understand more about mod and the arithmetic it induces.

## The mod function

The **mod function** has many applications in computer science. It is used for simple and complex ***cryptography, calendars and clocks, random number generators, and hash tables*** for a start. We will then use the ***mod function, to generate shift ciphers and more general linear ciphers***. If ***n is an integer that is greater than 1, and a is any integer***, then **a mod n**.

A **mod n** is defined when n is negative but we’ll restrict our attention to n > 1. In this case, **a mod n** is ***always an integer between 0 and n − 1***. In Scheme the mod function is given by (modulo a n). is the integer remainder when a is divided by n. In fact, **a mod n** is defined ***when n is negative but we’ll restrict our attention to n > 1***. In this case, a mod n is always an integer between 0 and n − 1. In Scheme the mod function is given by (modulo a n).

Example 5.1

1. 17 mod 5 = 2

17 divided by 5 is 3; the remainder is 2.

1. 8 mod 5 = 3

8 divided by 5 is 1; the remainder is 3.

1. 55 mod 5 = 0

55 divided by 5 is 11; the remainder is 0.

1. 4 mod 5 = 4

4 divided by 5 is 0; the remainder is 4.

1. 37 mod 17 = 3

37 divided by 17 is 2; the remainder is 3.

**How do we evaluate a mod n when a is negative**? Remember that as long as n > 1, the values of a mod n must be between 0 and n − 1. In general, a mod n is the unique integer between 0 and n − 1 that satisfies **a = q · n + a** ***mod n for some integer q***.

Example 5.2

1. −17 mod 5 = 3

**−17 = −4 · 5 + 3**

1. −8 mod 5 = 2

**−8 = −2 · 5 + 2**

1. −55 mod 5 = 0

**−55 = −11 · 5 + 0**

1. −4 mod 5 = 1

**−4 = −1 · 5+ 1**

1. −37 mod 17 = 14

**−37 = −3 · 17 + 14**

## Properties of mod

Let **n be an integer greater than 1**, and let **a and b be any integers**, then

1. If **a mod n = b mod n** then there is an integer k such that **a − b = k · n.**
2. **(a + b) mod n = ((a mod n) + (b mod n)) mod n**
3. **(a · b) mod n = ((a mod n) · (b mod n)) mod n**
4. **−a mod n = n − (a mod n)**

## Modular Arithmetic

Once we fix an integer n greater than 1, the properties of mod , we cited above, allow us to talk about arithmetic mod n on the set Zn of integers from 0 to n − 1. We define

1. **a + b = (a + b) mod n**

**(A + B) mod C = (A mod C + B mod C) mod C**

**Let A=14, B=17, C=5**

**(14+17)mod 5 = ((14 mod 5) + (17mod5)) mod 5**

**(14 mod 5 + 17 mod 5) mod 5**

**= (4 + 2) mod 5**

**= 1**

1. **a × b = (a × b) mod n**

**(A \* B) mod C = (A mod C \* B mod C) mod C**

**Example for Multiplication:**

**Let A=4, B=7, C=6**

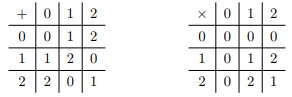
**(A \* B) mod C = (A mod C \* B mod C) mod C**

**= (A \* B) mod C**

**= (4 \* 7) mod 6**

**= 28 mod 6**

**= 4**

Consider these + and × tables for arithmetic mod 3.

Arithmetic mod 3 has some very nice properties. If a, b, and c are in Z3 (the set {0, 1, 2}) then

**closure**: a + b and a × b are in Z3

**commutativity**: a + b = b + a and a × b = b × a

**associativity**: (a + b) + c = a + (b + c) and (a × b) × c = a × (b × c)

**identity** +: 0 is an additive identity a + 0 = a for all a ∈ Z3

**identity** ×: 1 is a multiplicative identity a × 1 = a for all a ∈ Z3

**inverse** +: Every a ∈ Z3 has an additive inverse b ∈ Z3 such that a + b = 0

**inverse** ×: Every non-zero a ∈ Z3 has an multiplicative inverse b ∈ Z3 such that a × b = 1

**distributive** law: c × (a + b) = (c × a) + (c × b)

## Powers mod n / Modulus Exponentiation

We often have to compute powers of numbers mod n. The **RSA Encryption Algorithm**

