FALL SEMESTER 2020 - '21

MAIL ENCRYPTOR

A PROJECT REPORT

Submitted by
Anubhav Singh Guleria(18BCE0186)
Nitin Pramod Ranjan(18BCE0272)
Somesh Mishra(18BCE2525)

Course Code: CSE 3501
Course Title: Information Security Analysis and Audit

Under the guidance of

Dr.Chandramohan B
VIT, Vellore

INDEX

Acknowledgement

We are thankful to VIT, Vellore for providing us with the necessary guidelines and environment for building this project. We also express our gratitude to Dr Chandramohan B, our professor for the CSE3501 course for his guidance and expertise. We also express our sincere appreciation for the team and its members for working and completing this project.

Anubhav Singh Guleria
Nitin Pramod Ranjan
Somesh Mishra

1.ABSTRACT

Email encryption has for long being an active field of research ever since the advent of the field. Companies and institutions providing email services work on end-encryption based on user authentication alone. Other than that, email services have relied on a zero-encryption transmission. This is because most encryption mechanisms including depreciated encrypted techniques and modern techniques like AES and SHA take a certain amount of time to be executed upon the text. This creates inconvenience for the users. However, research suggests that in the era of increasing digital communication, encryption becomes a primary need for every user. For this, most organisations turn to custom security protocols or tools that require additional payment to the email service provider. So, the authors have tried to provide a minimal cost tool that allows three layers of authentication and an additional encryption with little time consumption for securing email transactions.

2. INTRODUCTION

ANS Mail Encryptor is a custom text encryptor built on Python-Django framework and uses a simple customised smtp service to allow any user logged in to its system and also to this program to send and receive emails directly through gmail service. The encrypted message itself is encrypted into a string. The software has been built atop Python 3.8.6 and uses Django 3.1.2 framework.

The admin of the server can look at all transactions, public keys and the keys involved in encrypting messages and user profiles.

The only limitation of this project at this moment is that the creators assume that there is some private mechanism by which the users communicating are sharing the decryption key.

3. TASK DISTRIBUTION

Name of Task	**Description**	**Executed by**
Literature Study	A set of research papers, articles and proposals were studied. Most of the material was collected from IEEE or scopus-indexed journals. The necessity of the project optimum encryption tools were studied.	Somesh Mishra
Browser compatibility	Different browsers shall support hashing, mathematical operations and python frameworks at different levels.Seven browsers - Opera, Firefox, Firefox Developer Edition, Chrome, Chromium, Edge and Internet Explorer - were analysed and the results were compiled.	Somesh Mishra
AES encryption	AES encryption was studied, customised to the needs of the UI and programmed in javascript. This is used to encrypt the message to be sent in the email.	Anubhav Singh Guleria
SHA-256 encryption	SHA-256 encryption was applied upon password to generate private keys. A javascript code was written for the purpose.	Nitin Ranjan
DH encryption	The encrypted message is encrypted in a small string that carries the name of the sender at its end. For this purpose, DH algorithm is used. A JS code was written for this purpose	Nitin Ranjan
Elliptical Curve Key Generator	Elliptical Curve Cryptography is being used to generate public key pairs ***for each message*** for the two users involved in the email transaction. A JS code was written for this purpose.	Anubhav Singh Guleria
Public key generation	Public Keys for every user is being generated using Elliptical Curve Cryptography. This key is a statutory entity of recognition for all users on the server and is unique to each user.	Anubhav Singh Guleria
Server Establishment	A Django server was established	Somesh Mishra

	that can incorporate all the HTML and JS tools mentioned so far. It also allows for the creation of a database and a superuser for analysis and audit of the server.	
Connecting SMTP with this server	SMTP is a custom open-source library that can be integrated with JS and front-end tools. However, this project required us to integrate it with every user on custom demand. Under normal conditions, the SMTP user and password are mentioned in the source script itself.	Somesh Mishra
Custom User Key	Every user has a custom key that is constant and appended to the tail of the string generated by the DH script.	Somesh Mishra
Decryption DH	DH script is decrypted to generate an encrypted string that requires AES decryption.	Nitin Ranajan
Decryption SHA-256	The keys and passwords are decrypted using the SHA-256 decryptor module. This has nothing to do with the message itself, but will grant the user the right to access the message.	Nitin Ranjan
Decryption AES	The message is decrypted completely and the receiver will be able to view it.	Anubhav Singh Guleria
Audit mechanism	The admin will be able to audit all users, messages and public-private keys on the network.	Anubhav Singh Guleria
Analysis mechanism	The admin will be able to assess the messages, keys involved, if encryption-decryption is functional and if some network error occurs.	Nitin Ranjan

4. LITERATURE SURVEY

[1]This paper emphasised on the existing problems emails generally face .This paper further implied the technological advancement’s effects on emails and how they are getting vulnerable by the minute. The paper also elaborated the ease of access to the networks has made an exposed leakage for some irresponsible parties who have the competencies to steal the information while the delivery streams take place. The email user is advised to add another security act such as encrypting toward the email contents before it being sent using ESP service. In this paper AES and 3DES cryptography method successfully implemented to secure email text messages. Email message being encrypted first using both algorithms, and then being evaluated and analyzed from various aspects. Evaluation's results shows that AES is better in terms of compile time, while 3DES is better in terms of increasing message's size after the encryption process, but the change in the addition of bytes is not significant. So based on the results of tests, email users are recommended to use AES encryption.

[2] This paper was based on applying Advanced Encryption Standard (AES) algorithm to text based data mainly .txt files. This paper further explained the sublime simplicity and effectiveness of the algorithm on server based encryption tools and further implied the difficulty of hackers to get the real data when encrypting by AES algorithm. Till date there is no evidence of cracking this algorithm. AES has the ability to deal with three different key sizes such as AES 128, 192 and 256 bit and each of these ciphers has 128 bit block size. This paper will provide an overview of the AES algorithm and explain several crucial features of this algorithm in detail and demonstrate some previous research that has been done on it compared to other algorithms such as DES, 3DES, Blowfish etc. This paper also demonstrates a pseudo code for the implementation of the text based encryption which we incorporated in our project.

[3] The authors have discussed the Advanced Encryption Standard (AES) algorithm, one of the most common and widely symmetric block cipher algorithms currently in use. The algorithm employs a unique salt and structure method that makes it one of those algorithms which have no generic means for cracking as of now. In this paper, the authors have provided an overview of AES algorithm and explain several crucial features of this algorithm while also demonstrating some previous researches that have been done on it with comparing to other algorithms such as DES, 3DES, Blowfish etc.

[4] The authors analysed various encryption methodologies and tested it on a standard email message while simulating a virus attack. The authors thereby conclude that of all the detection techniques currently in use, Generic signature Scanning is computationally cheapest while also offering a high degree of efficiency. For analysis, they have suggested that heuristic-based techniques are highly accurate. However, they have also suggested that owing to the current computational power available with most hackers and system engineers, heuristic based techniques need to be improved in terms of speed.

[5] Using Python as the programming language, the authors have used both ECC and RSA and compared them over a set experimental data. They have further discussed ECC built upon a ElGamal algorithm and this was done over a server-client model. They have concluded that modular mathematics based ECC is highly efficient and faster in key generation. They further theorized on the basis of the end-to-end server encoding using the particular algorithm which we tried and actually incorporated into our key generation, the encryption keys were based of this research paper.

[6] The authors have used AES-256 and SHA-256 on a cloud server to test the efficiency and compatibility of the same. They have tried to implement integrity and verification of the data stored on the cloud by proposing a new methodology that works towards cloud servers. The method is fairly simple and involves an upload followed by key generation followed by system encryption of the files after a hash key is generated. This encrypted file is then forwarded to a central database. After conducting a reliability control test, the authors concluded that there is little or no bug involved in the project and a strong data integrity is maintained. All these tests were done using Laravel framework.

