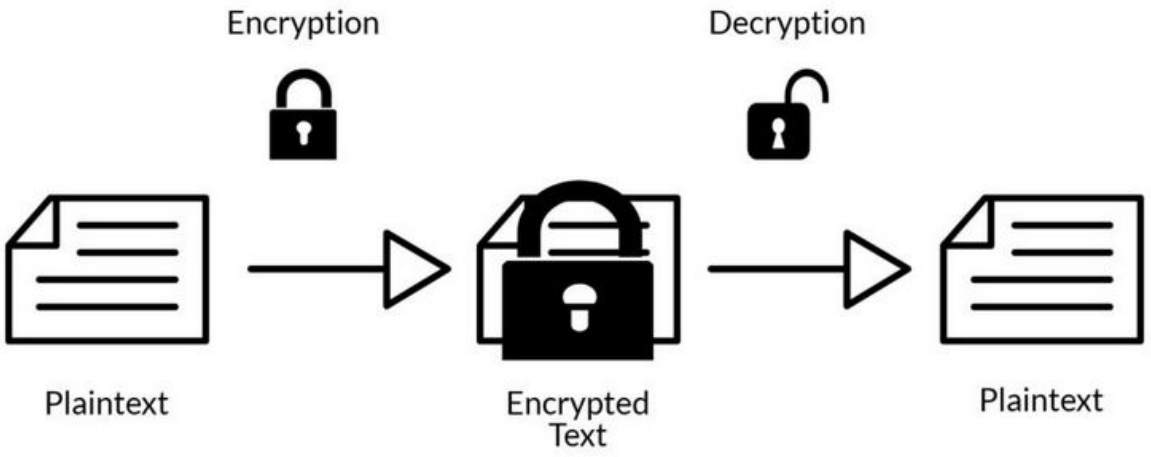
Encryption & Decryption Algorithms

CS 218 : Data Structures and Algorithms

IIT GOA



- By Manan Jain & Yuvraj Agrawal

Sophomores in Computer Science & Engineering

Under guidance of Prof. Milind Sohoni

INDEX

Abstract

Data is increasingly central to our personal lives, economic prosperity, and security. That data must be kept secure. Just as we lock our homes, restrict access to critical infrastructure, and protect our valuable business property in the physical world, we rely on encryption to keep cybercriminals from our data. Proposals to regulate this crucial form of protection — however well-intended — could weaken our security.

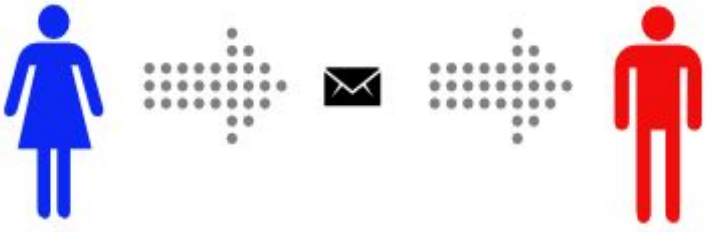
Digital security is becoming increasingly important to protect us as we bank, as we shop, and as we communicate. Our digital world is constantly under attack by cybercriminals and at the core of that security lies encryption. It is a secure envelope that keeps hackers from reading our personal communications.

In this project, we will begin with understanding the basics of encryption and decryption. Followed by learning RSA and AES algorithms. Then we will use these algorithms and model a real life situation of secure data transmission between army personnel.

INTRODUCTION

Encryption can be understood using the following example :

Alice wants to send a private message to Bob, and the only easy way they have to communicate is via postal mail.



Unfortunately, Alice is pretty sure that the postman is reading the mail she sends.

That makes Alice sad, so she decides to find a way to send messages to Bob without anyone else being able to read them.

So, she wants us to develop a way of encrypting the letter; of hiding its true meaning so the postman can't read it, but Bob can.

So before devising an algorithm to help Alice, let's look into the basics of Encryption and Decryption.

Encryption and Decryption

Encryption is a process that encodes a message or file so that it can only be read by certain people. Encryption uses an algorithm to scramble, or encrypt, data and then uses a key for the receiving party to unscramble, or decrypt, the information. The message contained in an encrypted message is referred to as plaintext. In its encrypted, unreadable form it is referred to as ciphertext.

Decryption is a way to change encrypted information back into plaintext. This is the decrypted form.

Key: Random string of bits created specifically for scrambling and unscrambling data. These are used to encrypt and/or decrypt data. Each key is unique and created via algorithm to make sure it is unpredictable. Longer keys are harder to crack. Common key lengths are 128 bits for symmetric key algorithms and 2048 bits for public-key algorithms.

The two main kinds of encryption are **symmetric encryption** and **asymmetric encryption**. Asymmetric encryption is also known as public key encryption.

Let’s continue the example of Alice and Bob to understand these kinds

Symmetric-Key Encryption

Alice decides to put the message inside a lockbox, then mail the box to Bob. She buys a lockbox and two identical keys to open it. But then she realizes she can’t send the key to open the box to Bob via mail, as the mailman might open that package and take a

copy of the key.
Instead, Alice arranges to meet Bob at a nearby bar to give him one of the keys. It's inconvenient, but she only has to do it once.

After Alice gets home she uses her key to lock her message into the lockbox.

Then she sends the lockbox to Bob. The mailman could look at the outside, or even throw the box away so Bob doesn't get the message – but there's no way he can read the message, as he has no way of opening the lockbox.

Bob can use his identical key to unlock the lockbox and read the message.

This works well, and now that Alice and Bob have identical keys Bob can use the same method to securely reply.
Meeting at a bar to exchange keys is inconvenient, though. It gets even more inconvenient when Alice and Bob are on opposite sides of an ocean.

Asymmetric/Public-Key Encryption

This time, Alice and Bob don't ever need to meet. First Bob buys a padlock and matching key.

Then Bob mails the (unlocked) padlock to Alice, keeping the key safe.

Alice buys a simple lockbox that closes with a padlock, and puts her message in it.

Then she locks it with Bob's padlock, and mails it to Bob.

She knows that the mailman can't read the message, as he has no way of opening the padlock. When Bob receives the lockbox he can open it with his key, and read the message.