**RSA (Rivest–Shamir–Adleman)** is one of the ***first public-key cryptosystems and is widely used for secure data transmission***. In such a cryptosystem, the encryption key is public and it is different from the decryption key which is kept secret (private). In RSA, this ***asymmetry is based on the practical difficulty of the factorization of the product of two large prime numbers***, the "factoring problem". Anyone can use the public key to encrypt a message, but with currently published methods, and if the public key is large enough, only someone with knowledge of the prime numbers can decode the message feasibly. **RSA** is a relatively ***slow algorithm***, and because of this, it is less commonly used to directly encrypt user data. More often, RSA passes encrypted shared keys for symmetric key cryptography which in turn can perform bulk encryption-decryption operations at much higher speed.

We can easily **compute powers mod n when the exponent is itself a power of 2 by using the property**

**(a · b) mod n = ((a mod n) · (b mod n)) mod n.**

The idea is to alternate evaluating mod n and squaringAlternative evaluation of mod n in squaring

**Example:**

Suppose we want to calculate 290 mod 13

* 290= 250 \* 240
* 290 mod 13 = (250 \* 240) mod 13
* 290 mod 13 = (250 mod 13 \* 240 mod 13) mod 13
* 290 mod 13 = ( 4 \* 3 ) mod 13
* 290 mod 13 = 12 mod 13
* 290 mod 13 = 12

Example 5.4

1. 372 mod 3 = (37 mod 3)2 = 12 = 1
2. 11532 mod 7 = (115 mod 7)32 = ((115 mod 7)4 ) 8 = (4 mod 7)8 = (16 mod 7)4 = (2 mod 7)4 = (16 mod 7) = 2

Using modular multiplication rules:

**A2** mod C = **(A \* A)** mod C = (**(A mod C)** \* **(A mod C)**) mod C

We can use this to calculate 7256 mod 13 **quickly**

**71** mod 13 = **7**  
**72** mod 13 = (**71 \*71** ) mod 13 = (**71 mod 13 \* 71 mod 13**) mod 13

We can *substitute* our previous result for **71 mod 13** into this equation.

**72** mod 13 = (**7 \*7**) mod 13 = **49** mod 13 = **10**  
**72 mod 13 = 10**

**74** mod 13 = (**72 \*72**) mod 13 = (**72 mod 13** \* **72 mod 13**) mod 13

We can *substitute* our previous result for **72 mod 13** into this equation.

**74** mod 13 = (**10 \* 10**) mod 13 = **100** mod 13 = **9**  
**74 mod 13 = 9**

**78** mod 13 = (**74 \* 74**) mod 13 = (**74 mod 13** \* **74 mod 13**) mod 13

We can *substitute* our previous result for **74 mod 13** into this equation.

**78** mod 13 = (**9 \* 9**) mod 13 = **81** mod 13 = **3**  
**78 mod 13 = 3**

We continue in this manner, substituting previous results into our equations.

**...after 5 iterations we hit:**

**7256** mod 13 = (**7128 \* 7128**) mod 13 = (**7128 mod 13** \* **7128 mod 13**) mod 13  
**7256**mod 13 = (**3 \* 3**) mod 13 = **9** mod 13 = **9**  
**7256mod 13 = 9**

This has given us a method to calculate **AB mod C** quickly provided that **B is a power of 2**.

However, we also need a method for fast modular exponentiation when **B is not a power of 2**.

To compute **ammod n** when m is not a power of 2, we use the binary representation of m to express m as a sum of powers of 2 and the rule (see C.1.1) for the product of two powers with the same base. In general, m can be expressed as

exponential value in mod for a^m * modn

Where bi = 0 or 1 for 0 ≤ i ≤ k.