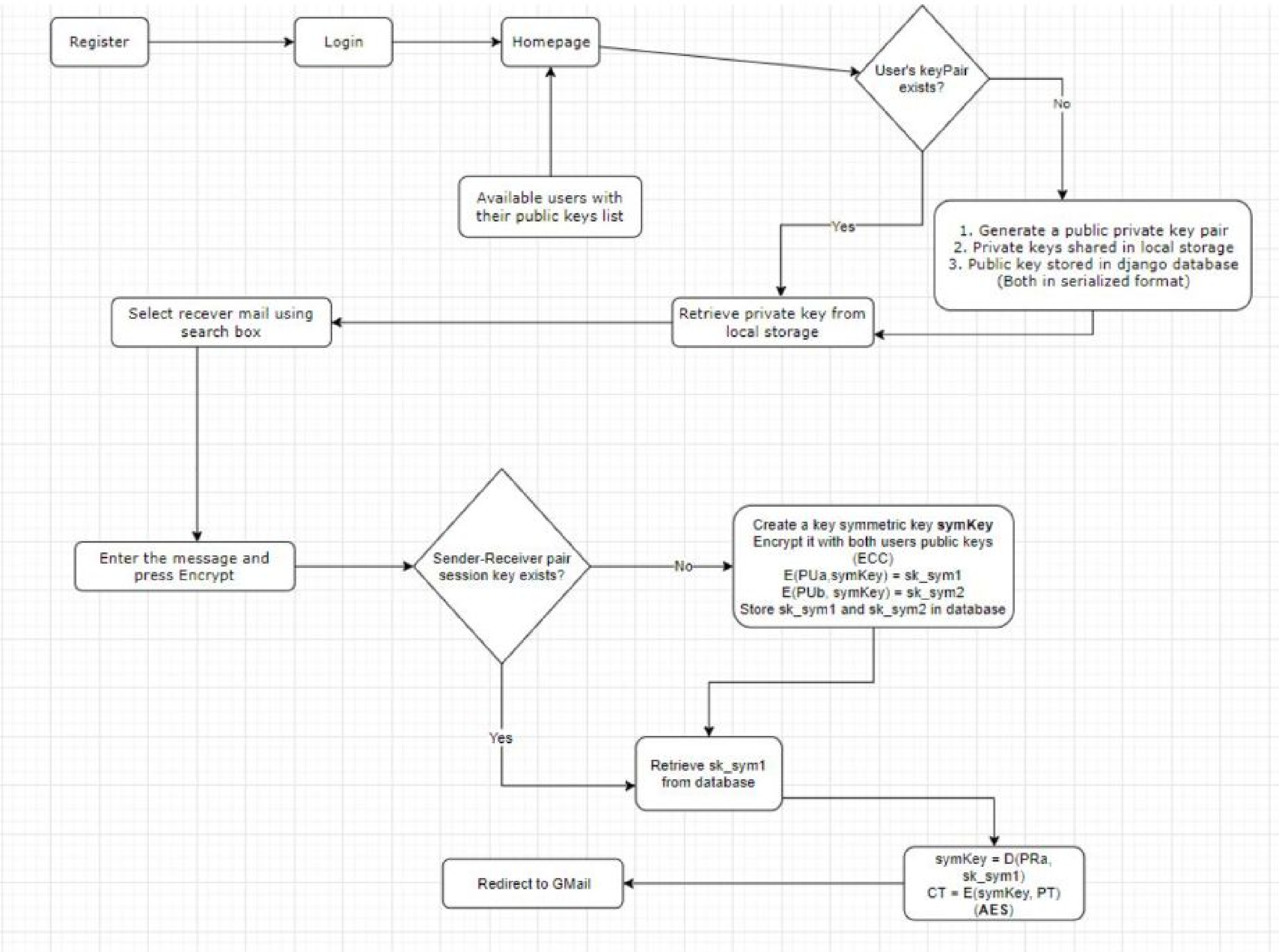
[7] The authors have discussed DH algorithms to create a key exchange protocol on a multi-peer server. The authors have pointed out that DH algorithm is in fact limited in its scope because it is very computation intensive. A modular arithmetic equation based DH algorithm has been proposed with a memory unit to store session keys. The project has very high efficiency and has reduced space and time complexity, while also being computationally cheaper in all respects.

[8] The authors have discussed Random Number Generators, Pseudo-Random number generators and compared them over CPU execution time and computational cost. They then propose the usage of a combined Key derivative

function using hashing and random and pseudo random key generators. This has been implemented using Blowfish block cipher and HMA cipher. Argon2id keying material. And finally, entropy is introduced in the hash function of the key generation. This results in the generation of a more secure key generation mechanism.

5. PROPOSED METHODOLOGY

Project Flow Chart



5.1 AES ALGORITHM

AES is based on a design principle known as a substitution–permutation network, and is fast in both software and hardware. Unlike its predecessor DES, AES does not use a Feistel network. AES is a variant of Rijndael which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, Rijndael *per se* is specified with block and key sizes that may be any multiple of 32 bits, with a minimum of 128 and a maximum of 256 bits.

AES operates on a 4 × 4 column-major order array of bytes, termed the *state*. Most AES calculations are done in a particular finite field.

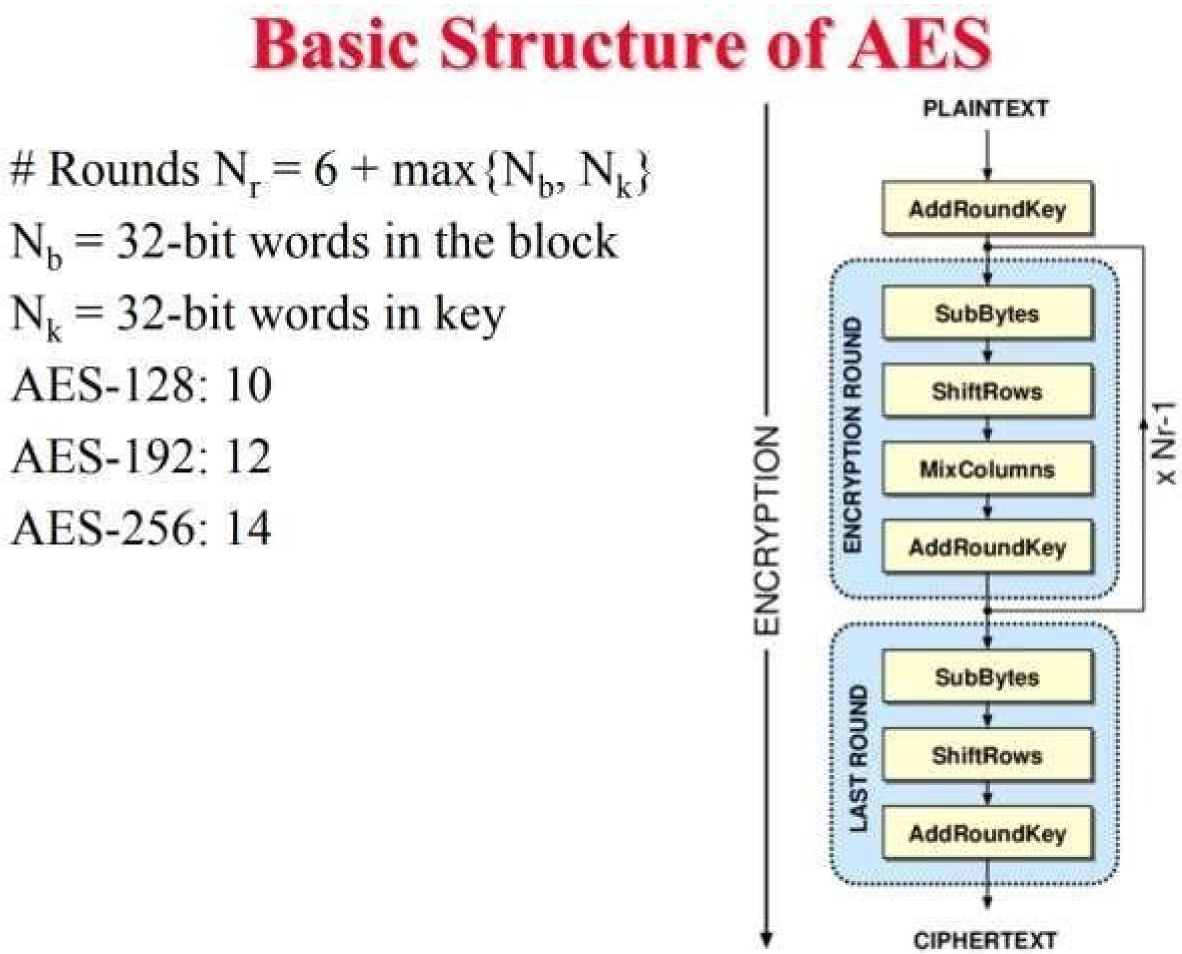


Figure 1.

For instance, if there are 16 bytes, b0 to b15 then they are represented in the form of a matrix like:

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

The key size used for an AES cipher specifies the number of transformation rounds that convert the input, called the plaintext, into the final output, called the ciphertext. The number of rounds are as follows:

10 rounds for 128-bit keys.

12 rounds for 192-bit keys.

14 rounds for 256-bit keys.

Each round consists of several processing steps, including one that depends on the encryption key itself. A set of reverse rounds are applied to transform ciphertext back into the original plaintext using the same encryption key.

1. KeyExpansion—round keys are derived from the cipher key using Rijndael's key schedule. AES requires a separate 128-bit round key block for each round plus one more.
2. Initial round key addition:
   - *AddRoundKey—each byte of the state is combined with a block of the round key using bitwise xor.*
3. 9, 11 or 13 rounds:
   - *SubBytes—a non-linear substitution step where each byte is replaced with another according to a lookup table.*
   - *ShiftRows—a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.*
   - *MixColumns—a linear mixing operation which operates on the columns of the state, combining the four bytes in each column.*
   - *AddRoundKey*
4. Final round (making 10, 12 or 14 rounds in total):
   - *SubBytes*
   - *ShiftRows*
   - *AddRoundKey*

S-BOX

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 2.

SUBSTITUTE BYTES

Each byte is replaced by byte indexed by row (left 4 Each byte is replaced by byte indexed by row (left 4-bits) & bits) & column (right 4 column (right 4-bits) of a 16x16 table bits) of a 16x16 table

EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

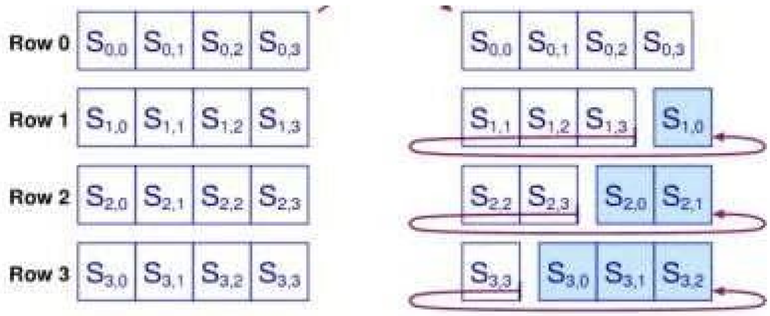
→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

Figure 3.

SHIFT ROWS

- the 1st row is unchanged.
- The 2nd row does 1 byte circular shift to the left row does 1 byte circular shift to left.
- 3rd row does 2 byte circular shift to left 3rd row does 2 byte circular shift to left.
- The 4th row does a 3 byte circular shift to the left 4th row does a 3 byte circular shift to left.



87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

Figure 5.

