

Design of an Encryption Chat Application Based on ChaCha20-Poly1305

Yurui Feng *

School of Cyber Science and Engineering, Huazhong University of science and technology,
Wuhan, 430074, China

* Corresponding Author Email: U202112079@hust.edu.cn

Abstract. This study introduces the meticulous design and development of a chat application fortified with the robust ChaCha20-Poly1305 encryption algorithm. The selected cryptographic algorithm is renowned for its efficacy in ensuring top-notch security. In our application, ChaCha20 serves as the backbone for stream encryption, ensuring that the message content remains confidential and impenetrable to eavesdroppers. Simultaneously, Poly1305, an esteemed message authentication code, guarantees the integrity and authenticity of each chat message. It ensures that the messages sent are genuine and have not been tampered with during transit. Together, the integration of these two components constructs a fortified wall against potential security breaches, safeguarding user communications. This not only instills confidence in users regarding the privacy of their conversations but also sets a benchmark in chat application security. The overarching aim is to offer an impeccable blend of user-friendly experience and top-tier security, revolutionizing the way secure communications are perceived and utilized.

Keywords: Encrypted chat application, ChaCha20-Poly1305, Text Encryption.

1. Introduction

In a digitized era, the importance of privacy and security cannot be understated. As advancements in information and communication technologies progress at a rapid pace, reliance on diverse communication channels to share sensitive information—from personal details to business secrets—becomes inescapable. Yet, with these digital exchanges come inherent risks: data breaches, eavesdropping, and malicious incursions.

Encrypted chat software emerges as an essential shield in this landscape, protecting communication privacy and confidentiality. Using sophisticated encryption algorithms, these tools ensure that exchanged information remains secure in transit, warding off third-party interception and potential data exposure. Beyond personal exchanges, encrypted chat applications hold paramount importance in business and governmental sectors, defending sensitive data against looming cyber threats.

However, these protective digital bubbles present challenges. While offering enhanced security, they also ignite debates, especially concerning lawful access and the regulatory gaze of law enforcement. These discussions often center around the precarious balance between individual privacy rights and overarching societal security.

This paper delves deep into the significance, core functionalities, and challenges surrounding encrypted chat software, highlighting evolving trends and anticipating future innovations. Additionally, this exploration considers the intricate dance between privacy rights and security imperatives in encrypted chat software's design and deployment, offering insights into best practices to counteract potential vulnerabilities. Through such analysis, the goal is to elucidate the pivotal role encrypted chat software plays in a digitized world and underscore the importance of safeguarding communication privacy for both individuals and institutions.

2. Encrypted Chat Applications

In this section, the analysis focuses on existing publicly available encrypted chat software. By comparing their similarities and differences, the features, protocols, and algorithms used in encrypted chat applications are dissected and examined.

2.1. Key Features of Encrypted Chat Software

With the advancement of cryptographic technology, most encrypted chat software now adopts the Signal Protocol for communication, utilizing end-to-end encryption for message transmission. The sender and receiver can encrypt and decrypt messages using pre-negotiated keys and each other's publicly available public keys. To prevent interception and decryption of information, a critical aspect is verifying the identities of the communicating parties, often accomplished through digital certificates, key pairs, PIN codes, PKI, and similar methods [1].

Moreover, mainstream encrypted chat applications available today often incorporate features for message destruction and self-destruction. This functionality enables users to set a lifespan for messages, making them visible within this timeframe. Once the message expires, it is automatically destroyed and no longer accessible. Typically, a timer is set at the time of sending, and while the message is stored locally on the device, it is simultaneously counting down. Once the specified time interval elapses, the message is automatically removed from the user's device, rendering it invisible. Therefore, messages are not permanently stored on the user's device. Preceding transmission, messages undergo end-to-end encryption. As mentioned earlier, only the intended recipient can decrypt and read the message content. This feature is fundamental for ensuring the privacy and security of messages and constitutes a core functionality of encrypted chat applications.

In the realm of encrypted chat software, the choice of platform should not hinder its widespread adoption. Hence, cross-platform support is indispensable for encrypted chat applications. It offers users convenience and flexibility, aiding in expanding user bases and facilitating collaboration. Message synchronization allows users to manage information across different devices. Particularly crucial for businesses and teams, cross-platform support ensures seamless communication, addressing compatibility concerns and proving highly beneficial for remote work and geographically dispersed teams.

