ABSTRACT

Through the project "End to end encryption" we aimed at understanding the concepts used to implement an end-to-end encrypted chat. End-to-end encryption is relevant to the world we live in these days as many social media chat applications like WhatsApp and Signal are making use of such encryption techniques. This project therefore aides us in understanding the importance of Information Systems and Security. In this project we tried to implement a simple encrypted chat application followed by implementation of the Signal protocol's Extended Triple Diffie Hellman, Double Ratchet and Encryption using AES CBC(Cipher Block Chaining). As a result, our understanding on the working of an end-to-end encrypted chat has solidified.

1. Introduction

What is data security?

Data security is a set of processes and practices designed to protect your critical information technology (IT) ecosystem. This included files, databases, accounts, and networks. Effective data security adopts a set of controls, applications, and techniques that identify the importance of various datasets and apply the most appropriate security controls.

What is data security important?

Data security is critical to public and private sector organizations for a variety of reasons.

•First, there are legal and moral obligations that state that companies have to protect their user and customer data from falling into the wrong hands.

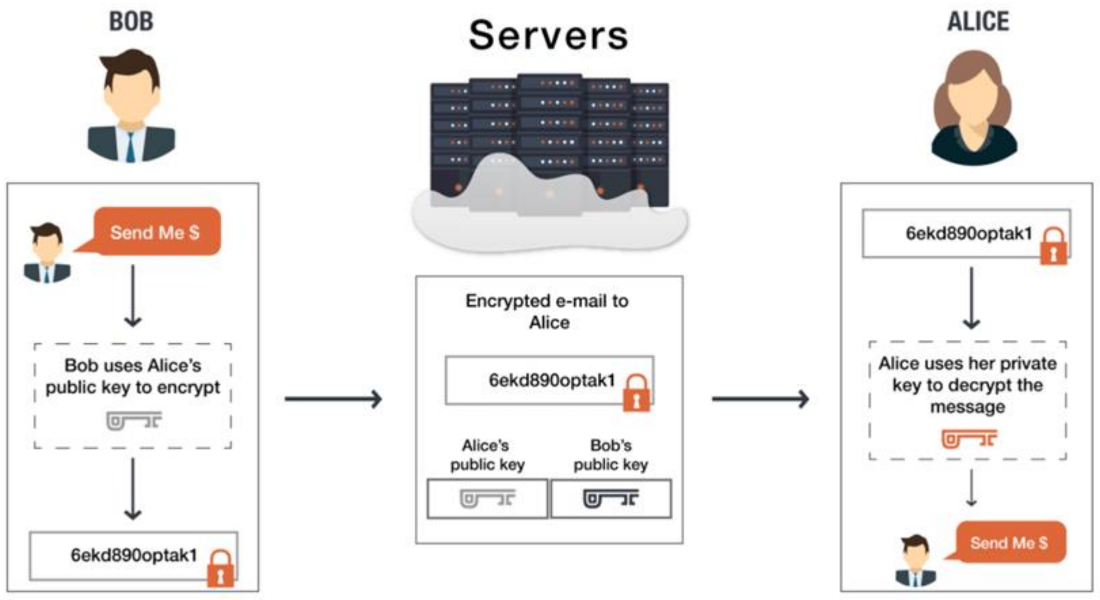
•Secondly a company's reputation is at risk in the case of data breach or hack. If they do not take data security seriously, their reputation can be permanently damaged in the event of a publicized, high-profile breach or hack. Furthermore, they will need to spend time and money to assess and repair the damage, as well as determine which business processes failed and what needs to be improved.

End to End Encryption: The gold standard for communication protection

- What is end to end encryption and how does it work?
- Why end-to-end encryption is important and what it protects against?
- Can end-to-end encryption be hacked?
- What are the advantages of end-to-end encryption?

1. What is end to end encryption and how does it work?

- In an end-to-end encrypted system, the only people who can access the data are the sender and the intended recipient(s) – and no one else. Neither hackers nor unwanted third parties can access the encrypted data on the server.
- In true end-to-end encryption, encryption occurs at the device level. That is, messages and files are encrypted before they leave the phone or computer and isn't decrypted until it reaches its destination. As a result, hackers cannot access data on the server because they do not have the private keys to decrypt the data. Instead, secret keys are stored with the individual user on their device which makes it much harder to access an individual's data.
- The security behind end-to-end encryption is enabled by the creation of a public-private key pair. This process, also known as asymmetric cryptography, employs separate cryptographic keys for securing and decrypting the message. Public keys are widely disseminated and are used to lock or encrypt a message. Private keys are only known by the owner and are used to **unlock or decrypt the message.**


BOB
Servers
ALICE
Send Me $
Bob uses Alice's public key to encrypt
6ekd890optak1
Encrypted e-mail to Alice
6ekd890optak1
Alice's public key
Bob's public key
6ekd890optak1
Alice uses her private key to decrypt the message
Send Me $

2. Why end-to-end encryption is important and what it protects against?

- End-to-end encryption is important because it provides users and recipients security for their email and files from the moment the data is sent by the user until the moment it is received by the recipient. It also ensures that no third party can read the exchanged messages.

- Services like Gmail, Yahoo or Microsoft enable the provider to access the content of users' data on its servers because these providers hold copies to the decryption keys. As such, these providers can read users' email and files. In Google's case, its possession of decryption keys has enabled them in the past to provide the Google account holder with targeted ads.
- By contrast, in well-constructed end-to-end encrypted systems, the system providers never have access to the decryption keys.

3. **Can end-to-end encryption be hacked?**
    - As the *Department of Homeland Security* has written:
      Given that attackers will go after low hanging fruit like where the data is stored, a solution that does not protect stored data will leave information extremely vulnerable. When practitioners use end-to-end encryption however the data that is stored on the server is encrypted. Even if a hacker were to access it, all they would get is gibberish.

    - The DHS goes on to state in its report:
      Attacking the data while encrypted is just too much work [for attackers].

4. **What are the advantages of end-to-end encryption?**
    - **Ensures your data is secure from hacks**: With end-to-end encryption, you are the only one who has the private key to unlock your data. Data on the server can't be read by hackers because they do not have the private keys to decrypt the information.
    - **Protect your privacy**: Providers like Google and Microsoft can read your data. When you use their service, data is decrypted on their servers. If data is decrypted on their servers, then hackers and unwanted third parties can read it, too.
    - **Protects admins**: Since admins don't hold the decryption keys to decrypt the data, any attack that targets administrators will come up short.

Is End-to-End Encryption relevant to us today?

Yes, it is. When it comes to E2E communication over the Internet, the best example would be messengers like WhatsApp, iMessage, and Signal (in which E2EE is turned on by default) or Telegram, Allo, and Facebook's Secret Conversation where E2EE is enabled by a special switch.

2. Literature Survey

S.No.	Title	Journal	Encryption Method	Remark
1.	Securing Instant Messages With Hardware-Based Cryptography and Authentication in Browser Extension	IEEE Access, Vol. 8, Pages. 95137-95152, 2020	Integrating hardware-based public key cryptography into Converse.js enabled with the Extensible Messaging and Presence Protocol (XMPP)	● It is able to guarantee confidentiality, authenticity and integrity to the IM application in a manner not predictable by unauthorized third parties.
2.	Virtual network function deployment and service automation to provide end-to-end quantum encryption	IEEE/OSA Journal of Optical Communications and Networking, Vol. 10, No. 4, Pages. 421-430, April 2018	A node is designed to provide Quantum Key Distribution-enhanced security in end-to-end services and analysis of control plane requirements for service provisioning in transport networks	● QKD is a novel technique to provide synchronized sources of symmetric keys between two separated domains. It is based on the fundamental laws of quantum physics, which makes it difficult to copy the quantum states exchanged between both endpoints.
3.	Modified Blind Source Separation for Securing End-to-End Mobile Voice Calls	IEEE Communications Letters, Vol. 22, No. 10, Pages. 2072-2075, Oct. 2018	Modified BSS algorithms are used for secured communications between the two ends of any voice call without changing existing mobile network infrastructure and also the key generation process limits the bandwidth occupied by the encrypted speech signal.	● It is a feasible, low cost and portable end-to-end voice call secure system. Although there was some degradation in signal to noise ratio of the modified BSS but overall security was unaffected.

4.	Resilient End-to-End Message Protection for Cyber-Physical System Communications	IEEE Transactions on Smart Grid, Vol. 9, No. 4, Pages. 2478-2487, July 2018	Long term keys per node are given by the REMP Protection authentication server. End-to-end authenticators per message are sent that consist of a message sender's identity and a message authentication code	● Compared to conventional group security schemes, this method has improved end-to-end security strength in terms of confidentiality, integrity, authentication and key exposure resilience while preserving scalability and extensibility.
5.	End-to-End Security for Local and Remote Human Genetic Data Applications at the EGA	IEEE/ACM Transactions on Computational Biology and Bioinformatics, Vol. 16, No. 4, Pages. 1324-1327, 1 July-Aug. 2019	Files are individually encrypted before archival using an encryption format (AES-CTR). Standardization to access of genomic data by providing powerful APIs and corresponding tools and toolchain integration	● It provides end-to-end security for genomic data, where data is kept secure at rest as well as during transfer and while providing advanced random access modes both for data at rest and via REST API
6.	A Certificateless One-Way Group Key Agreement Protocol for End-to-End Email Encryption	IEEE Access, vol. 9, pp. 90677-90689, 2021.	This paper presents a certificateless one-way group key agreement protocol.	● This paper proposes a method with which email security can be improved. Key features of the method are: 1) It is certificateless and therefore it forgoes the key escrow problem. Additionally, there is no need for a public key certificate infrastructure; 2) one way group key agreement is provided

				which eliminates the back-and-forth message exchange; 3) *n*-party group key agreement (not just 2- or 3- party).
7.	An Approach for End-to-End E2E Security of 5G Applications	2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity) , IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS), 2018, pp. 133-138,	Propose an intuitive 3D ECCDH PAKE protocol that assures high speed and security with less key length.	● The need to End-to-End encryption(E2E) in 5G systems has been highlighted in this paper. ● A framework has been designed for measuring the many applications with high security and trust presented by E2E security for various services. ● Heterogeneity and openness of 5G networks makes it susceptible to security vulnerabilities such as identity, confidentiality, integrity and availability. ● 3D ECCDH PAKE protocol has been proposed ● Proposed system design comprises 3 phases i.e. initialization, registration and authentication and key exchange. ● With the assistance of the 3D PAKE

				protocol, obtaining the sensitive information by the attacker is infeasible.
8.	An implementation of a lightweight end-to-end secured communication system for patient monitoring systems	2018 Emerging Trends in Electronic Devices and Computational Techniques (EDCT), 2018, pp. 1-5,	Lightweight version of AES encryption	● In a Patient Management System(PMS), the wireless link between body sensors and the user's device is susceptible to threat by eavesdroppers or malicious users. ● To overcome this, the researchers proposed and developed an end-to-end secure PMS with a lightweight encryption protocol since sensors cannot handle algorithms requiring high computational power. ● MQ telemetry transport (MQTT) as a transport protocol rather than hypertext transport protocol (HTTP) for its simplicity and lightweight features. ● This implementation can be used as a template for testing other prominent security algorithms and transport protocols, such as CoAP.
9.	Light-weight Internet-of-Things Device	IEEE Internet of Things Journal	Proposes a novel lightweight IoT device	● The paper suggests that there is a need to

