

Proyecto Final del Curso Seguridad en Computación 2024-1

Rodrigo Salazar

Abril Vento

19 de julio de 2024

1. Introducción

1.1. Motivación

La encriptación end-to-end es actualmente uno de los tipos de encriptación que se consideran más seguros para la mensajería instantánea. En parte ello es porque al realizar la encriptación únicamente en los "extremos" de la comunicación entre un emisor y un receptor [1], elimina al servidor como un punto débil [2]. En otra, porque cuando es aplicada junto a algoritmos de encriptación híbridos, "puede aumentar la eficiencia de la encriptación, la organización de llaves y la seguridad," [3] con lo cual resuelve los principales problemas que se tiene con AES. Sin embargo, las implementaciones de este tipo de encriptación no son perfectas. "La forma en la que WhatsApp maneja los mecanismos de respaldo no proveen verdadera encriptación end-to-end," indican Singh, Chauhan y Tewari [4]. Este es uno de los servicios de mensajería instantánea más populares actualmente. Que no sea capaz de aplicarlo como se debería, indica una debilidad en su parte.

Vista la situación, se ve la importancia de hacer un proyecto sobre el tema, que pueda resaltar las ventajas de este tipo de encriptación y un ejemplo de su uso. El objetivo planteado para este trabajo es elaborar un sistema de mensajería instantánea que utiliza encriptación end-to-end mediante la implementación del *Extended Triple Diffie-Hellman*.

2. Marco Teórico

2.1. End-to-End Encryption

A la fecha, no existe una definición formal del término [5], debido a su cambiante significado a través de los años [2]. A nivel informal, sin embargo, hay muchas definiciones. Según Scheffler y Mayer (2023) [6], la encriptación *end-to-end* (E2EE de ahora en adelante) es una comunicación entre al menos un emisor y uno o más receptores, mediante un canal encriptado de *extremo a extremo*.

Esto quiere decir que la data es encriptada en todas las estaciones por las que pasa durante su transmisión por una red [7], y que solo puede ser descryptada por los participantes de la comunicación cuando llega a sus extremos. [8] Una ilustración de los componentes principales en esta encriptación se muestra en la figura 1 (recuperada de Bandt, 2024 [2]).

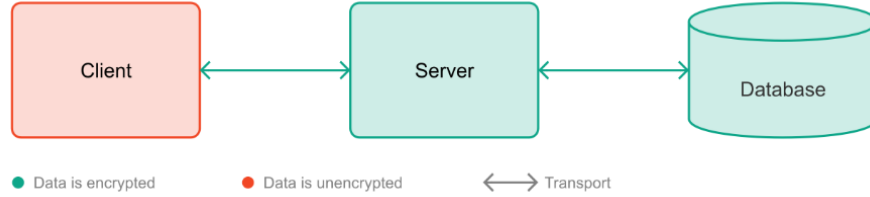


Figura 1: Diagrama resumen de la E2EE

Dada su naturaleza, la E2EE, entre otras, posee las siguientes características [6, 2]:

- **Confidencialidad:** el contenido cifrado no es vulnerable a ataques de texto plano por parte de atacantes en la red ni los operadores del canal de transmisión de los mensajes
- **Integridad:** permite que el receptor puede corroborar si el mensaje que ha recibido no ha sido modificado.
- **Autenticación:** los extremos de la comunicación pueden corroborar sus identidades mutuamente usando secretos criptográficos de largo plazo.
- **Disponibilidad:** los mensajes pueden leerse en varios dispositivos.
- **Deniability:** el origen de un mensaje encriptado no puede ser probado.
- **Forward Secrecy:** las llaves utilizadas durante la encriptación solo funcionan para un tiempo limitado, por lo cual los atacantes no pueden descryptar mensajes más adelante incluso si consiguiesen acceso a la llave.

