**­­­­­­­INF143A Exam Questions**

**Classic ciphers**

(Encrypt or decrypt messages, explain how the cipher works)

* Caeser cipher
  + Shift the alphabet against another alphabet x places to the right, where x is a number from 1 to 26. This can also be called the key. When you reach the end, loop the remaining alphabet.

Key: 2

Message: HEI

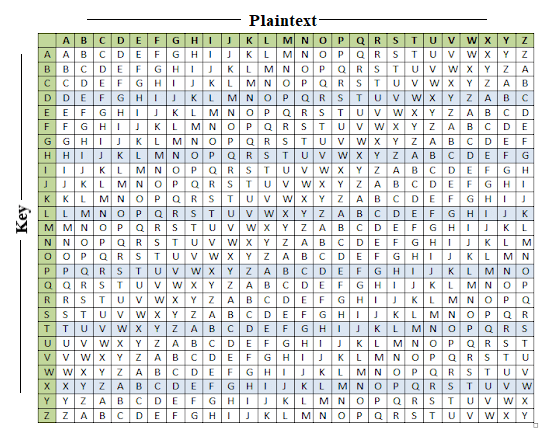
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Y Z A B C D E ­­­­F G H I J K L M N O P Q R S T U V W X

Cipher: FCG

* Vigenere cipher
  + You have a matrix of columns and rows, where the data is alphabets (see picture). You have a message and a key, as well as a key stream. You look up the first letter of the message in the columns of the matrix, and the first letter of the keystream in the rows of the matrix and find the mapping of these two – this is the first letter of the cipher.

Matrix:



Key: APPLE

Message: VIGENERE

Key-stream: APPLEAPL

Cipher: VXVPREGP

* Monoalphabetic substitution cipher
  + Works the same as caeser but instead of putting in a alphabet shifted x places to the right, you now put in the shuffled alphabet instead. This is your key. You place this shuffled alphabet under the real alphabet, and every letter from the real alphabet will always correspond to a letter in the shuffled alphabet.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

I L W S H E F A D B G X K V M T C U Z R N Y Q O P J

Key: I L W S H E F A D B G X K V M T C U Z R N Y Q O P J

Message: HEI

Cipher: AHD

* Playfair cipher
  + You have your plaintext. You split it into pairs of two, where the same letter can’t be twice in the pair. If the same letter occurs, replace it with a random letter. If the last pair is incomplete, add a z at the end.

Input1 example: hello -> “he” “lx” “lo”

Input2 example: baked -> “ba” “ke” “dz”

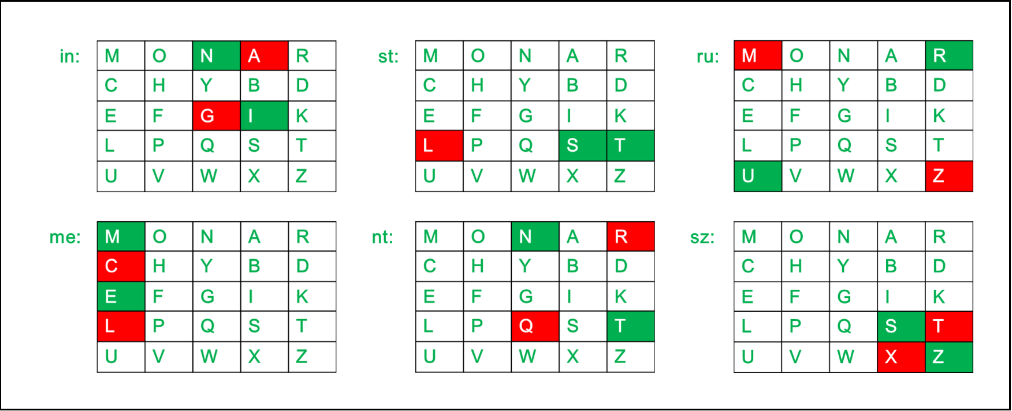
You then create a 5x5 matrix where the start of the matrix is your key, and the remaining spots are filled with the alphabet. Letters can’t repeat. J is often discarded. If J is in the plaintext, I am discarded instead.