	Authentication Encryption and Key Distribution using End-to-End Neural Cryptosystems		authentication, encryption, and key distribution approach using *neural cryptosystems* and *binary latent space*	protect IoT devices from the attacks of hijackers and hence proposes a neural cryptosystem to ensure safety. ● The neural cryptosystems use three types of E2e encryption schemes: symmetric, public-key, and without keys. ● Promising results were shown in terms of *Encryptor, Decryptor, and Public-key Generator Loss and Plaintext Reconstruction Error, Hamming distance between public and private key,Hamming between plain text and cipher text and Security analysis of ciphertext.*
10.	Security Analysis of End-to-End Encryption for Zoom Meetings	IEEE Access, vol. 9, pp. 90677-90689, 2021	This paper studies the E2EE adopted by Zoom which implements AES-GCM, key derivation using HKDF algorithm, Diffie-Gellman(DH) over Curve22519 nd EdDSA over Ed25519	● To protect communications between participants Zoom Video Communications rolled out their End to End Encryption (E2EE) in 2008. ● By analysing their cryptographic protocols ,this study conducted thorough security evaluations of the E2EE of Zoom(version 2.3.1) ● They found several attacks more powerful

				than those anticipated by Zoom in their whitepaper. ● One such attack identified was that if insiders collude with meeting participants, they can impersonate any Zoom user in target meetings, whereas Zoom indicates that they can impersonate only the current meeting participants.
11.	Post-quantum Secure Group Messaging	2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus), 2021, pp. 2323-2326, doi: 10.1109/ElConRus51938.2021.9396513.	In this paper, they suggested ways and primitives for creating a group key based on isogenies of elliptic curves and have named such protocol as an extended Double Ratchet Protocol.	● In this paper they have discussed the needed security properties and existing ways in creating group chats. ● The existing ways have been implemented in Signal,Threema and What's App Applications but failed to provide all the properties needed,even against attacks using classical computers. ● So,they have come up with the Double Ratchet Protocol, in which the main idea is that the key changes with every new message. ● Each participant stores three key chains: 1)The root chain. 2)The sending chain.

				3)The receiving chain. ● Further studies have to be done on post-quantum group schemas.
12.	Partitioned Private User Storages in End-to-End Encrypted Online Social Networks	2020 15th International Conference for Internet Technology and Secured Transactions (ICITST), 2020, pp. 1-8, doi: 10.23919/ICITST51030.2020.9351335.	This paper proposed a scheme to split encrypted user storages into multiple storages. Each partition can be reconstructed with the help of other people of the OSN.	● This approach allows reconstructing of login details even if the server or administrator are unaware. ● If a storage is recovered a part of the OSN can be accessed again. ● Main advantage is that no knowledge about login details is present on the server. ● The partitions have a group of chat rooms. ● Each peer in each chatroom receives multiple shares for a compartmented secret sharing scheme and an additional threshold secret sharing scheme. ● Several attack scenarios on the scheme have also been analyzed.
				● In this paper, they compared the existing whistleblowing platforms with AVISPA(Automated Validation of Internet Security Protocols and Applications)tool and summarized that they are UNSAFE, whereas the proposed architecture turns out

				to be SAFE. ● It performs end-to-end encryption client-side using JavaScript. ● It has main 6 entries namely: 1)The Whistleblowers 2)The platform/software. 3)The Media. 4)The Server administrator. 5)The Editors. 6)The Browser Extension. ● It is flexible enough to be customized and incorporated into existing systems.
13.	A Simple and Robust End-to-End Encryption Architecture for Anonymous and Secure Whistleblowing	2019 Twelfth International Conference on Contemporary Computing (IC3), 2019, pp. 1-6, doi: 10.1109/IC3.2019 .8844917.	This paper proposed a simple and robust architecture to enable secure and anonymous whistleblowing.	
14.	An end-to-end cryptography based real-	2021 16th Iberian Conference on Information Systems and Technologies (CISTI), 2021, pp. 1-6, doi: 10.23919/CISTI5	This paper proposed a chat application which tries to implement an innovative way of	● In this paper, most used message applications such as Signal,Telegram,Threema, have been chosen to identify and evaluate the characteristics of encryption mechanisms. ● The proposed chat application has two new approaches namely: 1)Server does not have any knowledge of the key and the message content in the Room created. 2)There is no data persistence,that is once

	time chat	2073.2021.9476399.	sending messages with end-to-end-encryption.	the Room is inactive it is not possible to access the data later. ● The disadvantages are: 1)The symmetric key which is managed by the user would allow all messages when the Room is active. 2)Usage of TLS instead of DTLS as TLS send messages at a slower rate compared to DTLS.
15.	Public-Key Encryption Secure Against Related Randomness Attacks for Improved End-to-End Security of Cloud/Edge Computing	IEEE Access, vol. 8, pp. 16750-16759, 2020, doi: 10.1109/ACCESS.2020.2967457.	This paper proposed some methods of constructing secure public-key encryption schemes against related randomness attacks.	● This paper proposed two schemas namely: 1)RRA-SECURE PKE SCHEME FROM RKA-SECURE ONE WAY FUNCTION. 2)RRA-SECURE PKE SCHEME AGAINST ARBITRARY FUNCTION. ● In terms of efficiency, the encryption algorithm of the first proposed scheme is inefficient, while the decryption algorithm of it is very efficient. ● The first scheme secure against arbitrary function is actually a publicly deniable encryption scheme so that it is inefficient at present.
16.	SMS Encryption Using 3D-AES Block Cipher on	IEEE 2013 International	Use of 3D-AES block cipher	● From the experiment, the 3D-AES has low

	Android Message Application	Conference on Advanced Computer Science Applications and Technologies (ACSAT) - Kuching, Malaysia (2013.12.23-2013.12.24)	symmetric cryptography algorithm for SMS transfer securing.	encryption time when message size is more than 256 bits. It can be indicated that an SMS encryption application using the 3D-AES block cipher will be proposed running after 256 bits.
17.	Chat Message Security Enhancement on WLAN Network Using Hill Cipher Method	2020 8th International Conference on Cyber and IT Service Management (CITSM). doi:10.1109/citsm50537.2020.92688	The author makes a system that can be used to secure message exchange information chat on WLAN networks with cryptographic techniques namely Security Of Chat Message On WLAN Network By Using Hill Cipher Method	● By using a method Hill cipher Binomial coefficient can be used as a key matrix ● The hill cipher key matrix must be an invertible matrix. ● Each encrypted message must have a multiple of n, which is the number of columns in the key matrix. ● The results of the same plaintext encryption using different key matrices will produce different ciphertexts. ● Hill Cipher's algorithm is quite well used in encrypting message text data because it is a linear combinator. Hill Cipher is able to secure text data but vulnerable to statistical attacks, but this weakness can be reduced by adding a key length. the longer the key will be more difficult to perform known plaintext attacks.

18.	More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema	[IEEE 2018 IEEE European Symposium on Security and Privacy (EuroS&P) - London, United Kingdom (2018.4.24-2018.4.26)	This paper describes the group communication protocols of Signal, WhatsApp, and Threema and thereby presents three fundamentally different protocols for secure and instant group communication to enable further scientific analyses.	● The protocols are analyzed by applying a security model and thereby reveal several insufficiencies showing that traceable delivery, closeness and thereby confidentiality of their group chat implementations are not achieved.
19.	Power system real time data encryption system based on DES algorithm	2021 13th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA). doi:10.1109/icmtma52658.2021.0005	The real-time data encryption system of the power system is optimized by the hybrid encryption system based on the DES algorithm.	● The real-time data encryption of the power system adopts the triple DES algorithm, and double DES encryption algorithm of RSA algorithm to ensure the security of triple DES encryption key, which solves the problem of real-time data encryption management of power systems. ● Java security packages are used to implement digital signatures that guarantee data integrity and non-repudiation. ● Experimental results show that the data encryption system is safe and effective.
20.	Classifying service flows in the encrypted skype traffic	IEEE ICC 2012 - 2012 IEEE International Conference on Communications -	This paper proposes a classification method for Skype encrypted traffic based on the	● The classification method for Skype encrypted traffic based on the Statistical Protocol IDentification

		Ottawa, ON, Canada (2012.06.10-2012.06.15)	Statistical Protocol IDentification (SPID)	(SPID) analyzes statistical values of some traffic attributes. ● They have evaluated their method on a representative dataset to show excellent performance in terms of Precision and Recall

3. Modules and Descriptions

Module 1: Simple Encrypted Chat

Encryption Technique

A substitution technique is one in which the letters of plaintext are replaced by other letters or by numbers or symbols. If the plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with cipher text bit patterns.

Caesar cipher (or) shift cipher

The earliest known use of a substitution cipher and the simplest was by Julius Caesar. The Caesar cipher involves replacing each letter of the alphabet with the letter standing 3 places further down the alphabet.

e.g., Plain text : pay more money Cipher text: SDB PRUH PRQHB

Note that the alphabet is wrapped around, so that letter following „z" is „a". For each plaintext letter p, substitute the cipher text letter c such that

C =E(p) = (p+3) mod 26

A shift may be any amount, so that general Caesar algorithm is

C = E (p) = (p+k) mod 26

Where k takes on a value in the range 1 to 25. The decryption algorithm is simply P = D(C) = (C-k) mod 26

We tried to implement a similar encryption technique for our chat application.