2.2. Comparative Analysis of Existing Encrypted Chat Software

This section provides a comparative analysis of the top encrypted chat applications, as per the rankings on the iOS and Google Play app stores by PCMag. The focus is on Signal Private Messenger, WhatsApp, and Telegram, which emerged as the top three encrypted chat applications in the rankings. The analysis includes an overview of their features, encryption methods, user experience, and platform compatibility, offering insights into their respective strengths and capabilities. This analysis is essential for users seeking to choose the most suitable encrypted chat application based on their preferences and needs [2]. As shown in Table 1.

Table 1. Comparative Analysis of Signal, WhatsApp and Telegram

	Signal	WhatsApp	Telegram
Encryption Methods	Signal Protocol	Signal Protocol, TextSecure	MTPROTO
Advantages	Strong in Privacy and Encryption Aspects	Comprehensive Communication Tool	Large Social Communities and Scalability
Applicable to	Privacy-Conscious Users	Staying Connected with a Broad Network of Contacts	Users Needing to Organize a Large Number of Individuals

Regarding the Signal Protocol, subsequent sections delve into a comprehensive analysis. Controversies surrounding TextSecure, employed by WhatsApp, and MTPROTO, adopted by Telegram, deserve mention.

Initially crafted by Open Whisper Systems, the TextSecure protocol was later integrated into WhatsApp, forming the backbone of its end-to-end encryption. Operating as an encryption protocol influenced by the Signal Protocol, its open-source nature encourages independent security experts to scrutinize its source code, affirming its reliability. To summarize, Text Secure, as implemented in WhatsApp, stands as a potent encryption tool, bolstering user privacy and security. Such advanced protections propelled WhatsApp to the forefront of encrypted chat applications [3].

MTPROTO, Telegram's home-grown protocol, blends symmetric with asymmetric encryption, ensuring message confidentiality during transit. It embraces forward secrecy; this principle ensures that, even with compromised encryption keys, past conversations remain confidential. Continual key updates further encrypt subsequent interactions, preserving historical message integrity. While MTPROTO's design prioritizes security, its unique, proprietary origins paired with undisclosed elements have sparked debate. Its initial unavailability for public examination prompted skepticism from encryption specialists and security researchers [4]. Yet, with its transition to open-source and an upgrade to version 2.0 in April 2021 [5], prior reservations were largely placated. However, fresh contentions about MTPROTO 2.0 surfaced. Resistance grew concerning the adoption of unconventional temporary protocols and cryptographic strategies, as the absence of wider examination could reveal latent system vulnerabilities. Further complicating matters, all messages traverse proprietary, closed-source servers, marking them as potential security weak points [6].

2.3. Signal Protocol

The Signal Protocol is an open-source encryption protocol used to implement end-to-end encrypted communication, primarily focused on ensuring message privacy and security. Developed by Open Whisper Systems, the Signal Protocol is widely utilized in various communication applications, including Signal, WhatsApp, and other encrypted chat tools [7]. It encompasses several key features: forward secrecy, end-to-end encryption, key exchange, multi-platform support, and complete open-source verifiability [8]. In the subsequent sections, this paper will delve into the algorithms employed by the Signal Protocol, including DH, Curve25519, HMAC, AES, SHA-256, and ECDH.

2.3.1. Diffie-Hellman Key Exchange (DH Key Exchange)

The Signal Protocol employs the Diffie-Hellman key exchange algorithm to negotiate session keys. This secure protocol allows two communicating parties to share an encryption key without directly transmitting the key. Additionally, the Signal Protocol utilizes double Diffie-Hellman key exchange, meaning that new temporary key pairs are generated for each message exchange, ensuring forward secrecy [9].

2.3.2. Curve25519

Curve25519 is a secure elliptic curve cryptography algorithm used for Diffie-Hellman key exchange. The Signal Protocol utilizes the Curve25519 curve to perform key exchange, ensuring the security and efficiency of the key negotiation process [10].

2.3.3. AES (Advanced Encryption Standard)

The Signal Protocol employs the AES encryption algorithm to encrypt message content. AES is a symmetric encryption algorithm that uses the same key for both encryption and decryption of data. Signal utilizes a session key (derived from the Diffie-Hellman key exchange) to perform encryption and decryption operations on the message content [11].

2.3.4. HMAC (Hash-based Message Authentication Code)

HMAC is a technique used to ensure message integrity. The Signal Protocol utilizes HMAC to verify whether messages have been tampered with during transmission [12].