This only works to send messages in one direction, but Alice could buy a blue padlock and key and mail the padlock to Bob so that he can reply.
Or, instead of sending a message in the padlock-secured lockbox, Alice could send Bob one of a pair of identical keys.

Then Alice and Bob can send messages back and forth in their symmetric-key lockbox, as they did in the first example.

This is how real world public-key encryption is often done.

- Bob generates a key pair, consisting of his public key (red padlock) and private key (red key).
- Bob then publishes his public key, and Alice fetches it (Bob mails his padlock to Alice).
- Alice then generates a temporary symmetric key (the pair of orange keys) and uses Bob's public key (red padlock) to securely send it to Bob.
- Bob then uses his private key (red key) to unlock his copy of the symmetric key (orange key).
- Bob and Alice can then use those symmetric keys to securely send messages back and forth.

This solves Alice's problem. Now let's learn about some famous and secure encryption and decryption algorithms.

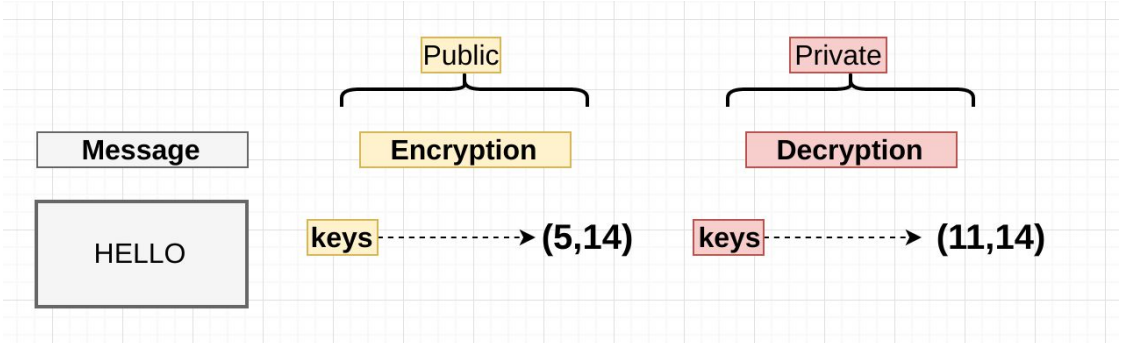
RSA (Rivest–Shamir–Adleman) Algorithm:

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm, which means that a key pair will be generated , a **public** key and a **private** key , you keep your private key secure and pass around the public one.

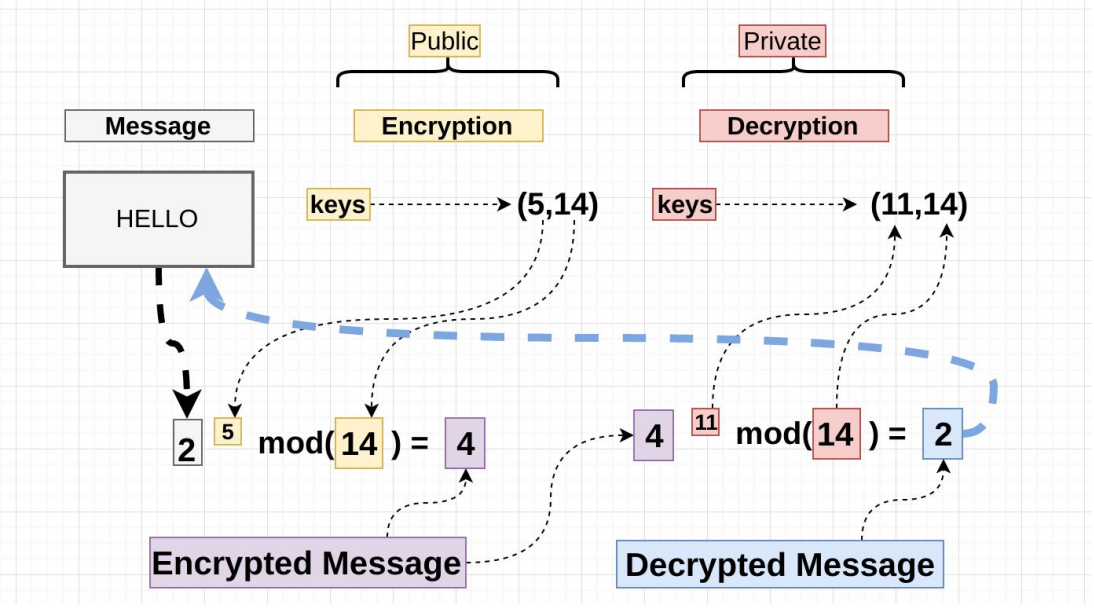
The algorithm is based on the fact that finding the factors of a large composite number is difficult: when the factors are prime numbers, the problem is called prime factorization. It is also a key pair (public and private key) generator.

Example: Working of RSA

We've got a message ("HELLO") , and we've picked two tuples with two numbers each.



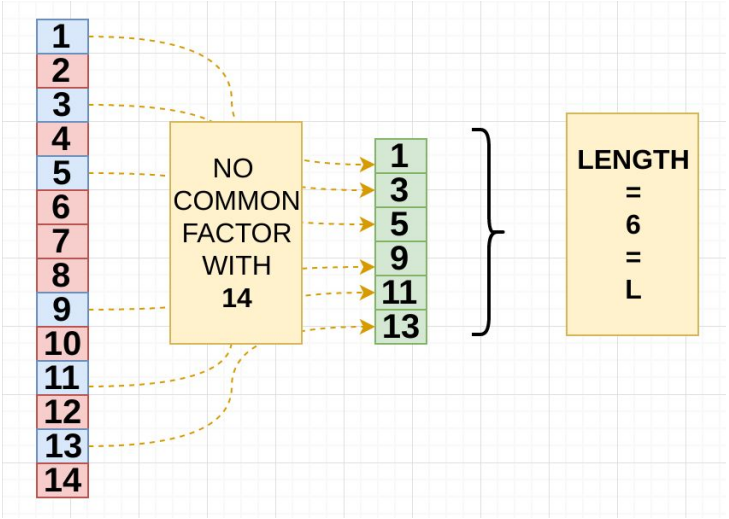
Obviously there's no arithmetic operation we can perform with strings , so the message has to be converted it to something , so let's say "HELLO" converts using some conversion algorithm to "2"