You then look for the first letter of message in the matrix and find corresponding cipher letter:

**If both the letters are in the same column**: Take the letter below each one (going back to the top if at the bottom).

**If both the letters are in the same row**: Take the letter to the right of each one (going back to the leftmost if at the rightmost position)

**If neither of the above rules is true**: Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle



Message: APPLE

Key: RANDOM

Matrix:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| R | A | N | D | O |
| M | B | C | E | F |
| G | H | I | K | L |
| P | Q | S | T | U |
| V | W | X | Y | Z |

Split message: “AP” “PL” “EZ”

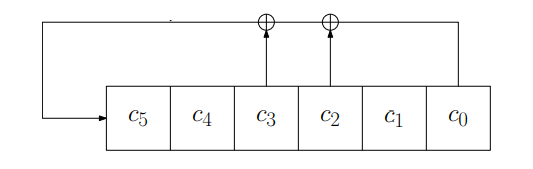
Cipher: RQ UG FY

**Stream ciphers**

Stream cipher is a type of cipher where a keystream is generated from a key, and the key is XORed with the plaintext. To decipher you XOR the key with the cipher.

LFSR can be represented in 3 ways:

If we have f(c0, . . . , c5) = c0 + c2 + c3 we would get

* Polynomial: x6 + x3 + x2 + 1
* Diagram: 

LFSR and primitive polynomial

If the LFSR covers all non-zero states on one loop, **it is primitive**. If any state is repeated prematurely, it is **not primitive**.

So for 0 0 0 0 0 1 we would need all 2^6 – 1 (minus the zero state, it lies on its own loop) states to be on one loop for it to be primitive.

Clocking

Clocking the LFSR with polynomial: x6 + x3 + x2 + 1 starting with 0 0 0 0 0 1

0 0 0 0 0 1

1 0 0 0 0 0

0 1 0 0 0 0

0 0 1 0 0 0

1 0 0 1 0 0

and so on…

Attacks on LFSR:

If an LFSR is used directly as keystream generator, the resulting stream cipher can be very easily broken with a known plaintext attack.

Suppose that the attacker knowns the first e.g. 100 bits of a plaintext message, and the corresponding 100 bits of the encrypted ciphertext message. By XOR-ing them together, he can recover the first 100 bits of the keystream. The attacker now wants to recover the connections in its feedback line. Suppose that the attacker somehow knows the degree m of the LFSR. If s0, s1, . . . , sn, . . . are the bits of the kestream, then the attacker knows that we have:



where a0, . . . , am−1 are the coefficients from the associated polynomial.

**Block ciphers**

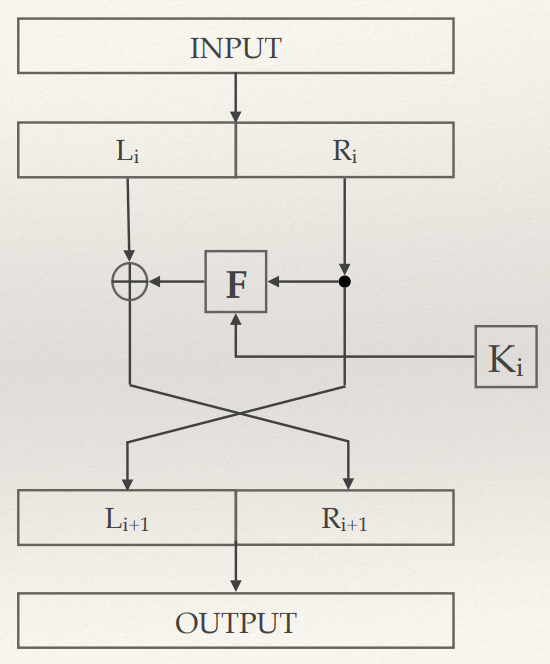
Feistel Network

The network consists of n+1 rounds. They key K is used to calculate the subkeys K0, K1, … , Kn. In each round, the round input is combined with the key using a round function F to produce the round input. The input of the first round is the plaintext, and the output of the last round is the ciphertext.

Single round:

Li+1 = Ri

Ri+1 = Li XOR F(Ri, Ki)



Decryption:

Decryption is the same as encryption, but in reverse:

Ri = Li+1

Li = Ri+1 XOR F(Li+1, Ki)

Meet in the middle attack:

This is a type of bruteforce attack where you calculate all possible keys used for encryption. You also know the plaintext before encryption and the ciphertext after 2 rounds of encryption. You encrypt the plaintext with every possible key and store the results. You then decrypt the ciphertext with every possible key and store the result. You then check every single middle value of the encryption and decryption, if you find a value that is the same for both, you have found a key pair that encrypts to the given cipher.

