Messenger

October 2024

Contents

1 Introduction

In today's digital age, many people want to have private and anonymous communication. The reasons are: people want to speak without censorship, people want to express themselves without fear of judgment or repercussion, people want to hide their personal data, people want to prevent trackers from collecting information, etc. The privacy and anonymity issues are currently the most critical for users communicating online. Usually, well-known messengers are lacking in it. Only a few companies, such as Signal [Sig], Matrix [Mat], or SimpleX [Evg24], can offer secure communication between two parties without data disclosure. This paper will cover an analysis of the problems that arise when developing private and anonymous messengers, describe known solutions and algorithms, and give a list of the best solutions for anonymous messengers that currently exist.

2 Related work

Sometimes, privacy of a conversation means that the content of the conversation is known only to participants of the conversation. This property is relatively easy to achieve by encrypting the messages. The other property, anonymity, protects users. In general, anonymity describes actions, like sending or receiving messages, where the acting person's identity is unknown. Most of the time, the identity of a user is associated with a collection of his metadata. It can be ephemeral information, like public key or one time email, IP address, it can be a user's personal information, such as a real name, age, preferences, or health condition, it can be user location, it can be any other information like names of family or friends, it can be some secret information like bank account passwords, etc. Sometimes, anonymity is divided into recipient and sender anonymity. There are also different adversary modes with different purposes and capabilities, which should be taken into account.

Anonymity isn't cheap. It requires more time, money, and research than usual communication. Researchers have already found many ways to send a message from User A to User B securely and anonymously. Unfortunately, not without compromises.

2.1 Key Agreement

Messengers achieve privacy by using End-to-End Encryption (E2EE), i.e., encryption on a user side. Most of them use key-agreement protocols based on the Diffie-Hellman key-exchange scheme or some modification. Here are some popular key agreement protocols:

1. **X3DH** [Sig]. It is a key agreement protocol with mutual authentication, which Signal had previously used. X3DH is designed for asynchronous settings, by storing some public information on a server. Another benefit of the protocol is key compromise prevention.
2. **PQXDH** [Sig]. PQXDH is an improvement of the X3DH protocol. Signal uses it now. PQXDH provides post-quantum forward secrecy and a form of cryptographic deniability. Unfortunately, it still relies on the difficulty of the discrete log problem for mutual authentication in this protocol revision.
3. **PQDR** [Evg24]. It is a SimpleX chat SMP protocol implementation, which is a modification of PQXDH.
4. **ZRTP** [ZJC11]. This key agreement protocol was previously used by Signal. ZRTP performs a DH key exchange during call setup in the media path and is transported over the same port as the Real-time Transport Protocol media stream, which has been established using a signaling protocol such as Session Initiation Protocol. However, it requires confirming a Short Authentication String between users, which isn't always convenient.
5. **Noise framework** [Tre18]. It is a framework, which provides 59+ key exchange protocols with varying authentication and secrecy properties. All of them are a combination of standard Diffie–Hellman key exchange, with the usage of static, ephemeral, and pre-shared keys in different configurations.
6. **Ibex** [PPD23]. The protocol, which is used in Threema. it is similar to algorithms in Noise framework.
7. **TGDH** [KPT02]. TGDH is a protocol, for group key exchange. It works by

associating users with leaves in a binary tree, then computing a secret in every non-leaf node using Diffie-Hellman key exchange between its two children. The group's common secret is the secret in a root. With a created secret, it is possible to remove or append the user efficiently (in $\mathcal{O}(\log(n))$ DH key exchange operations). There are similar ideas in [CGCG+17, PPS11a].

There are also protocols, which assumes, that the key is generated on a server and then transmitted by a secure channel. Most broadcast encryption schemes are an example. This way is obviously insecure for most of the cases.

2.2 Encryption

Even if the key is successfully transferred to the receiver, the adversary still can break the system if the encryption scheme isn't secure. Also, messengers must be resistant to key compromise, which can be achieved with forward secrecy and a backward secrecy properties. They can be achieved by frequent key rotation and old keys removal.

Most messengers use a Double Ratchet algorithm, which provides forward and backward secrecy, by using a special Key Derivation Function (KDF). KDF is a basic and essential component that takes some source of initial keying material and derives from it one or more cryptographically strong secret keys. Since the Double Ratchet requires updating several keys, the best way to solve this problem is to use HMAC-based KDF (HKDF). HKDF follows the "extract-then-expand" paradigm, where the KDF logically consists of two modules. The first stage takes the input keying material and "extracts" from it a fixed-length pseudorandom key K. The second stage "expands" the key K into several additional pseudorandom keys (the output of the KDF). The following messengers use the Double Ratchet or similar scheme:

1. Signal
2. Matrix (has its own scheme name – Olm)
3. iMessage (PQ3 Protocol)
4. SimpleX
5. Towns (MLS protocol)
6. Wire (MLS protocol)

Most protocols use the original Double Ratchet scheme published by Signal. This is due to its conciseness and convenience. Creators of Matrix, simplify chains by combining sent and received chains into one (for each user, these chains have their own view, since the sender will store sent messages in the corresponding sending chain, and the recipient in the receiving chain) in which the index decides about the sender.

2.2.1 Group encryption

The next problem with encryption is group chats, where the sender wants to send encrypted messages to many users. It's a really difficult task, because, in this case, we can't use a Double Ratchet scheme. An alternative option would be to use a symmetric Ratchet, that can be used for communication between many users. The Megolm protocol in Matrix implements this approach. Each user in this protocol has a ratchet chain with four values, counter and key pair. There can be many sessions in a conversation. The public key helps to authenticate the different sessions. There will be no general

description of the algorithm, as it is essentially a part of Olm library (use only symmetric Ratchet), which is described below. But, note that this approach has some lacks in the protection of messages. Also, there is a problem of race condition, when multiple users send the message at the same time. However, there is a way to deal with it in small groups.

Other approach for group chat is Signal Private Group System [CPZ20]. The system stores group information privately, but canonicalized on a server. Clients in a group could share a symmetric key and use it to encrypt group information so that it can be stored on the service, but in a way that is completely opaque to the service. All these steps can be performed by anonymous credentials. With an anonymous credential scheme, the service could issue authentication credentials to clients. Those clients could later prove possession of a credential, as well as facts about attributes bound into the credential, without revealing anything else. In Signal's case, the issuer and the verifier would be the same party (the Signal service), which raises the possibility of using more efficient MAC-based keyed-verification anonymous credentials [CMZ14]. For a better understanding, let us describe how this scheme looks like in an example:

1. Suppose Alice has an AuthCredential for her UID, and a GroupMasterKey (only known by group members, not the server) for some particular group. The server stores an encrypted membership list for the group. Each entry in the membership list is an encryption of some UID with the GroupMasterKey.
2. To add Bob to the group, Alice must first prove to the server that she is allowed to make this change. Alice provides a zero-knowledge proof to the server that she possesses an AuthCredential matching some particular entry.
3. Alice presents a randomized form of the credential and uses Schnorr's protocol and Fiat-Shamir transform to prove a relationship to the ciphertext without revealing anything else.
4. After Alice proves to the server that she matches some entries, she sends the server a new entry encrypting Bob's UID. Alice also sends Bob the GroupMasterKey via an encrypted Signal message.

Now that Bob is a member of the group, he proves he is a member using his AuthCredential, then downloads all the entries and decrypts them with the GroupMasterKey and can see other group members.

The third way to communicate in a group is MLS protocol [BBR$^+$23], which uses TGDH as an algorithm for key agreement. Using TGDH, users build an Asynchronous Ratcheting Tree (ART), which gives a common secret group key. The key will be used to compute a secret tree. It has the same structure as the ratcheting tree. Values in the secret tree are computed from root to leaves using KDF. Values in leaves are further used to retrieve keys for application and handshake symmetric ratchets of a user associated with a leaf. The example of a Secret Tree is shown on a *fig.* 1. The first ratchet (Application Ratchet) is for encrypting and decrypting messages, using key K_i and nonce N_i, from the user, and the second (Handshake Ratchet) will be used to operate the group (encrypt commands and approvals). This protocol provides forward secrecy and post-compromise security.