The interesting bit is how we come about those numbers , and how (5,14) is related to (11,14)

The details of the Decryption/Encryption pair:

1. Pick two prime numbers , we will pick 2 and 7 , let's call them p and q.
2. Multiply P and Q , and that becomes the modulus. i.e. N = p*q = 2*7 = 14.
3. Make a list between 1 and 14 and remove the factors which are common with 14:



Now there's an easy way to get this and that is:

(Q-1) * (P-1) = L

=> (7-1)*(2-1) = 6 = L

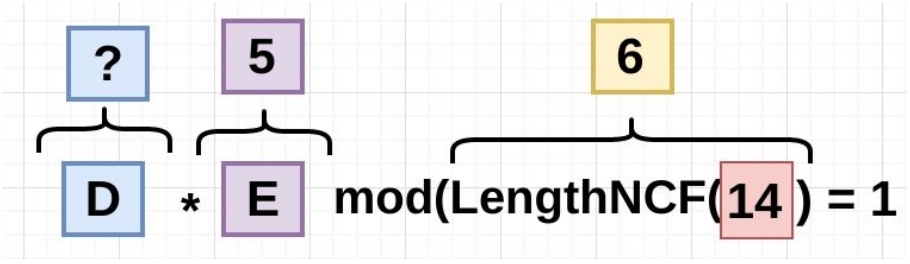
4. Now we get to pick the encryption key , in the example was (5,14) , we know 14

is the modulus.

So for the encryption key there's a few rules:

- it's got to be between 1 and L
  [2,3,4,5]
- Coprime with L (6) and the Modulus (14) , the answer is 5 , there's no other possibility .
- So there we came to a conclusion of why we picked (5,14)

5. The Decryption part , In the example we've picked (11,14) , again 14 is the modulus but where does 11 come from? , from now on let's call it D , let's find out why D is 11:



So the Decryptor(11) multiplied by the Encryptor(5) modulus the length of the non-common factor with the modulus(14) has to be equal to 1.

6. So we know if we multiply D * E and E is 5 , D will need to be a common factor of 5. We've made a list of numbers from 1 to 50 and filtered the ones that when multiplied by E and modulo by LengthofNonCommonFactors (N)/ LNCF(N) are equals 1.
   [5, 11, 17, 23, 29, 35, 41, 47]
7. So let's see if these can decrypt the message. We require to prove following function:

   isTrue(4 ** D % 14 == 2) <---Decrypted message
8. And, all the above values of D yield true.
   Applying the function to all the Decryption keys that follow the rule (D * E % LNCF(N) = 1) , decrypted the message successfully .

This Algorithm inputs a message, encrypts it and decrypts it successfully.

AES (Advanced Encryption Standards) Algorithm:

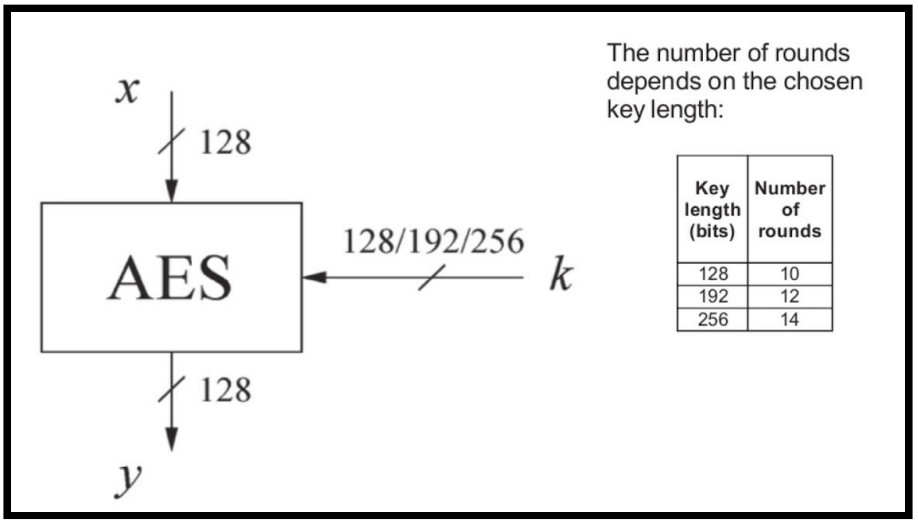
AES (also known by its original name **Rijndael)** comprises a series of linked

operations, some of which involve replacing inputs by specific outputs and others involve shuffling bits around.

AES includes three block ciphers: AES-128, AES-192 and AES-256.

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix –

There are 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. A round consists of several processing steps that include substitution, transposition and mixing of the input plaintext to transform it into the final output of ciphertext.



How does AES work?

The AES encryption algorithm defines numerous transformations that are to be performed on data stored in an array. The first step of the cipher is to put the data into an array -- after which, the cipher transformations are repeated over multiple encryption rounds.

The first transformation in the AES encryption cipher is **substitution** of data using a substitution table; the second transformation **shifts data rows**, and the third **mixes columns**. The last transformation is performed on each column using a different part of the **encryption key**.

Understanding AES in detail:

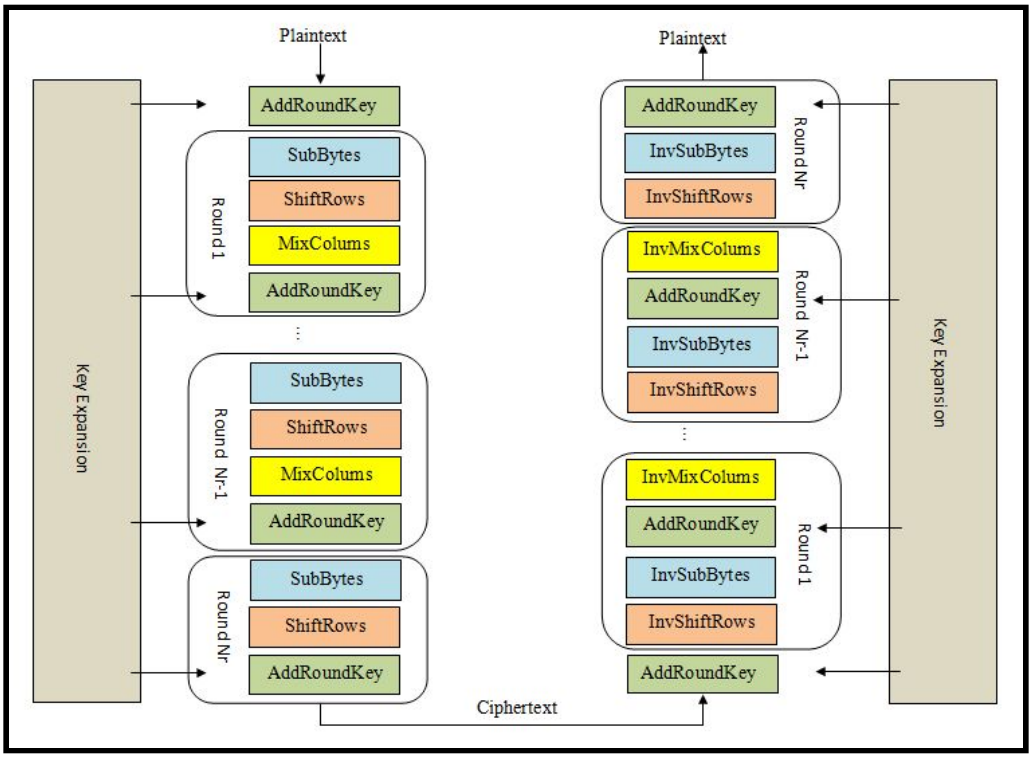
First, the data is divided into blocks.
The block size of AES is 128-bits, so it separates the data(plaintext - which you wish to encrypt) into a four-by-four column of sixteen bytes (there are eight bits in a byte and

16 x 8 = 128). For ex: data = “buy me some potatoes please”

b	m	o	p
u	e	m	o
y		e	t
	s		a

Consider this the first block of the plaintext. The message after this will be encoded in the next block. This block undergoes some rounds of cipher transformations to convert into cipher text/block.

Cipher Transformations:



1. Add round key

In this step, because it is the first round, our initial key is added to the block of our message:

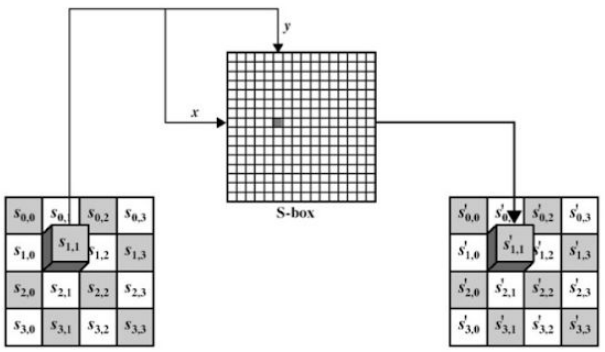
b	m	o	p
u	e	m	o
y		e	t
	s		a

k			i
e	a	b	n
y	r	o	g
s	e	r	1

This is done with an XOR cipher, After converting each block to its binary representation and applying XOR, Let's say that this mathematical operation gives us a result:

h3	jd	zu	7s
s8	7d	26	2n
dj	4b	9d	9c
74	el	2h	hg

2. Substitute bytes

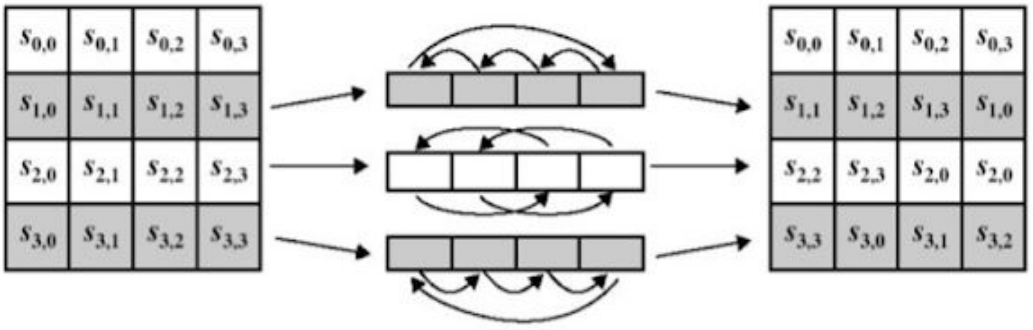


The Rijndael S-box is a substitution box (lookup table) used in the Rijndael cipher, which the Advanced Encryption Standard (AES) cryptographic algorithm is based on. There are two boxes, one forward (for encryption) and one inverse (for decryption).

In this step, each byte is substituted according to a predetermined table. This system is a bit more complicated and doesn't necessarily have any logic to it. Instead, there is an established table that can be looked up by the algorithm, which says, for example, that h3 becomes jb, s8 becomes 9f, dj becomes 62 and so on. After this step, let's say that the predetermined table gives us:

jb	**n3**	*kf*	n2
9f	**jj**	*1h*	js
74	**wh**	*0d*	18
hs	**17**	*d6*	px

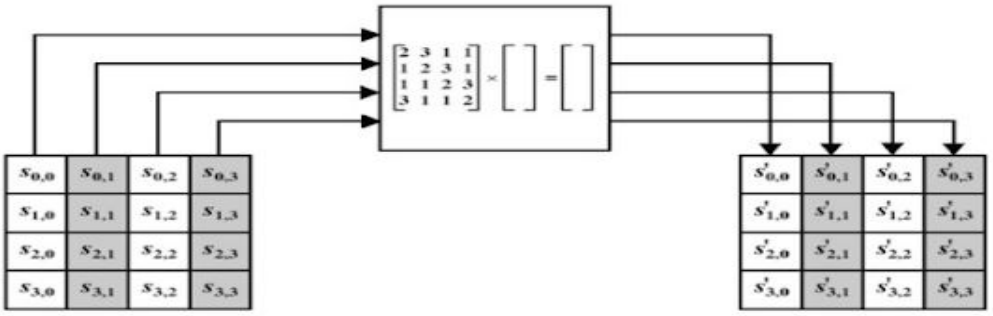
3. Shift rows



Shift rows is a straightforward name, and this step is essentially what you would expect. The second row is moved one space to the left, the third row is moved two spaces to the left, and the fourth row is moved three spaces to the left. This gives us:

jb	**n3**	*kf*	n2
jj	*1h*	js	9f
0d	18	74	**wh**
px	hs	**17**	*d6*

4. Mix columns



Each byte is mapped to a new value which depends on the value of all four of the bytes in the column. This is a matrix transformation, the coefficients of the matrix chosen to mix bytes the most. The following matrix is used :