Encrypt(plaintext, key\_n) = x

Decrypt(cipher, key\_m) = y

If(x==y) we have found a key-pair (key\_n, key\_m) that encrypts to the given cipher

3DES and meet-in-the-middle attack

The short key-length of DES can be remedied by encrypting the plaintext with DES thrice, each time with a different key. The total length of the key is now 3 x 56 = 168 bits. Triple DES (3DES) and DES with key whitening (DESX) are modifications to DES that overcome the security problems arising from the insufficient key length.

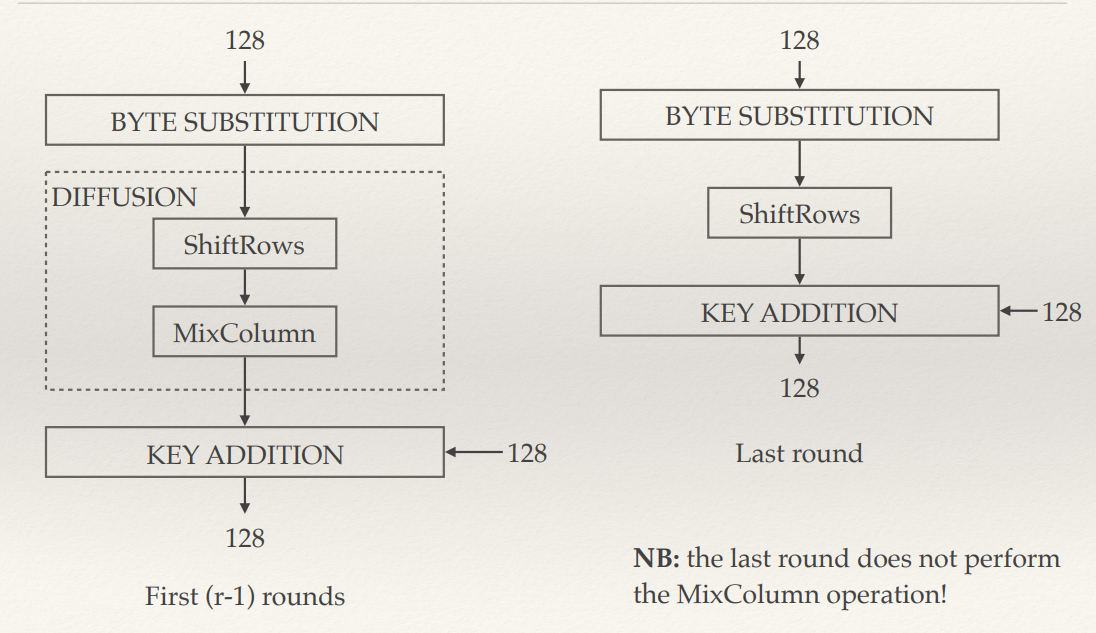
DES is not secure

No efficient analytical attacks have been found against DES, but the short key length (56 bits) allowed for successful brute force attacks with the increase in computational power.

AES single round

Each round of AES consists of several layers which serve to introduce confusion or diffusion; there are three types of layers which all process 128-bit blocks:

* Key addition layer: XOR’s the round key with the input
* Byte substitution layer (S-Box): introduces confusion
* Diffusion layer: provides diffusion, and consists of two sublayers:
  + ShiftRows
  + MixColumn