Also, it is possible to create a scheme, which delegates the transmission of the messages like [Wak, Gos, TBTW$^+$22]. The idea is for one user to send the message to k random users. Then, those users will also sends message to k random users, and so on, until

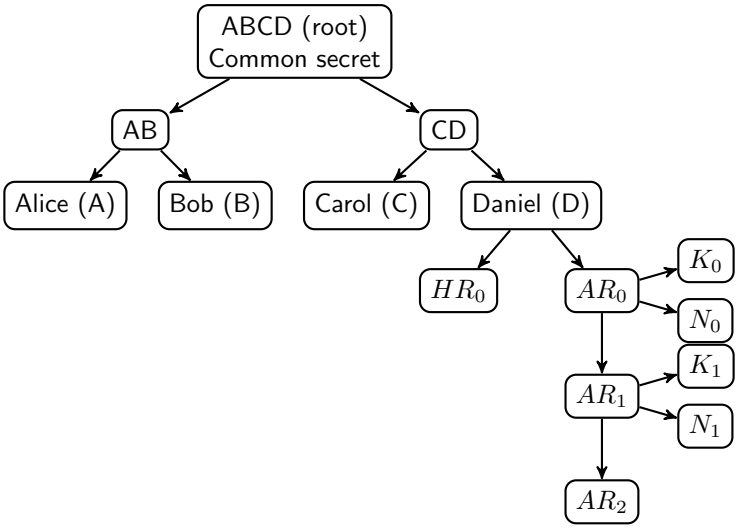

Figure 1: Secret Tree [BBR+23] for a Four-Member Group.

the user receives the message, he saw. The approach has several flaws like: the user (or his proxy) must be always online, malicious users can disrupt the system by delaying the messages or spamming, there is no forward secrecy. The advantage of the scheme is a better performance for a huge groups (10^6 users and more) in comparison with other schemes.

A separate mention should be made of post-quantum security, which is a popular topic today. Majority of the algorithms are based on key encapsulation mechanisms. So far, there are no optimal schemes that could use key encapsulation in the double ratchet algorithm. The implementation of this idea in iMessenger is suboptimal because encapsulation keys are quite large and computing with them is expensive. Of course, it is possible to achieve optimum through special settings in the OS, but in the prism of cross-platform applications, this is impossible to achieve.

2.2.2 Broadcast encryption

There is a broadcast encryption, which is similar to group chats. The main difference is the field of usage. It can be used in streaming platforms, where one sender (theoretically a server with powerful hardware in comparison to receivers) transmits messages to many users. It is satellite communications, play-TV systems, distributions of copyrighted materials, etc. Broadcast encryption requires a revocation mechanism to revoke the user's access to new messages. To reduce the message size, Broadcast Encryption (BE) schemes inherit Key Encapsulation Mechanism (KEM). Concretely, the general form of the message in Broadcast Encryption (BE) is (S, Hdr, C), where S is a set of receivers, Hdr is an object, which is used to retrieve one-time decryption key k, and C is a ciphertext, which can be decrypted with one time key k. Hdr is the main component, which brings the most bandwidth overhead. That is why, most researchers ignore other parameters and have the size of header as the main directions in research.

The main security property of broadcast encryption is collusion resistance. Intuitively, it means that a group of revoked users, can't decrypt new messages.

Definition 2.1 (Collusion security)**.** Let U be a set of all users. A broadcast encryption scheme is said to be (t, n)-collusion secure if, for any subset $R \subseteq U$, $|R| = r$, of revoked users, with $r \leqslant t$ and $|U| = n$, users in R can by no means infer information about the broadcast message.

Definition 2.2 (Full collusion security)**.** A broadcast encryption scheme is considered fully collusion secure when it is (n, n)-collusion secure.

The straightforward method for the sender is to encrypt the message with an ephemeral key, and then encrypt the ephemeral key for every legitimate receiver in the system with the receiver static key. Assume, there are n users and r of them are revoked. If the keys of legitimate users aren't compromised (what is a common assumption), the revoked user can't decrypt the message. So, the message will consist of the payload C (encrypted text) and header Hdr, which contains $(n - r)$ ciphertexts of a key for each legitimate receiver, which is too big to use in real applications. It is better to append a set of legitimate users to the message, or somehow mark ciphertexts in the header to speed up decryption, because else, users will need to use brute force, by trying to decrypt all the messages in the header.

There are different improvements to the scheme proposed above. The main idea, which improves the message length, is to divide the set of all users into different subsets, link some random independent keys to subsets, and for every subset, give the subset key to users, who are in the subset. Then the receiver in the subset can encrypt the message encrypted with the subset key. So, the sender can encrypt the message for a set of users instead of one user. The problem with the method is the number of keys in the user storage is 2^n. That is the reason, no one is using all possible subsets. However, fewer subsets will affect the size of the message. Naor et al. [NNL01] propose Complete Subtree (CS) and Subset difference (SD) schemes, which are two similar ways to choose subsets, which are mentioned above. In his schemes, users are associated with leaves of a perfect binary tree. Every node of the tree is enumerated from leaves to root. Complete Subtree (CS) scheme utilizes sets S_i. Each S_i is a set of all users in leaves of the subtree rooted at i-th node. The header length, i.e. the number of keys, is reduced to $r \log(n)$, while the user storage is $\log(n)$ keys. In Subset Difference (SD) scheme, sets are denoted as $S_{i,j}$. $S_{i,j}$ is a set of users in leaves, which are descendants of the node i, but not descendants of the node j. It reduces the header length to $2r - 1$, but user storage increased to $1/2 \log^2(n)$ keys. Bhattacherjee et al. [BS13], extend Complete Subtree (CS) and Subset difference (SD) schemes for use with k-ary trees. Here, sets are denoted as $S_{i,J}$, where J is a set of indexes. $S_{i,J}$ denote users in leaves, which are descendants of the node i, but not descendants of any node in J. It improved the header length, but increased the size of user storage. Halevy et al. [HS02] propose a Layered Subset Difference (LSD) scheme, which improves user key storage size for the Complete Subtree (CS) scheme to $\mathcal{O}(\log^{1+\varepsilon}(n))$, where $\varepsilon \geqslant 0$, but increases the header size. It reduces user storage size by using some of the sets in the Subset Difference (SD) scheme as a combination of others.

NNL-based symmetric schemes can't be used for broadcast by two users in the same system. The main reason is that the sender must know the secret symmetric keys assigned to every subset to send a message, which means, it is impossible to revoke the sender. With asymmetric schemes, everyone who knows the public key can broadcast the messages. Dodis et al. [DF03] describes how to use an Identity-Based Encryption

(IBE) scheme with Complete Subtree (CS), Subset Difference (SD) and Layered Subset Difference (LSD) schemes. Here, identity is the identity of a subset, so the user will store keys for identities of subsets. Unfortunately, it increases the user storage size.

The next problem with NNl-based schemes, is that they work best when $r << n$. In another case, when $(n - r) << n$, those schemes give a trivial solution, namely encrypting the broadcast message under each recipient's key. Boneh et al. [BGW05] propose a Public Key Broadcast Encryption Scheme which is good to broadcast to small sets of users compared to n.

The Boneh-Gentry-Waters scheme [BGW05, BGW06] is considered one of the simplest and most flexible, full-collusion resistant implementations using public key and pairing. They operate with parameter B, which scales the public key and ciphertext to the desired size, settings $= \lfloor\sqrt{n}\rfloor$ that enables to achieve the optimal public key and ciphertext sizes of $\mathcal{O}(\lambda\sqrt{N})$. It has good estimates on BN256, but note that this scheme is suitable for encrypting a small set of people from the whole system. This scheme can also be modified to achieve CCA security, but its main disadvantage is that it is vulnerable to adaptive adversaries. in [BGW05], an "adaptive adversary" is the adversary, which could request user keys adaptively.

The term "adaptive adversary" is conceptual and isn't standardized. However, there is a work by Phan et al. [PPS11b], which describes adaptive adversaries with different levels of adaptiveness. To be precise, they propose IND-Dynx-Ady-CCAz security, where x-Dynamic describes user join operation usage, y-Adaptive describes user corrupt operation usage, and CCA-z describes key decapsulation operation usage. x, y, and z variables denote the level of operations usage, i.e. when and how they can be used.