Implementation Details: The following project was implemented using -

(1) Hypertext Markup Language (HTML)

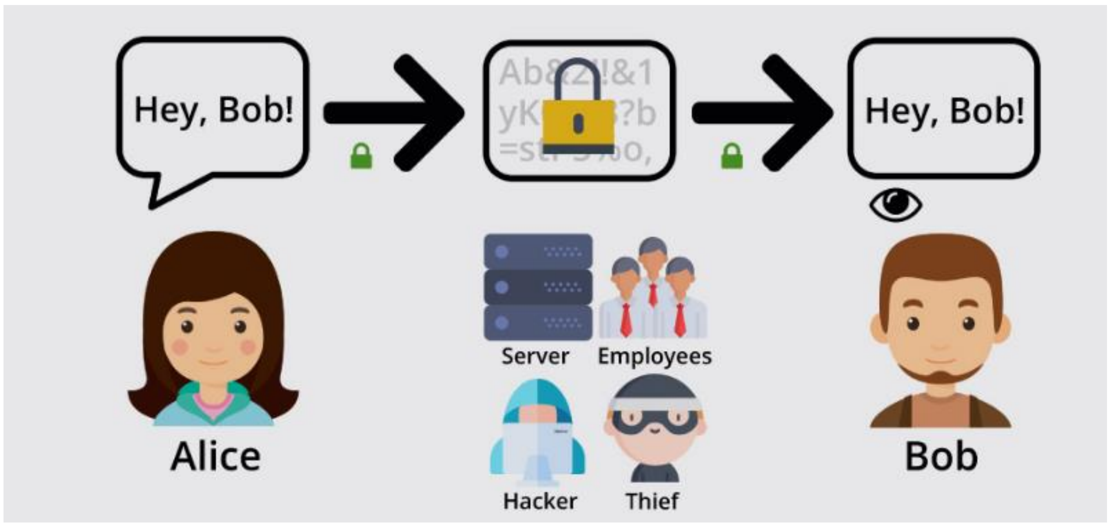
It is used for creating a responsive web page for the chat application. Divisions were made for various chat windows for sender, receiver and hacker. Input chat messages were taken using textarea and sent for encryption using a class name. Three independent keys were taken and assigned a unique id which were also sent for the encryption and decryption process.

(2) Javascript

It allows us to add dynamic behavior to the webpage and helps in the validation process. All the input messages and keys were used here for computation of encrypted text. It's then sent to the receiver and will be decrypted only if the keys from the sender side matches to that of the receiver side. Even if the hacker tries to interfere it's impossible to get plaintext without knowing all the three keys.

(3) Cascading Style Sheets (CSS)

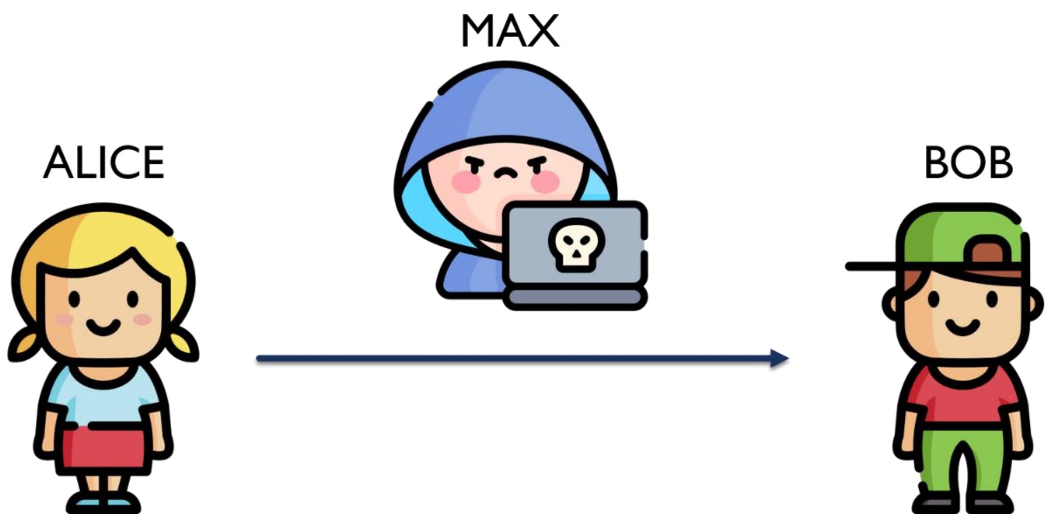
It is easier to make the web pages presentable using CSS. The application's user interfaces were developed using it. We can control various styles, such as the text color, the font style, the spacing among paragraphs, column size and layout, background color and images, design of the layout, display variations for distinct screens and device sizes, and many other effects as well.



Module 2: Understanding and partial implementation of Signal protocol

WhatsApp uses open-source Signal Protocol developed by Open Whisper Systems. Signal Protocol uses primitives like

- Double Ratchet Algorithm
- Prekeys
- Triple Diffie Hellman
- Curve25519
- AES
- HMAC_SHA256



In the further documentation we will be referring the **Sender** as **Alice**, the **Receiver** as **Bob** and **the Hacker as Max.**

Understanding Primitives

Prekeys: These are Curve25519 key pairs generated on device during install time. There is one Signed Prekey Pair and several one time prekey pairs. The Identity Public key and the Public keys of the prekey pairs are signed by a long term Identity Secret Key(Curve25519 private key correspond to Identity public key) and sent to the server during registration. Server stores these keys along with Identity Public key.

CURVE25519: Elliptic curve used as part of Diffie Hellman Key Exchange Protocol. Its security is based on difficulty of discrete logarithm problem in large finite groups.

HMAC_SHA256: It is keyed cryptographic hash function. Apart from the value to be hashed, this function also takes input a key. Unlike hash function which is easy to compute for a given input, keyed hash function requires knowledge of the key. Here it is used as a key derivation function and MAC.

Diffie Hellman: Diffie Hellman Key Agreement or Triple Diffie Hellman Handshake allows two parties to agree on a shared secret over public channel (Max is able to listen to message exchanged by Alice and Bob).

Extended Triple Diffie-Hellman:

The Extended Triple Diffie-Hellman (or X3DH) algorithm is used to establish the initial shared secret key between Alice and Bob, based on their public keys and using a server. Bob has already published some information on a server and Alice wants to establish a shared secret key with Bob in order to send him an encrypted message, **while Bob is not online.** Alice must therefore be able to perform the key exchange using simply the information stored on the server. The server can also be used to store messages by either of them until the other one can retrieve them.

Bob needs to generate several X25519 key pairs ahead of time:

- IK_b is Bob's long-term identity key. It is published to the server and is used to identify Bob.
- SPK_b is Bob's signed pre-key. The public key is published along with the signature Sig(IK_b, SPK_b) using Bob's identity key, therefore proving that Bob has access to the private key of IK_b. This key should be re-generated and updated by Bob every few weeks or months in order to provide better forward secrecy.
- OPK_b, OPK_b', OPK_b''… are Bob's one-time pre-keys. Each one's public key is published to the server and can be fetched by another party who wishes to communicate with Bob, after which it is deleted from the server. He can have as many as he wishes up to a limit defined by the server.

Similarly, Alice must generate and own the following:

- IK_a is Alice's long-term identity key. It is published to the server and is used to identify Alice.
- EK_a is Alice's ephemeral key which is generated simply for the upcoming DH with Bob's keys.

Alice then downloads a bundle from the server, which includes Bob's identity public key, signed public pre-key and its signature, and one of Bob's public pre-keys.

Then, she verifies the downloaded signature using IK_b. If the signature matches she can go ahead with establishing the secret. She calculates the following four secret outputs, using Diffie-Hellman:

DH1=DH(IK_a, SPK_b)

DH2=DH(EK_a, IK_b)

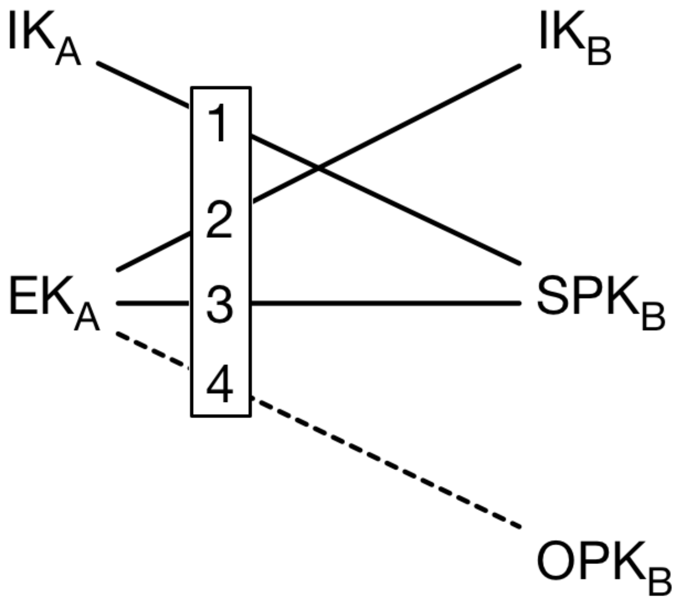
DH3=DH(EK_a, SPK_b)

DH4=DH(EK_a, OPK_b)

The first two of these secrets provide mutual authentication: The identity keys of both parties are used in them. Therefore if one of the parties tries to use a different identity key, they will arrive at a different result. The last two of these secrets provide forward secrecy as they are unique to this exchange.

By concatenating the four secrets and applying a KDF Alice arrives at the shared key that will be later used to initialize her ratchets.

SK = KDF(DH1 || DH2 || DH3 || DH4)



Afterwards, Alice sends Bob the public key of EK_a via the server, as well as her public identity key IK_a and the identifier of Bob’s one-time pre-key that she used (OPK_b). She also sends him the first encrypted message: IK_a || IK_b, which Bob will use to verify the identity keys of both parties.

Once Bob comes online, he will know one of his one-time pre-keys has been used by Alice to establish a new shared key. He will fetch IK_a and EK_a from the server. He must also know that IK_a belongs to the real Alice as well, as stated before. As he knows the private components of IK_b, SPK_b and OPK_b he can perform the same Diffie-Hellman calculations as Alice did using her public keys, and should therefore arrive at the same SK as Alice.

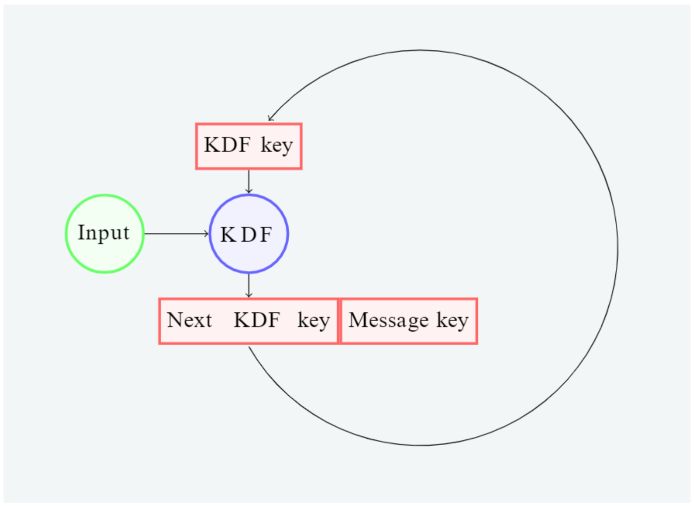
Double Ratchet:

Ratchet is the name of a device that moves only in one direction. Double Ratchet uses two cryptographic Ratchets, i.e., deriving new keys from current keys and moving forward, while forgetting old keys. The two ratchets used are Symmetric and Diffie Hellman Ratchet.

