

Phoenix Architecture Review

Threat model and hacking assessment report
V.1.0

Prepared for:
Phoenix R&D



1	Executive summary	3
1.1	Engagement overview	3
1.2	Observations and Risks.....	3
1.3	Recommendations.....	3
2	Evolution suggestions	4
2.1	Evolve the attacker model.....	4
2.2	Evolve the protocol description.....	4
2.3	Evolve the security posture	4
3	Motivation and scope	5
4	Methodology	7
4.1	Threat modeling and attacks	7
4.2	STRIDE Categories and Threat Scenarios:.....	8
4.3	Risk assessment using STRIDE	9
4.4	STRIDE threat mapping.....	10
5	Protocol Design	11
5.1	Architecture overview	11
5.2	Registration	13
5.3	Group creation	14
5.4	Connection	16
5.5	Add users to a group	18
5.6	Message fan-out.....	20
6	Findings summary	21
6.1	Risk Profile	21
6.2	Issue Summary	22
7	Findings details	23
7.1	SRL-1-SPA: Friendship tokens can be passed on to maliciously add users to a group	24
7.2	SRL-2-SPO: User identity confusion for client credential authenticated calls	26
7.3	SRL-3-PRI: Server can link users to Privacy Pass token redemption	28
7.4	SRL-4-SPO: Existing friendship tokens can be overwritten by other users.....	29
7.5	SRL-5-PRI: Queue sizes allow inferring the number and size of messages.....	31
7.6	SRL-6-PRI: FQDNs of federated servers visible to honest-but-curious QS.....	32
7.7	SRL-7-PRI: Insufficient anonymity set size in small instances.....	33
8	Detailed recommendations	34
8.1	Attacker	34
8.2	Protocol description	35
8.3	Security posture	36
9	Bibliography.....	37
10	Appendix A: SRLabs technical services.....	38

Disclaimer

This report describes the findings and core conclusions derived from the audit carried out by Security Research Labs (SRLabs) within the agreed-on timeframe and scope as detailed in the following sections.

Please note that this report does not guarantee that all existing security vulnerabilities were discovered in the codebase exhaustively and that following all evolution suggestions may not ensure all future concepts or code to be bug free.

Version:	V.1.0
Prepared For:	Phoenix R&D
Date:	October 24, 2024
Authors:	Marc Heuse Tobias Mueller Bruno Produit

Timeline

The Phoenix architecture received a thorough security assessment by Security Research Labs in summer 2024.

Date	Event
June 20, 2024	Initial engagement
October 31, 2024	Final report delivered to Phoenix R&D

Table 1. Audit timeline

1 Executive summary

The Phoenix protocol [1] is a federated, state-of-the-art secure messaging protocol that requires minimal metadata. Besides federation, Phoenix tries to differentiate itself from other secure messengers by placing special emphasis on the security and privacy of users against snapshot attacks to minimize data that law enforcement can obtain with a court order. Phoenix builds on Message Layer Security (MLS) [2] and provides three actors: (1) the Authentication service (AS) to provide identities, (2) the Delivery service (DS) to accept and distribute messages, and (3) the Queuing service (QS) for messages to be retained for clients to fetch them.

Security Research Labs reviewed the protocol and can attest that Project Phoenix has proactively implemented effective security measures and taken security and privacy seriously. These measures have significantly strengthened the protocol's security posture.

During the review, SRLabs identified seven issues, confirmed them with Phoenix, and proposed mitigations. There were no critical issues and three issues of high severity, all of which are already resolved.

1.1 Engagement overview

This work describes the result of the architecture review of the Phoenix protocol performed by SRLabs. SRLabs provides specialized audit services in the security ecosystem.

During this review, Phoenix provided access to relevant documentation and supported the research team effectively. The protocol design was reviewed by SRLabs to assure that the Phoenix protocol is resilient to hacking and abuse.

1.2 Observations and Risks

The research team identified seven issues ranging from high to low severity levels. Most identified issues were related to privacy or spamming potential. Phoenix, in cooperation with the auditors, has already addressed all identified issues.

1.3 Recommendations

We recommend placing strong emphasis on closing the gap between the specification and the current implementation. It is important to ensure that the specification is extended to cover all functionalities that have been implemented but remain undocumented. Additionally, a thorough investigation should be conducted to verify that the implementation aligns with the details outlined in the specification. Ensuring consistency between the specification and implementation will not only prevent misunderstandings but will also enable engineers to implement features more reliably and securely. A review will help identify any discrepancies and ensure that both the specification and implementation are in complete sync.

2 Evolution suggestions

The results of the audit show that Phoenix is designed with security and privacy in mind. To ensure that Phoenix is secure against further unknown or yet undiscovered threats, we recommend considering the evolution suggestions described in this section. This section presents a summary of our suggestions for further improving the Phoenix architecture. A more detailed presentation can be found in section 8.

2.1 Evolve the attacker model

The Phoenix protocol is designed to primarily defend against the “snapshot attacker”. The snapshot attacker is very common in the messaging space and is characterized by the ability to read data at a certain point in time but not continuously monitor the operations. We recommend developing strategies around a stronger adversary to ensure a robust and trustworthy architecture.

We recommend designing safeguards around the concept of the “honest-but-curious attacker”. This would further strengthen Phoenix’s commitment to privacy and favor their security position against other private messaging systems. This type of attacker not only can read data and but also keep historical records at their discretion. This attacker model reflects potential privacy risks more accurately than the snapshot attacker, and incorporating its respective design safeguards can in turn enhance user trust.