Another example of asymmetric BE scheme which uses a binary tree is the Interval scheme of Lin et al. [LCL$^+$10]. It uses interval of users $\{i, i + 1, i + 2, ..., i + k\}$ as a set for subset cover. Another example of Public Key Broadcast Encryption Scheme is Combinatorial Subset Difference (CSD) of Kim et al. [KLLO17], which instead of a binary tree, represents a set as a binary mask with the exclusion of another mask. There is a secret key size improvement of the scheme by Lee et al. [LKO19]. The scheme name is BESTIE.

The Public-key BE scheme with dynamic properties [DPP07] is a full-collusion resistant scheme, which allows decoders to store constant-size decryption keys and features $\mathcal{O}(r)$ ciphertext size. This is of particular interest when r is small (i.e. $r < \sqrt{n}$). The scheme can provide encryption for a huge number of users. It also supports a dynamic structure, but if the number of revoked users is large, the complexity of the computations will increase. There are two versions of this scheme: the first has a group manager responsible for inviting new members to join or permanently removing undesired members in a very efficient way, and the second, where the broadcaster and the group manager are the same entity. The second version is more dynamic due to the non-linear size of the group encryption key.

In an IBE system, the public key of a user may be an arbitrary string like an e-mail address or other identifier. This eliminates certificates altogether. The sender could just encrypt the message with the identity of the recipient without having to first obtain his public key (and make sure that the obtained public key is the right one). Of course, users are not capable of generating a private key for an identity themselves. For this reason, there is a trusted party called the private key generator (PKG) who does the system

setup (which leads to the key escrow problem). To obtain a private key for his identity, a user would go to the PKG and prove his identity. The PKG would then generate the appropriate private key and pass it on to the user. A hierarchical identity-based broadcast encryption (HIBBE) [YZW+14] scheme is an identity-based broadcast encryption (IBBE) scheme in a hierarchical environment. Hierarchical Identity-based broadcast encryption (HIBBE) allows a root PKG to distribute the workload by delegating private key generation and identity authentication to lower-level PKGs. In a HIBBE scheme, a root PKG only needs to generate private keys for domain-level PKGs, who in turn generate private keys for users in their domains at the next level. Authentication and private key transmission can be done locally. Another advantage of HIBBE schemes is damage control: disclosure of a domain PKG's secret does not compromise the secrets of higher-level PKGs. One way to deal with key escrow is trust distribution by multiple trust authorities [BDS10].

The Privacy-preserving identity-based broadcast encryption scheme [HPH12] is a very interesting scheme that solves the problem of privacy for users receiving broadcast content. Recall that Identity-based BE schemes are public key crypto systems that can use any string as a public key of each receiver. This scheme uses bilinear pairings and the one-way anonymous key agreement approach to achieve privacy for receivers. The disadvantage of this scheme is that decryption time may be linear in the number of receivers because the decryption algorithm must try decrypting the ciphertext until it retrieves c_i and decrypts successfully (a recipient must attempt $\frac{n}{2}$ decryption trials on average). This problem could be solved efficiently by labeling each c_i with shared secret information s_i, so it doesn't require any decryption trials.

The BE using Sum-Product decomposition of Boolean functions scheme [DA24] is a new approach that relies on simple and consensual security assumptions (resilience of a block cipher and PRF) with lower bandwidth requirements, at the price of a significant increase in key storage and potentially heavy computations at the emitter facility. This scheme is similar to the adaptive CS where the binary tree structure is regenerated at every broadcast in order to optimize bandwidth, but it doesn't compute a partition of the set S. The set S is considered as a Boolean function and is computed with a $\Sigma\Pi$-factorization, yielding subgroups S_i that cover S. It doesn't preclude one user from belonging to two distinct S_i's as this does not affect the security. For large values of l (the number of variables in the function f), the computation of the $\Sigma\Pi$-form of f quickly becomes the bottleneck of this procedure. For $l > 12$ Quine-McCluskey can't work efficiently to find the smallest sum of products, so authors of the paper, highly recommend alternative algorithms (like ESPRESSO). But, in that case, these alternative algorithms can find a good but possibly non-optimal solution. In conclusion, this algorithm uses fascinating techniques, but it is completely non-scalable to a large set of users.

The problem with the centralization of setup is hard to resolve. There are several proposals to deal with it. Qianhong et al. [WQZ+11] presented a Contributory Broadcast Encryption (CBE) primitive, which is a hybrid of Group Key Agreement (GKA) and Broadcast Encryption (BE). The main idea is to use Aggregatable Broadcast Encryption (ABE). They use an encryption scheme with key-homomorphic property, i.e., two pairs of asymmetric keys (sk_1, pk_1) and (sk_2, pk_2) can be aggregated to $(sk_1 \odot sk_2, pk_1 \otimes pk_2)$. Because of the way the broadcast scheme works, users don't need to know other users secret keys. The overhead of the setup procedure is $\mathcal{O}(n^2)$. However, in most cases, this pro-

cedure needs to be run only once and this can be done offline before online transmission of secret session keys. The scheme can be run assuming that the communication channels between members are authenticated during the setup stage. The setup sub-protocol requires only one round to send the public key to all users. The general structure of decentralized Dynamic Broadcast Encryption (DBE) of Phan et al. [PPS11a] is another example of a decentralized Broadcast Encryption (BE). Their concrete scheme uses a Complete Subtree (CS) [NNL01] with a 3-to-8 parallel Diffie-Hellman key exchange (similar to Tree-based Group Diffie-Hellman (TGDH) [KPT02]) and Cramer-Shoup Public Key Encryption (PKE). In general, the complexity of the proposed algorithm is similar to the standard Complete Subtree (CS) scheme [NNL01]. The difference is in the appending of a user, which requires $\log(n)$ key exchanges, while the revocation of a user requires $\log^2(n)$ key exchanges at worst. Kolonelos et al. [KMW23] proposed two Distributed Broadcast Encryption based on BGW [BGW05] and another BE, which is similar to BGW. They are based on two different assumptions: BDHE and k-Lin. The idea of the first one is to compute the master key t as $t = \sum_{i=1}^{n} t_i$, where t_i is chosen by the i-th user. The user will compute a secret key for him and cross-terms for others to decrypt messages. The second one uses the fact that it is possible to "push" a part of a secret key to public parameters and mix them with some user-created secret value. Those schemes have "distributed" in their name, but all the users participate in the creation of public parameters.

Broadcast Encryption (BE) algorithms with their properties are shown in tables *tab.* 1 and *tab.* 2.

There are schemes similar to BE. For example, Corrigan-Gibbs et al. [CGBM15] presented Riposte. It allows many users to anonymously post messages to a shared "bulletin board", maintained by a small set of minimally trusted servers (at least one of them is honest). Riposte can be used to anonymously publish comments or emails. The main idea is to use homomorphic (s, t)-Distributed Point Function ((s, t)-DPF), which allows you to divide the message into s parts in a way, that only by having $(t + 1)$ parts, the message can be recovered by computing the sum of DPF Eval function outputs. With the use of the $(s, s - 1)$-Distributed Point Function, the user can divide his message into s parts, encrypt i-th part for i-th server and send it. Upon receiving the part P_i, the server computes Eval(P_i) and saves the result in the database. Then at the end of the epoch, for example, 4 hours, servers publish their databases and compute the final result, which is a set of published messages. It is impossible to find out who actually published the concrete message by the final result. The more servers wait, the more users will participate and the harder it will be to find the location of the sender.

2.2.3 Encrypted calls and video-calls

Encrypted calls and video calls seem to be the next most difficult level after encrypted text chats. They require stable encryption of packets (in the case of video calls, it can be two streams), control of delays for each user, and your bandwidth. Those properties are necessary to ensure two things: end-to-end encryption and scale to many participants.

For this, Signal wrote a new SFU (Selective Forwarding Units) from scratch in Rust. It scales to 40 participants with ease (perhaps more in the future). It works as follows: Each participant sends its media to a server. The server "forwards" the media to other

participants without viewing or altering it. It works better than Server mixing, which isn't compatible with end-to-end encryption, and Full mesh, which does not scale to many participants. The obstacles to the implementation of SFU are as follows:

- The capacity of each participant's Internet connection is constantly changing. SFU must constantly and carefully adjust how much it sends each participant to be "just right".
- The SFU cannot modify the media it forwards; each participant must send the SFU multiple resolutions of video, and the SFU must constantly and carefully switch between them.