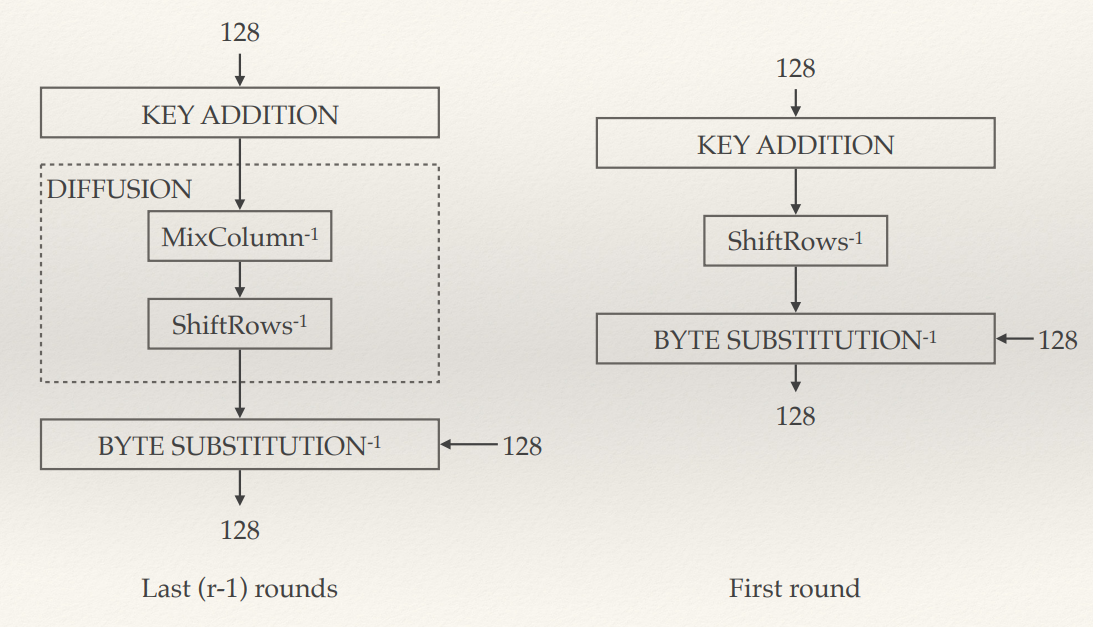
AES decryption

Like with DES, the encryption is “reversed” round by round, with the first round of decryption “cancelling” the last round of encryption, etc.

Unlike DES, encryption is not the same as decryption, i.e. it is not sufficient to simply reverse the key schedule

Each of the four layers (byte substitution, ShiftRows, MixColumn, and key addition) defines an inverse operation

The inverse operations are applied in the reverse order with respect to encryption



**Hash functions**

A hash function is a function which accepts a message of arbitrary length as input, and outputs a fingerprint of fixed length. They can be used to map data of arbitrary size to fixed-size values. The values returned by a hash function are called hash values, hash codes, digests, or simply hashes. The values are usually used to index a fixed-size table called a hash table.

Desirable properties for hash functions

* Pre-image resistance:

If h(x) = y, then y is called the image of x, and x is a pre-image of y via h. A hash function is said to have pre-image resistance if it is a one-way function. This means that it is easy to compute y = h(x) given x, but it is very difficult to compute x such that h(x) = y given y.

* Collision resistance:

Given some fixed input x1, it is practically difficult to find x2 with h(x1) = h(x2). You shouldn’t find a x2 that has the same hash as a message x1.

* Strong collision resistance:

It should be difficult to find any two values x1 and x2 with h(x1) = h(x2).

Build block cipher from a hash function

A cryptographically secure hash function h(x) can be converted to a symmetric cipher operating on blocks of size b.

Suppose a shared key K is agreed upon and M1, …, Mn, is the sequence of blocks to be encrypted. One chooses an initialization vector M0 = IV. To encrypt block Mi, one computes the hash h(K || Mi-1), takes the first b bits of the hash, and XOR-s them with Mi.

**Digital signatures**

Basic principles

Intuitively, digital signature schemes are like asymmetric ciphers but “in reverse”. In the case of an asymmetric encryption scheme, anyone can encrypt a message (using the public key), but only the intended recipient is able to decrypt it (using the private key).

In the case of a signature, only the legitimate sender of a message should be able to send it (using a secret key), but anyone should be able to verify the signature (using a public key). Indeed, virtually all digital signature schemes are based on modifications of asymmetric encryption schemes. In some cases, the modifications can be quite substantial; while in others, like the RSA digital signature scheme below, the digital signature is a straightforward adaption of the corresponding encryption scheme.

Digital signatures allow for authentication (verify who is sending the message) and non-repudiation (prove that the sender did send the message).