Another recommendation is analyzing data accessibility levels for each attacker in a worst-case scenario. Detailed enumeration of accessible and inferable data based on the attacker’s access level helps identify potential vulnerabilities which can then be addressed.

2.2 Evolve the protocol description

We recommend evolving the specification by taking a client's perspective. Our suggestions include describing core use-cases, e.g. revolving around processes for a comprehensive overview of message-sending and API call requirements, to provide clarity to implementors and reviewers. To further provide clarity for users and reviewers, we recommend explaining the use of every sort of persisted data, including a technical justification for why a data item is required to operate the service. Conveying the semantics and necessity of data items like user activity timestamps can significantly contribute towards a more secure data management strategy. Finally, centering a description around the user’s client also facilitates streamlining the existing parts of the specification.

2.3 Evolve the security posture

We recommend evolving the defensive mechanisms of the architecture, particularly in relation to the storage of private or secret key material and their leak prevention. Guiding implementors towards safeguard measures such as keeping these keys offline or in an Hardware Security Module (HSM) can substantially reduce security risks. Potential denial-of-service (DoS) vectors should also be quantified with regards to the computational, network, and I/O overhead to facilitate a risk assessment. Further, post-compromise security scenarios should be apprehended to mitigate the effects of forceful disclosure of user credentials or server operators surrendering their keys. Finally, advocating for cryptographic primitives and schemes that are not encumbered by patents ensures that the development and operation can proceed with fewer barriers.

3 Motivation and scope

This report presents the security audit conducted by SRLabs as part of the Open Tech Fund's Security Lab initiative [3]. This document focuses solely on the analysis of the Phoenix protocol documentation. The implementation [4] is reviewed in a subsequent phase.

From a technical perspective, Phoenix's core functionality is to provide an MLS-based privacy-friendly federated messaging service. Message Layer Security is a protocol for the secure exchange of messages [2]. However, MLS does not specify how exactly various components behave. The Phoenix protocol fills in this gap by striving to require as little data about its users or their behavior as possible.

The Phoenix protocol is described in a publicly available document [1] which serves to guide engineers in implementing a secure version of the federated protocol by providing clear definitions of essential components and their interaction. SRLabs examined this documentation considering its audience to assess the protocol's security and ensure that the documentation highlights potential security issues that could arise during implementation.

The core component of the documentation is the chapter titled "Specification" which describes the services of the Phoenix home server. These services are the scope of our review: The Authentication service (AS), the Delivery service (DS), the Queuing service (QS), and their interaction as well as persisted data.

The scope of this review also includes the client's interactions with the Phoenix services. We consider These interactions include registering an account, creating a group, creating a connection, adding a user to a group, and sending a message. Finally, the review's scope also includes the messages exchanged during interactions as well as their cryptographic protection.

This review does not include network-related issues in its scope. Network issues include DNS hijacking and the delay, reordering, or retransmission of network packets. Furthermore, the risks of analyzing traffic patterns are recognized as orthogonal to providing a messaging platform and are excluded from the scope of this review. We also do not consider cases of compromise of any party's secret or private key. For example, the AS posting their private key or clients sharing their credentials. [Table 2](#) provides a view of actors and assets and whether they are in the scope of this review.

Actor	Asset	In scope
User	2FA token	Yes
	Username	
	Attention & Motivation	
	Resources	
	Availability	
	Friendship token	
Client	Client to server authentication key material	No
	Local QS user record	
	Client to client authentication key material	
Home server	Group state	Yes
	KeyPackage	
	Client to server authentication key material	
	server to server authentication key material	
	Usernames	
	Message queue	
Home-server Operator	2FA	No
	Home server admin credentials	
	Home server configuration	
	Hosting cost	

Federated Home server	Server to server authentication key material	No
Federated User		No
Federated Client	Client to client authentication key material Local QS user record	No
Federated Home server Operator	Allow-list configuration	No

Table 2. Actors and assets in scope

4 Methodology

This report details the security assurance results for the Phoenix messaging protocol, aiming to create transparency through threat modeling, security design review, and remediation support. We applied the methodology described in this section when performing feature reviews, which aligns with the STRIDE framework [5] for threat modeling.

4.1 Threat modeling and attacks

The goal of the threat model framework is to determine specific areas of risk in Phoenix protocol. Familiarity with these risk areas can provide guidance for the design of the implementation stack, the actual implementation of the stack, and the security testing. This section introduces how risk is defined and provides an overview of the identified threat scenarios.

The risk level is categorized into low, medium, and high and considers both the hacking value and the damage that could result from successful exploitation. The risk of a threat scenario is calculated by the following formula:

$$Risk = Damage \times Hacking Value = \frac{Damage \times Incentive}{Effort}$$

The *Hacking Value* is also categorized into low, medium, and high, and considers the incentive of an adversary, as well as the effort required to successfully execute the attack. The hacking value is calculated as follows:

$$Hacking Value = \frac{Incentive}{Effort}$$

While incentive describes what an adversary might gain from performing an attack successfully, effort estimates the complexity of this attack. The degrees of incentive and effort are defined as follows:

Incentive:

- **Low:** Attacks offer the hacker little to no gain from executing the threat.
- **Medium:** Attacks offer the hacker considerable gains from executing the threat.
- **High:** Attacks offer the hacker high gains by executing this threat.