So, the solution of these problems is to combine the following techniques:

- **Simulcast and Packet Rewriting**. Simulcast – is the simultaneous sending of layers by each participant to the SFU. This enables SFU to switch between different resolutions. But, because the VP8 video codec is the most used among the devices and since VP8 doesn't support SVC, the SFU must do something to transform 3 layers into 1. Signal uses packet rewriting – is the process of altering the timestamps, sequence numbers, and similar IDs that are contained in a media packet that indicate where on a media timeline a packet belongs. It transforms packets from many independent media timelines (one for each layer) into one unified media timeline (one layer).
- **Congestion Control**. It is a mechanism to determine how much to send over a network. The inputs into the stream pipeline are data about when packets were sent and received, which are called acks. The outputs of the pipeline are changes in how much should be sent over the network, which we call the target send rates. To briefly describe the scheme, the recipient sends periodic updates when he has received the package. And the sender combines this information with his own, calculating the delay time.
- **Rate Allocation**. It helps determine which layers need to forward. If the budget isn't enough, users must prioritize. To aid in prioritization, each participant tells the server what resolutions it needs by requesting a maximum resolution.

End-to-end encryption is performed on the client side (completely opaque to the server). Signal implementation exists in the RingRTC library. The contents of each frame are encrypted before being divided into packets (by using AES256-CTR), similar to SFrame. The key distribution and rotation mechanism occurs as follows:

1. When a client joins the call, it generates a key and sends it to all other clients of the call over Signal messages (which are themselves end-to-end encrypted) and uses that key to encrypt media before sending it to the SFU.
2. Whenever any user joins or leaves the call, each client in the call generates a new key and sends it to all clients in the call. It then begins using that key 3 seconds later (allowing for some time for clients to receive the new key).

Unfortunately, this implementation is the only one described in detail so far. Usually, the protocols refer to the WebRTC library, which they use for encrypted calls and video calls. The description of this library is beyond the scope of our paper, but a few words about E2EE should be noted. Web Real-time Communication is a profound, flexible streaming protocol, and an open-source technology suitable for offering uninterrupted and bidirectional messaging, audio, and video chats in real-time between browsers and devices, but it has some vulnerabilities of security. When your computer establishes

a connection using WebRTC, it creates a proprietary, peer-to-peer pathway. At the beginning of the peer-to-peer connection establishment, both devices agree to bypass certain security measures, like firewalls and network access translation (NAT) devices to get the most direct connection path possible. This can bypass your standard security measures, which creates a potentially unsecured connection. That is why recommended to use WebRTC with E2EE. There are three mandatory WebRTC encryption specifications: Secure Real Time Protocol (SRTP), secure encryption key exchange, and secure signaling. These protocols encrypt the data sent through WebRTC, protect the encryption keys, and secure the web server connection. All of these encryption specifications are required for every WebRTC session.

2.3 Authentication

User authentication is a process that verifies the user's identity. Similarly, message authentication verifies that a message wasn't altered. In general, authentication is a challenging task. If we are working in the space of anonymous messengers, where servers don't store confidential information about users, or at least information that can confirm identity, then it is impossible to use this system to confirm that you are communicating with the correct user because of the "Man-in-the-middle" attack. It is generally recommended to use alternative encrypted communication channels or to meet in person to verify the shared secret. That is the reason messengers like Signal, Matrix, Status app, SimpleX, etc. are using out-of-band messages, like QR-codes or links, for verification.

In some cases, such as ZRTP, where authentication is completed prior to communication with usage of a hash commitment scheme, the system can generate a Short Authentication String (SAS). Then it can be interpreted into emoji. The use of hash commitment in the DH exchange constrains the attacker to only one guess to generate the correct SAS in his attack, so SAS can be quite short. A 16-bit SAS provides the attacker only one chance out of 65536 of not to be detected.

2.4 The problem of initial key exchange

The problems of authentication and possible man-in-the-middle described in the previous section, caused by an earlier problem – initial key exchange. If the initial exchange is correct, it means that your communication channel is established with the right person.

In order to get the correct public key of another user, which no one could have changed along the way, messenger must have a service that contains these ID and key mappings. It means that messenger should operate with user's identity. But we are interested in anonymity, so we have two approaches: use zero knowledge proofs with sophisticated algorithms or send a minimum amount of personal data, most likely pre-encrypted, which will not critically affect the user's anonymity. The best system for us would be one that use only the first approach or a combination of both approaches.

The service of messenger names and keys can have a similar structure to ENS (Ethereum Name Service). ENS is a decentralized system that runs on the Ethereum blockchain and which translates cumbersome Ethereum addresses into readable names. We assume that there may be better examples of approaches than ENS, because the main drawback of that is its public nature.

Next, we want to describe in more detail how the key exchange will be performed. Suppose user A wants to establish connection with user B, with whom it is in the same chat room. First, user A requests the user's public key from the service (to avoid any problems with incorrect nicknames, he can contact the service through the same group where they both are). Before the service starts searching for the public key, it asks user B if he agrees to be contacted by user A. If user B give a negative answer, the procedure is canceled. In another case, if he accepts the connection, user B creates proof that this is really his account, and it would be interesting if only by using this proof the system can obtain the public key from the name and key service. Finally, user A receives the public key of user B. It is possible for a service to encrypt the public key of user B by the public key of user A if we want to protect that.

2.5 Anonymity

It is hard to achieve anonymity in Messenger because strong anonymity unavoidably leads to decrees in performance and usability, or can affect something else. Also, the messenger must maintain security, i.e. protect user data on the network, and prevent malicious users from disrupting the system. For more, there is a problem with tracking the malicious users, because the system is anonymous.

In some cases, anonymity can be evaluated in terms of anonymity set [PK01]. Anonymity set is the set of users who, from the perspective of an adversary, may plausibly be associated with a message or transaction of interest. Transactions are more anonymous when the anonymity set is larger, and less anonymous if the anonymity set is smaller. This simple definition leads to conclusion, that the big number of users in a system is the main component of anonymity. Also, anonymity can be defined as amount of metadata disclosed [JSH24].

The first problem with anonymity is registration [JSH24]. Some centralized systems require a phone number or email to register a user. The use of a phone number, as the basis for ownership of an identity key, weakens security against user accounts being compromised. This weakness stems from the centralized service providers (i.e. telecommunications service providers), because they have direct control of specific users' phone numbers. Additionally, methods such as SIM swapping attacks, service provider hacking, and phone number recycling may be exploited by lower-level actors. Two-factor authentication through a phone call and registration PIN lock means that an attacker needs access to both the phone number and the registration PIN code to modify identity keys. However, in most messengers the feature is off by default, and for more, for users to be able to recover the registration PIN code, it will turn off after some period of user inactivity. As the advantages of using the phone number for registration, is that it prevents Denial of Service (DoS) or Sybil attacks.

Using phone numbers as the basis for account registration also greatly weakens privacy for typical users. In most countries, users must provide personally identifying information such as a passport, driver's license or identity card to obtain a phone number. It permanently maps users' identities to their phone numbers. These identity mappings are kept in private databases, which can be queried by governments or the service providers that own them. There are also a number of web scrapers and indexers which automatically scrape phone numbers associated with individuals. These scrapers may target sources

such as leaked databases, public social media profiles, and business phone numbers in order to link individuals to specific phone numbers. Since the primary method of initiating contact with a user is to know the user's phone number, this immediately strips away user anonymity. Other detractors of phone number account systems include: limiting the ability of a single user to establish multiple identities; and preventing high-risk users without access to a phone number from accessing the service.