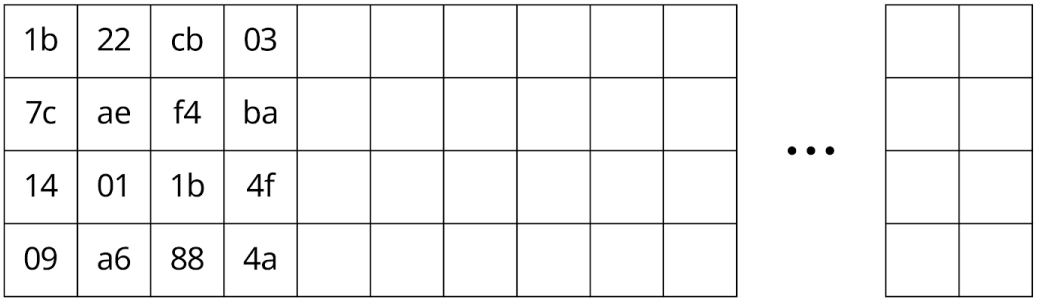
$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

Each column is operated on individually. Each byte of a column is mapped into a new value that is a function of all four bytes in the column. Let's say that the operation gives us this result:

ls	j4	2n	ma
83	28	ke	9f
9w	xm	3l	m4
5b	a9	cj	ps

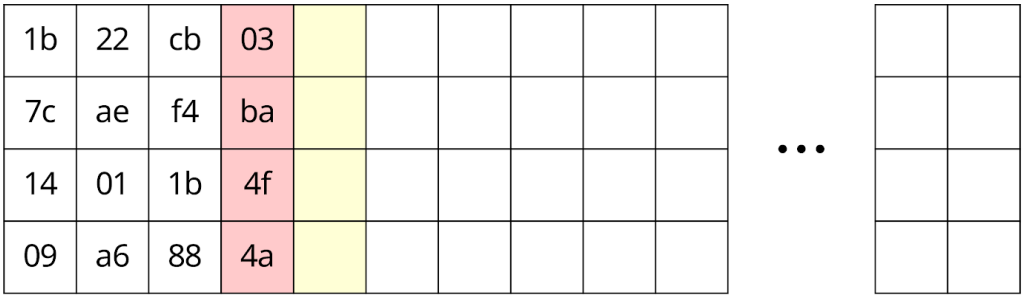
Key Expansion

Now we will use key expansion to generate new round keys from our original keys which will be used in the further rounds.



round generation - 1

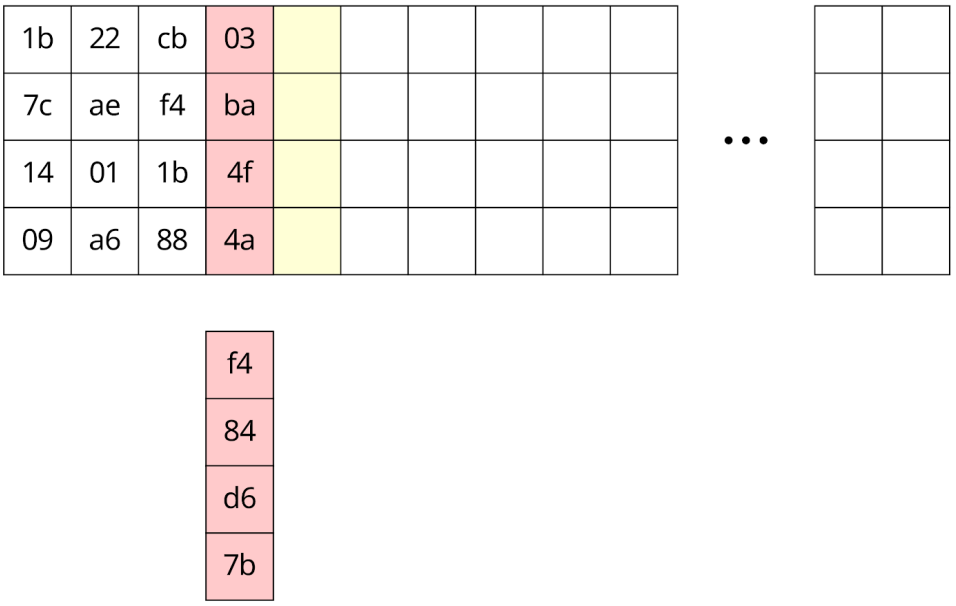
1 - As explained earlier, we have a private key represented as a two-dimensional array, and each block has 1 byte.