Effort:

- **Low:** Attacks are easy to execute. They require neither elaborate technical knowledge nor considerable amounts of resources.
- **Medium:** Attacks are difficult to execute. They might require bypassing countermeasures, the use of expensive resources or a considerable amount of technical knowledge.
- **High:** Attacks are difficult to execute. The attacks might require in-depth technical knowledge, vast amounts of expensive resources, bypassing countermeasures, or any combination of these factors.

Incentive and effort are divided according to Table 3.

Hacking value	Low incentive	Medium incentive	High incentive
High effort	Low	Medium	Medium
Medium effort	Medium	Medium	High
Low effort	Medium	High	High

Table 3. Hacking value measurement scale

Hacking scenarios are classified by the risk they pose to the system. Conversely, the *Damage* describes the negative impact that a given attack, if performed successfully, would have on the victim. The degrees of damage are defined as follows:

- **Low:** Risk scenarios would cause negligible damage to the user of the Phoenix protocol.

- **Medium:** Risk scenarios pose a considerable threat to Phoenix's functionality or privacy.
- **High:** Risk scenarios pose an existential threat to Phoenix network functionality or privacy.

Damage and hacking value are divided according to Table 4.

Risk	Low hacking value	Medium hacking value	High hacking value
Low damage	Low	Medium	Medium
Medium damage	Medium	Medium	High
High damage	Medium	High	High

Table 4. Risk measurement scale

After applying the threat model framework to the Phoenix system, we identified the different threat scenarios according to STRIDE. These are outlined in the following section.

4.2 STRIDE Categories and Threat Scenarios:

The STRIDE framework is a widely used security model that helps identify potential threats in software systems. It stands for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege, each representing a different category of security risks.

We have expanded the STRIDE framework for this engagement by including two additional categories that better reflect all possible threats: Privacy Breaches, and Spam and Annoyances. This enhanced approach allows for a more comprehensive analysis, ensuring that all relevant security and privacy concerns are thoroughly addressed. These categories are described below:

Spoofing

Spoofing refers to when an attacker poses as another legitimate user or entity within the Phoenix messaging protocol. This could allow unauthorized access to communications, enabling the attacker to send or receive messages as someone else. For example, the attacker might impersonate a trusted user to intercept sensitive information or initiate malicious conversations, thus undermining the identity verification process within the protocol.

Tampering

Tampering involves the unauthorized alteration of data as it is transmitted within the Phoenix messaging protocol. A potential threat scenario could involve an attacker intercepting and modifying the content of messages, potentially altering commands, instructions, or even the meaning of conversations. This could result in disrupted communication, misinformation, or the injection of malicious data into the system, which affect the integrity of the message exchange.

Repudiation

Repudiation threats concern the denials of action taken within the Phoenix protocol, where an attacker denies having sent or received specific messages. Without proper mechanisms to log and verify the authenticity of sent and received communications, the attacker could claim plausible deniability for harmful actions, such as sending malicious commands or fraudulent messages. Ensuring non-repudiation in the Phoenix protocol is essential for verifying the authenticity of message exchanges.

Information Disclosure

Information disclosure occurs when sensitive data, such as message content, user credentials, or metadata, is exposed to unauthorized parties. For the Phoenix messaging protocol, this could involve an attacker accessing private conversations or sensitive details about message routes. An example scenario might involve the attacker using a vulnerability to extract private messages from a server or

intercepting encrypted messages during transmission. Such information disclosure could lead to data breaches, privacy violations, or further exploitation of sensitive data.

Denial of Service (DoS)

Denial-of-service attacks in the context of the Phoenix messaging protocol aim to prevent legitimate users from sending or receiving messages by overwhelming the system with excessive requests or exploiting protocol vulnerabilities. An attacker could flood the message queues, exhaust resources, or take advantage of a specific weak point to disrupt communication between users. This could render the messaging protocol unavailable to legitimate users, affecting the overall functionality of the system.

Elevation of Privilege

Privilege escalation attacks occur when an adversary gains a higher level of permissions within the Phoenix messaging protocol than that which they are authorized to have. For example, the attacker might exploit a flaw in the protocol to elevate their permissions from a standard user to an administrator, thus granting them access to sensitive information or allowing them to manipulate the message flow. This threat represents a serious risk to the integrity and control of the system.

Privacy Breaches

Privacy breaches involve the unauthorized access to, collection, or exposure of personal or sensitive information. This can lead to identity theft, financial loss, or other privacy violations. Privacy breach prevention is crucial for Phoenix as the federated nature of the app could expose data across multiple servers if not properly secured.

Spam and Annoyances

Spam and annoyances refer to the unsolicited, excessive, or disruptive communications, notifications, or actions that not only degrade the user experience but can also burden system resources and compromise efficiency. For Phoenix, this includes the risk of spam spreading across federated nodes, potentially overwhelming users and servers.

4.3 Risk assessment using STRIDE

Each identified STRIDE threat is assessed based on the likelihood of exploitation and the potential impact on the Phoenix messaging protocol. For each category, the attacker's incentive and effort required to exploit a vulnerability are used to estimate the hacking value, as described earlier. Similarly, the damage posed by successfully exploiting a threat scenario is calculated to assess overall risk.