Por lo general, el tipo de encriptación que se utiliza para el E2EE, es de tipo híbrida, en la que se encripta una llave simétrica con un método asimétrico y luego se usa la llave simétrica para encriptar la cantidad más amplia de datos. Esto permite solucionar tanto el problema causado por el uso de la misma llave en los métodos simétricos, como amortiguar la lentitud de los métodos asimétricos. [2]

Al momento de elegir qué algoritmo utilizar para la encriptación, es importante descartar inmediatamente aquellos considerados inseguros, es decir para los que se hallan encontrado métodos para romper la encriptación mediante un criptoanálisis. Además de ello, factores como la longitud de la llave, qué tanto se ha estudiado el algoritmo y que sea utilizable de manera práctica deben ser

considerados. Entre las recomendaciones de la *Federal Office for Information Security* (BSI por sus siglas en alemán) para algoritmos simétricos que cumplan con estas propiedades, se encuentra AES. [9]

En cuanto a asimétricos, promueve el uso del *Integrated Encryption Schemes* (IES) [9], un esquema basado en Diffie-Hellman. Específicamente entre las opciones que da, se encuentra la *Elliptic Curve Integrated Encryption Schemes* (ECIES). [2] Un ejemplo de encriptación por curvas criptográficas (ECC), es el algoritmo *Curve25519*. También llamado *X25519*, ha sido diseñado para su uso con el esquema de Diffie-Hellman de curva elíptica (ECDH por sus siglas en inglés). El *Extended Triple Diffie-Hellman* utiliza el ECDH para hacer cálculos, pero también incluye más parámetros de llaves pública para aumentar la seguridad. [10] Esto lo hace entonces el algoritmo ideal para probar la encriptación end-to-end.

2.2. X3DH

El protocolo de acuerdo de llaves *Extended Triple Diffie-Hellman* (X3DH de ahora en adelante) consiste en el establecimiento de una llave privada compartida entre dos partes que se autentican mutuamente entre ellas usando llaves públicas, de forma tal que provee *forward secrecy* y negación criptográfica. [11] Este es usado en el protocolo *Signal* para la aplicación de mensajería instantánea del mismo nombre. Los principales parámetros que considera son una curva, un hash y un tipo de información. Para lo primero, considera las siguientes llaves elípticas públicas (con sus respectivas llaves privadas):

- IK_A - Llave de identidad de Alice
- EK_A - Llave efímera de Alice
- IK_B - Llave de identidad de Bob
- SPK_B - *Prekey* firmada de Bob
- OPK_B - *One-time prekey* de Bob

Las *prekeys* son mensajes del protocolo que Bob publica al servidor antes de que Alice ejecute el protocolo. La de tipo *SPK* es cambiada de manera periódica. Se genera un conjunto de *OPKs* y una al azar es empleada para cada ejecución del protocolo y eliminada tras su uso. Cuando quedan pocas se vuelve a generar otro conjunto. Las llaves de identidad son de largo plazo, y son lo que utilizan los usuarios para distinguirse unos de otros. Un par de llaves efímeras (pública y privada) se generan en cada ejecución del protocolo. Una vez ejecutado el protocolo de manera exitosa, el resultado es una llave secreta *SK* de 32-bytes compartida por Alice y Bob. [11]

En términos generales, X3DH tiene tres fases: [11]

1. Bob publica su llave de identidad y prekeys a un servidor

2. Alice consigue un "prekey bundle" del servidor, y lo utiliza para enviarle un mensaje inicial a Bob
3. Bob recibe y procesa el mensaje inicial de Alice.

Estos pueden separarse en 6 si se quiere ser un poco más detallado. Se muestra el funcionamiento de manera gráfica en la figura 2.

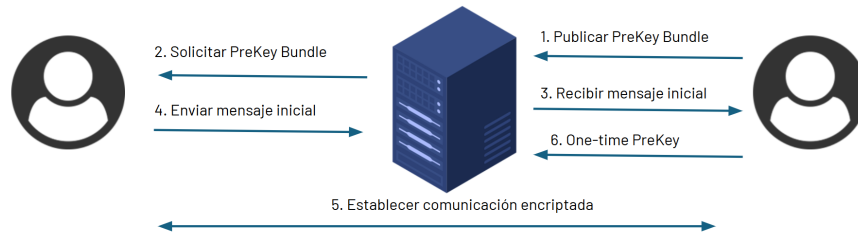


Figura 2: Funcionamiento del X3DH

Para realizar una comunicación, Alice y Bob deben tener un acuerdo de llaves entre ellos. La forma en la que se realizan los cálculos de Diffie-Hellman entre llaves para esta operación se visualiza en la figura 3.