MIX COLUMNS

Effectively a matrix multiplication in GF(2 Effectively a matrix multiplication in GF(28) using) using prime polynomial prime polynomial

m(x) =x m(x) =x8+x4+x3+x+1

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

$$(\{02\} \bullet \{87\}) \oplus (\{03\} \bullet \{6E\}) \oplus \{46\} \oplus \{A6\} = \{47\}$$

Figure 6.

5.2 SHA 256

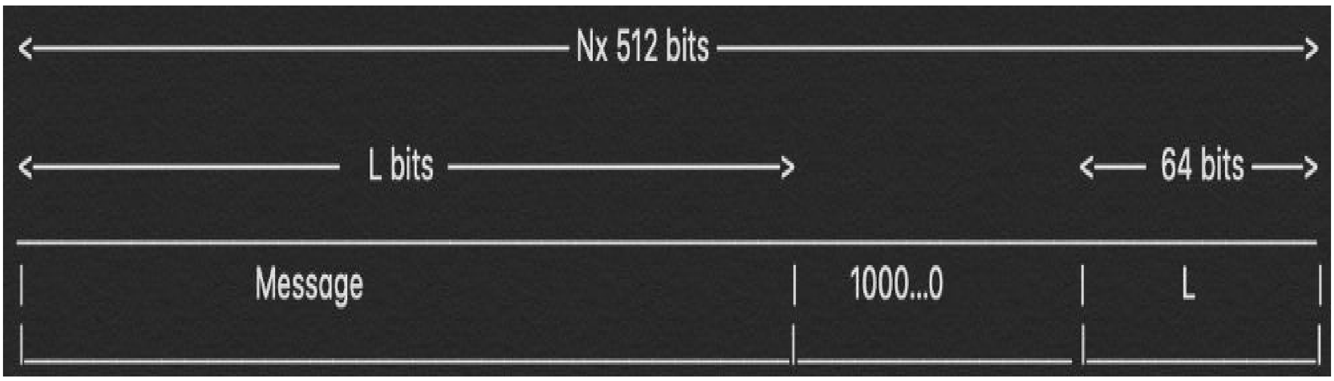
1. Append : Padding bits

First step of our hashing function begins with appending bits to our original message, so that its length will be same to the standard length required for the hash function. To do so we proceed by adding few bits to the message that we have in hand. The number of bits we add is calculated as such so that after addition of these bits the length of the message should be exactly 64 bits less than a multiple of 512. Let me depict it to you in mathematical terms for better understanding.

```
M + P + 64 = n x 512

i.e M = length of original message
    P = padded bits
```

The bits that we append to the message, should begin with '1' and the following bits must be '0' till we are exactly 64 bits less than the multiple of 512.



2. Append : Length bits

Now that we have appended our padding bits to the original message we can further go ahead append our length bits which is equivalent to 64 bits, to the overall message to make the entire thing an exact multiple of 512.

We know that we need to add 64 more bits, the way to calculate these 64 bits is by calculating the modulo of the original message i.e. the one without the padding, with 2^{32}. The message we obtain we append those length to the padded bits and we get the entire message block, which must be a multiple of 512.

3. Initialize the buffers

We have our message block on which we will begin to carry out our computations to figure out the final hash. Before we begin with that I should tell you that we need certain default values to be initialized for the steps that we are going to perform.

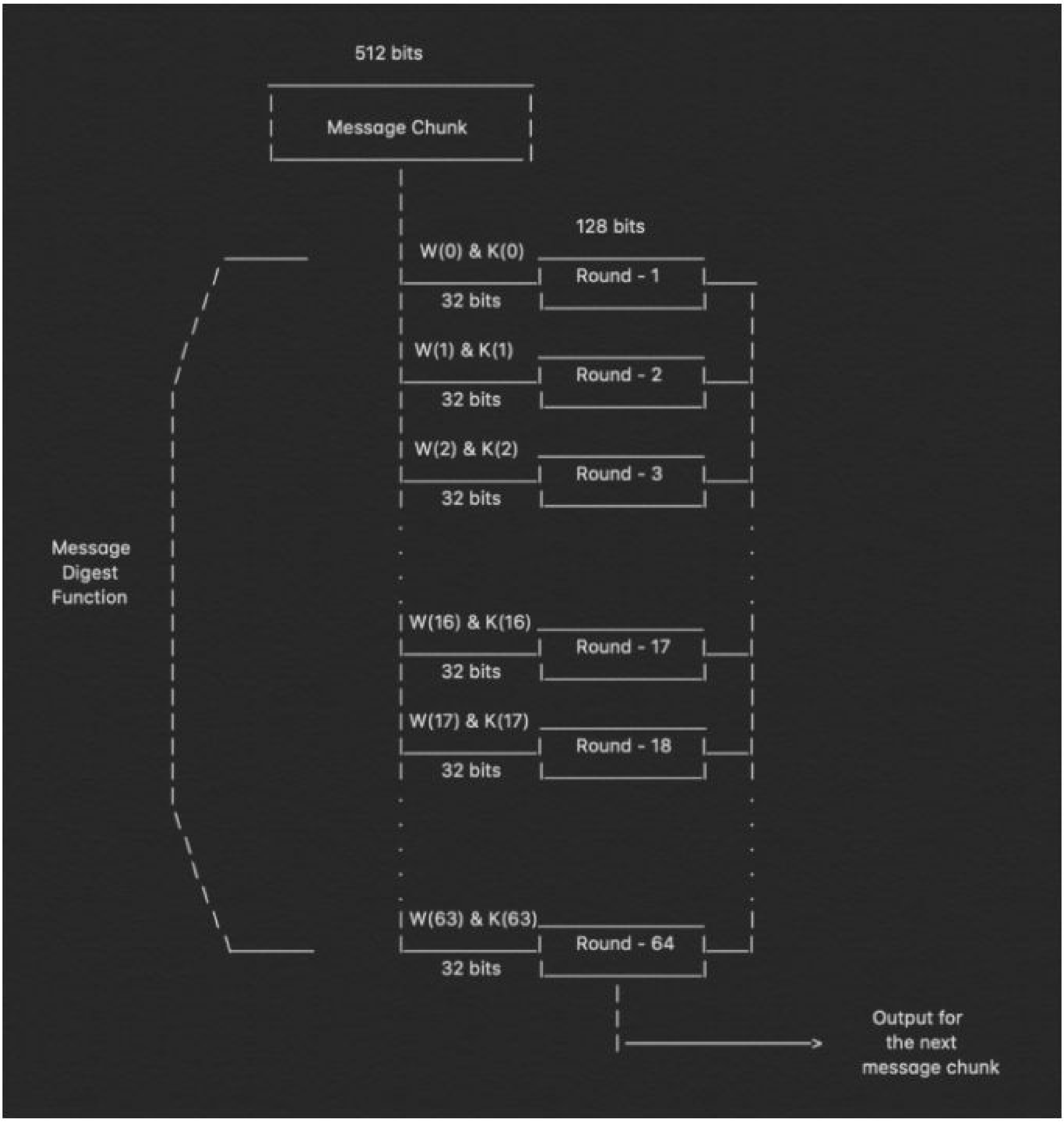
```
a = 0x6a09e667
b = 0xbb67ae85
c = 0x3c6ef372
d = 0xa54ff53a
e = 0x510e527f
f = 0x9b05688c
g = 0x1f83d9ab
h = 0x5be0cd19
```

Keep these values in the back of your mind for a while, in the next step everything will be clearly understandable to you. There are more 64 values that need to be kept in mind which will act as keys and are denoted by the word 'k'.

```
k[0..63] :=
   0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
   0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
   0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
   0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
   0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
   0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
   0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
   0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
```

4. Compression Function

So, the main part of the hashing algorithm lies in this step. The entire message block that we have 'n x 512' bits long is divided into 'n' chunks of 512 bits and each of these 512 bits, are then put through 64 rounds of operations and the output obtained is fed as input for the next round of operation.