ba
4f
4a
03

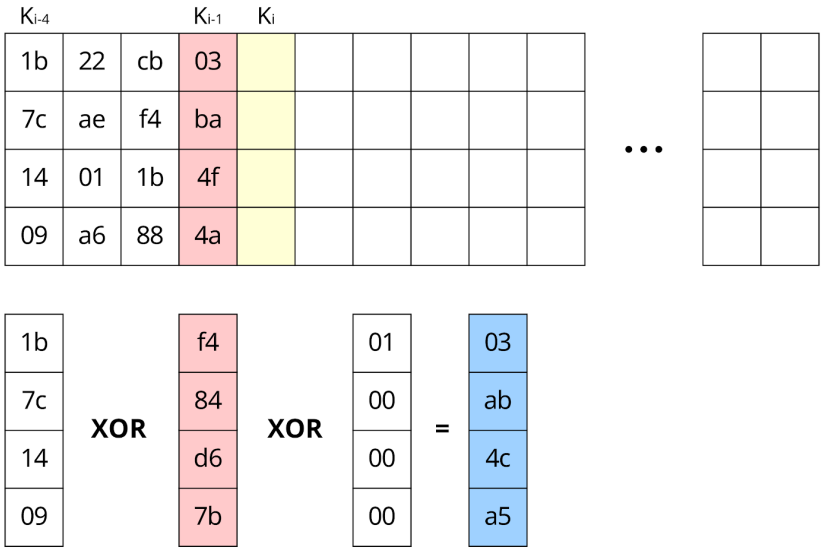
round generation - 2

2 - First take the right-most column, and execute circular upward shift



round generation — 3

3. In the same way as we did before in substitute bytes step, substitute bytes using S-BOX



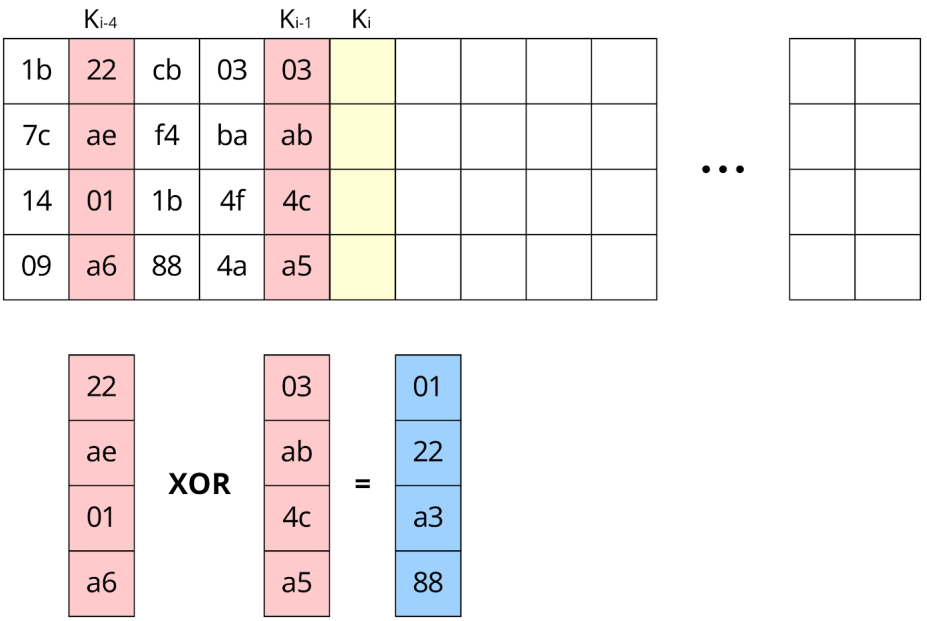
round generation - 4

4 Then do XOR operation with K_(i-4) columns and take the predefined value from rcon table, and do XOR operation again. The result is our first column of the current round.

Values of rc_i in hexadecimal

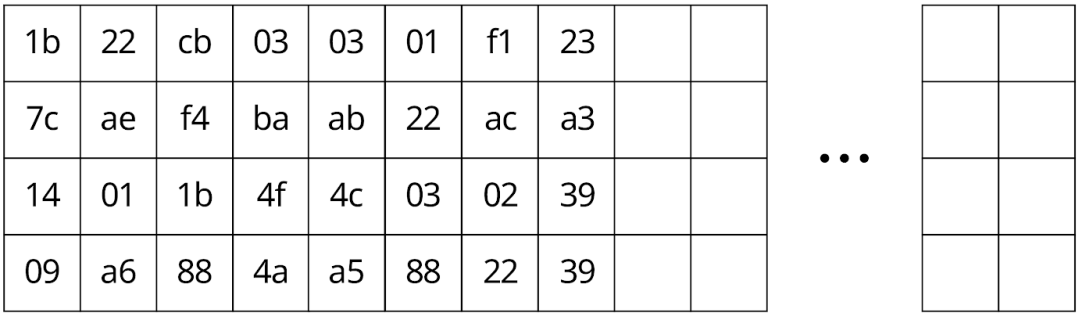
i	1	2	3	4	5	6	7	8	9	10
rc_i	01	02	04	08	10	20	40	80	1B	36

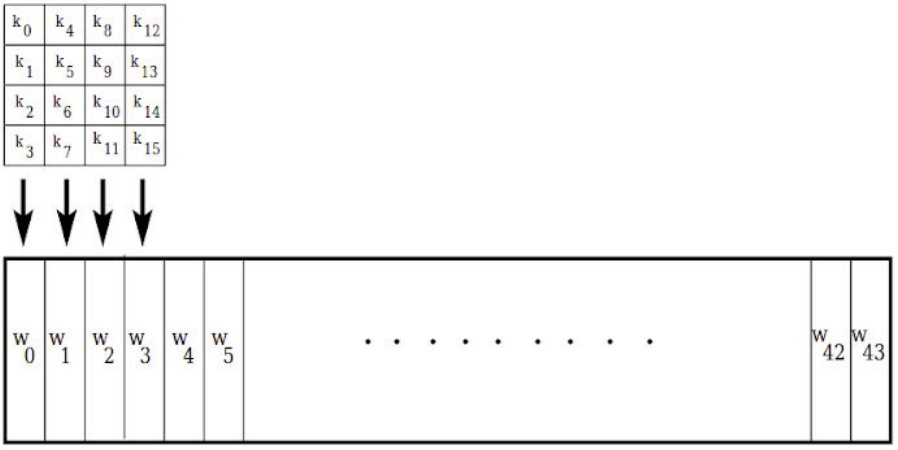
The round constant rcon_i for round i of the key expansion is the 32-bit word



round generation - 5

5 - Generating the 2nd, 3rd and last column of the round is rather simple, just do XOR operation on K_(i-1) and K_(i-4) columns.





The AES key expansion algorithm takes as input a 4-word key and produces a linear array of 44 words. Each round uses 4 of these words since each key consists of 4 words.

Add round key (again)

We take the result of our mixed columns and add the first round key that we derived:

ls	j4	2n	ma
83	28	ke	9f
9w	xm	3l	m4
5b	a9	cj	ps

+

14	29	1h	s5
h9	9f	st	9f
gt	2h	hq	73
ks	dj	df	hb

Let's say that this operation gives us the following result:

9d	5b	28	sf
ls	df	hf	3b
9t	28	hp	8f
62	7d	15	ah

Many more rounds...