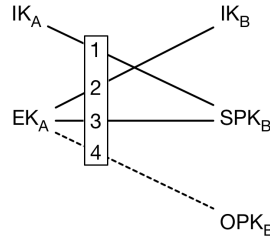


Figura 3: Intercambio de llaves en X3DH

3. Desarrollo

3.1. Usabilidad y Modelo Mental

El producto a realizar es una aplicación de mensajería instantánea “privada”, en la cual el mensaje es encriptado antes de salir del computador de un usuario emisor *Alice* y solo se desencripta al llegar al computador de un usuario receptor *Bob*. Se garantiza la privacidad **si y sólo si tanto Alice como Bob han validado mutuamente sus llaves públicas (información de contacto) de antemano**. Una ilustración se muestra en la figura 4.



Figura 4: Representación Gráfica del Modelo Mental

3.2. Diseño del Sistema

En la figura 5, se muestra la arquitectura general de la propuesta. Consiste de un único módulo de chat en el que dos clientes pueden ingresar mediante CLI, y se conectan con el servidor del chat usando un canal de *secure web sockets*. Dicho servidor está también conectado a una base de datos en MongoDB, donde se almacena una lista de usuarios y los mensajes enviados.

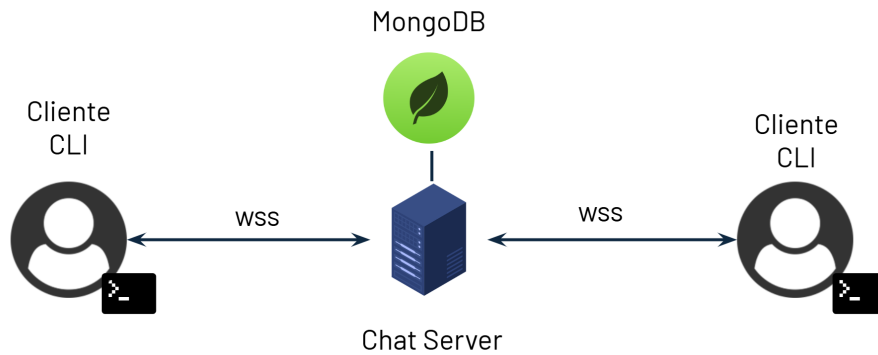


Figura 5: Arquitectura

Se consideraron también los requerimientos listados en la tabla 1.

ID	Prioridad	Requerimiento
FR01	MUST HAVE	Enviar (y recibir) mensajes de forma asíncrona
FR02	MUST HAVE	Mostrar la llave pública de largo plazo
FR03	NICE TO HAVE	Ver el recibo de estado del mensaje
NFR01	MUST HAVE	Permitir una comunicación en tiempo real (< 200ms)
NRF02	SHOULD HAVE	Manejar más de 1000 solicitudes de websockets en paralelo
NFR03	SHOULD HAVE	Tener un rendimiento de más de 200 mensajes por segundo

Cuadro 1: Tabla de Requerimientos

3.3. Modelo de Amenazas

Se consideran tres principales componentes en el modelo de amenazas: cliente, red y servidor. A continuación se listan los riesgos y respectivas soluciones para cada uno de ellos en la tabla 2

Una de las principales brechas de seguridad que se consideró para el diseño es la **filtración de secreto**, es decir la situación en la que el actor malicioso obtiene acceso al secreto utilizado para formar las sesiones de JSON Web Tokens (JWT), de forma tal que puede forjar nuevas sesiones. Analizando el sistema planteado en la etapa presente, se observa que el impacto de este ataque en la integridad es nulo, pues no se está dando opción de modificar data existente. Sin embargo, a nivel de privacidad, si bien el contenido de los mensajes no puede filtrarse, es posible que un atacante pueda observar la lista de mensajes recibidos. Y a nivel de disponibilidad, existe el caso en el que el atacante pueda inundar el servidor con nuevos OTP que generen mensajes inválidos.

Visto ello, se propone incluir una forma de invalidar el secreto anterior y que los nuevos secretos sean actualizados en los servidores. En caso que un daño se haya producido, se realiza una revisión de los logs para analizar el impacto a los usuarios y dependiendo de qué tan grave es, se hace un rollback y/o eliminación de filas en la base de datos para remediarlo.