Symmetric Ratchet

The symmetric ratchet is the first ratchet type that we discussed before. The symmetric ratchet is implemented with a KDF chain using the HKDF algorithm, which ensures that the output will be cryptographically secure.

On the initial step, the KDF function is supplied with a secret key and some input data, which can be a constant. The output of the KDF is another secret key. This new key is split into two parts: The **next KDF key** and the **message key**. This is a turn of the "ratchet": The internal state of the ratchet (the KDF key) is changed and a new message key is created.



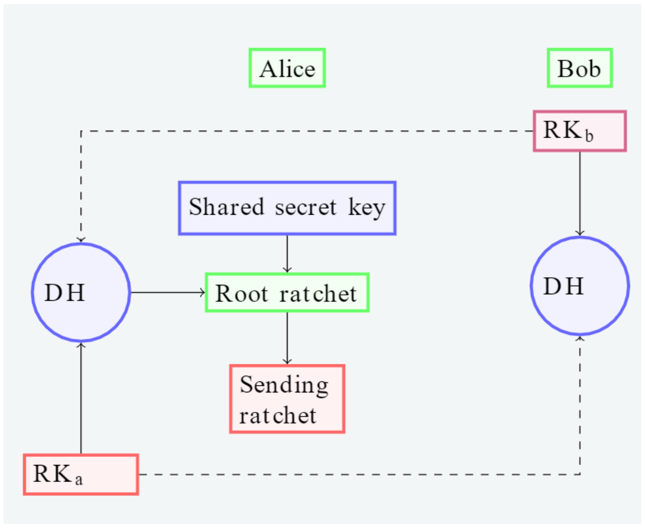
Because the output of the KDF algorithm is cryptographically secure, it's hard to reconstruct the input key of the KDF given the output key. This means that an attacker can't reconstruct older keys even if the current state and message key is leaked. They can however decrypt a single message by using the message key. In addition, by changing the input parameter on each step, we are also guaranteed break-in recovery: An attacker can't deduce the next state of the ratchet by only knowing the current state if they don't know what the input is. If the input is constant however an attacker can sync with the ratchet and decrypt all future messages.

Having performed the X3DH algorithm, both Alice and Bob have now arrived at a common shared secret key. That is now used to establish their session keys by using the Double Ratchet algorithm.

Diffie-Hellman Ratchet

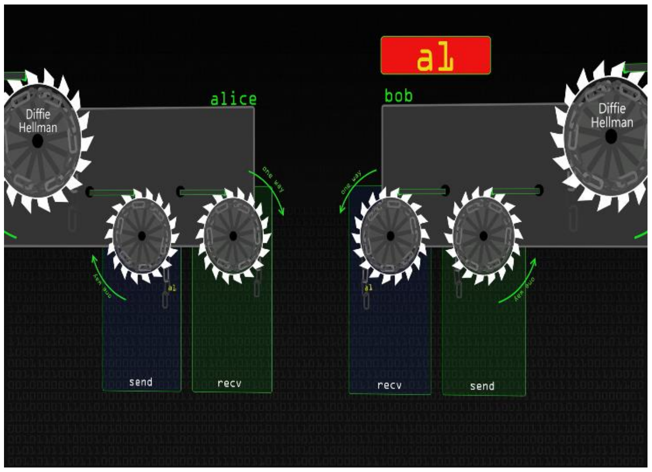
The second ratchet type is the Diffie-Hellman ratchet which is used to reset the keys used for the sending and receiving ratchets of both parties to new values.

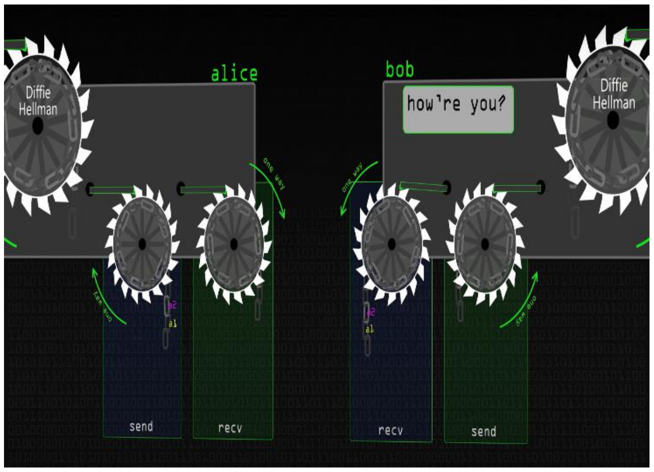
Before receiving a message from Alice, Bob must initialize a new ratchet key pair RK_b and advertise the public key to Alice. Upon learning Bob's public key, Alice will then generate her own key pair RK_a and calculate DH(RK_a, RK_b). This value will be used as the input to turn Alice's root symmetric ratchet once, yielding a new key. This key will then be used to initialize Alice's sending symmetric ratchet.



This process signifies a single turn of the DH ratchet, as in each step one party's key is renewed and the old one is forgotten. It can be performed as often as the two parties like in order to provide break-in recovery. In practice it's done with every single message.

With that, we can add the code for maintaining the DH ratchet by both Bob and Alice. We don't need a new construct for the DH ratchet, as it's sufficient to keep an X25519 key pair for each user.





4. Implementation

Software / Platform used

- Software – Visual Studio Code (Version 1.40 or higher)
- Operating System – Windows 10
- Browser – Chrome
- Python editor: Spyder

Code

HTML Code:

```
<!DOCTYPE html>

<html lang=”en”>

<head>

  <meta charset=”UTF-8”>

  <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>

  <link rel=”stylesheet” href=”style.css”>

  <title>Encryption with javascript</title>

</head>

<body>
```

```
  <div class=”container”>
    <div class=”phone phone-2”>
      <div class=”header header__phone1”>
        <div class=”back-btn phone-2”>
          <span class=”top-line”></span>
          <span class=”mid-line”></span>
          <span class=”bottom-line”></span>
        </div>
        <div class=”profile-photo phone-2”></div>
        <div>
          <div class=”profile-name phone-2”>Bob</div>
          <div class=”active-status phone-2”>online</div>
        </div>
      </div>
      <div class=”chats chats__phone1”>
      </div>
      <div class=”text-area text-area__phone1”>
        <input type=”textarea” placeholder=”Type Message Here” class=”message-
box”></input>
        <button class=”send-button”>Send</button>
      </div>
    </div>
    <div class=”encryptKey”>
      <h2>key</h2>
      <input id=”0” type=”number” value=”0” min=”0” max=”99”>
      <input id="1" type="number" value="0" min="0" max="99">
      <input id="2" type="number" value="0" min="0" max="99">
    </div>
    <div class="phone phone-2">
    <div class="header header__phone1">
      <div class="back-btn phone-2">
        <span class="top-line"></span>
        <span class="mid-line"></span>
        <span class="bottom-line"></span>
      </div>
      <div class="profile-photo phone-2"></div>
      <div>
        <div class="profile-name phone-2">Hacker</div>
        <div class="active-status phone-2">online</div>
      </div>
    </div>
    <div class="chats chats__phone2">
    </div>
    <div class="text-area text-area__phone1">
      <input type="textarea" placeholder="Type Message Here" class="message-
box"></input>
      <button class="send-button">Send</button>
    </div>
  </div>
  <div class="decryptKey">
```

```
    <h2>Key</h2>

    <input id=”0” type=”number” value=”0” min=”0” max=”99”>

    <input id=”1” type=”number” value=”0” min=”0” max=”99”>

    <input id=”2” type=”number” value=”0” min=”0” max=”99”>

  </div>

    <div class=”phone phone-2”>

      <div class=”header header__phone1”>

        <div class=”back-btn phone-2”>

          <span class=”top-line”></span>

          <span class=”mid-line”></span>

          <span class=”bottom-line”></span>

        </div>

        <div class=”profile-photo phone-2”></div>

        <div>

          <div class=”profile-name phone-2”>Ashley</div>

          <div class=”active-status phone-2”>online</div>

        </div>

      </div>

      <div class=”chats chats__phone3”>

      </div>

      <div class=”text-area text-area__phone1”>

        <input type=”textarea” placeholder=”Type Message Here” class=”message-
box”></input>

        <button class=”send-button”>Send</button>

      </div>
```

```
   </div>
  </div>
  <script src=”main.js”></script>
</body>
</html>
```

CSS Code:

```
* {
  padding: 0;
  margin: 0;
  box-sizing: border-box;
  color: white;
}
.container {
  height: 100vh;
  margin: auto;
  display: flex;
  justify-content: space-between;
}

.phone {
  display: flex;
  justify-content: space-between;
  height: 100vh;
  flex-direction: column;
  background-color: #f8f8ff;
  width: 28%;
}

.header {
  display: flex;
  align-items: center;
  height: 55px;
  width: 100%;
  background: #4e0dea;
  padding: 4px;
}

.back-btn {
  width: 35px;
  height: 100%;
  padding-top: 10px;
  cursor: pointer;
}
.back-btn span {
  display: block;
  height: 3px;
  width: 75%;
  background: #e6e8e7;
  margin: 5px;
```

```
}


span.top-line,
span.bottom-line {
  width: 45%;
  border-radius: 30px;
}
span.top-line {
  transform: rotate(-42deg);
  transform-origin: 10px 4px;
}


span.bottom-line {
  transform: rotate(42deg);
  transform-origin: 10px -1px;
}


.profile-photo {
  width: 41px;
  height: 90%;
  background: rgb(255, 255, 255);
  border-radius: 50%;
  cursor: pointer;
}
```

```
.profile-name {
  padding-left: 10px;
  font-size: 20px;
}
.active-status {
  padding-left: 5px;
  font-size: 11.5px;
}

.chats {
  flex: 1;
  overflow-y: scroll;
  background: linear-gradient(45deg, #b8b9f5, #234654);
}

.chat {
  padding: 8px 10px;
}
.chats::-webkit-scrollbar {
  width: 5px;
}

.chats::-webkit-scrollbar-thumb {
  background: blanchedalmond;
}
```

```
.chats::-webkit-scrollbar-track {
  background: #8e97ff;
}

.chats::-webkit-scrollbar-track:hover {
  background: #8e97ff;
}
.chat-sent {
  display: flex;
  margin-left: auto;
  max-width: 85%;
}
.chat-received {
  display: flex;
  margin-right: auto;
  max-width: 85%;
}

.message-sent {
  margin-left: auto;
  background: green;
  padding: 10px;
  border-radius: 20px;
}
.message-received {
  padding: 10px;
  border-radius: 20px;
  background: #a9a9af;
}
.text-area {
  display: flex;
  width: 100%;
  background: linear-gradient(45deg, #b8b9f5, #234654);
}

.text-area input {
  display: inline;
  padding-left: 20px;
  padding-right: 90px;
  outline: none;
  width: 100%;
  height: 45px;
  color: black;
  border-radius: 20px;
  border: none;
}
.send-button {
  width: 80px;
  border-radius: 20px;
  border: none;
  color: rgb(0, 0, 0);
}
.send-button:focus {
  border: none;
}
.encryptKey input::-webkit-scrollbar-thumb {
  display: none;
}

.encryptKey h2 {
  color: black;
}
.decryptKey h2 {
  color: black;
}
.encryptKey input {
  display: block;
  width: 65px;
  height: 35px;
  color: black;
  margin-top: 10px;
}
.decryptKey input {
  display: block;
  margin-top: 10px;
  width: 65px;
  height: 35px;
  color: black;
}
```