2.3.5. SHA-256 (Secure Hash Algorithm 256-bit)

SHA-256 is used to compute the hash value of messages, ensuring that messages have not been tampered with during transmission [13].

2.3.6. ECDH (Elliptic Curve Diffie-Hellman Key Exchange)

The Signal Protocol utilizes ECDH for key exchange. Specifically, it employs ECDH-ES (Elliptic Curve Diffie-Hellman Key Exchange - based on static keys) and ECDH-1PU (Elliptic Curve Diffie-Hellman Key Exchange - based on one-party usage keys) [14].

3. Building an Encrypted Chat System

3.1. First-Time Registration

When the user first opens the interface, they will be prompted with registration options, encouraging them to register and log in. Upon authenticating their phone number, username, and password, the user registration is completed. After the system obtains the user's password, it encrypts the password using salted hashing. The system stores the hashed value of the user's password with a unique "salt," which is a random value-added during hashing [15,16]. Unlike simple hash storage, salted hashing ensures each user has a unique salt, resulting in different hash values even if two users have the same password. Salting enhances security because attackers cannot identify identical passwords through simple hash comparisons. After these steps, the system randomly generates the necessary public and private key pair for encryption, used to encrypt and decrypt messages during subsequent information exchanges.

3.2. Login

Firstly, the user enters their username and password on the login page. At this point, the system retrieves the hash value and salt associated with the user from the database based on the username or unique identifier. Subsequently, using the same hash algorithm and the retrieved salt, the system hashes the password entered by the user. The hashed password is then compared with the hashed password stored in the database. If the two match, the user has provided the correct password and is allowed to log in.

3.3. Chat Functionality

3.3.1. Encryption Process

Obtain User Key Pair: Users generate a key pair (public key, private key) during registration. These key pairs are secure, unique, and each user's key pair is distinct.

Message Encoding: Users compose the message they want to send.

Generate Random Nonce: Generate a random nonce, unique for each message, used to create the ChaCha20 keystream. The nonce is unique for every message.

Key Derivation using HKDF: Utilize HKDF to derive keys [17]. Initially, extract entropy from shared keying material (The shared key material is established based on trusted key negotiation), generating an independent pseudo-random key. Subsequently, based on this pseudo-random key, derive additional keying material, including the keys and nonce required for ChaCha20.

Generate Keystream: Utilize the derived keys and nonce as input to generate the ChaCha20 keystream. This keystream will be used to encrypt the message.

Encrypt Message using ChaCha20: Employ the ChaCha20 keystream to encrypt the message, resulting in the encrypted message ciphertext.

Compute Poly1305 Authentication Tag: Use the encrypted message ciphertext and the user's public key as input to compute the Poly1305 authentication tag [18].

Send Encrypted Message and Authentication Tag: Transmit the encrypted message ciphertext, Poly1305 authentication tag, and nonce together to the recipient.

Encryption process is as shown in Figure 1.