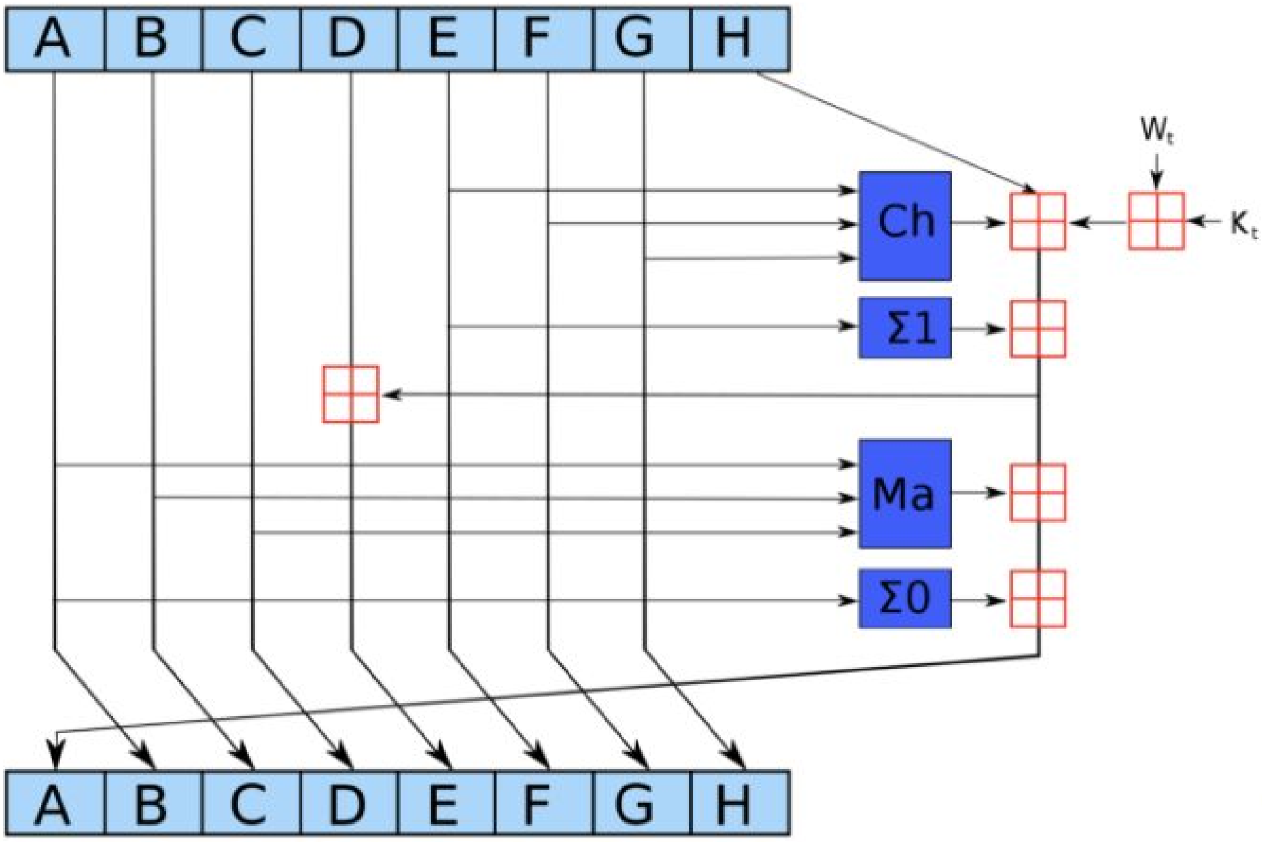


In the image above we can clearly see the 64 rounds of operation that is performed on a 512 bit message. We can observe that two inputs that we send in are W(i) & K(i), for the first 16 rounds we further break down 512 bit message into 16 parts each of 32 bit but after that we need to calculate the value for W(i) at each step.

```
W(i) = W^(i-16) + σ0 + W^(i-7) + σ1

where,
 σ0 = (W^(i-15) ROTR^7(x)) XOR (W^(i-15) ROTR^18(x)) XOR (W^(i-15) SHR^3(x))
 σ1 = (W^(i-2) ROTR^17(x)) XOR (W^(i-2) ROTR^19(x)) XOR (W^(i-2) SHR^10(x))
 ROTR^n(x) = Circular right rotation of 'x' by 'n' bits
 SHR^n(x)  = Circular right shift of 'x' by 'n' bits
```

Well now that we have a well established method to create the W(i) for any given of the 64 rounds let's dive in what happens in each of these rounds.



Depiction of a single "round"

In the image above we can see exactly what happens in each round and now that we have the values and formulas for each of the functions carried out we can perform the entire hashing process.

```
Ch(E, F, G) = (E AND F) XOR ((NOT E) AND G)
Ma(A, B, C) = (A AND B) XOR (A AND C) XOR (B AND C)
    ∑(A)    = (A >>> 2) XOR (A >>> 13) XOR (A >>> 22)
    ∑(E)    = (E >>> 6) XOR (E >>> 11) XOR (E >>> 25)
     +      = addition modulo 2³²
```

These are the functions that are performed in each of the 64 rounds that are performed over and over for 'n' number of times

5. Output

The output from every round acts as an input for the next round and this process keeps on continuing till the last bits of the message remains and the result of the last round for the n^{th} part of the message block will give us the result i.e. the hash for the entire message. The length of the output is 256 bits.

5.3 DH ALGORITHM

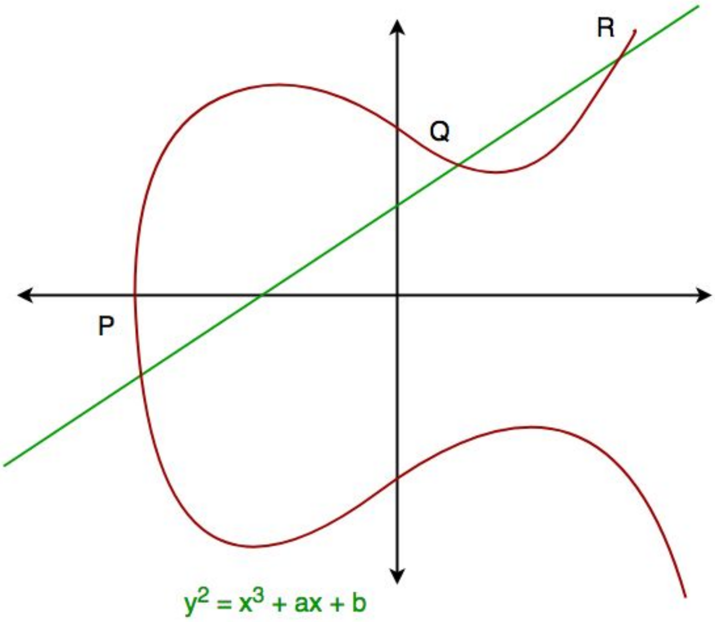
Elliptic Curve Cryptography (ECC) is an approach to public-key cryptography, based on the algebraic structure of elliptic curves over finite fields. ECC requires a smaller key as compared to non-ECC cryptography to provide equivalent security (a 256-bit ECC security has an equivalent security attained by 3072-bit RSA cryptography).

For a better understanding of Elliptic Curve Cryptography, it is very important to understand the basics of Elliptic Curve. An elliptic curve is a planar algebraic curve defined by an equation of the form

$$y^2 = x^3 + ax + b$$

The curve is non-singular; that is its graph has no cusps or self-intersections (when the characteristic of the Coefficient field is equal to 2 or 3).

In general, an elliptic curve looks like as shown below. Elliptic curves could intersect almost 3 points when a straight line is drawn intersecting the curve. As we can see the elliptic curve is symmetric about the x-axis, this property plays a key role in the algorithm.



This approach uses six tuple {P, a, b, G, n, h}

P = Field that the curve is define over

G = Generator point

a, b = Values define the curve

h = Co- factor

n = Prime order of G

5.3.1 Diffie-Hellman algorithm

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

- For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables one prime P and G (a primitive root of P) and two private values a and b.
- P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly, the opposite person receives the key and from that generates a secret key after which they have the same secret key to encrypt.

Working:

Let-

- Private key of the sender = X_s
- Public key of the sender = Y_s
- Private key of the receiver = X_r
- Public key of the receiver = Y_r

Using Diffie Hellman Algorithm, the key is exchanged in the following steps-

<u>Step-01:</u>

- One of the parties chooses two numbers ‘a’ and ‘n’ and exchanges them with the other party.
- ‘a’ is the primitive root of prime number ‘n’.
- After this exchange, both the parties know the value of ‘a’ and ‘n’.

Step-02:

- Both the parties already know their own private key.
- Both the parties calculate the value of their public key and exchange with each other.

Sender calculate its public key as-

$$Y_S = a^{X_s} \bmod n$$

Receiver calculate its public key as-

$$Y_r = a^{X_r} \bmod n$$

Step-03:

- Both the parties receive public keys from each other.
- Now, both parties calculate the value of the secret key.

Sender calculates secret key as-

$$\text{Secret key} = (Y_r)^{X_s} \bmod n$$

Receiver calculates secret key as-

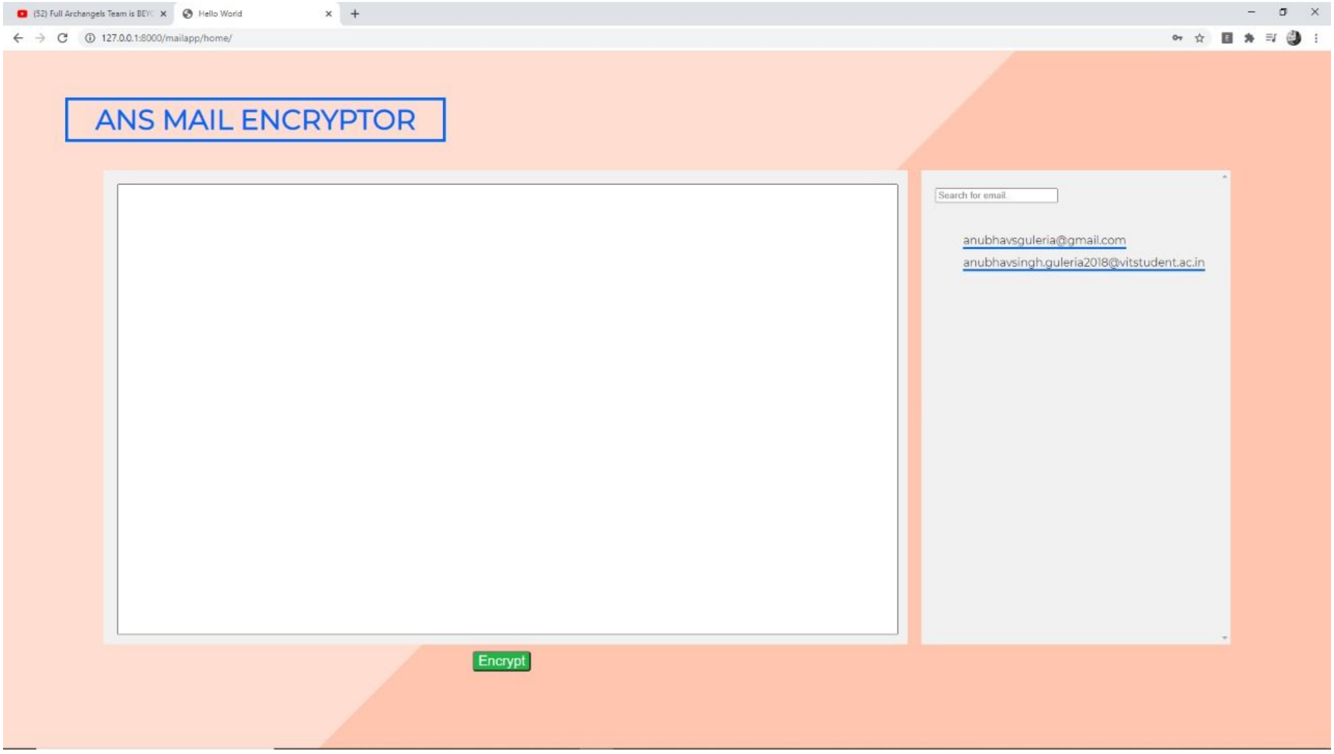
$$\text{Secret key} = (Y_S)^{X_r} \bmod n$$

Finally, both the parties obtain the same value of secret key.