JavaScript Code:

```
const messagebox = document.querySelector(“.message-box”);
const encrypt_mssge = document.querySelector(“.encrypted_message”);
const encryptKey = document.querySelectorAll(“.encryptKey input”);
const decryptKey = document.querySelectorAll(“.decryptKey input”);
const phone1 = document.querySelector(“.chats__phone1”);
const phone2 = document.querySelector(“.chats__phone2”);
const phone3 = document.querySelector(“.chats__phone3”);

let letters = [“ “,
  “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”,

“z”,

“a”,

“b”,

“c”,

“d”,

“e”,

“f”,

“g”,

“h”,

```
  “€”,
  “j”,
  “k”,
  “l”,
  “m”,
  “n”,
  “o”,
  “p”,
  “q”,
  “r”,
  “s”,
  “t”,
  “u”,
  “v”,
  “w”,
  “x”,
  “y”,
  “z”,
]

let encryptionKey = [0,0,0];
let decryptionKey = [0,0,0];
let messageText = [];

let cnt = 0;
let cnt1 = 0;

encryptKey.forEach(input => {input.addEventListener('click', function() {
  encryptionKey[input.id] = input.valueAsNumber;
})});
encryptKey.forEach(input => {input.addEventListener('keyup', function() {
  encryptionKey[input.id] = input.valueAsNumber;
})});
decryptKey.forEach(input => {input.addEventListener('click', function() {
  decryptionKey[input.id] = input.valueAsNumber;
})});
decryptKey.forEach(input => {input.addEventListener('keyup', function() {
  decryptionKey[input.id] = input.valueAsNumber;
})});

const bobSend = mssge => {
  let BobCht = document.createElement("div");
  let BobMssg = document.createElement("div");
  BobCht.classList.add("Bob", "chat", "chat-sent");
  BobMssg.classList.add("Bob", "message-sent");
  BobMssg.innerText = mssge;
  BobCht.appendChild(BobMssg);
  phone1.appendChild(BobCht);
}
```

```
const hckrReceive = mssge => {
  let text = '';
   mssge.map(letter => {
    text += letter;
  });
  let hckrCht = document.createElement("div");
  let hckrMssg = document.createElement("div");
  hckrCht.classList.add("Bob", "chat", "chat-sent");
  hckrMssg.classList.add("Bob", "message-sent");
  hckrMssg.innerText = text;
  hckrCht.appendChild(hckrMssg);
  phone2.appendChild(hckrCht);
  console.log(text);
}

const AshlyReceive = mssge => {
  let AshCht = document.createElement("div");
  let AshMssg = document.createElement("div");
  AshCht.classList.add("Ashley","chat", "chat-received")
  AshMssg.classList.add("Ashley", "message-received");
  AshMssg.innerText = mssge;
  AshCht.appendChild(AshMssg);
  phone3.appendChild(AshCht);
}
```

```
function encrypt (messge = [], lock) {
  let encryptedText = [];
  let hold = 0;
  messge.map(letter => {
    hold = lock[cnt] + letters.indexOf(letter);
    while(hold > letters.length – 1){
      hold -= letters.length  - 1;
    }
    encryptedText.push(letters[hold]);
    cnt++;
    if(cnt === lock.length – 1){
      cnt = 0;
    }
  });
  return encryptedText;
}

function decrypt(messages = [], unlockKey){
  let chr = “”;
  let hold = 0;
  messages.map(message => {
    hold = letters.indexOf(message) – unlockKey[cnt1];
    console.log(letters.indexOf(message), unlockKey[cnt1]);
    while(hold < 0){
      hold += letters.length – 1;
      console.log(hold);
     }
   chr += letters[hold];
   cnt1++;
    if(cnt1 === unlockKey.length – 1){
      cnt1 = 0;
     }
  });
  return chr;
}
const send = (mssge) => {
  let enkeys = [];
  messageToSend = encrypt(messageText, encryptionKey);
  enkeys = decrypt(messageToSend, decryptionKey);
  bobSend(enkeys);
  hckrReceive(messageToSend);
  AshlyReceive(enkeys);
  messagebox.value = "";
}

messagebox.addEventListener('keydown', function€ {
  if(e.keyCode === 8){
    messageText.pop()
}
  else if(e.keyCode === 13){
    send(messageText);
    messageText = [];
  }
  else{
    messageText.push(e.key);
  }
});
```

Extended Triple Diffie Hellman & Double Ratchet Algorithm and encryption using AES Mode CBC (Cipher Blocker Chaining):

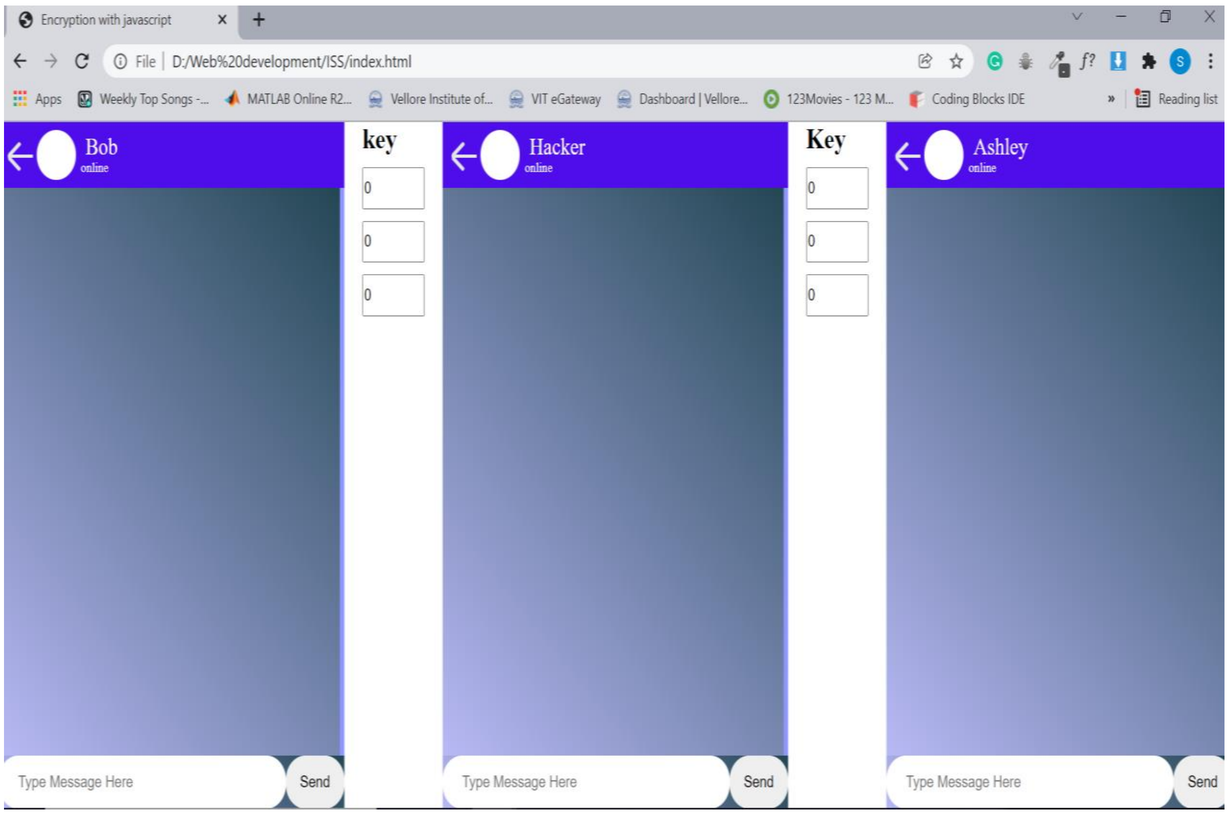
```
import base64
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric.x25519 import X25519PrivateKey
from cryptography.hazmat.primitives.asymmetric.ed25519 import \
        Ed25519PublicKey, Ed25519PrivateKey
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.backends import default_backend
from Crypto.Cipher import AES


def b64(msg):
    # base64 encoding helper function
    return base64.encodebytes(msg).decode('utf-8').strip()


def hkdf(inp, length):
    # use HKDF on an input to derive a key
    hkdf = HKDF(algorithm=hashes.SHA256(), length=length, salt=b'',
                info=b'', backend=default_backend())
    return hkdf.derive(inp)


class Bob(object):
    def __init__(self):
        # generate Bob's keys
        self.IKb = X25519PrivateKey.generate()
        self.SPKb = X25519PrivateKey.generate()
        self.OPKb = X25519PrivateKey.generate()

    def x3dh(self, alice):
        # perform the 4 Diffie Hellman exchanges (X3DH)
        dh1 = self.SPKb.exchange(alice.IKa.public_key())
        dh2 = self.IKb.exchange(alice.EKa.public_key())
        dh3 = self.SPKb.exchange(alice.EKa.public_key())
        dh4 = self.OPKb.exchange(alice.EKa.public_key())
        # the shared key is KDF(DH1||DH2||DH3||DH4)
        self.sk = hkdf(dh1 + dh2 + dh3 + dh4, 32)
        print('[Bob]\tShared key:', b64(self.sk))



class Alice(object):
    def __init__(self):
        # generate Alice's keys
        self.IKa = X25519PrivateKey.generate()
        self.EKa = X25519PrivateKey.generate()