The next anonymity problem is location. Messengers need to know the destination of a packer to deliver it, but the sender's location isn't necessary. A great solution is the Sealed Sender algorithm from Signal [Sig]. The main idea is to hide all information about the sender by encrypting the header and the body of the message. The server will only know to whom the message must be sent. The solution mentioned is good, but a stronger adversary can track the messages. The most common way to prevent it, is to redirect the user's messages through the sequence of other users, trusted servers, or some combination of them. Those servers can mix the messages and increase correlation complexity. Simplex Messaging Protocol (SMP) [Evg24] is a protocol, which uses one centralized server, to store and redirect messages. The server, has no need to know what those messages are or their format, so in addition, they should be end-to-end encrypted. To receive messages, the receiver creates a "queue" (a data abstraction for one way chat) and sends an out-of-band message to the sender. Then the sender can create his one way chat for bidirectional communication. For more, it is possible to send noise traffic to hide the time, when you are communicating. However, the construction puts a lot of trust into the server. The more trust distributed way to hide the user location is Tor [DMS04]. To send a message via Tor, the user makes preparations by choosing the number of Tor nodes (three nodes is enough) and establishing a connection with them via the Diffie-Hellman key exchange. The connection to the next node is done by redirecting the message from the previous node. Those servers will be arranged in the circuit (chain), which leads from the user to the receiver. At the creation of the circuit, the user and Tor node agree on a common secret key k_i, where i is the order number of the node in the circuit. Then, to send a message, the user encrypts that message for the circuit, with key k_i like an onion wrapper:

$$C = \mathrm{Enc}(k_1, ... \mathrm{Enc}(k_{n-1}, \mathrm{Enc}(k_n, M))...).$$

After the ciphertext C computation, the user can send the message to the first server, which decrypts it and sends the result to the next server in the sequence until it reaches the receiver. Tor has no end-to-end encryption by default, but sender and receiver can agree of some by themselves, to hide the message at the last node.

Tor provides a mechanism for hidden services through rendezvous points. Rendezvous points are a building block for location-hidden services (also known as responder anonymity) in the Tor network. Location-hidden services allow Bob to offer a TCP service, such as a web server, without revealing his IP address. This type of anonymity protects against distributed DoS attacks: attackers are forced to attack the onion routing network because they do not know Bob's IP address. Tor provides location-hiding for Bob by allowing him to advertise several onion routers (OR) (his introduction points) as contact points. Alice, the client, chooses an onion router (OR) as her rendezvous point. She connects to one of Bob's introduction points, informs him of her rendezvous point, and then waits for him to connect to the rendezvous point. The extra level of indirection also allows Bob to respond to some requests and ignore others.

Unfortunately, different research papers have shown [BJE+19] that Tor is vulnerable to traffic analysis, i.e., Replay Attack, Cell Counter-Based Attack, Correlation-based Traffic-Analysis attack, HTTP Pattern injection Attack, Probabilistic Attack, Packet timing watermarking attack, etc. Some attacks become infeasible because of the bigger number of Tor nodes today. Still, there are attacks, which allow adversaries controlling at least two nodes in the same circuit (first and last) to identify the user, who created the circuit. With the number of nodes in the Tor network, a targeted attack is improbable. The more nodes, the adversary can corrupt, the more powerful the adversary is and the higher the probability of attack. There is also a problem with directory authorities (containing a list of running nodes) and measurement authorities (measuring the bandwidth of nodes) [DHK21], which aren't fully decentralized. However, Tor has a good compromises between anonymity and performance. That is why Tor is a great tool, which is currently in use. For example, Ricochet [Ric] and Tox [Tox17] are messengers built on top of the Tor network. They utilize Tor's rendezvous points mechanism to provide anonymity for both users.

Traffic analysis resistant systems can be achieved by using cover traffic. In Vuvuzela [vdHLZZ15], this is done by increased bandwidth overhead and increased latency. Like Tor, Vuvuzela messages are encrypted in an onion wrapper for a set of servers, which hides the user's location. The main difference is that the user sends and receives some messages constantly (approximately, one per minute), which makes traffic analysis hard. Most of them are encrypted garbage text. Constant message flow, hide information about users. That is why the system works only if the user is always online. For real applications, the model of the adversary should be lightened. Nym [DHK21] is also a Tor like structure with cover traffic. Nym nodes send cover traffic at random intervals that follow a Poisson process. Cover traffic is optional for end users and services. Also, Vuvuzela and Nym are mixing user messages.

Those results can be generalized. There is an "Anonymity trilemma" [DMMK17], which claims, it is impossible for a communication system to have Strong Anonymity, Low Bandwidth Overhead, and Low Latency at the same time. Informally, they say, that no protocol can achieve strong anonymity if $2l\beta < 1 - \epsilon(\eta)$, where $\epsilon(\eta) = \frac{1}{\eta^d}$ for any positive constant d, where l is latency overhead and β is bandwidth overhead. They have more constraints for different scenarios and adversary models, for the receiver and sender.

Another problem in Tor is lack of incentivization. Tor's nodes are run by volunteers. Economical encouragement will make the network sustainable and healthy. For example, Nym [DHK21], Session [JSH24], Status [Sta], towns[Tow] have their own blockchains. In those systems, nodes are rewarded for redirection and encryption of messages. The disadvantage of it is the cost of the message.

All that remains is to prevent two problems: spoofing and spam. Each user can get a short-lived sender certificate that the sender can add to an encrypted message to prove their identity. To prevent abuse or spam, clients derive a 96-bit delivery token from their profile key and register it with the service. The service requires clients to prove knowledge of the delivery token for a user in order to transmit "sealed sender" messages to that user. But, note that the user should exchange tokens via another encrypted channel or in person at a meeting. Of course, the application supports a mode without token confirmation, but it enables us to receive incoming "sealed sender" messages from

non-contacts.

2.6 Session management

The use of the messenger shouldn't be limited to just one device. However, multi-device support brings a lot of complications, which can be avoided other way. The main problems are:

1. Multiple devices should share the same chat history, state, etc.
2. Messenger should provide Forward Secrecy (FS) and Post-Compromise Security (PCS).
3. At a certain time, the majority of user devices can be offline.
4. There should be a mechanism that combines user devices into a single entity.
5. Multi-device system shouldn't disclose user metadata (number of devices he has).
6. Multi-device system should support device revocation.

For multi-device messengers, it is important to differentiate between identities on the device level, where every device has its unique key, and the user level, where all devices use one key [DGGL21]. If a messenger utilizes identity on a device level, then, to send a message, the sender he is required to know all the device keys of the certain user, and send a message to all of them. On the other hand, user level identity usage hides all the communication between user devices. It can be done in different ways. There might be a one central device, which will act as a proxy, and mirror messages between user devices and other users (the central device is required to be always online). Also, there is a way to organize devices in a group.

Users might want to have a proof that the devices they are talking with belong to the same user. By utilizing identity on a user level, there is no need for proof at all, because of one key. If the messenger utilizes device level identity, then it is required to provide additional verification through cross-signing, providing with identity key, or other ways.

As soon as users add a new device, the chat history made by the already existing devices is considered as old [DGGL21]. For best usability, users expect to see their old chats on the new device. However, this brings different complications. Some messengers, like Threema and WhatsApp, have a mechanism to mirror all messages on a new device, but some of them, like iMessage, Signal, Wickr, and Wire, do not synchronize old chats with a new device, because of the privacy concerns.

2.7 Decentralized messaging systems

Most of the usual messengers are centralized. The main reasons are:

- Centralized systems are easier to create and maintain.
- Centralized systems are faster and give a better user experience.
- Centralized systems are cheaper.

In comparison with decentralized systems, centralized systems have next vulnerabilities:

- They have one point of failure.
- They are easier to compromise.
- They are more vulnerable to Denial-of-Service (DoS) attacks.
- Messenger owners and creators can disclose your data without your concern.

That is why, some creators of anonymous messengers are trying to build a decentralized system, by encouraging people to host their servers (nodes) creating a decentralized network. Sometimes it is voluntarily, and sometimes, there is a monetary encouragement.

Waku [Wak, TTBC22] is a family of peer-to-peer communication protocols that enable privacy-focused messaging. The Waku Network (TWN) consists of independent nodes running Waku protocols. The main protocols of Waku are:

- 14/WAKU2-MESSAGE, which describes the format of the message.
- 23/WAKU2-TOPICS, which describes the format of topics of the message. It helps to redirect and filter messages.
- 11/WAKU2-RELAY, which describes the way, nodes can communicate with each other. The node has two types of connection: connection for full messages and connection for gossiping. Gossiping is a process when one node says to another node, that she saw some message on some topic. Any node can request the full connection to get the message if it doesn't have it. It is a way to be sure that the network is connected. However, the communication channels are assumed to be secure.
- 17/WAKU2-RLN-RELAY and 32/RLN-V1 are WAKU RELAY extensions, which use Rate Limiting Nullifiers (RLN). Rate Limiting Nullifier is a protocol, that restricts the possible number of messages sent by the user to one per epoch (for example, 10 seconds) [TBTW$^+$22, Vac].
- 12/WAKU2-FILTER. A lighter-weight version of the relay protocol for resource-restricted devices, Waku Filter enables light nodes to only receive the messages they want from full nodes.
- 13/WAKU2-STORE. It enables devices not participating in the Waku Relay network to retrieve messages they missed while offline.