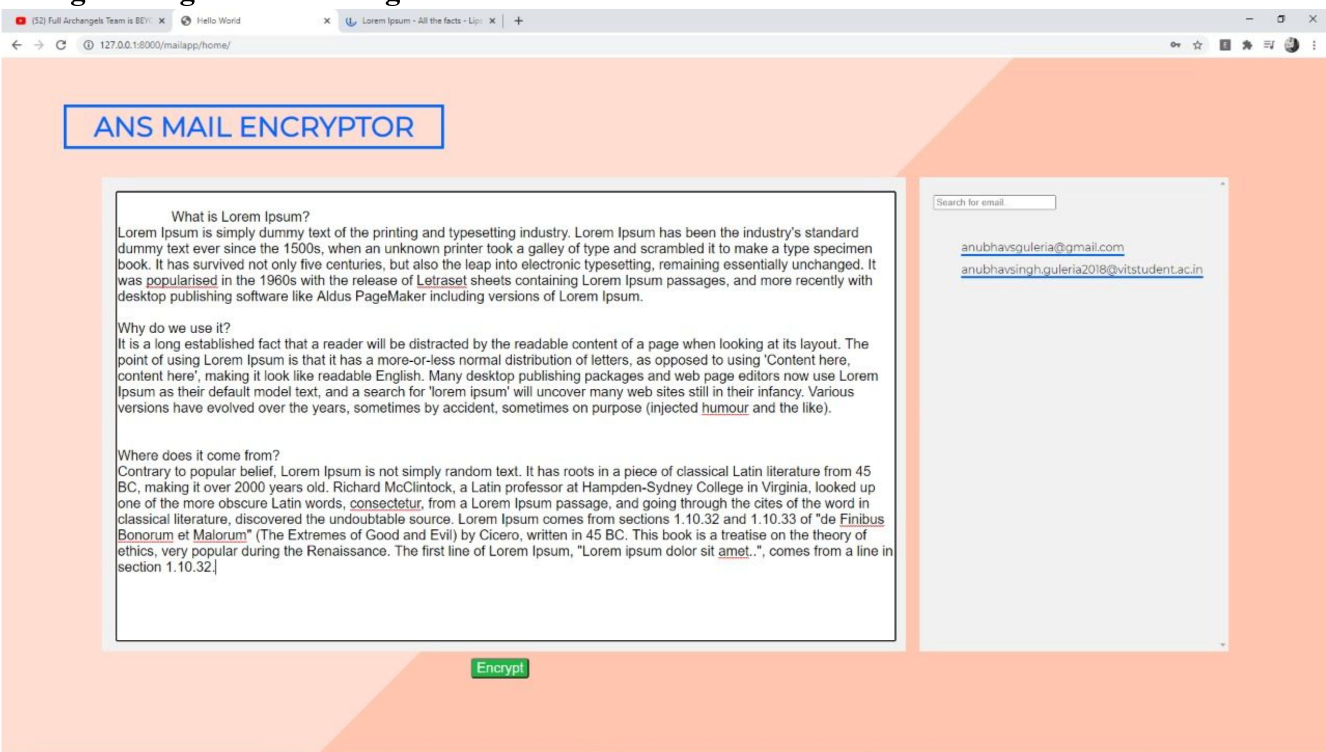
6. RESULTS

Chrome was found to be the most compatible browser of all the browsers we tested. Firefox, Edge and Chromium were close second, however firefox required some simple additional customisations to function properly.