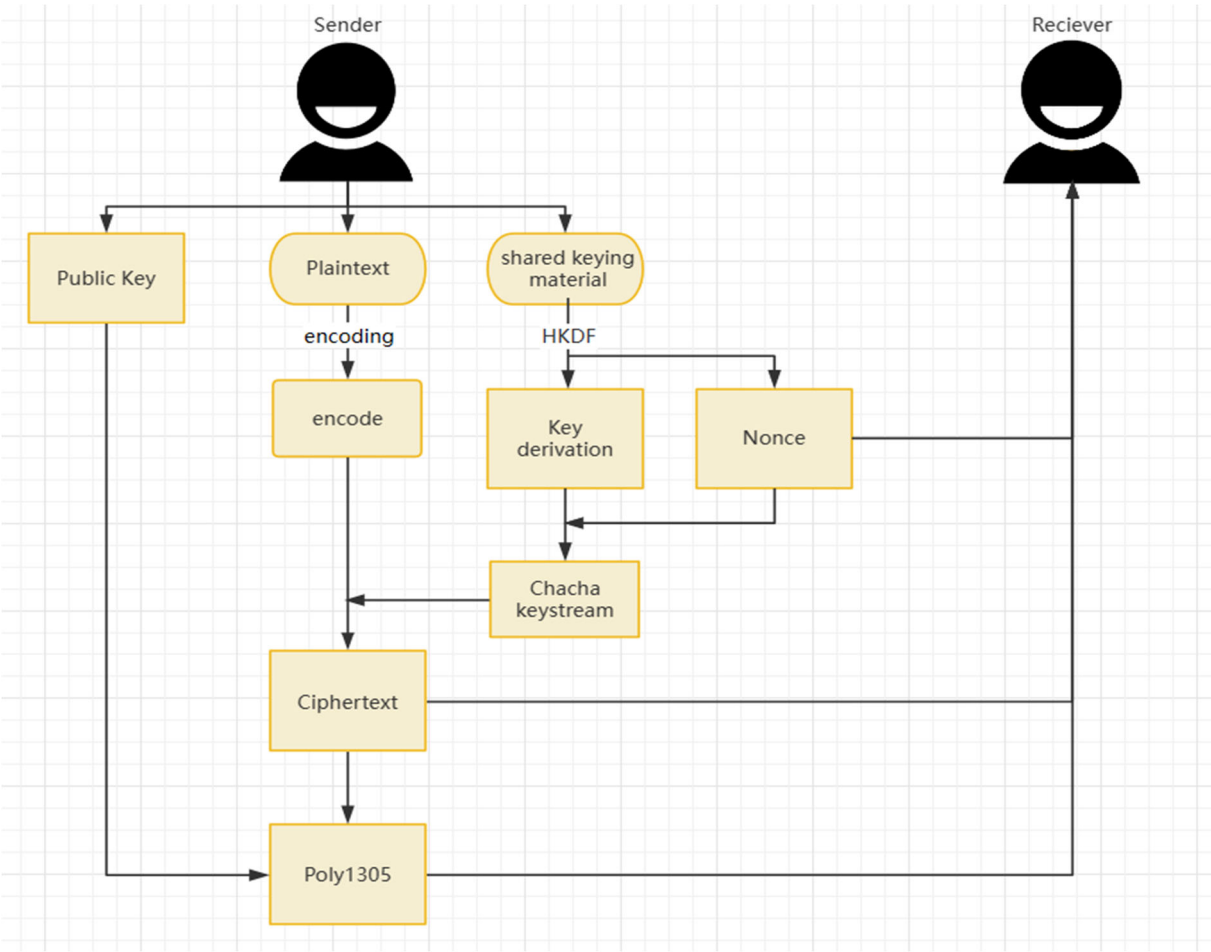


Figure 1. Encryption process

3.3.2. Decryption Process

Receive Encrypted Message and Authentication Tag: Receive the encrypted message ciphertext, Poly1305 authentication tag, and nonce.

Retrieve User Key Pair: Obtain the user's public key and private key.

Generate Keystream: Use the derived keys from HKDF and the received nonce as input to generate the ChaCha20 keystream.

Decrypt Message using ChaCha20: Utilize the ChaCha20 keystream to decrypt the encrypted message ciphertext, resulting in the original message.

Compute Poly1305 Authentication Tag: Use the decrypted message ciphertext and the user's public key to compute the Poly1305 authentication tag.

Verify Message Integrity: Compare the computed Poly1305 authentication tag with the received Poly1305 authentication tag to validate the message's integrity.

Display Decrypted Message: If the message integrity verification is successful, display the decrypted original message.

Decryption process is as shown in Figure 2.