After the last round key is added, it goes back to the byte substitution stage, where

each value is changed according to a predetermined S box table. Once that's done, it's back to shifting rows and moving each row to the left by one, two or three spaces. Then it goes through the mix columns equation again. After that, another round key is added.

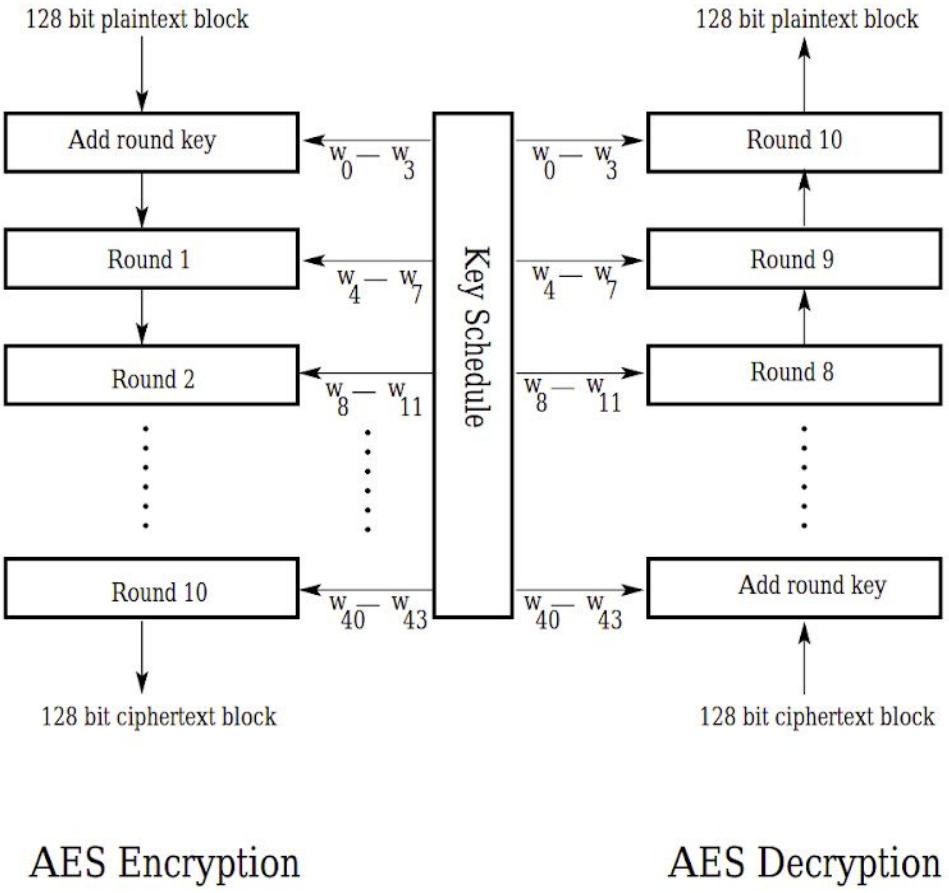
As we know, **AES has key sizes of either 128, 192 or 256-bits.** When a 128-bit key is used, there are nine of these rounds. When a 192-bit key is used, there are 11. When a 256-bit key is used, there are 13.

After these nine, 11 or 13 rounds, there is one additional round in which the data is only processed by the byte substitution, shift rows and add round key steps, but *not* the mix columns step. The mix columns step is taken out because at this stage, it would just be eating up processing power without altering the data, which would make the encryption method less efficient.

Once the data has gone through this complex process, your original **"buy me some potato chips please"** comes out looking something like **"ok23b8a0i3j293uivnfqf98vs87a"**. It seems like a completely random string of characters, but as you can see from these examples, it is actually the result of many mathematical operations being applied to it again and again.

Decryption will use the above transformations in an inverse order.

To make things clearer, the entire AES encryption process goes:



Problem Statement:

We are trying to model a typical situation in PoK.

The Army outposts regularly want to communicate and exchange sensitive information, but the data is at risk since it can be intercepted by the adversary.

Data breach is common and while delivering the data, the adversary might steal our data, interpret it and may use it against us. So we plan to generate a way to secure the communication among the army outposts.

Our solution:

We propose a way through which the army personnel first uses asymmetric encryption (RSA) to request and receive a key for further secure communication. And then the army personnel can record an audio message, convert it to hexadecimal and encrypt this data, use symmetric encryption (AES) and deliver it to his allies.

In layman terms, we, using the above method will try to develop a secure and fast

way of communication among the army outposts which can't be intercepted by the adversary.

Understanding the solution in detail:

- The sensitive information at the border needs to get delivered as quickly as possible to the concerned authorities, which will give them time to plan, prepare and act according to the situation.
- So, we are working on audio transmission. As the army javaan could record audio messages at ease and deliver them quickly.
- After recording the audio, the software will convert the audio into hexadecimal format.
- Then the software will contact the concerned department for setting up encryption and will use the RSA algorithm to interchange private AES key for further communication.
- Once, the AES key gets delivered, secure communication can take place. Now the hexadecimal encrypted file gets transferred from one end to another.
- Decryption and reconversion of the file then takes place at the other end.
- Now, the audio message is ready to be heard on the other side.