RSA digital signature scheme

RSA digital signature scheme consists of a setup, signature and verification that looks as followed:

Setup:

1. Pick 2 large primes **p** and **q**
2. Compute n = pq and m = ϕ(n) = (p – 1) ( q – 1)
3. Pick an integer e in Zm with gcd(e, m) = 1
4. Take d = e^-1 (mod m)
5. Publish (n, e) as the public key, and preserve (d) as the private key

Signature:

1. Given a message x in Zn
2. Compute s = x^d (mod n)
3. The signed message is (x,s)

Verification:

1. Given a pair (x, s) of a message and a signature
2. Compute x’ = s^e (mod n)
3. Accept the message if and only if x = x’

RSA digital signature example

Setup:

Let (p, q) = (**3, 11**)

Let **e = 3**

n = p \* q = 3 \* 11

**n = 33**

m = ϕ(n) = (p – 1) ( q – 1) = (3 – 1) (11 – 1)

**m = 20**

Find d such that d = e^-1 (mod m) = 3^-1 (mod 20)

d \* 3 (mod 20) = 1 (mod 20)

7 \* 3 (mod 20) = 1 (mod 20)

21 (mod 20) = 1 (mod 20)

1 = 1

**d = 7**

Signature:

Sign the message **x = 4**

s = x^d (mod n) = 4^7 (mod 33) = 16384 (mod 33)

**s = 16**

Pair: (x, s) = (4, 16)

Verification:

x’ = s^e (mod n) = 16^3 (mod 33) = **4**

Since x’ = x -> 4 = 4, we have confirmed that the message was sent from who claimed they sent it.

Elgamal digital signature scheme

In contrast to the RSA digital signature scheme above (which is essentially the same as RSA encryption but “in reverse”), the Elgamal digital signature scheme is somewhat different from the Elgamal encryption scheme.

Setup:

1. Pick large prime p
2. Pick a generator g of Zp
3. Choose d in {2,3,…, p-2}
4. Compute β = g^d (mod p)
5. The public key is (p, g, β) and the private key is (d)

Signature:

1. Pick ephemeral key Ke in {1,2,…,p-2} with gcd (Ke, p-1) = 1
2. Compute r = g^Ke (mod p)
3. Compute s = (x – dr) K^1e (mod p -1)
4. The signed message is (x, (r, s))

Verification:

1. Compute t = β^r \* r^s (mod p)
2. Accept only if t = g^x (mod p)

Elgamal digital signature example:

Setup:

Let

**p = 29**

**g = 2**

**d = 12**

β = g^d (mod p) = 2^12 (mod 29)

**β = 7**

Public key: (p, g , β) = (29, 2, 7)

Private key: (12)

Signature:

Pick ephemeral key **Ke = 5** for signing the message **x = 26**

r = g^Ke (mod p) = 2^5 (mod 29)

**r = 3**

s = (x – dr) Ke^-1 (mod p -1) = (26 – 36) 17 (mod 28) = -170 (mod 28)

**s = 26**

Signed message: (x, (r, s)) = (26, (3, 17)

Verification:

To verify, we compute:

t = β^r \* r^s (mod p) = 7^3 \* 3^26 (mod 29) = 22

**t = 22**

Accept only if t = g^x (mod p)

g^x (mod p) = 2^26 (mod 29) = 22

**t = g^x** -> 22 = 22 and we know that this message is authentic.

**Asymmetric cryptography**

Difference between asymmetric and symmetric:

Symmetric: a single secret key is shared by both participating parties; this same key is used for both encryption and decryption.

Asymmetric: both participating parties have their own pair of keys, private and public key. The private key is not made public, but the public key can be made public.

Pros and cons:

* Asymmetric reduces the numbers of keys needed for communication between many participants, symmetric would need a private key for each participant parties, while asymmetric only would need to generate a private and public key pair, where the public one is given to the parties who wants to communicate with the user.
* Asymmetric tend to be slower than symmetric.
* Asymmetric schemes are not used for encrypting large data since it tends to be slow, but rather used for communicating the secret keys for using symmetric encryption.
* Asymmetric is more safe than symmetric due to the number and usage of keys.

Symmetric encryption algorithms:

* AES
* DES
* 3DES

Asymmetric encryption algorithms:

* RSA cryptosystem
* Elgamal
* Diffie-Hellman

RSA cryptosystem:

The RSA cryptosystem is a asymmetric algorithm that works like this:

Key generation:

Every participant needs to have a key pair (K\_public, K\_private).