Componente	Riesgo	Solución
Usuario	Manejo inadecuado de secretos Robo de sesiones Prácticas de seguridad inadecuadas	Guías al usuario sobre principales consideraciones de seguridad JWT con fecha de expiración
Red	Man in the Middle Denial of Service	Protocolos de seguridad en transporte Limitación de solicitudes por usuario
Servidor	IDOR SQLi Divulgación de información Filtrado de secretos Broken Auth	Uso de herramientas de análisis estático de código Aislamiento de componentes críticos Autorización de acceso a recursos Uso de librerías reconocidas Actualización periódica de secretos

Cuadro 2: Tabla de riesgos y soluciones por componente

4. Implementación

4.1. Descripción

Como detalles de la implementación:

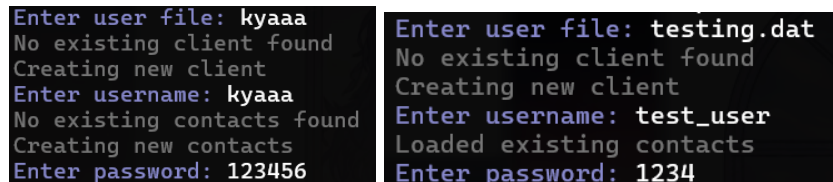
- Diffie-Hellman: x25519 Curve
- AEAD: Galois/Counter Mode AES
- KDF: pbkdf2

Los usuarios poseen, de manera local, archivos `.dat` con toda la información de su usuario incluyendo nombre, contraseñas y llaves. También cuentan con un archivo `contacts.json` donde almacenan el nombre y llave pública de sus contactos. La información de contacto se comparte también en un formato JSON que contiene el nombre y la llave pública de dicho usuario. Es esto lo que se utiliza para añadir contactos.

Los OPKs se crean -por motivos demostrativos- de 5 en 5. Por la naturaleza educativa de este aplicativo, cada vez que quedan solo 3, se envía un mensaje indicando que quedan pocos. En una versión oficial de este programa, se mantendría dicha funcionalidad como abstracta y no se mostraría dicho mensaje.

Se utilizó pretty print para poder hacer la interfaz de CLI más atractiva y que el usuario pueda entenderla mejor.

A continuación, algunas capturas de la ejecución del programa.



```
Enter user file: kyaaa
No existing client found
Creating new client
Enter username: kyaaa
No existing contacts found
Creating new contacts
Enter password: 123456

Enter user file: testing.dat
No existing client found
Creating new client
Enter username: test_user
Loaded existing contacts
Enter password: 1234
```

Figura 6:

- (a) Iniciando el programa por primera vez
- (b) Creando un segundo usuario

En la figura 6, se muestra el programa siendo corrido en una computadora por primera vez. Como no existe un user file previo, crea uno con el nombre especificado y pide crear un username y una contraseña después de ello.

El usuario usa su archivo `.dat` y su contraseña para autenticarse ante el sistema. Una vez corroborada su identidad e iniciada su sesión, procede a mostrar un menú de opciones.

La primera opción muestra la lista de contacto del usuario, colocando tanto su nombre como su llave pública, como se observa en la figura 8.

Para añadir un contacto a la lista de un usuario, este requiere tener el archivo JSON del mismo. En la figura 9, se muestra el uso de la opción 6 para obtener


```

PS C:\Users\rodri\Rodrigo\UTEC\2024-1\Seguridad\final\E2EE-chat\e2ee_client> go run .
Enter user file: utec.dat
Loaded existing client
Loaded existing contacts
Enter password: 1234

=== Menu ===
# OPTIONS
1 List Contacts
2 Add Contact
3 Remove Contact
4 Send Message
5 Receive Messages
6 Share My Contact
7 Help
8 Exit
Enter choice:
Low OTP. Sending more.

```

Figura 7: Usuario iniciando sesión

```

PS C:\Users\rodri\Rodrigo\UTEC\2024-1\Seguridad\final\E2EE-chat\e2ee_client> go run .
Enter user file: u2.dat
Loaded existing client
Loaded existing contacts
Enter password: mypass

=== Menu ===
# OPTIONS
1 List Contacts
2 Add Contact
3 Remove Contact
4 Send Message
5 Receive Messages
6 Share My Contact
7 Help
8 Exit
Enter choice: 1

=== Contacts ===
# Username Public Key
0 user1 Q2ZN/h0CNK0xQqSkUvrcj5k9UmRuIbIdOL++IAigVQA=
-----
1 utec +v960JceQCjJzkQF95krmuQX53h3tVhQf0lBeLk+diw=