4.4 STRIDE threat mapping

The STRIDE threat modeling framework is a systematic approach to identify and analyze potential threats to a system. Threats are categorized and addressed by considering six distinct types, corresponding to the STRIDE acronym.

Scope category	Asset	Hacking value	Easiness of attack	Overall risk
Spoofing	2FA Username Client to server authentication key material	High	Medium	High
Tampering	Message queue Group state KeyPackage	Medium	Critical	High
Repudiation	Message queue	Low	Low	Low
Information disclosure	Client to server authentication key material Message queue	High	High	High
Denial of service	2FA Client to server authentication key material	Low	Critical	Medium
Elevation of privilege	2FA Client to server authentication key material	High	Low	Medium
Privacy breach	Message queue Client to server authentication key material Group state KeyPackage	Critical	Critical	Critical
Spam and annoyances	Message queue	High	Critical	Critical

5 Protocol Design

This section provides an overview of the Phoenix architecture and its subprotocols for establishing an identity, managing groups, and sending messages. This overview is the basis for the findings presented in the next section.

5.1 Architecture overview

The Phoenix architecture is based on MLS. MLS defines how clients interact but deliberately leaves it open how, for example, identities are established or how clients fetch messages. These open ends are tied together by the Phoenix protocol.

The Phoenix architectural center is the home server which consists of three components: The Authentication service, the Delivery service, and the Queueing service:

- The Authentication service (AS) establishes the cryptographic identities of the users.
- The Delivery service accepts messages and distributes them to the relevant users by sending them to the Queueing Service (QS).
- The Queueing service (QS) retains messages until they are fetched by their users.

These components are meant to be independent so that they can be run by different entities. The consequence of this independence is a trust boundary between the components. For example, the QS must not learn which user is fetching messages. Another trust boundary exists between the users and the home server. For example, the server must not learn who is part of which group and who sends messages to which other users.

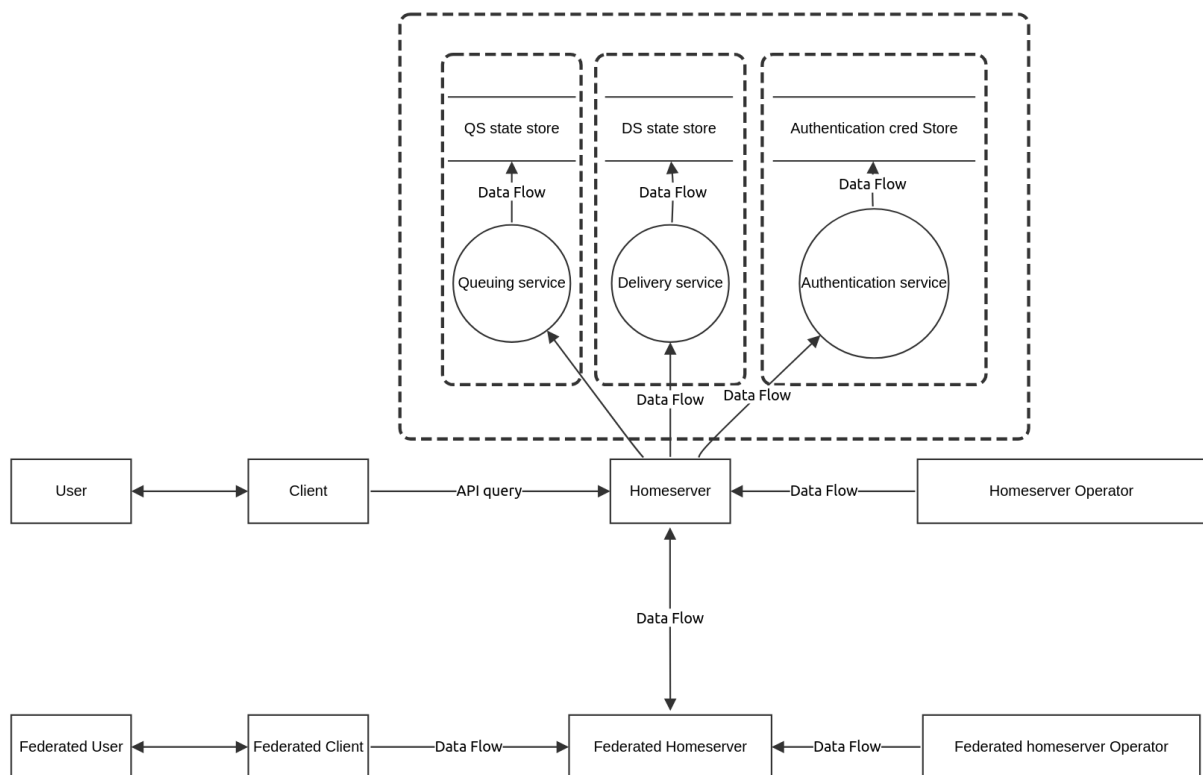


Figure 1. Phoenix architecture overview

Authentication service

The authentication service (AS) is the identity provider in the Phoenix architecture. It provides API calls pertaining to account management, such as registering accounts, deleting accounts, or changing account properties. A central addition to the MLS concepts is the concept of a *connection*, which is the ability to add users to groups. Connections are established by fetching connection packages which are kept by the AS for other users to fetch.