1. Choose two large primes p and q
2. Compute n = p \* q
3. Compute m = (p – 1) (q – 1)
4. Select e between 0 and m – 1 from Zm such that gcd(e, m) = 1
5. Find inverse d of e modulo m, i. e compute d = e^1 (mod m)
6. Set K\_public = (n, e) and K\_priv = (d)

Example:

1. We choose p = 11 and q = 17
2. n = 11 \* 17 = 187
3. m = 10 \* 16 = 160
4. Select e = 63 since gcd(63, 160) = 1
5. Compute d = 63^-1 (mod 160)

d \* 63 (mod 160) = 1 (mod 160)

127 \* 63 (mod 160) = 8001 (mod 160) = 1 (mod 160) = 1

d = 127

1. Public key is K\_public = (187, 63) and K\_private = (127)

Encryption:

The message can be assumed to be a number between 0 and n, i.e all elements of Zn. Large values of p and q adds security and allows for wider range of messages.

If plaintext message is x:

y = ENC(x, K\_public) = x^e (mod n)

Example:

With K\_public = (187, 63) and x = 100, we get:

y = 100^63 (mod 187) = 144

Decryption:

We can decrypt y if we know the private key K\_private = d:

x’ = DEC(y, K\_private) = y^d (mod n)

Example:

With y = 144:

x’ = 144^127 (mod 187) = 100 = x

RSA efficiency optimization

* **Square and multiply:** For instance, suppose that we want to compute x^34. We can, of course, multiply x by itself 33 times, obtaining x → x 2 → x 3 → · · · → x 33 → x 34; or, we can multiply as x → x 2 → x 4 → x 8 → x 16 → x 17 → x 34 , (7) which only requires 6 operations.
* **Short public exponents:** A strategy that is frequently used to speed up encryption using RSA is to select the public exponent e to be a number co-prime with n (so that it satisfies the requirements of RSA) and having small binary weight (so that exponentiation can be performed quickly).
* **Chinese remainder theorem:** The CRT is a useful result from number theory that allows us to “break up” a modular equation modulo n = pq into two equations, one modulo p and one modulo q, and to obtain the solution to the original equation by solving the two “smaller” equations separately.

DHKE – Diffie Hellmann Key Exchange

DHKE is an asymmetric encryption scheme based on the discrete logarithm problem (DLP), which is another well-known hard problem. As the name “key exchange” suggests, the DHKE scheme is primarily designed for exchanging keys to be used for communication through a symmetric cipher. In practice, the DHKE is used in SSH, TLS, and other cryptographic protocols.

We have two parties Bob and Alice. They want to agree on a secret key over an insecure channel.

Setup:

1. Choose large prime p
2. Choose a generator g of Zp
3. Public (p, g)

Key-exchange:

1. Alice picks a of Z\*p, bob picks b of Z\*p. These can be -1 or 1.
2. Alice computes A = g^a, Bob computes B = g^b
3. Alice and Bob sends A and B to each other (unencrypted)
4. Alice computes K = B^a, Bob computes K = A^b
5. The secret key is now K = A^b = B^a = g^ab

Elgamal encryption

The Elgamal encryption scheme’s idea is that the two parties agree of a masking key that is then used to encrypt and decrypt messages.

Key generation and exchange:

1. Bob chooses a large prime p and a generator g of Z\*p
2. Bob chooses a private key d of Z\*p that cant be -1 or 1
3. Bob computes the public key β = g^d, which he sends to Alice (unencrypted)

Encryption:

1. Alice chooses i of Z\*p where i cant be 1 or -1
2. Alice computes ephemeral key Ke = g^i
3. Alice computes masking key Km = β^i
4. Alice encrypts the message x by computing y = x\*Km
5. Alice transmits the pair (Ke, y) to Bob

Decryption:

1. Bob computes the masking key Km = (Ke)^d
2. Bob decrypts x = y\*Km^-1

Malleable: Finally, Elgamal is malleable, i.e. an attacker can manipulate the plaintext in a predictable way even without decrypting it.

Elliptic curves:

Elliptic curves allow us to shorten the key because of the discrete logarithm problem.