```

Figura 8: Opción 1: Lista de Contactos

```

1 List Contacts
2 Add Contact
3 Remove Contact
4 Send Message
5 Receive Messages
6 Share My Contact
7 Help
8 Exit
Enter choice: 2
Enter contact file: MyContact.json
Contact added

=== Menu ===
# OPTIONS
1 List Contacts
2 Add Contact
3 Remove Contact
4 Send Message
5 Receive Messages
6 Share My Contact
7 Help
8 Exit
Enter choice: 1

=== Contacts ===
# Username Public Key
0 user1 Q2ZN/h0CNK0xQqSkUvrcj5k9UmRuIbIdOL++IAigVQA=
-----
1 utec +v960JceQCjJzkQF95krmuQX53h3tVhQf0lBeLk+diw=

```

Figura 9: Opción 2: Añadir Contacto

```
=== Menu ===
# OPTIONS
1 List Contacts
2 Add Contact
3 Remove Contact
4 Send Message
5 Receive Messages
6 Share My Contact
7 Help
8 Exit
Enter choice: 3
Enter contact id:
0
Are you sure you want to remove contact:
=== Contact ===
Username: user1
Public Key: QZ2U/h0CHK0XqQsKJvrcjsk9Umru1bIdOL++IAIgvQA=
=====
Enter 'yes' to confirm:
yes
Contact removed
```

Figura 10: Opción 3: Remover Contacto

ello por el lado de un usuario, y por el otro, se tiene un usuario que utiliza ese archivo para añadir al otro usándola.

Para remover un contacto se elige a un usuario por su número de ID en la lista de contactos. (Figura 10).

```
=== Menu ===
# OPTIONS
1 List Contacts
2 Add Contact
3 Remove Contact
4 Send Message
5 Receive Messages
6 Share My Contact
7 Help
8 Exit
Enter choice: 4
Enter contact id: 1
Enter message: HelloWorld!
Message sent
```

Figura 11: Opción 4: Enviar Mensaje

Para enviar un mensaje, se debe seleccionar el usuario destinatario y luego colocar el contenido (Figura 11). Si el usuario receptor está conectado, este recibe notificaciones indicándole que tiene mensajes entrantes (Figura 12).

La opción 5 permite al usuario ver un historial de los mensajes que ha recibido. De ser el emisor alguien fuera de la lista de contactos del receptor, un mensaje de advertencia aparece previo al mensaje, advirtiéndolo ante ello. Esta medida de seguridad a nivel de usuario se visualiza en la Figura 13.

Otra medida de seguridad usable colocada es la información proveída por la opción 7. Además de cubrir con las medidas consideradas en el modelo de amenazas, refuerza el modelo mental del aplicativo al proveer una descripción de la misma. (Figura 14.)

El repositorio a la implementación completa se encuentra aquí: <https://github.com/tux550/E2EE-chat>.