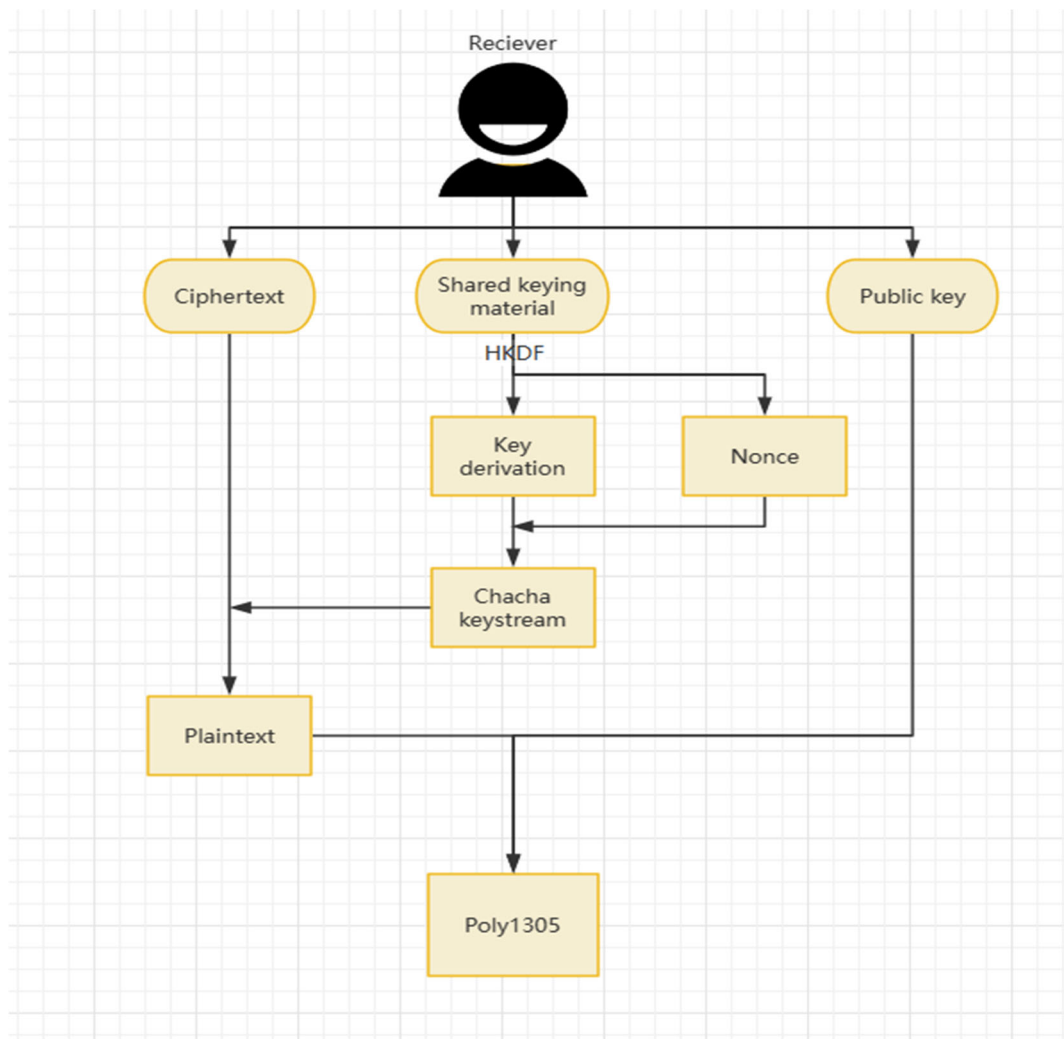


Figure 2. Decryption process

4. Feasibility Analysis

4.1. Key Transmission Analysis

ChaCha20-Poly1305 uses a symmetric key, unlike asymmetric encryption with a pair of public and private keys. Therefore, the recipient does not need to acquire the sender's private key to decrypt the ciphertext, avoiding the risk of private key exposure. This further prevents identity impersonation, message tampering, protocol security breaches, or other potential attacks. In the encryption and decryption process of ChaCha20-Poly1305, a cryptographically secure key derivation function (HKDF) is employed to generate the ChaCha20 key and nonce from the user's long-term keying material. This ensures the security and randomness of the keys [19].

4.2. Message Integrity Verification

After the message sender calculates the Poly1305 authentication tag, this tag is sent to the receiver. Upon receiving the message and decrypting it, the receiver uses the decrypted plaintext and the sender's public key to compute the Poly1305 authentication tag. Subsequently, the received Poly1305 authentication tag is compared with the computed Poly1305 authentication tag. This comparison allows the receiver to verify the integrity of the decrypted message [20].

4.3. Security Analysis

From the perspective of encryption algorithms, ChaCha20 is a secure symmetric encryption algorithm. It adopts advanced encryption principles, such as constructing the encryption algorithm using a pseudo-random function, ensuring the algorithm's security. Moreover, ChaCha20 allows the use of different key lengths, such as 128 bits and 256 bits, with longer keys providing higher security strength that is difficult to crack through exhaustive methods [21].

In terms of public awareness and authority, ChaCha20 has undergone extensive cryptographic analysis and review. It has been thoroughly examined and tested by cryptographic experts' multiple times, demonstrating its ability to prevent common attacks like differential cryptanalysis, linear cryptanalysis, and related-key attacks. This provides evidence of its security.