Algorithm:

The code is implemented in python. Follow the instructions.txt file present in the attached folder to run the program on your pc. To implement AES we have used the package pycryptodome. To install it, the following command can be used :

```
pip install pycryptodome
```

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
import binascii
import random
import math
import sys
import random
import string

def randomString(stringLength):
    letters = string.ascii_letters
    return ''.join(random.choice(letters) for i in
range(stringLength))
```

```
13.
14. def is_prime(x):
15.     if x < 2:
16.         return False
17.     elif x == 2:
18.         return True
19.     for n in range(2, int(math.sqrt(x))):
20.         if x % n ==0:
21.             return False
22.     return True
23.
24. def randprime():
25.  prime1= [i for i in range(500,1000) if is_prime(i)]
26.  n = random.choice(prime1)
27.  return n
28.
29. def gcd(a,b):
30.     if b==0:
31.         return a
32.     else:
33.         return gcd(b,a%b)
34.
35. def encrypt(plain_text, iv, key):
36.     obj = AES.new(key, AES.MODE_CFB, iv)
37.
38.     enc_text = obj.encrypt(plain_text)
39.     return enc_text
40.
41. def decrypt(enc_text, iv, key):
42.     obj = AES.new(key, AES.MODE_CFB, iv)
43.
44.     plain_text = obj.decrypt(enc_text)
45.     return plain_text
46.
47.
48. print('Hello there, Please wait while we set up encryption for
    you', '\n')
49. print('SENDER SIDE: ', '\n')
```

```
print('Beginning to set up RSA')

print('Generating two primes p & q for RSA encryption')

p = randprime()
q = randprime()
while(p == q):
    q = randprime()

print('Generated primes are: p = ', p, ' and q = ', q,)

#RSA code for generation of e & d (encryption & decryption pair)
n = p*q
l = (p-1) * (q-1)

for e in range(2,l):
    if gcd(e,l)== 1:
        break

for i in range(1,100):
    x = 1 + i*l
    if x % e == 0:
        d = int(x/e)
        break

print('Generated the encryptor e:', e, ' and the decryptor d:',
d, '\n')
print('Sending the encryptor key to the receiver and requesting
for AES key for further communications', '\n')

print('RECEIVER SIDE:')
print('Received public encryption key: ', e, 'and modulo number
n: ', n)

#Empty strings for storing 128-bit binary AES keys
AESkey=randomString(16)          #original AES key
encrypt_RSA=[]                   #Empty list to store ascii
values of encrypted AES string
```

```
decrypt_RSA=[]                          #Empty list to store ascii values of decrypted AES string
AES_key = [ord(c) for c in AESkey]  #Converting the string to a list of characters
#RSA function for Encryption
for i in range(16):
    encrypt_RSA.append(1)
    for j in range(e):
        encrypt_RSA[i] = encrypt_RSA[i]*AES_key[i]
        encrypt_RSA[i]  = encrypt_RSA[i]%n      #(key^e)mod_n

print('Delivering the Encrypted AES key to sender', '\n')

print('SENDER SIDE: ', '\n')
print('Received encrypted AES key')

#RSA function for Decryption
for i in range(16):
    decrypt_RSA.append(1)
    for j in range(d):
        decrypt_RSA[i] = decrypt_RSA[i]*encrypt_RSA[i]
        decrypt_RSA[i]  = decrypt_RSA[i]%n
#(encrypted_key^d)mod_n

AESkey_decrypted=''.join(chr(i) for i in decrypt_RSA)
#Reconverting the Ascii value list back to original string

print('Ready to begin secure transmission, AES setup completed')
print('AES key is: ', AESkey_decrypted)

iv = "TestMeInitVector"

# print('Converting your audio to hex')
filename = './Input/input.mp4'
with open(filename, 'rb') as f:
    content = f.read()
hexa = binascii.hexlify(content)
```

```
file=open("./Output/converted.txt", "wb")
file.write(hexa)
file.close()

print("Encrypting your audio using AES")

file1=open("./Output/encrypted.txt", "wb")
encdata = encrypt(hexa, iv, AESkey_decrypted)
file1.write(encdata)

print("Data received on other side, beginning decryption")
file2=open("./Output/decrypted.txt", "wb")
file2.write(decrypt(encdata, iv,AESkey_decrypted))

print("Converting .txt back to audio file")
filename2 = './Output/decrypted.txt'
with open(filename2, 'rb') as f:
    content = f.read()
file=open("./Output/decrypted.mp4", "wb")
file.write(binascii.unhexlify(content))
file.close()
```

Inputs and outputs

INPUTS:

1. Recorded audio message (Sender side)

OUTPUTS:

1. Encrypted hexadecimal message
2. Decrypted hexadecimal message
3. Audio message.

Algorithm Analysis

Describing the functions used:

- def randomString(stringLength) - generates a random string of letters in O(stringLength) time.
- def is_prime(x) - checks if a number is prime, with complexity O($\sqrt{n}$)
- def randprime() - function used to generate random prime number(for p & q of RSA) in O(is_prime())
- def gcd(a,b) - function returns the GCD of 2 inputs using Euclid's division lemma with the time complexity O(log(min(a,b))
- def encrypt(plain_text, iv, key) - function used to encrypt plain_text using AES algorithm in O(number of blocks used for plain_text) = O(size(plain_text)/128)
- def decrypt(enc_text, iv, key) - function used to decrypt enc_text using AES algorithm in O(number of blocks used for enc_text) = O(size(enc_text)/128)

Complexity analysis of complete algorithm:

Total complexity = complexity of finding random p & q + complexity of finding e (encryptor) + complexity of finding d (decryptor) + complexity of generating random AES key (16 bytes) + complexity of converting AES string to list of their ASCII values + complexity of RSA encryption + complexity of RSA decryption + complexity of reconverting decrypted AES key to character string + complexity of converting input to hexadecimal + complexity of encrypting data using AES + complexity of decrypting data using AES + complexity of converting hexadecimal to binary + C

Neglecting all the smaller terms, time complexity is mainly decided by,

Complexity of RSA decryption since the value of d is generally of the order 10^4 compared to e which is of the order 10. And complexity of 2* AES encryption (Since complexity of AES encryption and decryption will be the same and depend on the size of the Input audio file.)

The final complexity will depend on the size of input for AES.

If size of input for AES is large:

Complexity = O(number of blocks for input data) = O(input size/128)

Else:

Complexity costs us O(log(d)^3)

Where d is the RSA decryptor.

Scope of the algorithm and future work :

- This algorithm can be modelled for any type of file. Here we are just using audio.
- To improve the security, rather than using 128 bit we'll use 192 or 256 bit for additional security.

Limitations :

- RSA consumes a lot of time.
- RSA is only secure for very large prime numbers
- Due to advanced computing and invention of quantum computers , RSA may get compromised in the near future.

REFERENCES

1. How does RSA work? - HackerNoon.com
2. Cryptography with Alice and Bob
3. What is AES encryption (with examples) and how does it work?
4. Algorithms book by Dasgupta, Papadimitriou and Vazirani