Super Stealth Chat

Nathan Tiegiser Jonas Metzger

***Abstract*—Security is an ever-growing need in today's society. There are many malicious entities that seek to steal or cause other mischief. This type of behavior is especially apparent in applications that have chatting features. But, many existing chatting applications have security issues. For example, many applications do not perform end-to-end encryption for messages sent between users. Also, some applications store user data and users messages in databases which could be leaked. Super Stealth Chat is a secure chatting application that seeks to deal with these security issues. First, Super Stealth Chat takes a decentralized approach in its architecture. There is not central server that is always running that routes communication. This is simply done by the host computer of the chat room. Because of this, there is also no database in which data is stored whether it be user data or messages. Super Stealth Chat also makes use of both secret key cryptography (for user verification and initial data exchange) and public key cryptography (for messages between users). Even with the improvements, there are still many limitations that could be fixed by further implementation if given enough time.**

I. Introduction

A. Importance

The importance of computer security in communication based applications (or chatting applications) cannot be understated. Just like in the physical world, many people wish to have secure lines of communication over the internet. This importance can be easily understand by observing the CIA triad. Communication must first be confidential. For example, individuals discussing business opportunities or other private matters may want to keep over from seeing their communication. Second, the integrity of communication must be upheld. For instance, a malicious entity must not be able to alter communication between two parties. Lastly, communication must be available always (if not, most of the time). For example, breaking changes and other bugs must not cause a centralized server that communication relies on to have extremely long down times or other blocks. Other important factors that are less related to computer security include application speed, user friendliness, and developer friendliness. The best applications are quick, easy to use, and easily understood. Before building a new secure chatting application, we must first use these factors see where other went wrong, and where this new application can improve upon.

B. Previous Implementation

The original idea of our project was called vault talk and the major points of it's operation were:

- Use socket programming to connect the computers.
- The room would have a password. With this password and other common elements that each user would have, a encryption key would be made, which solves distribution of the key (encryption method had not been decided). Many other chatting service might not even require a password to get into the chat group, just the join link.
- There would be no server involved with messaging, only the users computers. This differs from most other chatting services because they mostly use servers to handle messaging. This had three main effects:
  - First was joining the room, which we planned to solve using a server to handle connecting people to a room, after which the server is not involved in the messaging thus not compromising security. This is somewhat similar to how other chatting software deal with this.
  - Second was distribution of messages. Most other chatting software sends the message to the server and the server handles getting the message to everyone (see figure 1) but for ours we were going to have each of the users computers send the message to each other computer in the room (see figure 2).
  - Third there would be no way to save messages after the chat ended, however for our program not saving messages was a feature so this needed no action. Most chatting software saves messages on their servers for various reasons, mainly around convenience for normal chatting but this is not very secure.
- There would be no accounts unlike most other chatting services that require you to make an account to use the service but rather there would be one time usernames entered when you make/join the room.
- To make/enter a room, a user would enter their username, the room password and if joining a room IP of the host user.

C. Reason for name

Our previous name Vault Talk was not very good as it was not very intuitive, it was not super clear what the app was based on the name alone. So we decided that a much better name was Super Stealth Chat which clearly gets the point across, it's a chatting app that is super stealthy.

D. Contributions

Before diving into the details behind the main security features of Super Stealth Chat, it is important to give an overview of the contributions is as made to field of chatting applications. First, unlike some applications, the messages in this application are end-to-end encrypted. Many other applications rely on single keys for encrypting messages, however,

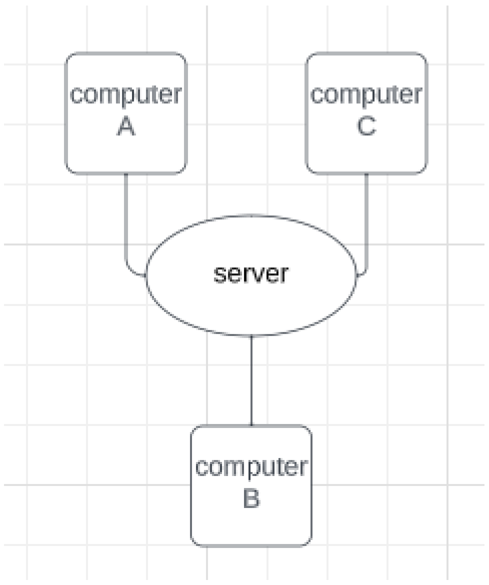

Fig. 1. Example of server communication.