    def x3dh(self, bob):
        # perform the 4 Diffie Hellman exchanges (X3DH)
        dh1 = self.IKa.exchange(bob.SPKb.public_key())
        dh2 = self.EKa.exchange(bob.IKb.public_key())
        dh3 = self.EKa.exchange(bob.SPKb.public_key())
        dh4 = self.EKa.exchange(bob.OPKb.public_key())
        # the shared key is KDF(DH1||DH2||DH3||DH4)
        self.sk = hkdf(dh1 + dh2 + dh3 + dh4, 32)
        print('[Alice]\tShared key:', b64(self.sk))
alice = Alice()
bob = Bob()
# Alice performs an X3DH while Bob is offline, using his uploaded keys
alice.x3dh(bob)
# Bob comes online and performs an X3DH using Alice's public keys
bob.x3dh(alice);
class SymmRatchet(object):
    def __init__(self, key):
        self.state = key
    def next(self, inp=b''):
        # turn the ratchet, changing the state and yielding a new key and IV
        output = hkdf(self.state + inp, 80)
        self.state = output[:32]
        outkey, iv = output[32:64], output[64:]
        return outkey, iv
class Bob(object):
    def __init__(self):
        # generate Bob's keys
        self.IKb = X25519PrivateKey.generate()
        self.SPKb = X25519PrivateKey.generate()
        self.OPKb = X25519PrivateKey.generate()
    def x3dh(self, alice):
        # perform the 4 Diffie Hellman exchanges (X3DH)
        dh1 = self.SPKb.exchange(alice.IKa.public_key())
        dh2 = self.IKb.exchange(alice.EKa.public_key())
        dh3 = self.SPKb.exchange(alice.EKa.public_key())
        dh4 = self.OPKb.exchange(alice.EKa.public_key())
        # the shared key is KDF(DH1||DH2||DH3||DH4)
        self.sk = hkdf(dh1 + dh2 + dh3 + dh4, 32)
        print('[Bob]\tShared key:', b64(self.sk))
    def init_ratchets(self):
        # initialise the root chain with the shared key
        self.root_ratchet = SymmRatchet(self.sk)
        # initialise the sending and recving chains
        self.recv_ratchet = SymmRatchet(self.root_ratchet.next()[0])
        self.send_ratchet = SymmRatchet(self.root_ratchet.next()[0])
class Alice(object):
    def __init__(self):
        # generate Alice's keys
        self.IKa = X25519PrivateKey.generate()
        self.EKa = X25519PrivateKey.generate()
 def x3dh(self, bob):
        # perform the 4 Diffie Hellman exchanges (X3DH)
        dh1 = self.IKa.exchange(bob.SPKb.public_key())
        dh2 = self.EKa.exchange(bob.IKb.public_key())
        dh3 = self.EKa.exchange(bob.SPKb.public_key())
        dh4 = self.EKa.exchange(bob.OPKb.public_key())
        # the shared key is KDF(DH1||DH2||DH3||DH4)
        self.sk = hkdf(dh1 + dh2 + dh3 + dh4, 32)
        print('[Alice]\tShared key:', b64(self.sk))
    def init_ratchets(self):
        # initialise the root chain with the shared key
        self.root_ratchet = SymmRatchet(self.sk)
        # initialise the sending and recving chains
        self.send_ratchet = SymmRatchet(self.root_ratchet.next()[0])
        self.recv_ratchet = SymmRatchet(self.root_ratchet.next()[0])


alice = Alice()

bob = Bob()

# Alice performs an X3DH while Bob is offline, using his uploaded keys

alice.x3dh(bob)

# Bob comes online and performs an X3DH using Alice's public keys

bob.x3dh(alice)

# Initialize their symmetric ratchets

alice.init_ratchets()

bob.init_ratchets()

# Print out the matching pairs

print('[Alice]\tsend ratchet:', list(map(b64, alice.send_ratchet.next())))

print('[Bob]\trecv ratchet:', list(map(b64, bob.recv_ratchet.next())))

print('[Alice]\trecv ratchet:', list(map(b64, alice.recv_ratchet.next())))

print('[Bob]\tsend ratchet:', list(map(b64, bob.send_ratchet.next())))


"""Diffie-Hellman Ratchet"""

class Bob(object):

    def __init__(self):

        # generate Bob's keys

        self.IKb = X25519PrivateKey.generate()

        self.SPKb = X25519PrivateKey.generate()
        self.OPKb = X25519PrivateKey.generate()


    def x3dh(self, alice):
        # perform the 4 Diffie Hellman exchanges (X3DH)
        dh1 = self.SPKb.exchange(alice.IKa.public_key())
        dh2 = self.IKb.exchange(alice.EKa.public_key())
        dh3 = self.SPKb.exchange(alice.EKa.public_key())
        dh4 = self.OPKb.exchange(alice.EKa.public_key())
        # the shared key is KDF(DH1||DH2||DH3||DH4)
        self.sk = hkdf(dh1 + dh2 + dh3 + dh4, 32)
        print('[Bob]\tShared key:', b64(self.sk))
        # initialise Bob's DH ratchet
        self.DHratchet = X25519PrivateKey.generate()


    def dh_ratchet(self, alice_public):
        # perform a DH ratchet rotation using Alice's public key
        dh_recv = self.DHratchet.exchange(alice_public)
        shared_recv = self.root_ratchet.next(dh_recv)[0]
        # use Alice's public and our old private key
        # to get a new recv ratchet
        self.recv_ratchet = SymmRatchet(shared_recv)
        print('[Bob]\tRecv ratchet seed:', b64(shared_recv))
        # generate a new key pair and send ratchet
        # our new public key will be sent with the next message to Alice
        self.DHratchet = X25519PrivateKey.generate()
        dh_send = self.DHratchet.exchange(alice_public)
        shared_send = self.root_ratchet.next(dh_send)[0]
        self.send_ratchet = SymmRatchet(shared_send)
        print('[Bob]\tSend ratchet seed:', b64(shared_send))
    def init_ratchets(self):
        # initialise the root chain with the shared key
        self.root_ratchet = SymmRatchet(self.sk)
        # initialise the sending and recving chains
        self.recv_ratchet = SymmRatchet(self.root_ratchet.next()[0])
        self.send_ratchet = SymmRatchet(self.root_ratchet.next()[0])


class Alice(object):
    def __init__(self):
        # generate Alice's keys
        self.IKa = X25519PrivateKey.generate()
        self.EKa = X25519PrivateKey.generate()


    def x3dh(self, bob):
        # perform the 4 Diffie Hellman exchanges (X3DH)
        dh1 = self.IKa.exchange(bob.SPKb.public_key())
        dh2 = self.EKa.exchange(bob.IKb.public_key())
        dh3 = self.EKa.exchange(bob.SPKb.public_key())
        dh4 = self.EKa.exchange(bob.OPKb.public_key())
        # the shared key is KDF(DH1||DH2||DH3||DH4)
        self.sk = hkdf(dh1 + dh2 + dh3 + dh4, 32)
        print('[Alice]\tShared key:', b64(self.sk))

        # Alice's DH ratchet starts out uninitialised
        self.DHratchet = None

    def dh_ratchet(self, bob_public):
        # perform a DH ratchet rotation using Bob's public key
        if self.DHratchet is not None:
            # the first time we don't have a DH ratchet yet
            dh_recv = self.DHratchet.exchange(bob_public)
            shared_recv = self.root_ratchet.next(dh_recv)[0]
            # use Bob's public and our old private key
            # to get a new recv ratchet
            self.recv_ratchet = SymmRatchet(shared_recv)
            print('[Alice]\tRecv ratchet seed:', b64(shared_recv))
        # generate a new key pair and send ratchet
        # our new public key will be sent with the next message to Bob
        self.DHratchet = X25519PrivateKey.generate()
        dh_send = self.DHratchet.exchange(bob_public)
        shared_send = self.root_ratchet.next(dh_send)[0]

        self.send_ratchet = SymmRatchet(shared_send)

        print('[Alice]\tSend ratchet seed:', b64(shared_send))


    def init_ratchets(self):

        # initialise the root chain with the shared key

        self.root_ratchet = SymmRatchet(self.sk)

        # initialise the sending and recving chains

        self.send_ratchet = SymmRatchet(self.root_ratchet.next()[0])

        self.recv_ratchet = SymmRatchet(self.root_ratchet.next()[0])

alice = Alice()

bob = Bob()

# Alice performs an X3DH while Bob is offline, using his uploaded keys

alice.x3dh(bob)

# Bob comes online and performs an X3DH using Alice's public keys

bob.x3dh(alice)

# Initialize their symmetric ratchets

alice.init_ratchets()

bob.init_ratchets()

# Initialise Alice's sending ratchet with Bob's public key

alice.dh_ratchet(bob.DHratchet.public_key())

def pad(msg):

    # pkcs7 padding
    num = 16 - (len(msg) % 16)

    return msg + bytes([num] * num)


def unpad(msg):

    # remove pkcs7 padding

    return msg[:-msg[-1]]

class Bob(object):

    def __init__(self):

        # generate Bob's keys

        self.IKb = X25519PrivateKey.generate()

        self.SPKb = X25519PrivateKey.generate()

        self.OPKb = X25519PrivateKey.generate()

    def x3dh(self, alice):

        # perform the 4 Diffie Hellman exchanges (X3DH)

        dh1 = self.SPKb.exchange(alice.IKa.public_key())

        dh2 = self.IKb.exchange(alice.EKa.public_key())

        dh3 = self.SPKb.exchange(alice.EKa.public_key())

        dh4 = self.OPKb.exchange(alice.EKa.public_key())

        # the shared key is KDF(DH1||DH2||DH3||DH4)

        self.sk = hkdf(dh1 + dh2 + dh3 + dh4, 32)

        print('[Bob]\tShared key:', b64(self.sk))