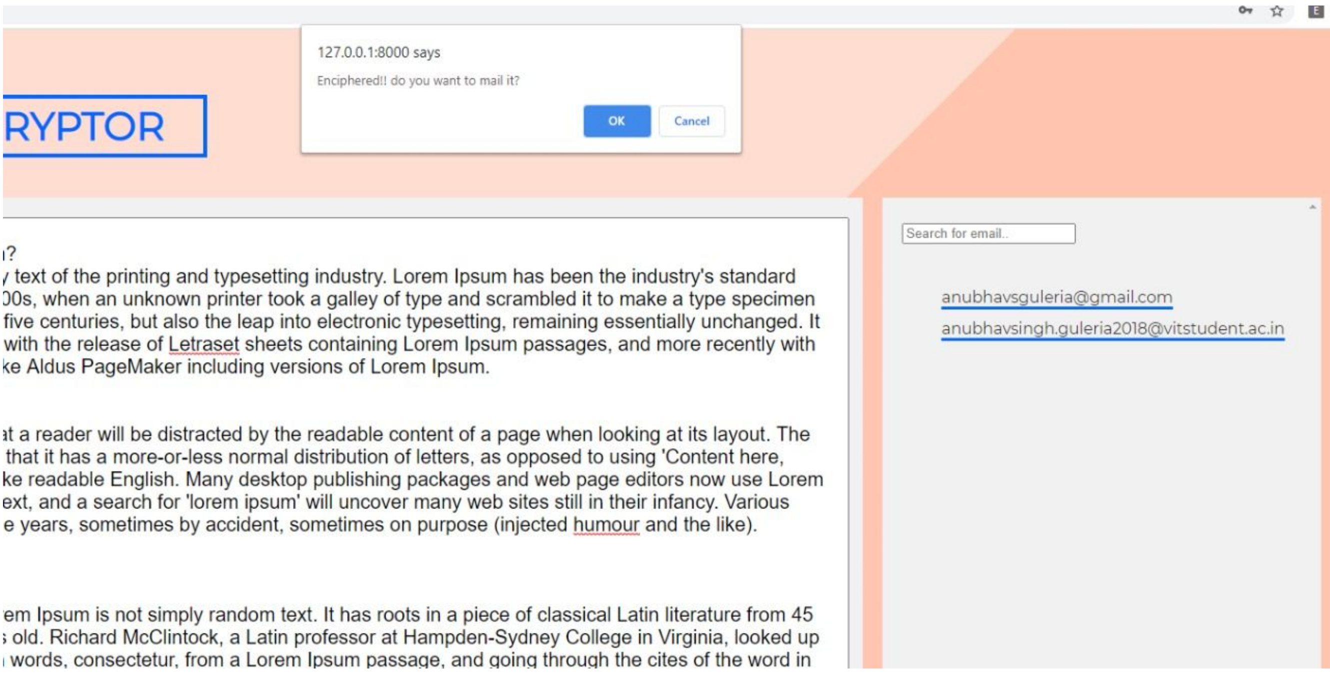
User Interface



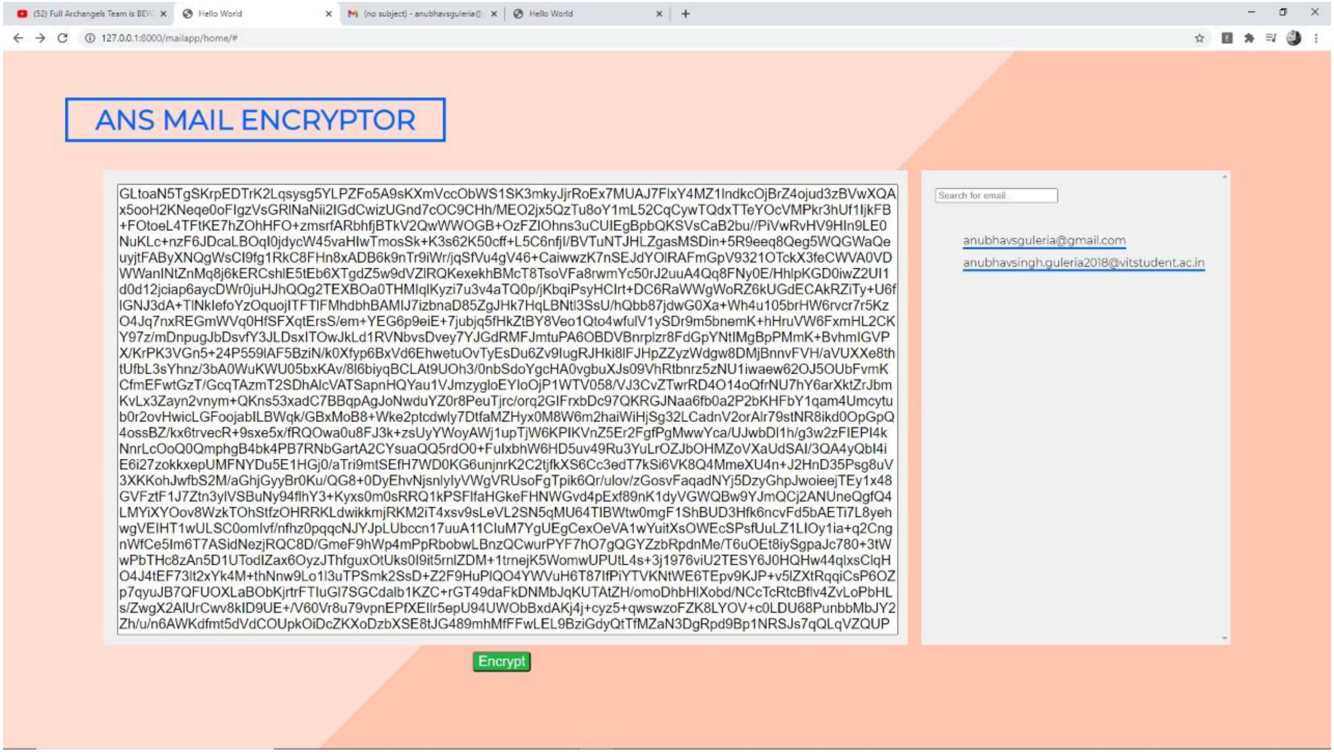
Filling Message in the message box



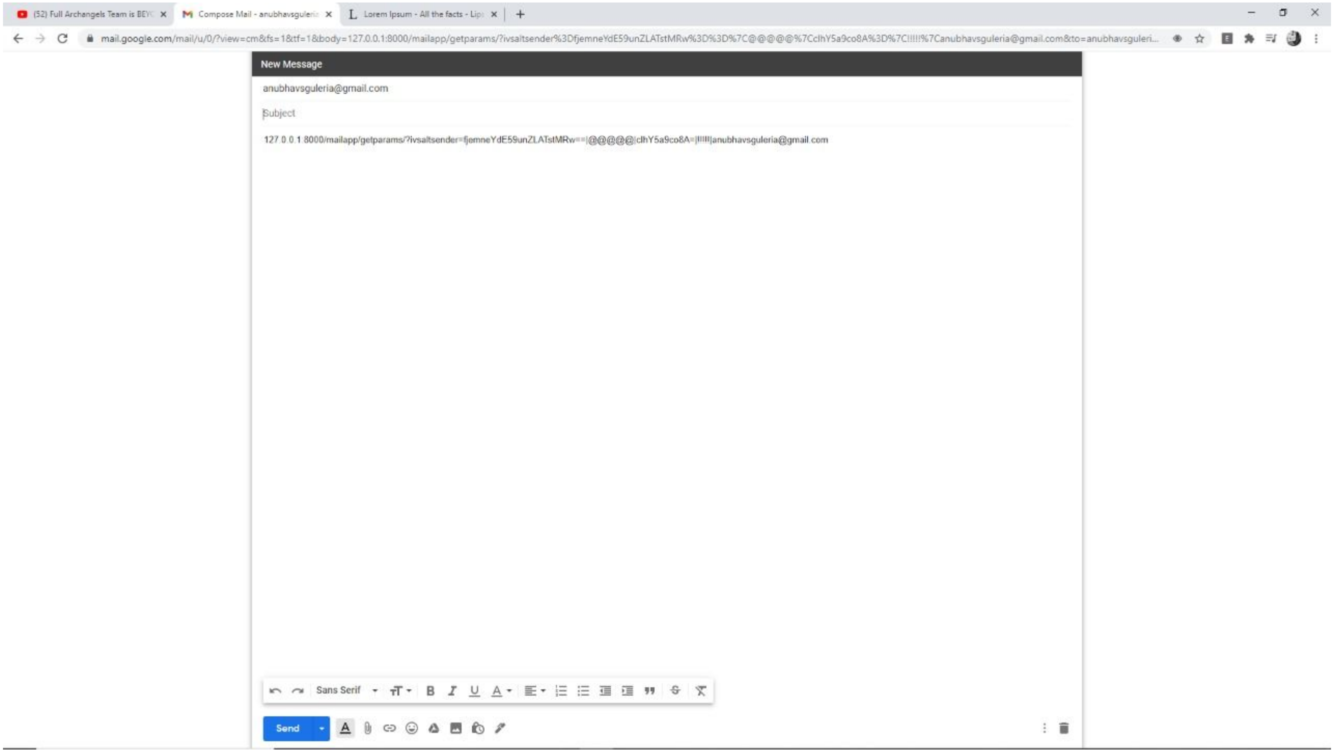
After clicking encrypt and selecting whom to send