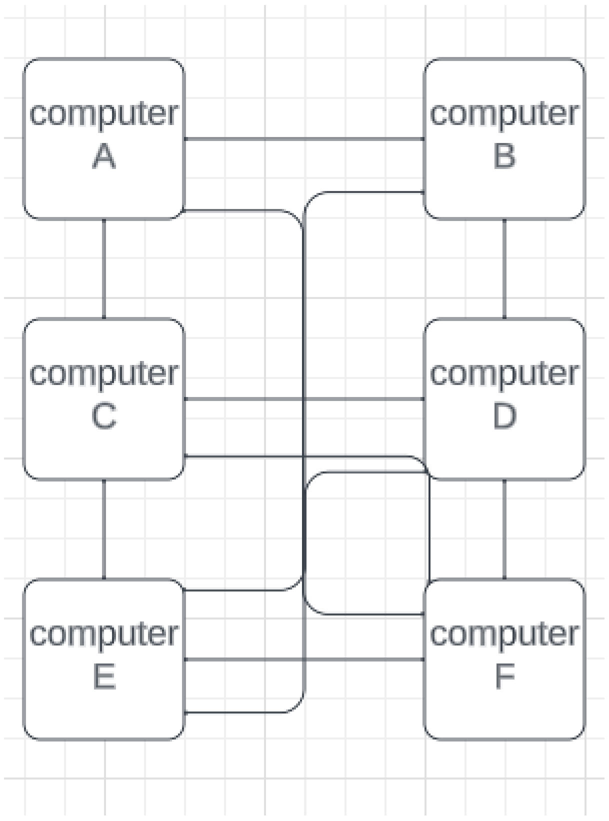

Fig. 2. Example of computers handling their own messages.

this application makes use of public key cryptography via RSA for encrypting messages. On top of that, AES is used this application for user verification (and initial data exchanges). Therefore, Super Stealth Chat making use of multiple encryption algorithms which makes it more diverse and more secure. Most applications with communication rely on central servers that are always running, however, this application does not. It simply relies on users computers and hosted proxies. Lastly, most applications save user data and messages which could potentially be viewed by anybody. This does seem like the most secure option, so this application eliminated saving any user and messages. Due to its importance, understanding the ways that Super Stealth Chat meets the CIA triad should be discussed. First and most obviously, all chat room messages are encrypted, so the confidentiality of the application holds. The next most obvious one would be availability. This application does not rely on central server that could potentially go down. So, the chat rooms should always be available unless a host ends the room. And lastly, the integrity of the chat messages is generally upheld because only users who know passwords and room URLs are allowed into the chat rooms. There is another feature that should be implemented to uphold integrity and confidentiality to a greater extent, but was simply overlooked. This feature will be discussed in a later section.

II. Technical Details

A. Basics

Before going into specific details on the features of our application, there are a couple of implementation-level details (that are not necessarily related to computer security) that we will go over. First, to build this application we utilized the Python programming language. Python is a language that everyone in the group understands, it provides some useful cryptography libraries, and it would also allow us to easily implement a graphic user interface (GUI). For the GUI we utilized Python's Tkinter library. Although the library is slightly limited, it gives us all the tools we need to create the GUI.

B. Communication Over Internet

The original plan to use socket programming had many issues but the one that cause the most issue was talking over different domains. Without having to make the users do things like setup port forwarding on their own routers or generally doing a bunch of crazy networking tricks which alone would be a project, there would be no way to using socket programming to commutation over the internet, it would work on local domains just fine. Figure 3 helps to show the issue, in domain A computer A and computer B can see each other clearly, so in the case socket programming is possible and not that bad, just make a socket connecting to the IP of the computer that you want to connect to. However there is an issue when it comes to trying to connect to computer C. Because it is on a different domain to connect computer A and computer C you would need to setup port forwarding in domain B to allow you to connect to computer C. All of this meaning that you could only talk over a local network, which is not very useful in the grand scheme of things.