5. Conclusion

In this paper, the described encryption and decryption processes provide a robust and secure method for ensuring the confidentiality and integrity of messages within the application. The process involves the generation of user-specific key pairs, random nonces, and the utilization of advanced cryptographic techniques like ChaCha20 and Poly1305. The encryption process begins with user key pair generation, message encoding, and the generation of unique nonces for each message. Key derivation using HKDF ensures the creation of independent and secure keys for encryption. The ChaCha20 keystream is then generated and employed to encrypt the message, with an additional Poly1305 authentication tag computed to verify message integrity. The encrypted message ciphertext, authentication tag, and nonce are transmitted to the recipient. The decryption process on the recipient's end is equally secure and follows a precise set of steps. Upon receiving the encrypted message, the recipient retrieves their user key pair, generates the ChaCha20 keystream using the provided nonce, and decrypts the message. The Poly1305 authentication tag is computed and compared to the received tag to validate message integrity. If the verification is successful, the decrypted original message is displayed. Overall, these encryption and decryption processes establish a strong foundation for safeguarding the confidentiality and integrity of messages, ensuring that only authorized parties can access and read the messages while protecting against tampering or unauthorized alterations during transmission.

References

- [1] Moyle, L., et al. (2019). # Drugsforsale: An exploration of the use of social media and encrypted messaging apps to supply and access drugs. *International Journal of Drug Policy*, 63, 101 - 110.
- [2] Eddy, M. (2023). The best secure messaging apps for 2023. *PCMag*. Retrieved from <https://www.pcmag.com/picks/best-secure-messaging-apps>.
- [3] Frosch, T., et al. (2016). How secure is TextSecure? 2016 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE.
- [4] Jakobsen, J., & Orlandi, C. (2016). On the CCA (in) security of MTProto. *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*.
- [5] Wikipedia contributors. (2023). Telegram (software). *Wikipedia, The Free Encyclopedia*. Retrieved from [https://en.wikipedia.org/wiki/Telegram_\(software\)](https://en.wikipedia.org/wiki/Telegram_(software)).
- [6] Miculan, M., & Vitacolonna, N. (2023). Automated verification of Telegram's MTProto 2.0 in the symbolic model. *Computers & Security*, 126, 103072.
- [7] Almuzaini, N. Z., & Ahmad, I. (2019). Formal analysis of the signal protocol using the scyther tool. 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS). IEEE.
- [8] WhatsApp. (n.d.). Privacy Policy. Retrieved from <https://www.whatsapp.com/privacy>.
- [9] Signal. (n.d.). The Double Ratchet Algorithm Diffie-Hellman ratchet. Retrieved from <https://signal.org/docs/specifications/doubleratchet/>.

- [10] Niasar, M. B., et al. (2020). Fast, small, and area-time efficient architectures for key-exchange on Curve25519. 2020 IEEE 27th Symposium on Computer Arithmetic (ARITH). IEEE.
- [11] Wikipedia contributors. (2023). Advanced Encryption Standard. Wikipedia, The Free Encyclopedia. Retrieved from https://en.wikipedia.org/wiki/Advanced_Encryption_Standard.
- [12] Wikipedia contributors. (2023). HMAC. Wikipedia, The Free Encyclopedia. Retrieved from <https://en.wikipedia.org/wiki/HMAC>.
- [13] IETF Datatracker. (2008). IPR Details: The United States of America as represented by the National Security Agency's general license statement. Retrieved from [URL not provided].
- [14] Wikipedia contributors. (2023). Elliptic-curve Diffie–Hellman. Wikipedia, The Free Encyclopedia. Retrieved from https://en.wikipedia.org/wiki/Elliptic-curve_Diffie–Hellman.
- [15] Gauravaram, P. (2012). Security Analysis of salt|| password Hashes. 2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT). IEEE.
- [16] Rathod, U., Sonkar, M., & Chandavarkar, B. R. (2020). An experimental evaluation on the dependency between one-way hash functions and salt. 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT). IEEE.
- [17] Wikipedia contributors. (2023). HKDF. Wikipedia, The Free Encyclopedia. Retrieved from <https://en.wikipedia.org/wiki/HKDF>.
- [18] Wikipedia contributors. (2023). ChaCha20-Poly1305. Wikipedia, The Free Encyclopedia. Retrieved from <https://en.wikipedia.org/wiki/ChaCha20-Poly1305>.
- [19] Nir, Y. (2015). ChaCha20, Poly1305, and Their Use in the Internet Key Exchange Protocol (IKE) and IPsec. RFC7634.
- [20] Sadio, O., Ngom, I., & Lishou, C. (2019). Lightweight security scheme for mqtt/mqtt-sn protocol. 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS). IEEE.
- [21] Serrano, R., et al. (2022). ChaCha20–Poly1305 Authenticated Encryption with Additional Data for Transport Layer Security 1.3. *Cryptography*, 6 (2), 30.