The Waku network isn't static. It can change, because of new users, or because some users decide to go online. That is why Waku has a peer discovery mechanism. In default implementation the ideal network peering degree, i.e., the number of connected peers, to which you send full messages, is 6 with anywhere from 4 to 12 being acceptable, so if the node peering degree is less than 4, it should find more peers, and if the node peering degree is more than 12, it should randomly remove some of them. In libp2p's implementation, each peer performs a series of checks every 1 second. There are four mechanisms to find a new node:

1. Static Peers. One can establish connections with the node using the node details, which are predefined (hard-coded) into one's node.
2. DNS Discovery. One can find a node in the DNS server and connect to it like to the static peer.
3. Discv5. Discv5 is a decentralized and efficient peer discovery mechanism for the Waku Network. It uses a Distributed Hash Table (DHT) to store Ethereum Node Records (ENR), providing resistance to censorship.
4. Peer Exchange. Peer Exchange enables requesting random peers from other network nodes without revealing information about their connectivity or neighborhood.

2.8 Spam protection in decentralized anonymous system

In centralized systems, creators can control spammers by detecting them and removing or restricting their access to the system. This approach can be used in decentralized system through voting, but anonymity makes it hard to track the spamer. One way to prevent spamers is to use a consensus protocol. For example, use Proof of Work (PoW) to make messaging a computationally costly operation, hence lowering the messaging rate of all the users, including the spammers.

Some messengers, like Waku[TBTW$^+$22, Vac], DarkIRC [dar24], etc., deal with spam using Rate-limit Nullifiers (RLN). The mechanism restricts the number of messages sent to one per epoch. For example, Waku has a restriction of 1 message per 10 seconds. It is done by utilizing Shamir's Secret sharing. It makes restoring a user's identity key possible if he sends more messages than the system defined. To prevent users from spamming, on the registration stage, the user sends some assets to the smart contract, so revealing his identity key allows others to withdraw his assets, making spam attacks expensive. Shamir [Sha79], proposed a simple (k, n) threshold scheme based on polynomial interpolation. Given k points in the 2-dimensional plane (x_i, y_i), $i = \overline{1, k}$. with distinct x_i's, there is one and only one polynomial $q(x)$ of degree $k-1$ such that $q(x_i) = y_i$ for all i. So, one can choose polynomial $q(x) = a_0 + a_1x + a_2x^2 + ... + a_{k-1}x^{k-1}$, where $D_0 = a_0$ is a shared secret and $D_i = q(i)$, $i = \overline{1, n}$ are secret keys given to n users. Then, with k secret keys D_i, it is possible to recover a shared secret.

Waku and DarkIRC utilize $(2, n)$-Shamir secret sharing to derive a share (x, y) of its private identity key sk. They, also, compute other values, which will be sent along with the message and (x, y). Finally, a user needs to prove in a zero-knowledge manner (through zkSNARKs) the following:

1. He belongs to the group, i.e., provides proof of membership. In Waku, he proves, that his secret key sk belongs to the identity commitment Merkle tree. However, Waku developers [TBTW$^+$22, Vac] are trying to replace it with distributed hash tables.
2. (x, y) is a valid share of its identity key.
3. Additional values are correctly calculated.

Another way to deal with spam is reputation. Users can ignore messages from other users, which have a low reputation. For better flexibility, one can use different thresholds. For example, if the user has a reputation less than the minimum threshold, ignore all his messages, but if it is greater than the minimum threshold but less than the publishing threshold, like in Waku, ignore all his messages, but accept all redirected messages. Trust is a metric with a similar purpose. Sometimes, $\text{trust}(A, B)$ is a boolean value, indicating if user A fully trusts user B. The better approach is to use trust, which results in a number in $[0, 1]$. It gives better flexibility.

Some researchers [ZG05] use distrust along with trust, which is the opposite of the last. It doesn't mean low trust. Distrust is one of the most controversial topics. Most approaches completely ignore It. If we decide to associate distrust as negative trust, i.e. $\neg\text{trust}(x)$, then we might get the next claim:

$$\neg\text{trust}(A, B) \land \neg\text{trust}(B, C) \models \text{trust}(A, C)$$

It is similar to a proverb, which says that the enemy of your enemy is your friend. In

some cases, it has a rather unwanted effect. The next issue is in the following claim:

$$\neg\text{trust}(A, B) \wedge \text{trust}(B, C) \models \neg\text{trust}(A, C)$$

This claim is more than questionable. Here, user A penalizes user C, simply for being trusted by a user, that A distrusts. This assumption is not sound and does not reflect expected real-world behavior. To avoid such unwanted results, one can scale or normalize the trust value, or simply not allow untrusted users to distribute trust at all.

To apply a reputation or a trust mechanism, the network must be associated with a directed graph. Every node is a user, and the edges between them are associated with their connections. Those edges can be weighted, meaning trust level or score between two users.

The computation of reputation can be done in different ways. Trust metrics may basically be subdivided into ones with global, and ones with local scope. Global trust metrics take into account all peers and trust links connecting them. Global trust ranks are assigned to an individual based on complete trust graph information. The basic intuition behind the approach is that nodes should be ranked higher the better the rank of nodes pointing to them. Trust metrics with local scope, on the other hand, take into account personal bias. Some researchers claim that only local trust metrics are "true" trust metrics, since global ones compute overall reputation rather than personalized trust. The second division refers to the place where trust relationships between individuals are evaluated and quantified. Local or centralized approaches perform all computations in one single machine and hence need to be granted full access to relevant trust information. The trust data itself may hereby be distributed over the network. Also, there are scalar and group trust metrics. Scalar metrics compute trust between two given users A and B, taken from set V of all users. On the other hand, group trust metrics generally compute trust ranks for sets of individuals in V or compute a set of trusted individuals.

In Gossipsub [Gos], which Waku is built on, the reputation is a local parameter computed locally. Score, describes the trust of user A in user B. It computes as a sum of parameters, that differ for different message topics t_i, which are publically visible. Message topics are strings, which are used for message routing. The formula is the next:

$$\text{Score}(B) = \text{TopicCap}\left(\sum_i t_i \sum_{j\in\{1,2,3,3b,4\}} w_j(t_i)P_j(t_i)\right) + w_5P_5 + w_6P_6 + w_7P_7$$

The parameters are defined as follows:

- w_j: weights of message for j-th parameter. It can be dependent on the topic t_i.
- $P_1(t_i)$: Time of the user in the network while listening to the messages on the topic t_i.
- $P_2(t_i)$: The number of messages on topic t_i first delivered by the user.
- P_3: Network Message Delivery Rate for a topic.
- P_{3b}: Network Message Delivery Failures for a topic.
- P_4: The number of invalid Messages for a topic from the user.
- P_5: Application-Specific score. This is the score component assigned to the user by the application itself, using application-specific rules.
- P_6: IP Colocation Factor, which is a threshold for the number of users using the same IP address.

- P_7: Behavioral Penalty.

In Gossipsub, every node internally maintains the reputation of its neighbors whenever an event of interest occurs. At a new connection of user A to user B, if they have ever met, user A can retrieve the score from the memory. If it is their first connection, user B will be assigned a new, default score, even if he was in the network for a long time.

There are alternatives to the approach that allow taking into account the opinion of others. The obvious downside of the approach is the fact, that you need to give some trust in other users. For more, the computation requires the knowledge of the network topology. Those alternatives are EigenTrust, Appleseed, TidalTrust, and others.

In comparison to Gossipsub, EigenTrust [KSGM03], user A, to compute the score of user C, firstly, compute general satisfaction function s_{AB} of neighbor users B. Satisfaction is a cumulative function, which for each interaction of A and B changes by the value in $\{-1, 0, 1\}$. Then the normalized values of general satisfaction, named normalized local trust values, will be used to compute the trust that user A places in user B. To compute the score of user C, user A, will run an algorithm, which computes the score using those opinions and normalized trust values. There will be values that the user A can't compute, so he will ask his neighbors to give them. Actually, The user A will ask every user in the network.

Ziegler et al. describe Appleseed [ZG05], which computes local group trust, i.e., for a given user, accuracy, and spreading factor, returns a function, which for each member of the group V, gives some score value. They are computed as energy flow distribution. Every user will receive some part of the energy. Then the user will save some portion of the energy for himself, and send the remaining part of it to its neighbors. Also, Appleseed has a mechanism to deal with distrust.