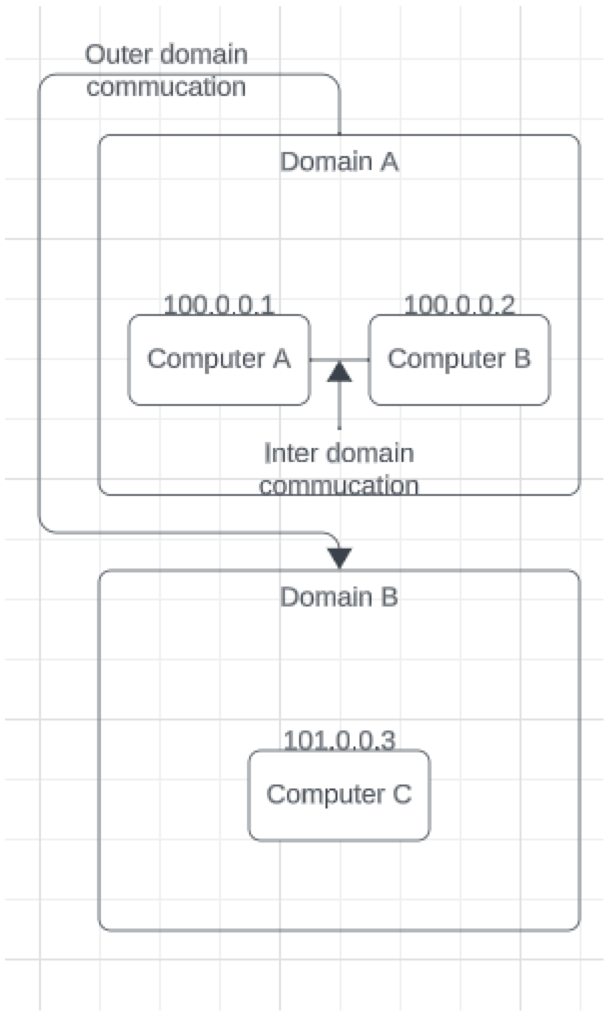

Fig. 3. Example showing the domain issue with socket programming.

To get around this issue, we used a service called Ngrok. Ngrok allows users to expose a port via a ”reverse proxy” (it is not really a reverse proxy) so that other computers can access it over the internet securely with https protocol. For use in this application, each user and the host of the chat room will have their own Ngrok proxy. The users send requests to the hosts Ngrok proxy, and the host sends requests to each of the users proxies. Figure 4 shows the idea of how nrgok works To implement Ngrok into our application, we used the Pyngrok library.

C. User Verification

The original plan was to use the password to the room and other common elements to make an encryption key. This was not used to encrypt messages as a different method was chosen for that but rather is use for user verification. The process of a user joining/making a room is:

- When a user host a room they set a password to the room, then the computer runs that password though 480,000 rounds of the SHA256 hashing algorithm making a room key of size 32. There is no salt applied to this hashing, this is a point of improvement.
- When a user tries to join a room the room password that they enter is run though the same process to get a key. Using this key the user will use AES to encrypt their information and then send it to the host.
- After the host receivers the join message they will then decrypt the received message using AES with the host’s key. If the password that the user trying to join entered was correct the decryption will be successful and output will be valid (see fig 5), which the host will store. If the password was wrong then the decryption will fail and will result in throwing an error and not letting the user into the room (see fig 6).
- If the user was allowed into the room then the host will reply to the join request with all of the needed information about the users in the room.

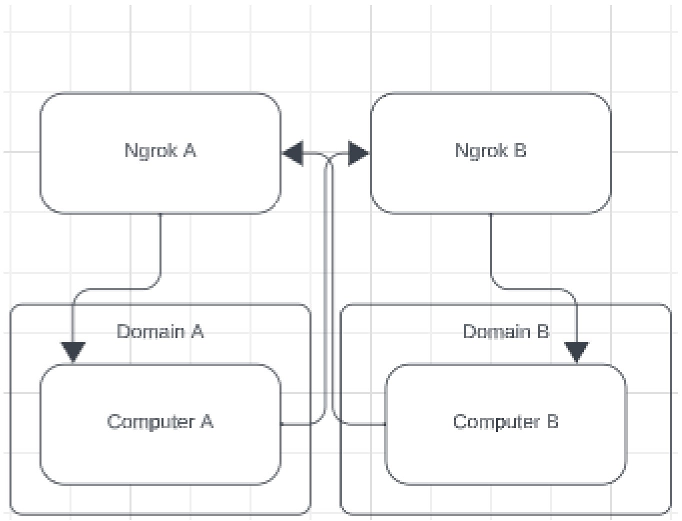

Fig. 4. Showing the idea of how ngrok works.

The error messages for failing to join a room are all the same weather it be because if wrong password or wrong URL. This could be seen as a good or bad thing. On one hand it makes attacks just that much harder, on the other when you can’t get into a room for some reason it’s hard to trouble shoot.