        # initialise Bob's DH ratchet

        self.DHratchet = X25519PrivateKey.generate()
```

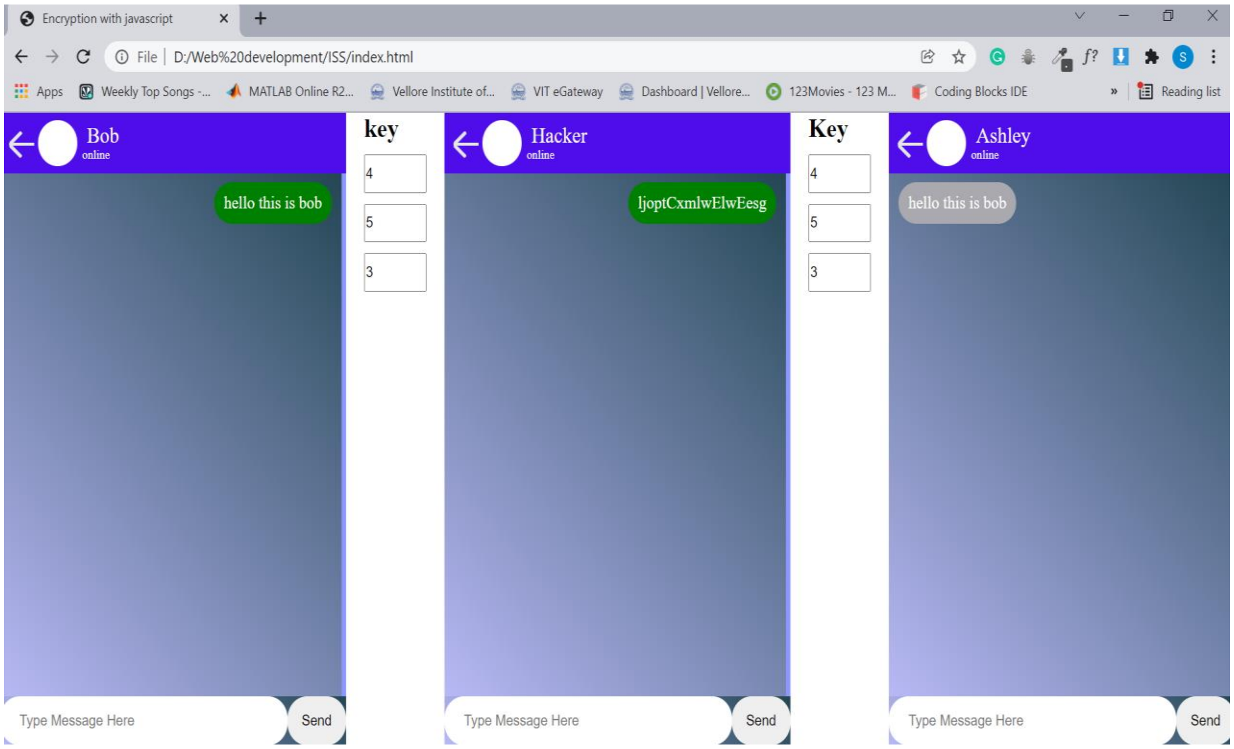
```
def dh_ratchet(self, alice_public):
        # perform a DH ratchet rotation using Alice's public key
        dh_recv = self.DHratchet.exchange(alice_public)
        shared_recv = self.root_ratchet.next(dh_recv)[0]
        # use Alice's public and our old private key
        # to get a new recv ratchet
        self.recv_ratchet = SymmRatchet(shared_recv)
        print('[Bob]\tRecv ratchet seed:', b64(shared_recv))
        # generate a new key pair and send ratchet
        # our new public key will be sent with the next message to Alice
        self.DHratchet = X25519PrivateKey.generate()
        dh_send = self.DHratchet.exchange(alice_public)
        shared_send = self.root_ratchet.next(dh_send)[0]
        self.send_ratchet = SymmRatchet(shared_send)
        print('[Bob]\tSend ratchet seed:', b64(shared_send))
  def init_ratchets(self):
        # initialise the root chain with the shared key
        self.root_ratchet = SymmRatchet(self.sk)
        # initialise the sending and recving chains
        self.recv_ratchet = SymmRatchet(self.root_ratchet.next()[0])
        self.send_ratchet = SymmRatchet(self.root_ratchet.next()[0])
   def send(self, alice, msg):
        key, iv = self.send_ratchet.next()
        cipher = AES.new(key, AES.MODE_CBC, iv).encrypt(pad(msg))
        print('[Bob]\tSending ciphertext to Alice:', b64(cipher))
        # send ciphertext and current DH public key
        alice.recv(cipher, self.DHratchet.public_key())
def recv(self, cipher, alice_public_key):
        # receive Alice's new public key and use it to perform a DH
        self.dh_ratchet(alice_public_key)
        key, iv = self.recv_ratchet.next()
        # decrypt the message using the new recv ratchet
        msg = unpad(AES.new(key, AES.MODE_CBC, iv).decrypt(cipher))
        print('[Bob]\tDecrypted message:', msg)
class Alice(object):
    def __init__(self):
         # generate Alice's keys
        self.IKa = X25519PrivateKey.generate()
        self.EKa = X25519PrivateKey.generate()
def x3dh(self, bob):
        # perform the 4 Diffie Hellman exchanges (X3DH)
        dh1 = self.IKa.exchange(bob.SPKb.public_key())
        dh2 = self.EKa.exchange(bob.IKb.public_key())
        dh3 = self.EKa.exchange(bob.SPKb.public_key())
        dh4 = self.EKa.exchange(bob.OPKb.public_key())
        # the shared key is KDF(DH1||DH2||DH3||DH4)
        self.sk = hkdf(dh1 + dh2 + dh3 + dh4, 32)
        print('[Alice]\tShared key:', b64(self.sk))

        # Alice's DH ratchet starts out uninitialised
        self.DHratchet = None
    def dh_ratchet(self, bob_public):
        # perform a DH ratchet rotation using Bob's public key
        if self.DHratchet is not None:
            # the first time we don't have a DH ratchet yet
            dh_recv = self.DHratchet.exchange(bob_public)
            shared_recv = self.root_ratchet.next(dh_recv)[0]
            # use Bob's public and our old private key
            # to get a new recv ratchet
            self.recv_ratchet = SymmRatchet(shared_recv)
            print('[Alice]\tRecv ratchet seed:', b64(shared_recv))
        # generate a new key pair and send ratchet
        # our new public key will be sent with the next message to Bob
        self.DHratchet = X25519PrivateKey.generate()
        dh_send = self.DHratchet.exchange(bob_public)
        shared_send = self.root_ratchet.next(dh_send)[0]
        self.send_ratchet = SymmRatchet(shared_send)
        print('[Alice]\tSend ratchet seed:', b64(shared_send))
     def init_ratchets(self):
        # initialise the root chain with the shared key
        self.root_ratchet = SymmRatchet(self.sk)
        # initialise the sending and recving chains
        self.send_ratchet = SymmRatchet(self.root_ratchet.next()[0])
        self.recv_ratchet = SymmRatchet(self.root_ratchet.next()[0])
    def send(self, bob, msg):
        key, iv = self.send_ratchet.next()
        cipher = AES.new(key, AES.MODE_CBC, iv).encrypt(pad(msg))
        print('[Alice]\tSending ciphertext to Bob:', b64(cipher))
        # send ciphertext and current DH public key
        bob.recv(cipher, self.DHratchet.public_key())
    def recv(self, cipher, bob_public_key):
        # receive Bob's new public key and use it to perform a DH
        self.dh_ratchet(bob_public_key)
        key, iv = self.recv_ratchet.next()
        # decrypt the message using the new recv ratchet
        msg = unpad(AES.new(key, AES.MODE_CBC, iv).decrypt(cipher))
        print('[Alice]\tDecrypted message:', msg)
alice = Alice()
bob = Bob()
at# Alice performs an X3DH while Bob is offline, using his uploaded keys
alice.x3dh(bob)
# Bob comes online and performs an X3DH using Alice's public keys
```

```
bob.x3dh(alice)

# Initialize their symmetric ratchets

alice.init_ratchets()

bob.init_ratchets()

# Initialise Alice's sending ratchet with Bob's public key

alice.dh_ratchet(bob.DHratchet.public_key())

# Alice sends Bob a message and her new DH ratchet public key

alice.send(bob, b'Hello Bob!')

# Bob uses that information to sync with Alice and send her a message

bob.send(alice, b'Hello to you too, Alice!')
```

5. Results

Result of simple encrypted chat using Ceaser Cipher Technique




Encryption with javascript
D:/Web%20development/ISS/index.html
Bob
online
key
Hacker
online
Key
Ashley
online
Type Message Here
Send

hello this is bob

4

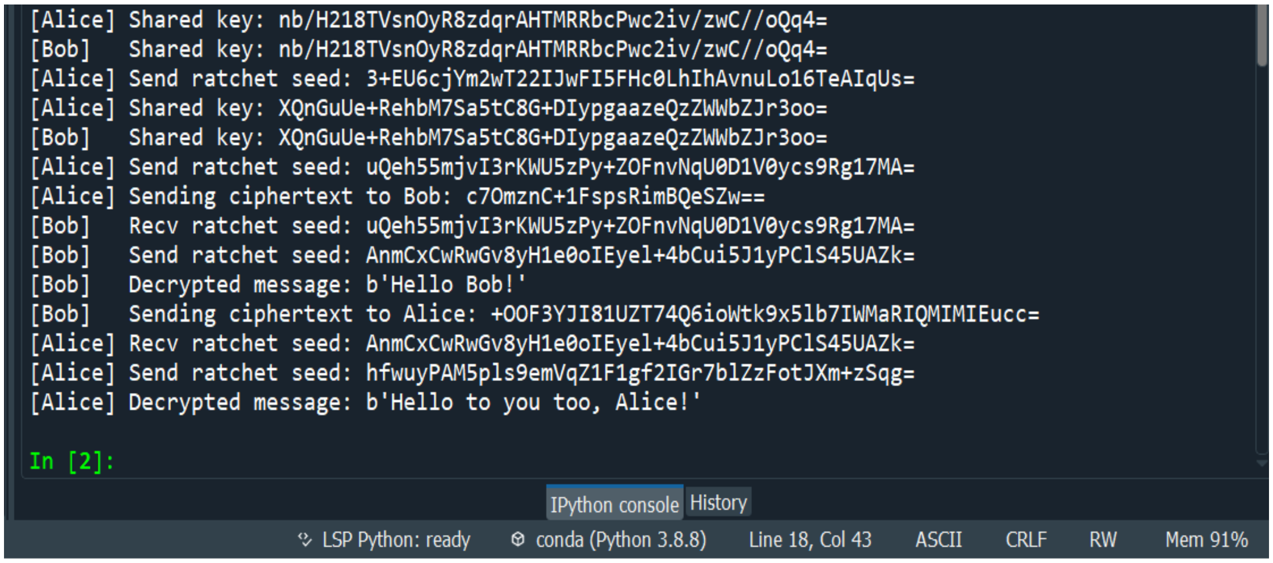
5

3

ljoptCxmlwElwEesg

hello this is bob

Result of Extended Triple Diffie Hellman , Double Ratchet and encryption using AES Mode CBC (Cipher Blocker Chaining):



```
[Alice] Shared key: nb/H218TVsnOyR8zdqrAHTMRRbcPwc2iv/zwC//oQq4=
[Bob]   Shared key: nb/H218TVsnOyR8zdqrAHTMRRbcPwc2iv/zwC//oQq4=
[Alice] Send ratchet seed: 3+EU6cjYm2wT22IJwFI5FHc0LhIhAvnuLo16TeAIqUs=
[Alice] Shared key: XQnGuUe+RehbM7Sa5tC8G+DIypgaazeQzZWWbZJr3oo=
[Bob]   Shared key: XQnGuUe+RehbM7Sa5tC8G+DIypgaazeQzZWWbZJr3oo=
[Alice] Send ratchet seed: uQeh55mjvI3rKWU5zPy+ZOFnvNqU0D1V0ycs9Rg17MA=
[Alice] Sending ciphertext to Bob: c7OmznC+1FspsRimBQeSZw==
[Bob]   Recv ratchet seed: uQeh55mjvI3rKWU5zPy+ZOFnvNqU0D1V0ycs9Rg17MA=
[Bob]   Send ratchet seed: AnmCxCwRwGv8yH1e0oIEyel+4bCui5J1yPClS45UAZk=
[Bob]   Decrypted message: b'Hello Bob!'
[Bob]   Sending ciphertext to Alice: +OOF3YJI81UZT74Q6ioWtk9x5lb7IWMaRIQMIMIEucc=
[Alice] Recv ratchet seed: AnmCxCwRwGv8yH1e0oIEyel+4bCui5J1yPClS45UAZk=
[Alice] Send ratchet seed: hfwuyPAM5pls9emVqZ1F1gf2IGr7blZzFotJXm+zSqg=
[Alice] Decrypted message: b'Hello to you too, Alice!'