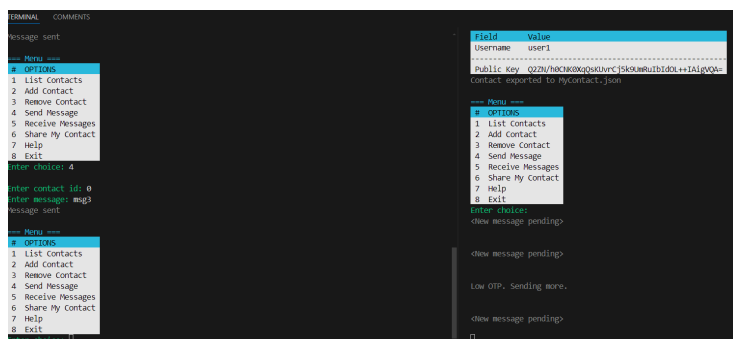


Figura 12: Notificaciones mensaje recibido

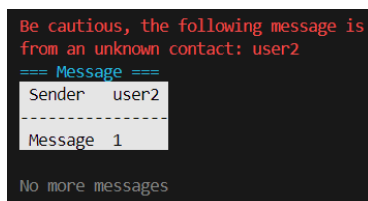


Figura 13: Abrir mensaje desconocido

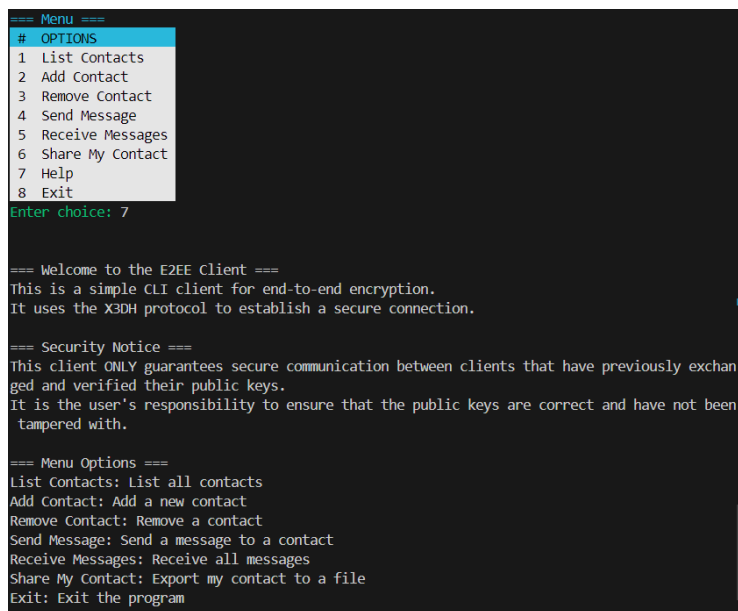


Figura 14: Opción 7: Ayuda

4.2. Evaluación

Como herramienta de análisis estático, se utiliza *SonarQube*. Además de su facilidad de incorporar al proyecto, también ha sido probada por múltiples usuarios, permite producir logs, y se actualiza constantemente. Esto agrega múltiples principios de seguridad de diseño a la solución final.

La integración de SonarQube para el sistema requiere la instalación de SonarQube CLI, Sonar Scanner y Java 17. Con esto cubierto, se puede configurar los puertos y host a usar por Sonar Scanner para poder ejecutarlo. Al hacerlo, se puede ingresar al portal web de Sonar, mediante el cual se puede crear un proyecto, su respectivo token y configurar sus propiedades. Al correr Sonarqube para el proyecto específico, se realiza un análisis completo que indica la calidad del código. Para el caso del aplicativo desarrollado, no se ha identificado ningún security hotspot, como se observa en la imagen 15. Además se ha identificado un grado de seguridad elevado en el aplicativo (figura 16).

Se tienen clientes en CLI que pueden comunicarse entre ellos. Se prueba enviar mensajes de uno al otro.

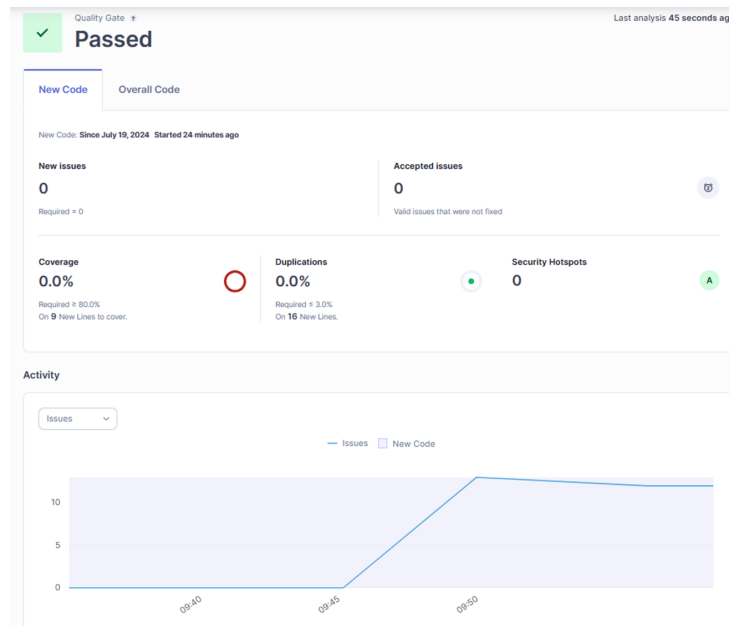


Figura 15: Captura de pantalla de la interfaz de Sonar para el análisis del código del proyecto