Additionally, the connection packages are tied to their own user. This direct relationship to a user is not always desirable as it allows for inferences on the user's identity when a contact is made. To prevent these inference, users can also upload a connection package without a reference to the actual user. Instead of binding the connection package to a user, the connection package is bound to an alias. The AS offers users to register unlimited aliases and does not keep an association between the alias and the user that registered it.

The AS also maintains a queue for the user and its aliases. These AS queues are used to route the connection packages. Once the users establish a connection, the queuing service queues are used for routing messages.

The AS also manages the issuance and redemption of privacy pass tokens. Privacy pass tokens are used to manage API quotas and maintain service stability for all clients. Traditionally, rate limiting is implemented by recording the time and user of an action. If a user causes too many actions in too short of a period, further requests will be blocked. Keeping a record of the user performing an action trivially enables the server to learn which user performs which action, e.g., sending a message. The use of privacy passes as a rate limiting mechanism ensures that the server can limit the number of actions any user can perform without knowing the individual user behind each action. This mechanism works by first handing out tokens which can be redeemed later by the users. This ensures the server limits the number of overall actions that can be performed rather than which actions an individual user performed. During redemption, the privacy pass mechanism ensures that the server does not learn which user is redeeming the token.

Delivery service

The delivery service (DS) maintains the groups required for communication among users. Groups are a core concept of MLS and central to Phoenix. The DS offers calls to create groups, add users, and delete groups. The DS maintains the state of groups and modifies it according to the client's request. The group's state is stored encrypted with a key provided by the client on every call. Storing the state encrypted protects it against an attacker capable of reading the persistent storage.

The DS also distributes group messages by sending them to the Queuing service. The DS fetches the group's members from the state. The members are pseudonyms that do not have a direct relationship to the user ID or the cryptographic identity maintained by the AS. Instead of user IDs, the group's state contains IDs for the member's queues. The DS forwards those queue IDs to the QS to let it retain the messages.

Queuing service

The Queuing service (QS) retains messages for clients to fetch. Clients register queues and continuously query the QS for new messages. The queues do not have a direct relationship with the user but are maintained as simple integers with a signing key. Users interacting with the queue need to present requests signed with the queue's signing key to prove their eligibility.

5.2 Registration

The user registration process is composed of a call to the registration interface followed by a call for provisioning the user's queues:

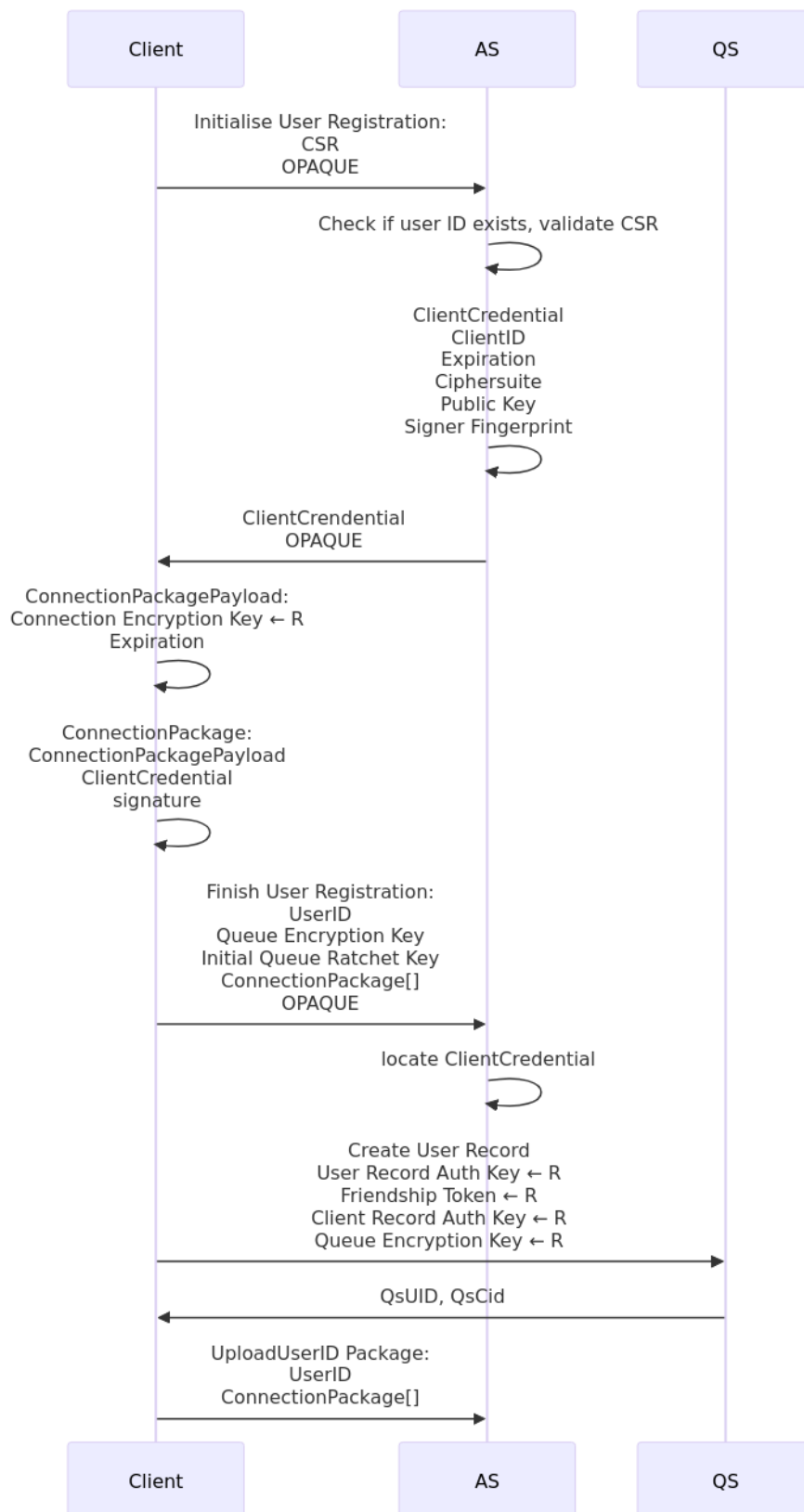


Figure 2. Registration process

5.3 Group creation

The core concept in MLS is a group. A group contains one or more users and is used to send messages.