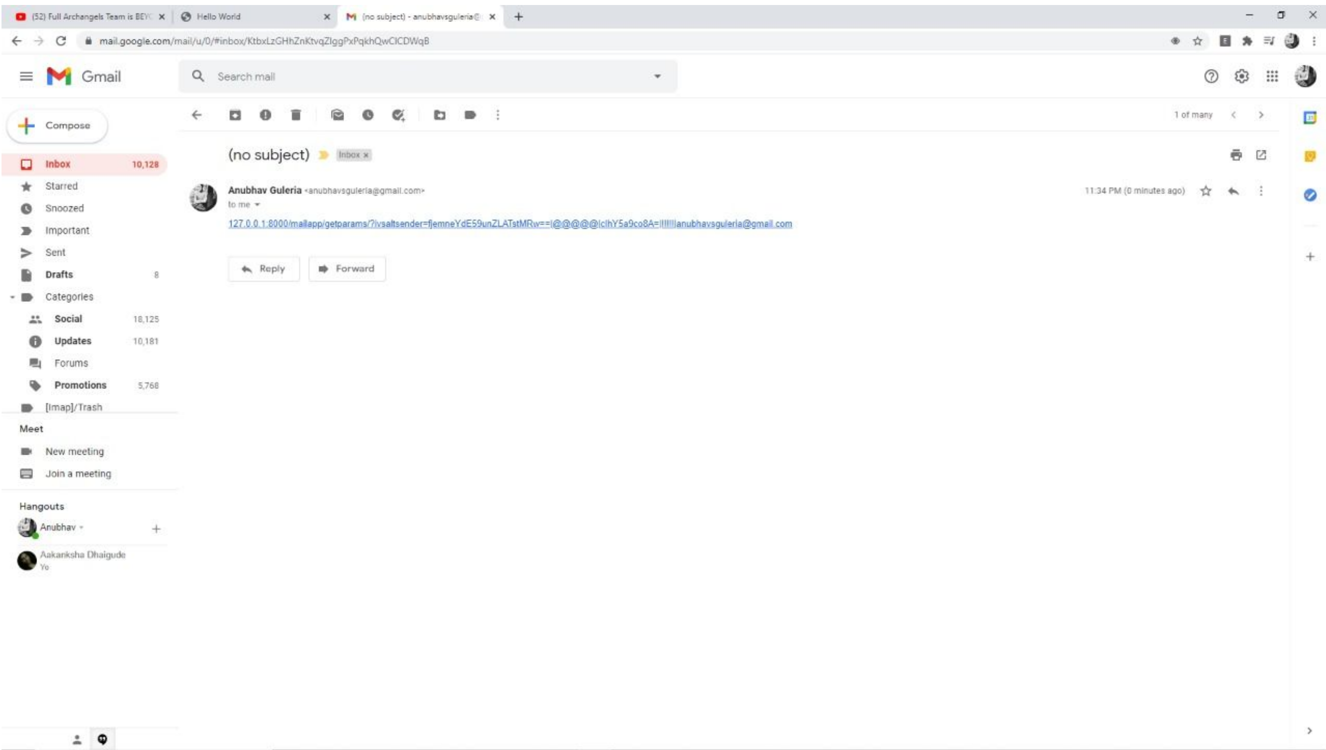
Encrypted message



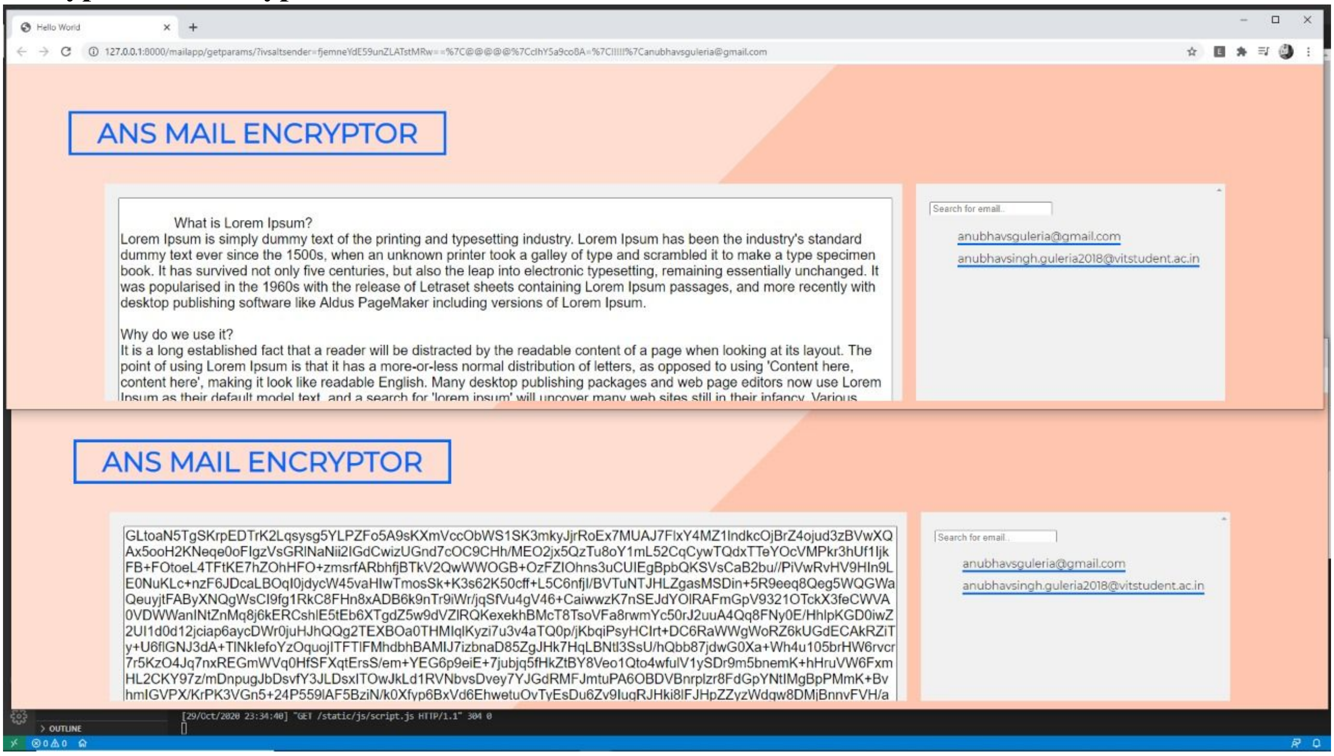
The Gmail Tab opens



Received email

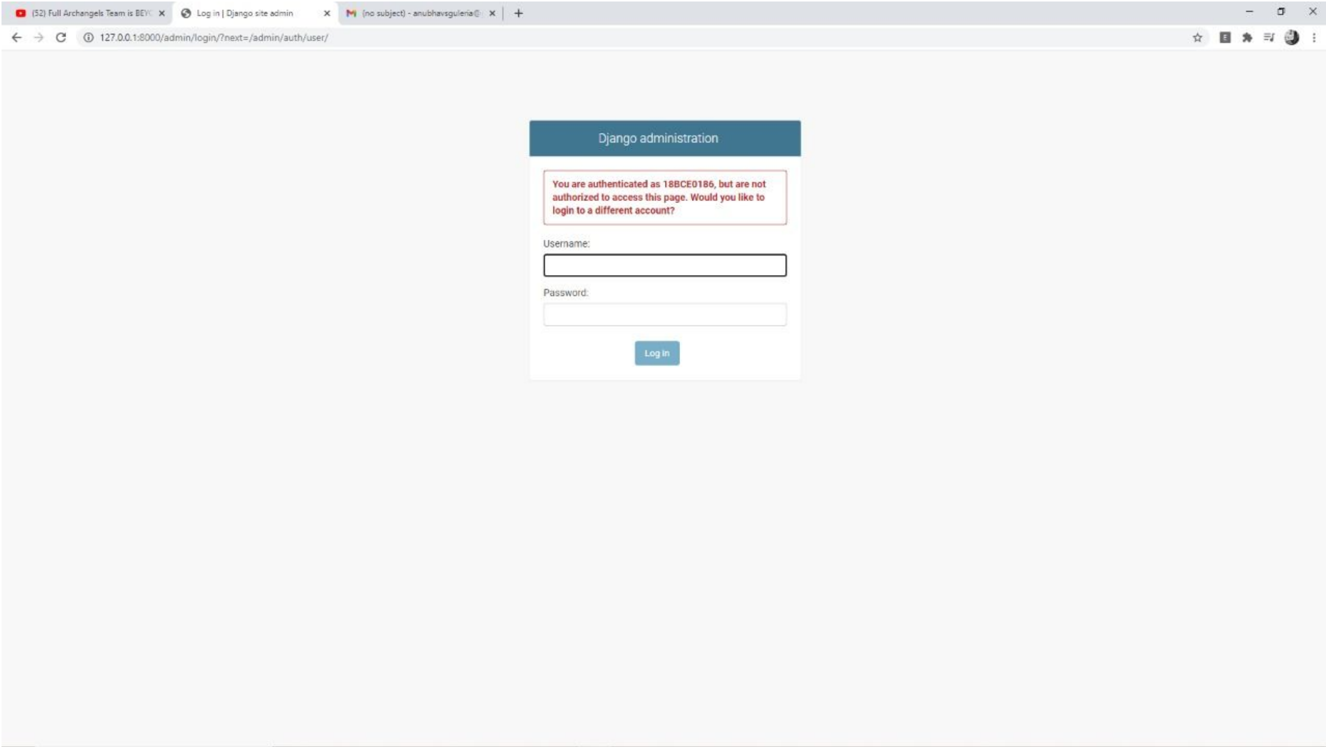


Encrypted v/s Encrypted mail

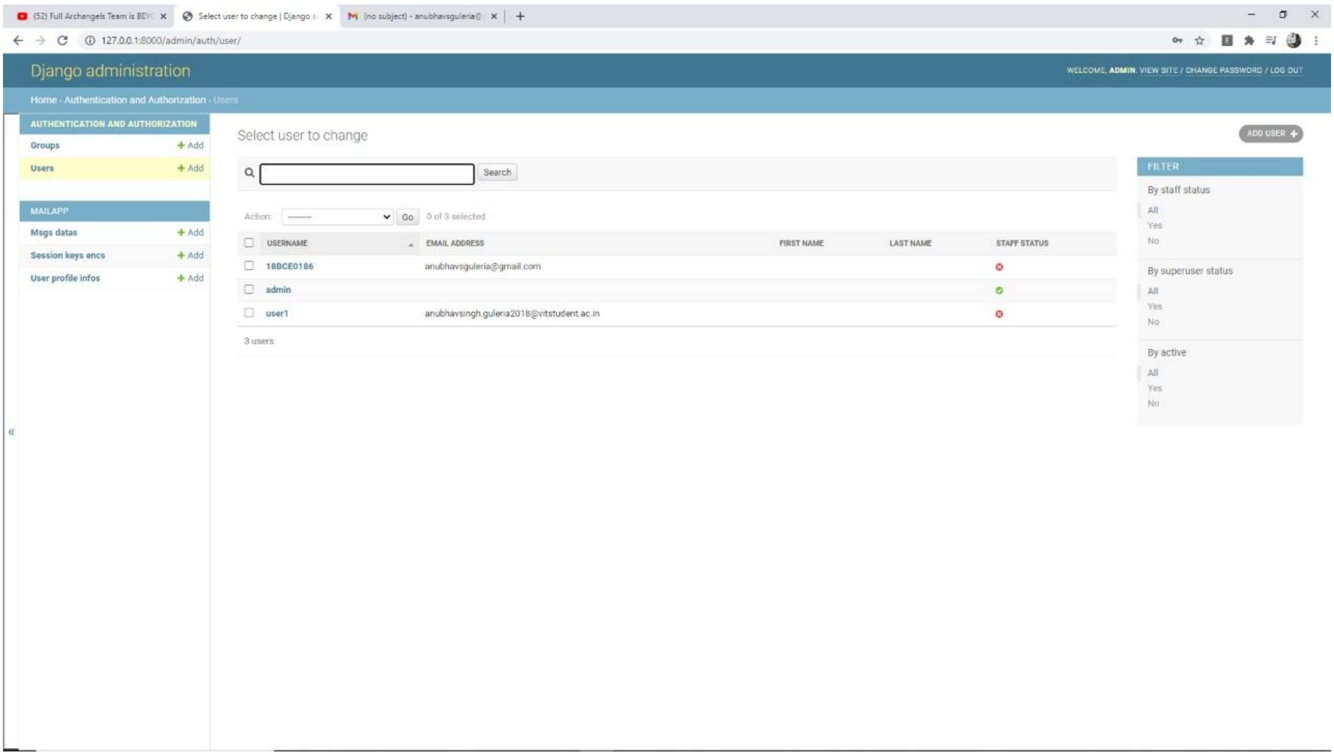


FOR ANALYSIS AND AUDIT

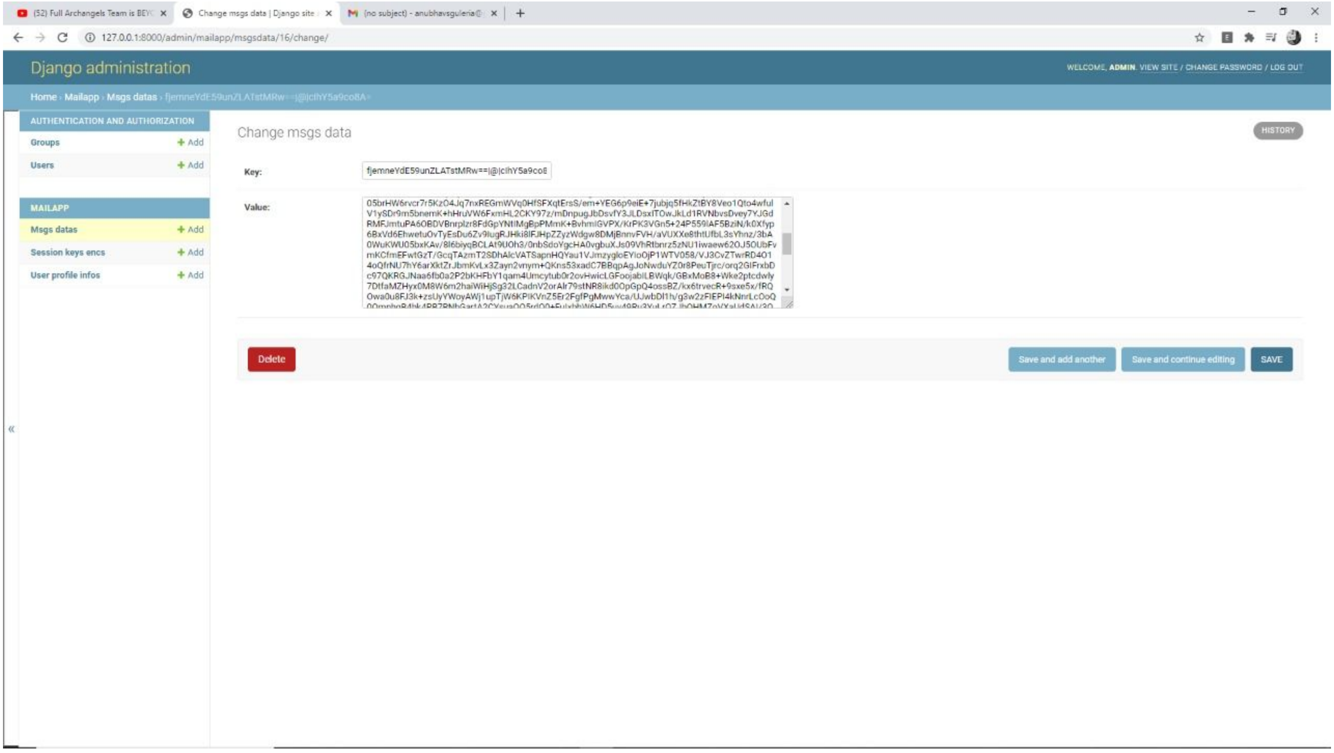
Logging into Admin account



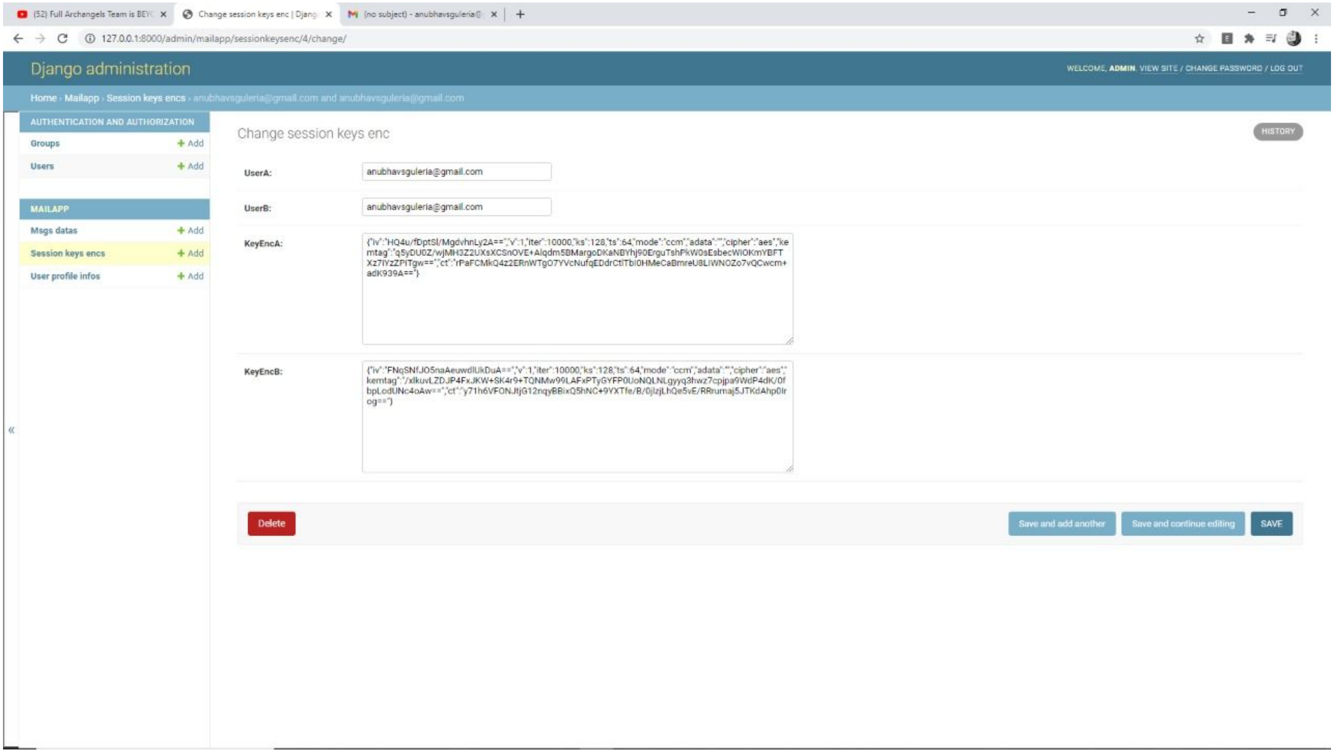
Users data as visible to the admin



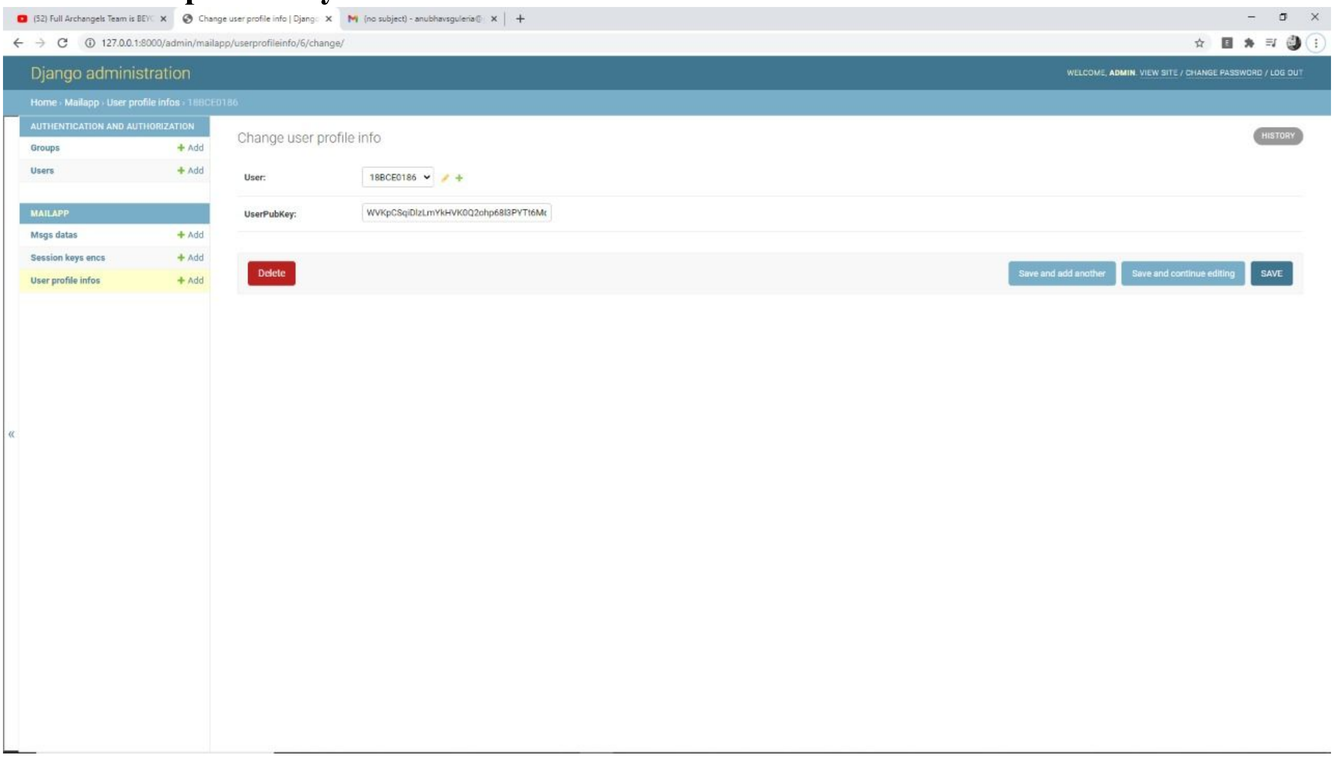
Message and keys as visible to the admin



List of keys in a transaction as visible to the user



User and its public key



6. CONCLUSION

The team has built a novel service for allowing multiple layers of encryption while also attempting to counter the chief problem that email service providers cite to avoid encryption - user convenience and time consumption. Javascript is highly integrable to any web application. That has certainly helped a great way through this project.
For future works, the project could incorporate a larger scale of users on a cloud based server or possibly a hardware accelerated server, which will lead the project to deliver the same efficiency as a traditional email service.

7. REFERENCES

1. "*Effectiveness comparison of the AES and 3DES cryptography methods on email text messages*" by Rini Indrayani, Subektiningsih and others, International Conference on Information and Communication Technology, IEEE, 2019