D. Chat Room Encryption

In place of the original plan of using one common key we decided to use the public key method RSA for encryption. When the host makes the room they will generate a random RSA key pair for the host user. The other users will make a random RSA key pair when they are trying to join the room and sends their public key to the host in the first message to the host. In the message that host replies with when a user is allowed into the room part of the information in that reply is the public keys to all of the users in the room. This reply is encrypted using the new users public key which brings up the size limited of RSA.

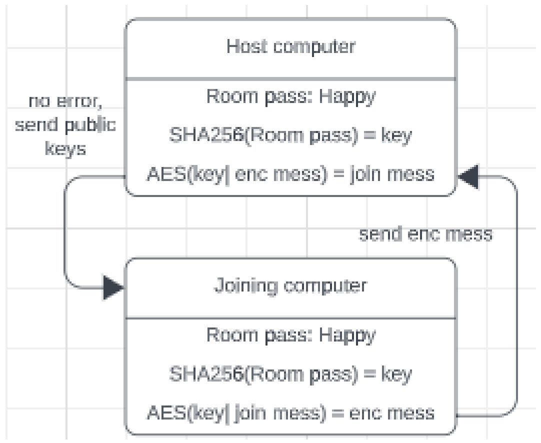

Fig. 5. In this case the user used the correct password so when the host decrypts the message valid json is the result which allows the user into the room.

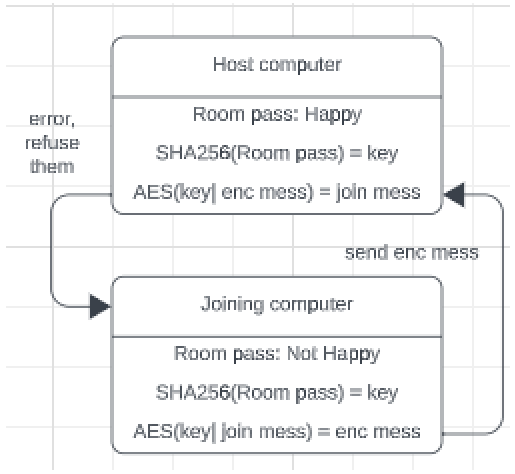

Fig. 6. In this case the user used the incorrect password so when the host decrypts the message invalid json is the result which does not allow the user into the room.

The size limit for RSA encrypting is around 256 letters which is to small to hold all of the information about all the users. To fix this the reply message split into chunks of 200 letters (number was semi randomly selected) and each of those are encoded using the RSA key which results in a 344 letter long string. All of these chunks are concatenated together and that string is what is sent to the user. Then on the user end the string is split into chunks of 344 letters and each of those are decrypted and concatenated together. This method is sort of required for providing a user a list of all the current public keys when there are more than two users in the room. But, as of now, this method is not implemented for normal messages as most messages are not really that big. However, instead of just enforcing a maximum length for messages, this method could be used in it's place which would be more user friendly.

E. Message Distribution

The original plan was for each computer to handle their own message (see figure 2) however this proved to be a bad idea for many reasons, mainly it would make the code very hard to deal with. So to make things simpler we decide to change the plan to be instead of that we would have the host's computer act like the server and distribute the messages. The main downside to this is that now the host can't just leave and the chat keep going.

There are two different cases that can happen when a user sends a message depending if they are the host or not:

- Case 1 which is demonstrated by figure 7 and this is when the host sends a message. The host will encrypt the message using each of the users public keys and then send each of the encoded messages to the corresponding user.
- Case 2 which is demonstrated by figure 8 and this is when a user sends a message (user A in the case of figure 8). The user will encrypt the message using each of the other users public keys (including the host) and then will make a json object that labels which user each of the encoded messages are for and then sends that as a json string to the host. Then when the host gets that message it will send each of the encoded messages to it's corresponding user holding the one for the host. This process reveals the usernames of all of the chatters, a point of improvement. When each user and host receivers their message they decode it with their private key and then display the message.

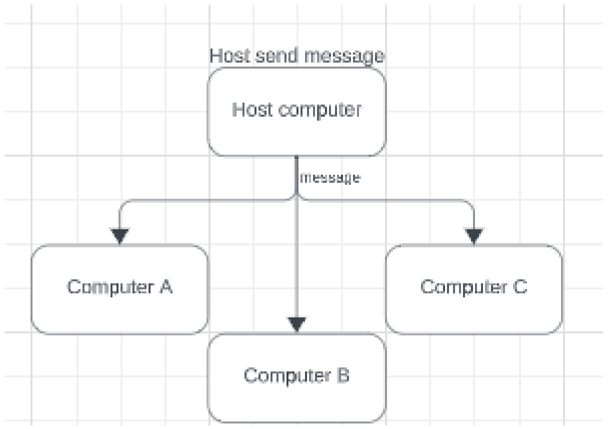

Fig. 7. The host sending a message.