**Modes of operation**

The encryption of a series of blocks of data can be performed in different ways; these modes of operation describe how the encryption of a given block depends on the other blocks of the plaintext.

Types of modes:

* ECB (Electronic codebook mode)
* OFB (Output feedback mode)
* CBC (Cipher block chaining mode)
* GCM (Galois counter mode)

ECB: Each block is encrypted individually. The main vulnerability is that the same plaintext block will always encrypt to the same ciphertext block. This allows a substitution attack to be performed even if the underlying cipher is secure and unbroken!

CBC: The blocks are “chained together” so that the encryption of Pi+1 depends on the previously encrypted Ci

GCM security: GCM provides authentication in addition to confidentiality, i.e. it provides a mechanism allowing the receiver (Bob) to verify that the message really originates from Alice.

**Modular arithmetic**

Compute something to the power of something module something (a^b mod x).

Example:

5^20 (mod 17):

Using square and multiply

5^2: 5^2 (mod 17) = 8

5^4: 8^2 (mod 17) = 13

5^8: -4^2 (mod 17) = 16

5^16: -1^2 (mod 17) = 1

5^20: 5^4 \* 5^16 (mod 17) = 1 \* 13 (mod 17) = 13

**Maths**

Finite fields

F2^6 means (0,1,1,0,0,1) with some bits 0 and 1

To construct finite field we need to use a irreducible polynomial of degree 6 if n = 6.

Smallest finite field / irreducible polynomial = F2

An irreducible polynomial has no roots in F2

**Addition** and **subtraction** in finite field is just XOR:

(0,1,1,0,0,1)

(1,1,1,0,0,1

= (1,0,0,0,0,0)

**Multiplication:**

(1, 0, 1) \* (0,0,1)

We have the irreducible polynomial of degree 3: x^3 + x ^1

a^3 + a + 1 = 0

=> a^3 = a + 1

(1, 0, 1) = 1 + a^2

(0, 0, 1) = a^2

(1 + a^2) \* (a^2) = a^2 + a^4 = a^2 + (a + 1)\*a

= ~~a^2~~ + ~~a^2~~ + 1 = 1 = (1,0,0)

Given a irreducible polynomial, we check if its primitive by converting the polynomial to alpha values like above:

x^3 + x ^1

a^3 + a + 1 = 0

=> a^3 = a + 1

We then check if all the elements in the finite field lays on one cycle (we get all the values in one cycle before it loops, if it loops prematurely, it is not primitive.

This is finite field F2^3 so we have 2^3 = 8 elements, minus one zero element so 7 non-zero elements.

a^0 = 1

a^1 = a

a^2 = a^2

a^3 = a + 1

a^4 = a \* (a + 1) = a^2 + a

a^5 = a \* (a^2 + a) = a^3 + a^2 = a + 1 + a^2

a^6 = a \* (a + 1 + a^2) = a^2 + a + a^3 = a^2 + ~~a + a~~ + 1 = a^2 + 1

--- stop here because we have 7 unique elements and the next one loops back 1

a^7 = a \* (a^2 + 1) = a^3 + a = ~~a + a~~ + 1 = 1

**Primitive** leads to **=> irreducible** leads to **=> has no roots**

Not the other way around

**Boolean functions**

Define (n,m function):

A b function is a function that takes an input of n bits and gives a output of m bits. Such function can be represented as a mapping between the vector space Fn^2 to the vector space Fm^2.

Attacks:

* Differential cryptanalysis/attacks
  + Works when: there is a strong correlation between the difference of pairs of inputs to the function, and the difference of their corresponding outputs
  + **How to prevent:** having low differential uniformity. Almost perfect non-linear is when the differential uniformity is 2.
* Linear cryptoanalysis:
  + Works when: one the Boolean function’s components is close to a linear function. Of course, this raises the question of what it means for one function to “close” to another. The usual notion is that of the Hamming distance. The Hamming distance dH(f, g) between two functions f and g is the number of inputs x for which f(x) != g(x).
  + **How to prevent:** the nonlinearity should be as high as possible in order to resist linear cryptanalysis
* **How to prevent both:** “Almost bent” or AB functions offer the best possible resistance to both differential and linear cryptanalysis.

Example of Boolean functions used in block ciphers:

* S-boxes from AES