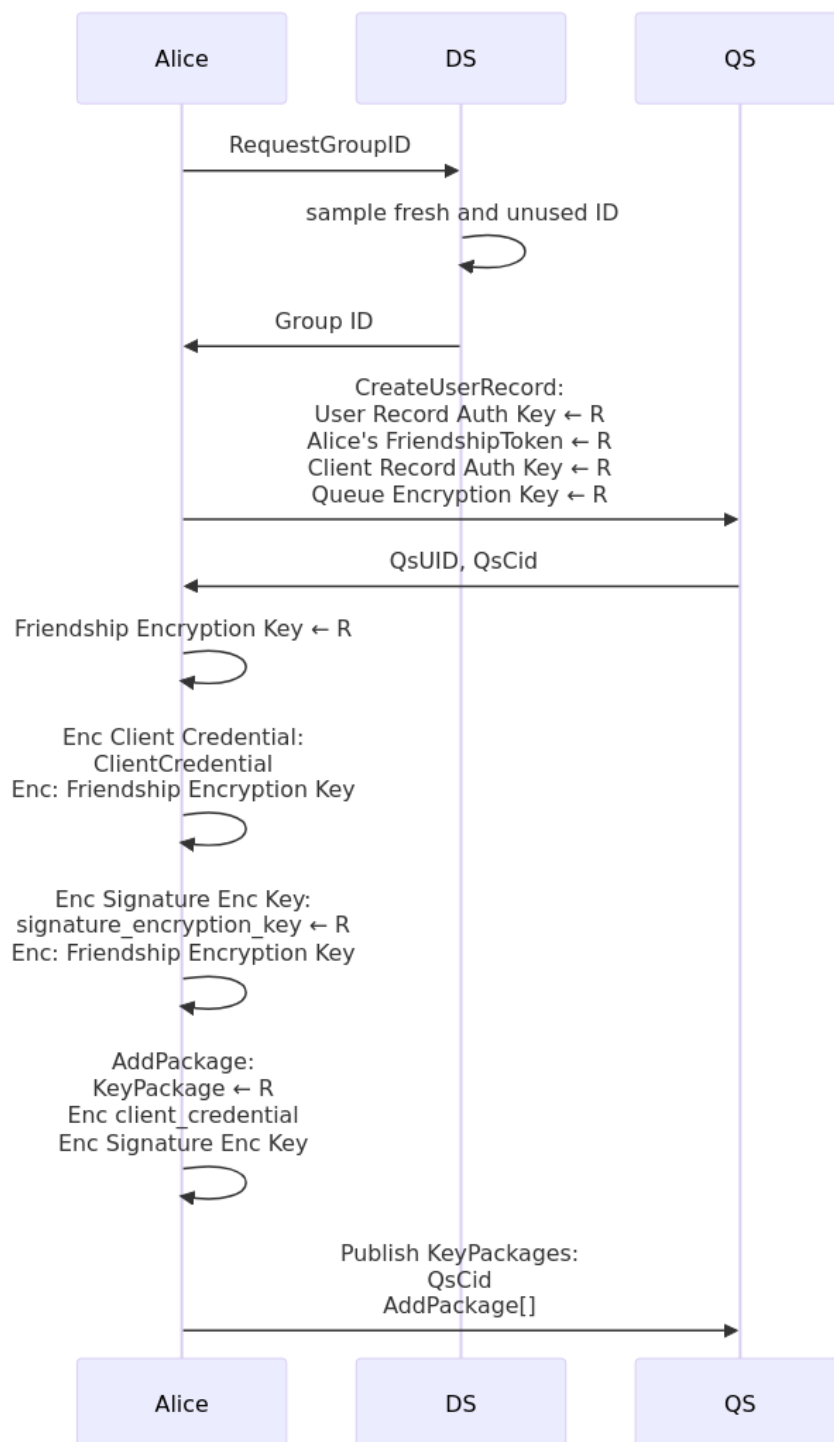


Figure 3. Group creation process 1/2

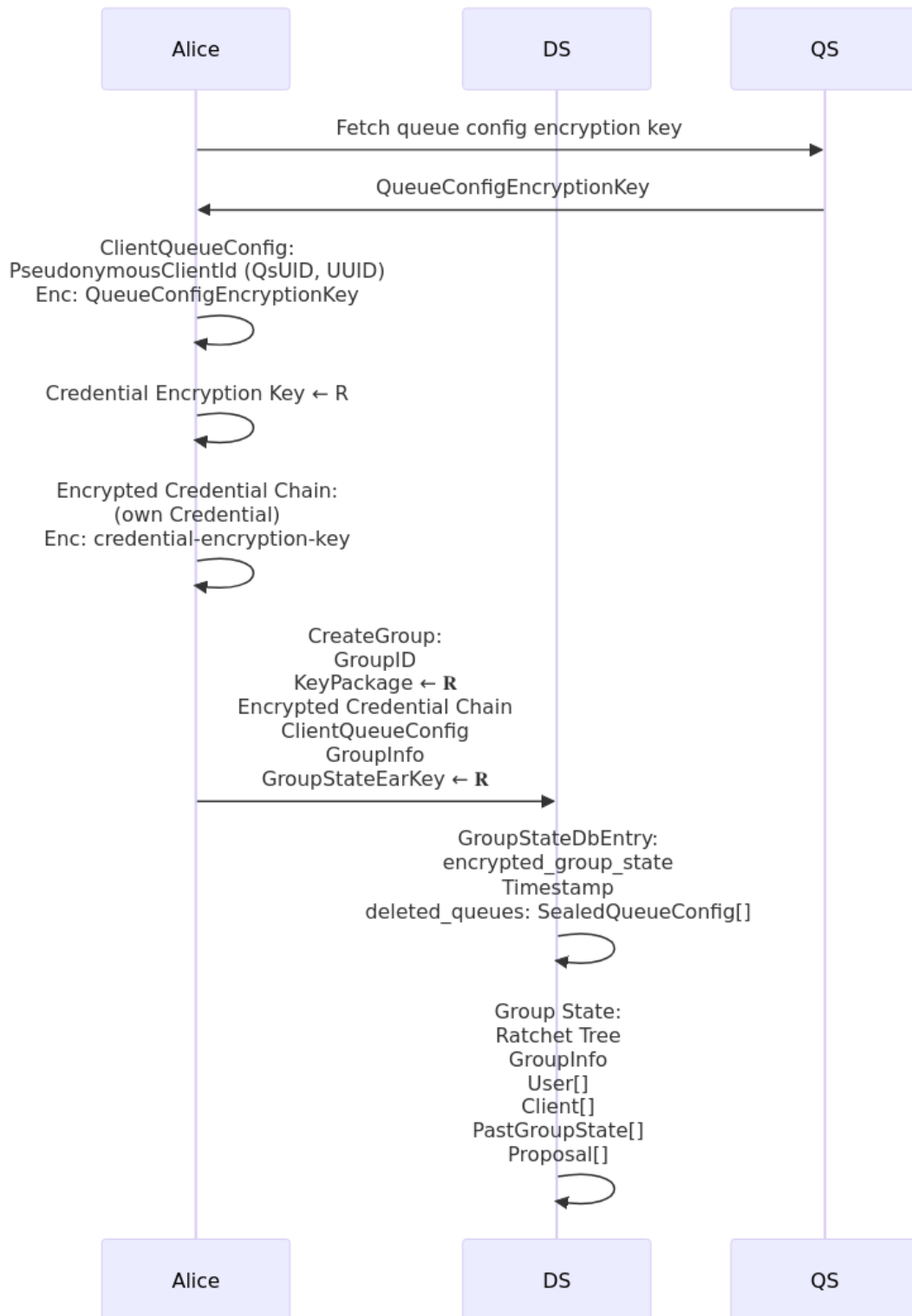


Figure 4. Group creation process 2/2

5.4 Connection

One of the core concepts of Phoenix is that of a user establishing a connection with another user. A connection is considered a requirement to be able to add a user to a group or send them a message. Two users have a connection once they are members of a dedicated two-user group. The connection is a vehicle for transmitting so-called “friendship keys” which are used to fetch key material required for adding the user to a group.