F. Images

As we were finishing up work on Super Stealth Chat, we were asked to also implement the ability to send images. And whether you are encrypting messages, images, or other data, it really all works in the same way. There was a problem that we did run into with this. The messages in the chat are all encrypted using the RSA algorithm, and so we decided to just do the same thing for images. Because of this decision, images appear to be very delayed. They will not show up on the another user in the chat room's screen for several seconds. This delay is even large when using larger images. This problem

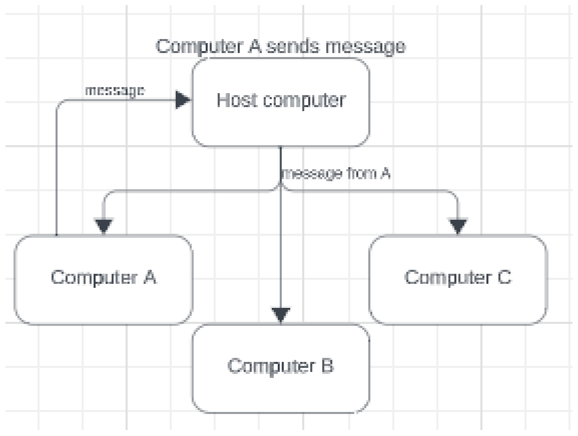

Fig. 8. User A sending a message.

is due to limitations with RSA, and this will be discussed in a later section.

G. Additional Security Aspects

Aside from the major security features like user verification and encryption, there are a couple other notable security aspects that should be presented. The first is the decentralized structure of the chat rooms. As previously stated, this application does not rely on a central server to sit in between users and direct communication. Yes, the chat room host is still the central entity that directs communication, however, the host only exists while the chat room is open, and most chat rooms won't exists for a very long time. So, malicious users who may want to disrupt communication or intercept messages will have a difficult time picking out the host to attack. Going along with the decentralized, no-server structure, the application also does not have a centralized place to store data. Whether, its user data (which only consists of usernames and public keys) or chat room data, none of it is stored in any database. This means that once a chat room is closed, all the chat room data is gone without a trace.

III. Conclusion

A. Recap

Before discussing some limitations and other concerns, here is a small recap. Super Stealth Chat is a secure, account-less, decentralized chatting application. It does contain and central server to direct communication, and there is not central database for saving any user data or chat room messages. This application also makes of multiple encryption algorithms including RSA for encrypting chat room messages and AES for the encryption of initial data and user verification.

B. Limitations

Although many ideas were implemented into Super Stealth Chat and work pretty well, there are some programming limitations and other limitations that need to be discussed. For programming limitations, the first to be examined is Python. Python is decent for building just about anything. However, when it comes to GUIs and applications with real-time-data transfer, the slowness of Python starts to become apparent. Another programming limitation that is actually not great because it is a huge part of schema. This limitation is the RSA encryption algorithm. RSA is really good for encrypting small amounts of data at a time. However, as chatting messages get long, this starts to become a problem as you have to start encrypting in chunks. And when it comes to encrypting images with RSA, it gets even worse. Images contain lots of data and you have to start encrypting lots of chunks. All of this is mostly due to the fact that RSA is kind of slow. The last programming limitation would be the requirement of a ”reverse proxy” for each user in a chat room. Ideally, any computer could just connect to any other computer that would be good and it would make implementing communication a lot easier. However, the technology just does not work that way. The port must be exposed to the internet via some method. Also, Ngrok is service and it is not entirely free. So, developing our own method of creating these proxies could be worth trying. The other limitations are not programming related but human related. The first is time. We did not have the same time frame as say a group building a real-world chatting application. With more time, we would be able to plan a little more and rush a little less. Another limitation is manpower. We did only have two people building this application. The last identified limitation is building this application with multiple people coding at the same time. Generally, two coders and work on separate parts of an application at the same time, however, every part of this application is so interrelated that doing this was not really possible. So, we just ended up having only one person code at any given time.

C. Future Implementation

While many of the core details were implemented mainly the security things there is still lot of extra things that could be implemented.