Further improvement of algorithms mentioned above is the Transitive Reputation Algorithm [Kir24]. It is designed to compute and propagate trust in a decentralized manner by leveraging the concept of relative trust between nodes. Here are the properties, that it satisfies:

1. Non-Destructiveness. Every attestation of trust from one peer to another should result in the recipient's trust score increasing or remaining the same from = the perspective of every other peer in the network.
2. Sybil-resistance and robustness towards malicious collectives. No peer should be able to create another peer whose reputation is greater than that of the peer itself, and no collective should be able to create a peer whose reputation is greater than that of the collective's most reputable member.
3. Balanced incentive structure. No peer should be able to raise or lower its trust score from the perspective of any other peer. Furthermore, no peer should be able to alter its standing relative to another peer from the perspective of a third observer.

Mathematically, this can be reduced to graph theory by describing each property in terms of a set of edges, a set of nodes, and weights. In addition, it uses a trust policy, which determines whether the set of weights originating from a given user is valid. Previously mentioned algorithms do not satisfy those properties fully. To calculate the node's reputation score, the algorithm finds nodes with maximum accumulated scores. Then, passing through all the nodes of such value, it makes a passing score down. As a result, the algorithm returns the score of the node.

A number of modifications can be made to the proposed algorithm to improve its

robustness:

- Transitive Trust Algorithm with Distrust (add negative reputation);
- Transitive Trust Algorithm with Node Disqualification;
- Telescoping Transitive Trust Algorithm (amplifies the change in reputation caused by each one of n nodes of similar reputation attesting positively to a single node as n grows large). The same accumulation method can be applied to the above variants.

There are other uncommon ways to compute the reputation. For example, reputation in Nym [DHK21] is proportional to the user's stake.

3 Discussion, comparison, results

Summing up the research, we can say that achieving the necessary properties, such as privacy and anonymity, requires a comprehensive approach. The main challenge is to combine all stages and structures, which is required to develop chats and messengers. This stems from the problem of handling data that shouldn't be stored by servers and the fact that this data is encrypted.

It is worth noting that some structures have no alternatives, while others, on the contrary, have a fairly wide list of possible variations. Unfortunately, they are all imperfect and try to balance security concerns.

There are a lot of different key agreement protocols. For one-to-one chats, the majority of anonymous messengers are using X3DH. The main reason is the possibility of sending a message to the offline user. The security properties of the protocol have been proved several times. Video calls and audio calls can be done only online. That is why messages are encrypted differently. The best algorithm is ZRTP.

The best choice for one-to-one message encryption is Double Ratched. It is resilient, it has backward secrecy, forward secrecy, and Break-in recovery properties. Also, it can work in asynchronous mode. Other algorithms are similar or are an extension of the Double Ratched, with over-complications.

Secure connection verification can be done only in real-life communication via QR-code or link. Other ways are insecure.

Before choosing the best broadcast encryption scheme, note that there are two uncompromising approaches: broadcast to a small subset of users from a big set and broadcast to users in a system, that has a small set of revoked users. Unfortunately, no existing algorithm can provide efficiency for both cases. In addition, it is necessary to remember about privacy and anonymity.

Therefore, this problem can have one compromise – choose three algorithms: for the approaches described above and one that preserves privacy.

For $n-r << n$ case, when broadcast to a small subset of users from a big set, the best solution is a Privacy Preserved IDBE scheme [HPH12], which provides privacy preserved property but leaves an open problem of linear ciphertext in the number of receivers. On the other hand, it has better decryption time results. The BGW scheme [BGW05] can be an alternative variant to choose. However, it is important to note that these schemes don't have full resistance to adaptive adversaries.

From results on tables *tab.* 1 and *tab.* 2, for $r << n$ case, the best solutions are

Combinatorial Subset Difference (CSD) [KLLO17] with fast encryption and decryption time, and Dynamic BE scheme (modified construction) [DPP07], with constant public and private keys.

4 Conclusions

We have analyzed general problems related to the implementation of anonymous messengers.

In our work, we singled out practical compromised results, which balance anonymity and complexity. Unfortunately, the problem of anonymous communication is still unresolved. Most of the solutions are a slightly changed version of one with some compromises. Only new concepts can drastically improve different system properties at the same time.

We expect that our work will help future researchers to find potential ways to improve existing schemes or introduce new ones.

References

[BBR+23] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. Technical Report 9420, RFC Editor, 2023.

[BDS10] Kent D. Boklan, Alexander W. Dent, and Christopher A. Seaman. Broadcast encryption with multiple trust authorities. In Michel Abdalla and Paulo S. L. M. Barreto, editors, *Progress in Cryptology – LATINCRYPT 2010*, pages 1–19, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. Cryptology ePrint Archive, Paper 2005/018, 2005.

[BGW06] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 3621 LNCS, 2006.

[BJE+19] Evers B., Hols J., Kula E., Schouten J., den Toom M., van der Laan R.M., and Pouwelse J.A. Thirteen years of tor attacks. `https://github.com/Attacks-on-Tor/Attacks-on-Tor`, 2019.

[BS13] Sanjay Bhattacherjee and Palash Sarkar. Tree based symmetric key broadcast encryption. Cryptology ePrint Archive, Paper 2013/786, 2013.

[CGBM15] Henry Corrigan-Gibbs, Dan Boneh, and David Mazieres. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015.

[CGCG+17] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. Cryptology ePrint Archive, Paper 2017/666, 2017.

[CMZ14] Melissa Chase, Sarah Meiklejohn, and Gregory M. Zaverucha. Algebraic macs and keyed-verification anonymous credentials. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2014.

[CPZ20] Melissa Chase, Trevor Perrin, and Greg Zaverucha. The signal private group system and anonymous credentials supporting efficient verifiable encryption. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2020.

[DA24] Aurélien Dupin and Simon Abelard. Broadcast encryption using sum-product decomposition of boolean functions. Cryptology ePrint Archive, Paper 2024/154, 2024.

[dar24] The darkfi book. `https://darkrenaissance.github.io/darkfi/`, 2024.

[DF03] Yevgeniy Dodis and Nelly Fazio. Public key broadcast encryption for stateless receivers. In Joan Feigenbaum, editor, *Digital Rights Management*, pages 61–80, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[DGGL21] Antonio Dimeo, Felix Gohla, Daniel Goßen, and Niko Lockenvitz. SoK: Multi-device secure instant messaging. Cryptology ePrint Archive, Paper 2021/498, 2021.

[DHK21] Claudia Díaz, Harry Halpin, and Aggelos Kiayias. The nym network the next generation of privacy infrastructure, 2021.

[DMMK17] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency—choose two. Cryptology ePrint Archive, Paper 2017/954, 2017.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, page 21, USA, 2004. USENIX Association.

[DPP07] Cécile Delerablée, Pascal Paillier, and David Pointcheval. Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4575 LNCS, 2007.

[Evg24] Poberezkin Evgeny. Simplex: messaging and application platform. `https://github.com/simplex-chat/simplexmq/blob/stable/protocol/overview-tjr.md`, 2024.

[Gos] gossipsub v1.0: An extensible baseline pubsub protocol. `https://github.com/libp2p/specs/blob/master/pubsub/gossipsub/gossipsub-v1.0.md`.

[HPH12] Junbeom Hur, Chanil Park, and Seong Oun Hwang. Privacy-preserving identity-based broadcast encryption. *Information Fusion*, 13, 2012.

[HS02] Dani Halevy and Adi Shamir. The lsd broadcast encryption scheme. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, pages 47–60, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[JSH24] Kee Jefferys, Maxim Shishmarev, and Simon Harman. Session: End-to-end encrypted conversations with minimal metadata leakage, 2024.

[Kir24] Jacob Kirmayer. Designing a subjective transitive reputation algorithm. *Ethereum Attestation Service*, 2024.

[KLLO17] Jihye Kim, Jiwon Lee, Seunghwa Lee, and Hyunok Oh. Combinatorial subset difference public key broadcast encryption scheme for secure multicast. Cryptology ePrint Archive, Paper 2017/408, 2017.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. Cryptology ePrint Archive, Paper 2023/874, 2023.