In [2]:
```

6. Conclusion

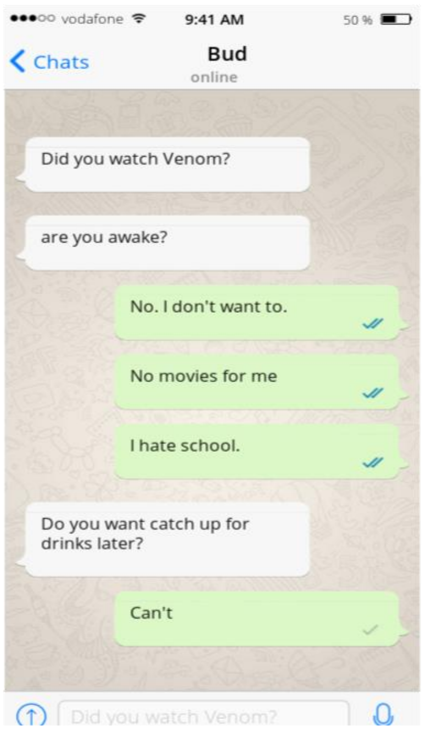
In the image, message in white boxes are ones received and green ones are the messages sent. Each pair of continuous boxes can help you visualize ratchet movements.

White -> White: Hashing Ratchet of Bob moves forward by 1 step (new message key and next chain keys are derived).

Green -> Green: Hashing Ratchet of Alice moves forward by 1 step (she derives new message key and chain key).

White -> Green: Alice's DH Ratchet moves forward by 1 step (she derives new root key and chain key)

Green -> White: Bob's DH Ratchet moves 1 step forwards (He derives new root key and chain key)



Did you watch Venom?

are you awake?

No. I don't want to.

No movies for me

I hate school.

Do you want catch up for drinks later?

Can't

Did you watch Venom?

This project was aimed at understanding End-to-End encryption. We achieved a basic understanding of this by implementing a simple encrypted chat web browser application. Then we tried to understand and implement a few concepts of the signal protocol used by WhatsApp. By implementing Extended triple Diffie Hellman and the Double ratchet protocol, we gained insight into the world of message encryption we use in our daily lives.

7. Team Members' Contribution

Team Leader (Reg.No. & Name): 19MIS0138 Laasya Yarlagadda

Register Number	Name	Contribution / Role in this Project
19MIS0137	Sabrina Manickam	4 research papers review, Research into the topic, Implementation (Understanding and execution of existing code)
19MIS0138	Laasya Yarlagadda	4 research papers review, Research into the topic, Implementation (Understanding and execution of existing code)
19MIS0186	James Joseph Thandapral	4 research papers review, Documentation
19MIS0212	Shambhavi Krishna Swarup	4 research papers review, Documentation, Implementation (Understanding and execution of existing code)
19MIS0235	Harikrishnan Jayakumar	4 research papers review, Documentation

References

[1] G. A. Pimenta Rodrigues, "Securing Instant Messages with Hardware-Based Cryptography and Authentication in Browser Extension," in IEEE Access, vol. 8, pp. 95137-95152, 2020

http://ieeexplore.ieee.org/document/9091122

[2] A. Aguado, V. Lopez, J. Martinez-Mateo, M. Peev, D. Lopez and V. Martin, "Virtual network function deployment and service automation to provide end-to-end quantum encryption," in IEEE/OSA Journal of Optical Communications and Networking, vol. 10, no. 4, pp. 421-430, April 2018

http://ieeexplore.ieee.org/document/8336694

[3] O. A. L. A. Ridha, G. N. Jawad and S. F. Kadhim, "Modified Blind Source Separation for Securing End-to-End Mobile Voice Calls," in IEEE Communications Letters, vol. 22, no. 10, pp. 2072-2075, Oct. 2018

http://ieeexplore.ieee.org/document/8428646

[4] Y. Kim, V. Kolesnikov and M. Thottan, "Resilient End-to-End Message Protection for Cyber-Physical System Communications," in IEEE Transactions on Smart Grid, vol. 9, no. 4, pp. 2478-2487, July 2018

http://ieeexplore.ieee.org/document/7576699

[5] A. Senf, "End-to-End Security for Local and Remote Human Genetic Data Applications at the EGA," in IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 16, no. 4, pp. 1324-1327, 1 July-Aug. 2019

http://ieeexplore.ieee.org/document/8713924

[6] J. Yeh, S. Sridhar, G. G. Dagher, H. Sun, N. Shen and K. D. White, "A Certificateless One-Way Group Key Agreement Protocol for End-to-End Email Encryption," 2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC), 2018, pp. 34-43, doi: 10.1109/PRDC.2018.00014.

https://ieeexplore.ieee.org/document/8639558

[7] K. A. Kumari, G. S. Sadasivam, S. S. Gowri, S. A. Akash and E. G. Radhika, "An Approach for End-to-End (E2E) Security of 5G Applications," 2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS), 2018, pp. 133-138, doi: 10.1109/BDS/HPSC/IDS18.2018.00038.

https://ieeexplore.ieee.org/document/8552296

[8] F. S. Chowdhury, A. Istiaque, A. Mahmud and M. Miskat, "An implementation of a lightweight end-to-end secured communication system for patient monitoring system," 2018 Emerging Trends in Electronic Devices and Computational Techniques (EDCT), 2018, pp. 1-5, doi: 10.1109/EDCT.2018.8405076.

https://ieeexplore.ieee.org/document/8405076

[9] Y. Sun, F. P. . -W. Lo and B. Lo, "Light-weight Internet-of-Things Device Authentication, Encryption and Key Distribution using End-to-End Neural Cryptosystems," in IEEE Internet of Things Journal, doi: 10.1109/JIOT.2021.3067036.

https://ieeexplore.ieee.org/document/9381407

[10] T. Isobe and R. Ito, "Security Analysis of End-to-End Encryption for Zoom Meetings," in IEEE Access, vol. 9, pp. 90677-90689, 2021, doi: 10.1109/ACCESS.2021.3091722.

https://ieeexplore.ieee.org/document/9462825

[11] J. Bobrysheva and S. Zapechnikov, "Post-quantum Secure Group Messaging," 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus), 2021, pp. 2323-2326, doi: 10.1109/ElConRus51938.2021.9396513.

http://ieeexplore.ieee.org.egateway.vit.ac.in/document/9396513

[12] F. Schillinger and C. Schindelhauer, "Partitioned Private User Storages in End-to-End Encrypted Online Social Networks," 2020 15th International Conference for Internet Technology and Secured Transactions (ICITST), 2020, pp. 1-8, doi: 10.23919/ICITST51030.2020.9351335.

http://ieeexplore.ieee.org.egateway.vit.ac.in/document/9351335

[13] H. Jayakrishnan and R. Murali, "A Simple and Robust End-to-End Encryption Architecture for Anonymous and Secure Whistleblowing," 2019 Twelfth International Conference on Contemporary Computing (IC3), 2019, pp. 1-6, doi: 10.1109/IC3.2019.8844917.

http://ieeexplore.ieee.org.egateway.vit.ac.in/document/8844917

[14] T. Melo, A. Barros, M. Antunes and L. Frazão, "An end-to-end cryptography based real-time chat," 2021 16th Iberian Conference on Information Systems and Technologies (CISTI), 2021, pp. 1-6, doi: 10.23919/CISTI52073.2021.9476399.

http://ieeexplore.ieee.org.egateway.vit.ac.in/document/9476399

[15] P. Liu, "Public-Key Encryption Secure Against Related Randomness Attacks for Improved End-to-End Security of Cloud/Edge Computing," in IEEE Access, vol. 8, pp. 16750-16759, 2020, doi: 10.1109/ACCESS.2020.2967457.

http://ieeexplore.ieee.org.egateway.vit.ac.in/document/8961999

[16] S. Ariffi, R. Mahmod, R. Rahmat and N. A. Idris, "SMS Encryption Using 3D-AES Block Cipher on Android Message Application," 2013 International Conference on Advanced Computer Science Applications and Technologies, 2013, pp. 310-314, doi: 10.1109/ACSAT.2013.68.

https://ieeexplore.ieee.org/document/6836597

[17] U. Indriani, H. Gunawan, A. Yugo Nugroho Harahap and H. Zaharani, "Chat Message Security Enhancement on WLAN Network Using Hill Cipher Method," 2020 8th International Conference on Cyber and IT Service Management (CITSM), 2020, pp. 1-5, doi: 10.1109/CITSM50537.2020.9268838.

https://ieeexplore.ieee.org/abstract/document/9268838

[18] P. Rösler, C. Mainka and J. Schwenk, "More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema," 2018 IEEE European Symposium on Security and Privacy (EuroS&P), 2018, pp. 415-429, doi: 10.1109/EuroSP.2018.00036.

https://ieeexplore.ieee.org/document/8406614

[19] Hong-Bao Qin and Xin Xu, "Solution to secure Instant Messaging based on hardware encryption," 2015 IEEE 16th International Conference on Communication Technology (ICCT), 2015, pp. 844-847, doi: 10.1109/ICCT.2015.7399959.

https://ieeexplore.ieee.org/document/7399959

[20] M. Korczyński and A. Duda, "Classifying service flows in the encrypted skype traffic," 2012 IEEE International Conference on Communications (ICC), 2012, pp. 1064-1068, doi: 10.1109/ICC.2012.6364024.

https://ieeexplore.ieee.org/document/6364024