- The last security issue there is when a user sends a message to the host to be distributed, the usernames of the people in the chat are revealed, this could be fixed.
- Another security feature that is not necessarily an issue but additional feature, would be message signatures and verification. When a user sends a message, they could provide a digital signature which could be verified by the host. Therefore, if the host finds an unverified signature, they can kick that user out of the room. This is a feature that would uphold the integrity of chatting much more.
- Users can't leave the room in a ”normal” fashion, they can force close the program but there is no functionality behind leaving the room.
- Much of the GUI is still not very refined and could stand for major improvement.
- Many of the normal shortcut keys are not implemented, mainly enter for things like sending messages.
- Another programming language that is quicker than Python and provides better tools for building GUIs could be used.

- RSA has a limit on the length of plain texts that it can encrypt/decrypt and it is a little slow. Using another public key cryptography algorithm could help decrease delays (cause by the time it takes to encrypt and decrypt) that are seen when sending or receiving messages, especially images.
- Currently, the ”reverse proxy” required for communication is only supported with Ngrok. Other ways of hosting this proxy or just the ability to forward port could be implemented.

D. Additional Security Concerns

Through the development of Super Stealth Chat, we realized that there are a couple of concerns related to the security of the application. First, since there is no centralized server, one may argue that the security is left in the hands of the users. And while this is partially true, most of the security still lies in the encryption of messages, the secrecy of the password, and the secrecy of the room url. On top of this, there is no saved data so an attacker would have to do whatever they are trying to do while the room is still open. This leads into the other major concern and that is saved data. Some may argue that this application could be used to communicate about malicious things without anyone knowing. Without saved data or logs about the existence of communication, there is no tell who is communicating and what they are communicating about. And while this is a legitimate concern, it is not the problem that Super Stealth Chat seeks to fix. Also, malicious communication happens all the time physical without anyone knowing, so if someone did not have application to do this, they could just do it in the real world.

REFERENCES

[1] N. Unger, “End-to-End Encrypted Group Messaging with Insider Security.” Available: https://uwspace.uwaterloo.ca/bitstream/handle/10012/17196/UngerNik.pdf sequence=3isAllowed=y

[2] S. Oesch et al., “User Perceptions of Security and Privacy for Group Chat,” Digital Threats: Research and Practice, Oct. 2021, doi: https://doi.org/10.1145/3491265.

[3] “More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema,” ieeexplore.ieee.org. https://ieeexplore.ieee.org/document/8406614

[4] P. Pansara, R. Patel, K. Shah, R. Jhaveri, and V. Parmar, “Chat Application Security: Implementing Blockchain-based End-to-End Encryption,” IEEE Xplore, Mar. 01, 2023. https://ieeexplore.ieee.org/document/10112278

[5] “A Secure End-to-End Mobile Chat Scheme,” ieeexplore.ieee.org. https://ieeexplore.ieee.org/document/7016118

[6] R. E. Endeley, “End-to-End Encryption in Messaging Services and National Security—Case of WhatsApp Messenger,” Journal of Information Security, vol. 09, no. 01, pp. 95–99, 2018, doi: https://doi.org/10.4236/jis.2018.91008.

[7] “End-to-End Encryption for Chat App with Dynamic Encryption Key,” ieeexplore.ieee.org. https://ieeexplore.ieee.org/document/9725597

[8] “OFF-THE-RECORD SECURE CHAT ROOM,” Proceedings of the Fourth International Conference on Web Information Systems and Technologies, 2008, doi: https://doi.org/10.5220/0001530500540061.

[9] “Discord Server Forensics: Analysis and Extraction of Digital Evidence,” ieeexplore.ieee.org. https://ieeexplore.ieee.org/document/9432654 (accessed Sep. 24, 2023).

[10] R. Yang, W. Lau, and T. Liu, “Signing into One Billion Mobile App Accounts Effortlessly with OAuth2.0.” Available: https://oauth.ie.cuhk.edu.hk/static/papers/eurobh16.pdf

[11] ”Data Integrity: Identifying and Protecting Assets Against Ransomware and Other Destructive Events” Available: nccoe.nist.gov/publication/1800-25/VolA/index.html

[12] K. Praghash, V. D. S. Eswar, J. Y. Roy, A. Alagarsamy and S. Arunmetha, ”Tunnel Based Intra Network Controller Using NGROK Framework For Smart Cities,” 2021 5th International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 2021, pp. 39-43, doi: 10.1109/ICECA52323.2021.9676036.

[13] A. Hamza and B. Kumar, ”A Review Paper on DES, AES, RSA Encryption Standards,” 2020 9th International Conference System Modeling and Advancement in Research Trends (SMART), Moradabad, India, 2020, pp. 333-338, doi: 10.1109/SMART50582.2020.9336800.