[KPT02] Yongdae Kim, Adrian Perrig, and Gene Tsudik. Tree-based group key agreement. Cryptology ePrint Archive, Paper 2002/009, 2002.

[KSGM03] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, page 640–651, New York, NY, USA, 2003. Association for Computing Machinery.

[LCL+10] Huang Lin, Zhenfu Cao, Xiaohui Liang, Muxin Zhou, Haojin Zhu, and Dongsheng Xing. How to construct interval encryption from binary tree encryption. In *Proceedings of the 8th International Conference on Applied Cryptography and Network Security*, ACNS'10, page 19–34, Berlin, Heidelberg, 2010. Springer-Verlag.

[LKO19] Jiwon Lee, Jihye Kim, and Hyunok Oh. BESTIE: Broadcast encryption scheme for tiny IoT equipments. Cryptology ePrint Archive, Paper 2019/1311, 2019.

[Mat] Matrix website. `https://matrix.org/`.

[NNL01] Dalit Naor, Moni Naor, and Jeff Lotspiech. Revocation and tracing schemes for stateless receivers. Cryptology ePrint Archive, Paper 2001/059, 2001.

[PK01] Andreas Pfitzmann and Marit Köhntopp. *Anonymity, Unobservability, and Pseudonymity — A Proposal for Terminology*, pages 1–9. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[PPD23] Gerhart Paul, Rösler Paul, and Schröder Dominique. Security of ibex, 2023.

[PPS11a] Duong Hieu Phan, David Pointcheval, and Mario Strefler. Decentralized dynamic broadcast encryption. Cryptology ePrint Archive, Paper 2011/463, 2011.

[PPS11b] Duong Hieu Phan, David Pointcheval, and Mario Strefler. Security notions for broadcast encryption. In Javier Lopez and Gene Tsudik, editors, *Applied Cryptography and Network Security*, pages 377–394, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Ric] Ricochet-refresh. https://github.com/blueprint-freespeech/ricochet-refresh.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[Sig] Signal website. https://signal.org/.

[Sta] Status website. https://status.app/.

[TBTW+22] Sanaz Taheri-Boshrooyeh, Oskar Thorén, Barry Whitehat, Wei Jie Koh, Onur Kilic, and Kobi Gurkan. Waku-rln-relay: Privacy-preserving peer-to-peer economic spam protection, 2022.

[Tow] Towns website. https://www.towns.com/.

[Tox17] The tox reference. https://zetok.github.io/tox-spec, 2017.

[Tre18] Perrin Trevor. The noise protocol framework. http://www.noiseprotocol.org/noise.html, 2018.

[TTBC22] Oskar Thorén, Sanaz Taheri-Boshrooyeh, and Hanno Cornelius. Waku: A family of modular p2p protocols for secure & censorship-resistant communication, 2022.

[Vac] Vac website. https://vac.dev/.

[vdHLZZ15] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, page 137–152, New York, NY, USA, 2015. Association for Computing Machinery.

[Wak] Waku website. https://docs.waku.org/.

[WQZ+11] Qianhong Wu, bo Qin, Lei Zhang, Josep Domingo-Ferrer, and Oriol Farràs. Bridging broadcast encryption and group key agreement, 12 2011.

[YZW+14] Chunli Yang, Shihui Zheng, Licheng Wang, Xiuhua Lu, and Yixian Yang. Hierarchical identity-based broadcast encryption scheme from lwe. *Communications and Networks, Journal of*, 16:258–263, 06 2014.

[ZG05] Cai-Nicolas Ziegler and Lausen Georg. Propagation models for trust and distrust in social networks. *Information Systems Frontiers*, 7:337–, 12 2005.

[ZJC11] P Zimmermann, A Johnston, and J Callas. Zrtp: Media path key agreement for unicast secure rtp. *Internet Engineering Task Force (IETF)*, 2011.

Appendix: A

Table 1: Symmetric key broadcast encryption comparison. Here: Hdr size is header size, n = the number of total users, and r = the number of revoked users, χ_k is the number of cyclotomic cosets modulo $(2^k - 1)$.

Name	Hdr size	User storage (secret key)
CS [NNL01]	$\mathcal{O}(r \log(\frac{n}{r}))$	$\mathcal{O}(\log(n))$
SD [NNL01]	$\mathcal{O}(r)$	$\mathcal{O}(\log^2(n))$
k**-ary SD** [BS13]	$\min(2r-1, n-r, \lceil n/k \rceil)$	$\mathcal{O}(\chi_k \log_k^2(n))$
LSD [HS02]	$\mathcal{O}(r)$	$\mathcal{O}(\log^{1+\varepsilon}(n))$, $\varepsilon > 0$
BE-SP [DA24]	$\leq r + \log(n)$ (empirical)	$\mathcal{O}(n)$

Table 2: Asymmetric key broadcast encryption comparison. Here: Hdr size is header size, τ_m denote point multiplication, τ_s denote points addition, τ_e denote pairings, τ_{enc} and τ_{dec} denote assymetric encryption and decryption algorithms, τ_p denote division and subtraction modulo p, n is the number of total users, r is the number of revoked users, λ is a system parameter, S_p is a bit size of an element in $\mathbb{Z}_p^*$, and S_1 is a bit size of an element in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}$.

Algorithm	PK Size	SK Size	Hdr size	Enc Time	Dec Time
SD (HIBE) [KLLO17, DF03]	$\mathcal{O}(\log(n))$	$\mathcal{O}(\log^3(n))$	$\mathcal{O}(r)$	$\mathcal{O}(r \log(n))\tau_m$	$\mathcal{O}(\log(n))\tau_m$
Interval [LCL+10]	$\mathcal{O}(\log(n))$	$\mathcal{O}(\log^2(n))$	$\mathcal{O}(r)$	$\mathcal{O}(r)\tau_m$	$\mathcal{O}(\log(n))\tau_m$
CSD [KLLO17]	$\mathcal{O}(\log(n))$	$\mathcal{O}(\log^2(n))$	$\mathcal{O}(r)$	$\mathcal{O}(r)\tau_m$	*const*
BESTIE [LKO19]	$\mathcal{O}(\log(n))$	$\mathcal{O}(\log(n))$	$\mathcal{O}(r)$	$\mathcal{O}(r)\tau_m$	$\mathcal{O}(\log(n))\tau_s$
BGW [BGW05]	$\mathcal{O}(\lambda\sqrt{n})$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda\sqrt{n})$	$\mathcal{O}(\lambda\sqrt{n})\tau_s$	$\mathcal{O}(\lambda\sqrt{n})\tau_s$
PP-IDBE [HPH12]	S_1	S_p	$\mathcal{O}(n)$	$\mathcal{O}(n)(\tau_m + \tau_e)$	*const*
Dynamic BE [DPP07]	$\mathcal{O}(n)$	*const*	$\mathcal{O}(r)$	$\mathcal{O}(r^2)\tau_m$	$\mathcal{O}(r)\tau_m$
Dynamic BE [DPP07]	*const*	*const*	$\mathcal{O}(r)$	$\mathcal{O}(r)\tau_m$	$\mathcal{O}(r)\tau_m$
CBE [WQZ+11]	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(r)$	$\mathcal{O}(r)\tau_s$	$\mathcal{O}(r)\tau_s$

Algorithm	PK Size	SK Size	Hdr size	Enc Time	Dec Time
Decentralized BE [PPS11a]	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$	$\mathcal{O}(r\log(\frac{n}{r}))$	$\mathcal{O}(r)\tau_{enc}$	*const*
Distributed BE 1 [KMW23]	$\mathcal{O}(n^2)$	*const*	*const*	$\mathcal{O}(n)\tau_s$	$\mathcal{O}(n)\tau_s$
Distributed BE 1 (Log. Updates) [KMW23]	$\mathcal{O}(n^2)$	*const*	$\mathcal{O}(\log(n))$	$\mathcal{O}(n)\tau_s$	$\mathcal{O}(n)\tau_s$
Distributed BE 2 [KMW23]	$\mathcal{O}(n^2)$	*const*	*const*	$\mathcal{O}(n)\tau_s$	$\mathcal{O}(n)\tau_s$
Distributed BE 2 (Log. Updates) [KMW23]	$\mathcal{O}(n^2)$	*const*	$\mathcal{O}(\log(n))$	$\mathcal{O}(n)\tau_s$	$\mathcal{O}(n)\tau_s$