2. "*Secure e-mail communication - Comparison and selection of encryption solutions using an utility value analysis approach*" by D. Fischer, B Markscheffel and K Scherr,

12th International Conference for Internet Technology and Secure Transactions, IEEE, 2018.

3. "*A simple construction of encryption for a tiny domain message"* by Rashed Mazumder, Astuko Miyaji and Chunhua Su, *51st Annual Conference on Information Sciences and Systems, IEEE*, 2017.
4. "*E-mail Security Framework through Various Virus Encryption Techniques*" by Ashutosh Prasad Bhatt, Dr Monika Sharma, *Proceedings of the International Conference on Intelligent Computing and Control Systems, 2019, IEEE*
5. "*Implementation of Elliptical Curve Cryptography over a Server-Client Network*" by Bore Gowda S B, *5th International Conference on Devices, Circuits and Systems, 2020*
6. "*Security audit in cloud-based server by using encrypted data AES-256 and Sha-256*" by M Husmi, H T Ciptaningtya, W Suadi, R M Ijihadie, R Anggoro, M F Salam and S Arifiani, *IOP Conference Series: Materials Science and Engineering, IOP Publsihing, Scopus*
7. "*Efficient Extended Diffie-Hellman Key Exchange Protocol*" by Arjun Singh Rawat and Maroti Deshmukh, *International Conference on Computing, Power and Communication Technologies, 2019, IEEE*
8. "*Generation of Symmetric Key Using Randomness of Hash Function*" by Kamana Sai Charan, Harsha Vardhan Nakkina, B R Chandravarkar, *11th ICCCNT, 2020, IEEE*