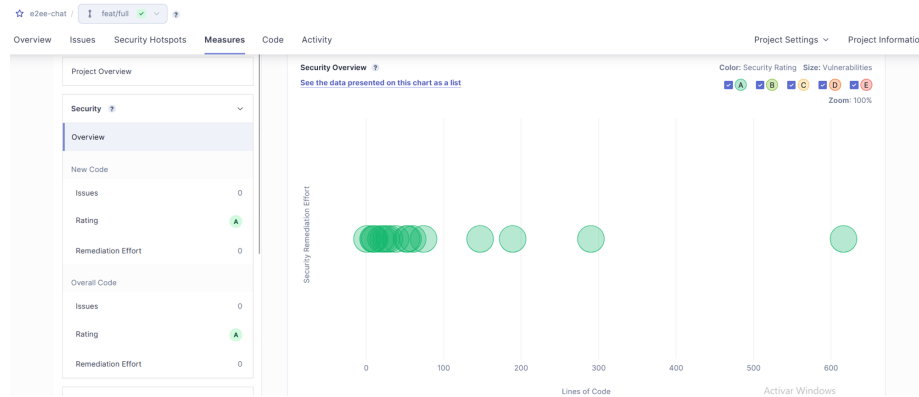


Figura 16: Enter Caption

5. Conclusión

Al desarrollar este trabajo y poder correr el código correctamente, se cumplió con el objetivo de elaborar un sistema de mensajería instantánea que utiliza encriptación end-to-end mediante la implementación del *Extended Triple Diffie-Hellman*.

Asimismo, se consolidaron conocimientos sobre diseños seguridad usable y diseños seguros, al requerirse ambos para diseñar el producto y ponerlo en ejecución.

Vista que la prioridad para este proyecto estuvo en la implementación del algoritmo de encriptación, para futuras instancias, podría ponerse mayor énfasis en otros aspectos. Algunas mejoras que pueden agregarse incluyen hacer una interfaz más elaborada y modificar la arquitectura para tener módulos separados para autenticación, mensajes y contactos.

Referencias

- [1] B. Hale and C. Komlo, “On end-to-end encryption,” Cryptology ePrint Archive, Paper 2022/449, 2022, <https://eprint.iacr.org/2022/449>. [Online]. Available: <https://eprint.iacr.org/2022/449>
- [2] T. Bandt, “End-to-end encryption: A technical perspective,” 2024. [Online]. Available: <https://thomasbandt.com/end-to-end-encryption-technical-perspective>
- [3] O. O. Blaise, O. Awodele, and O. Yewande, “An understanding and perspectives of end-to-end encryption,” in *IRJET*, vol. 8, no. 4, 2021.
- [4] R. Singh, A. N. S. Chauhan, and H. Tewari, “Blockchain-enabled end-to-end encryption for instant messaging applications,” in *2022*

- IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2022, pp. 501–506. [Online]. Available: <https://research-portal.uws.ac.uk/en/publications/blockchain-enabled-end-to-end-encryption-for-instant-messaging-ap>
- [5] M. Knodel, S. Celi, O. Kolkman, and G. Grover, “Definition of End-to-end Encryption,” Internet Engineering Task Force, Internet-Draft draft-knodel-e2ee-definition-11, Jun. 2023, work in Progress. [Online]. Available: <https://datatracker.ietf.org/doc/draft-knodel-e2ee-definition/11/>
- [6] S. Scheffler and J. Mayer, “Sok: Content moderation for end-to-end encryption,” 2023. [Online]. Available: <https://arxiv.org/abs/2303.03979>
- [7] M. Nieves, K. Dempsey, and V. Y. Pillitteri, “An introduction to information security,” National Institute of Standards and Technology (NIST), Tech. Rep., 2017. [Online]. Available: <https://doi.org/10.6028/NIST.SP.800-12r1>
- [8] Bundesamt für Sicherheit in der Informationstechnik (BSI), “Moderne messenger – heute verschlüsselt, morgen interoperabel?” Bundesamt für Sicherheit in der Informationstechnik (BSI), Tech. Rep., 2021. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/DVS-Berichte/messenger.pdf?__blob=publicationFile&v=8
- [9] —, “Kryptographische verfahren: Empfehlungen und schlüssellängen,” Bundesamt für Sicherheit in der Informationstechnik (BSI), Tech. Rep., 2024. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile&v=6
- [10] ChainX, “How does the signal protocol achieve end-to-end encrypted communication?” 2021. [Online]. Available: <https://chainx-org.medium.com/how-does-the-signal-protocol-achieve-end-to-end-encrypted-communication-a2b92ca13b54>
- [11] M. Marlinspike, “The X3DH Key Agreement Protocol,” 2016. [Online]. Available: <https://signal.org/docs/specifications/x3dh/>