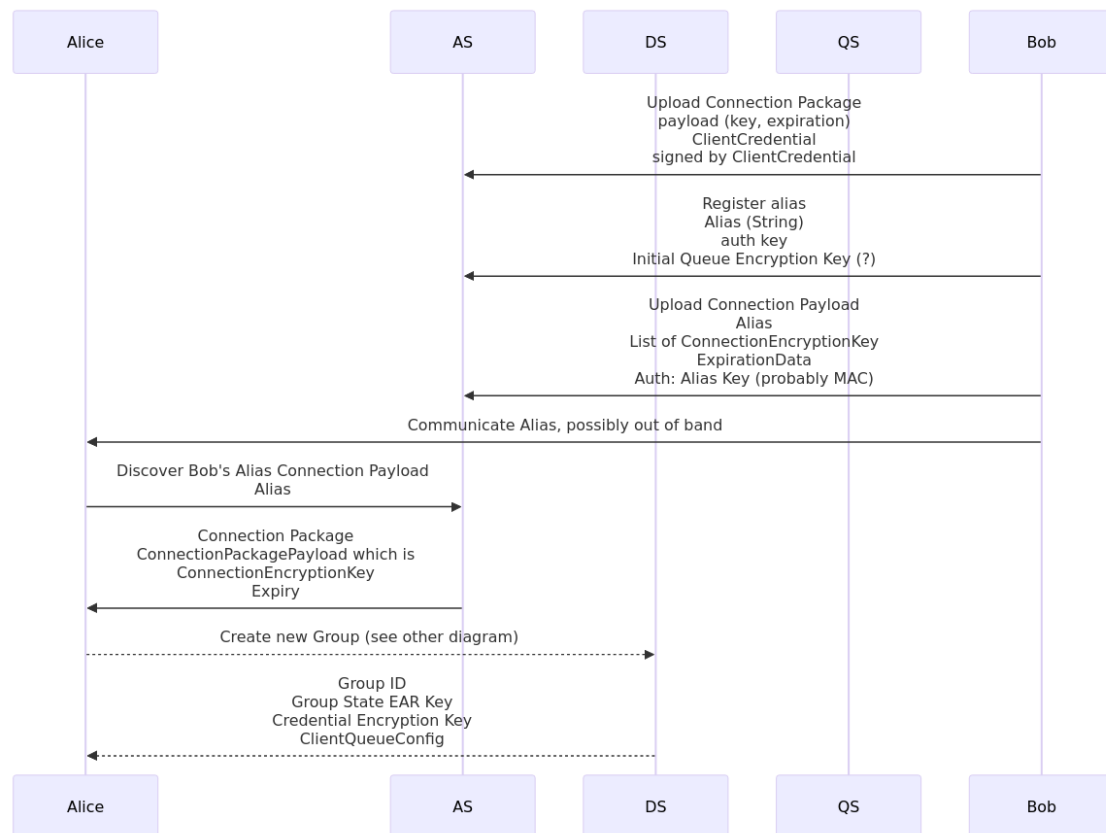


Figure 5. Connection establishment 1/2

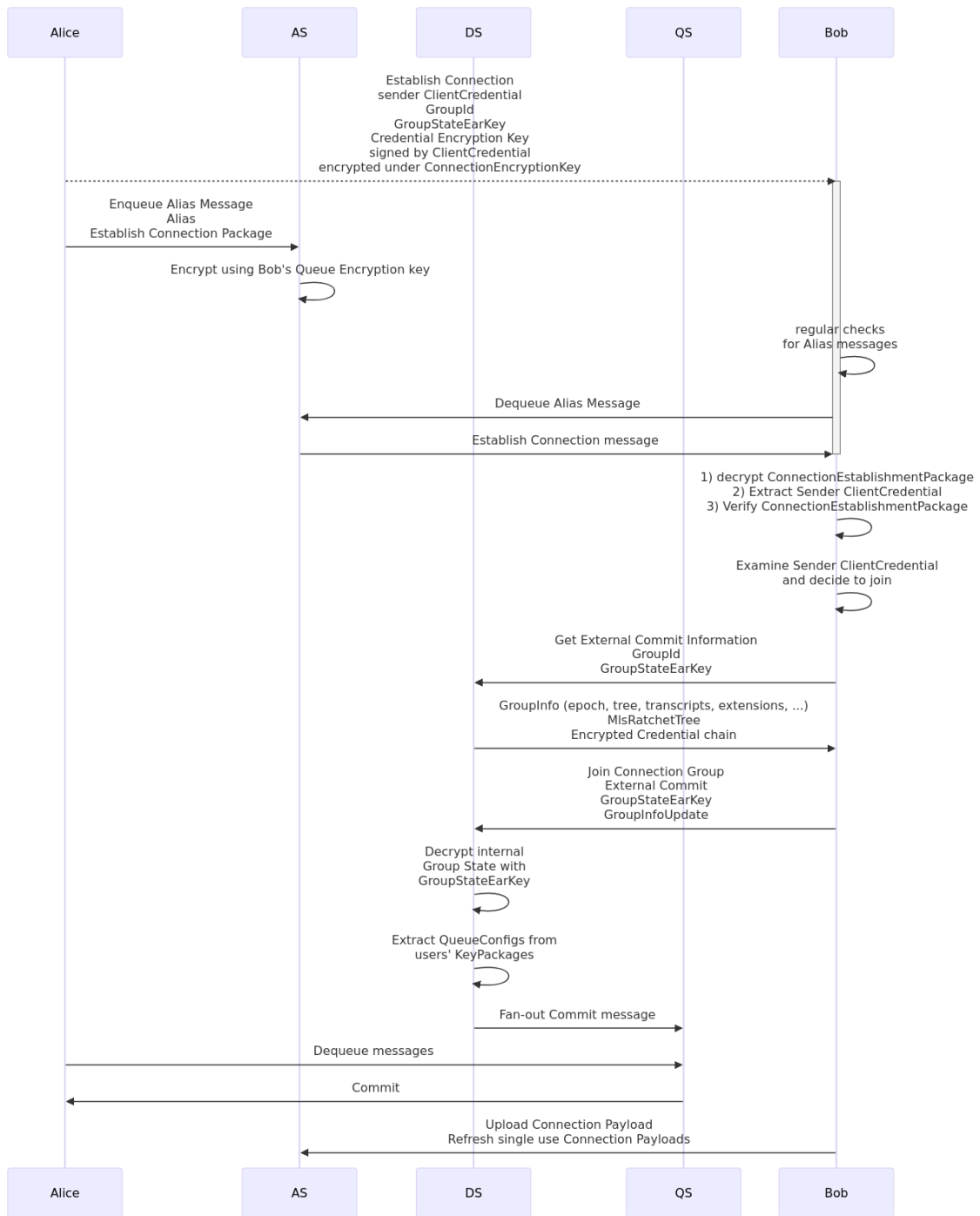


Figure 6. Connection establishment 2/2

5.5 Add users to a group

After creating a group and having established a connection, a user can add another user to the group for exchanging messages.

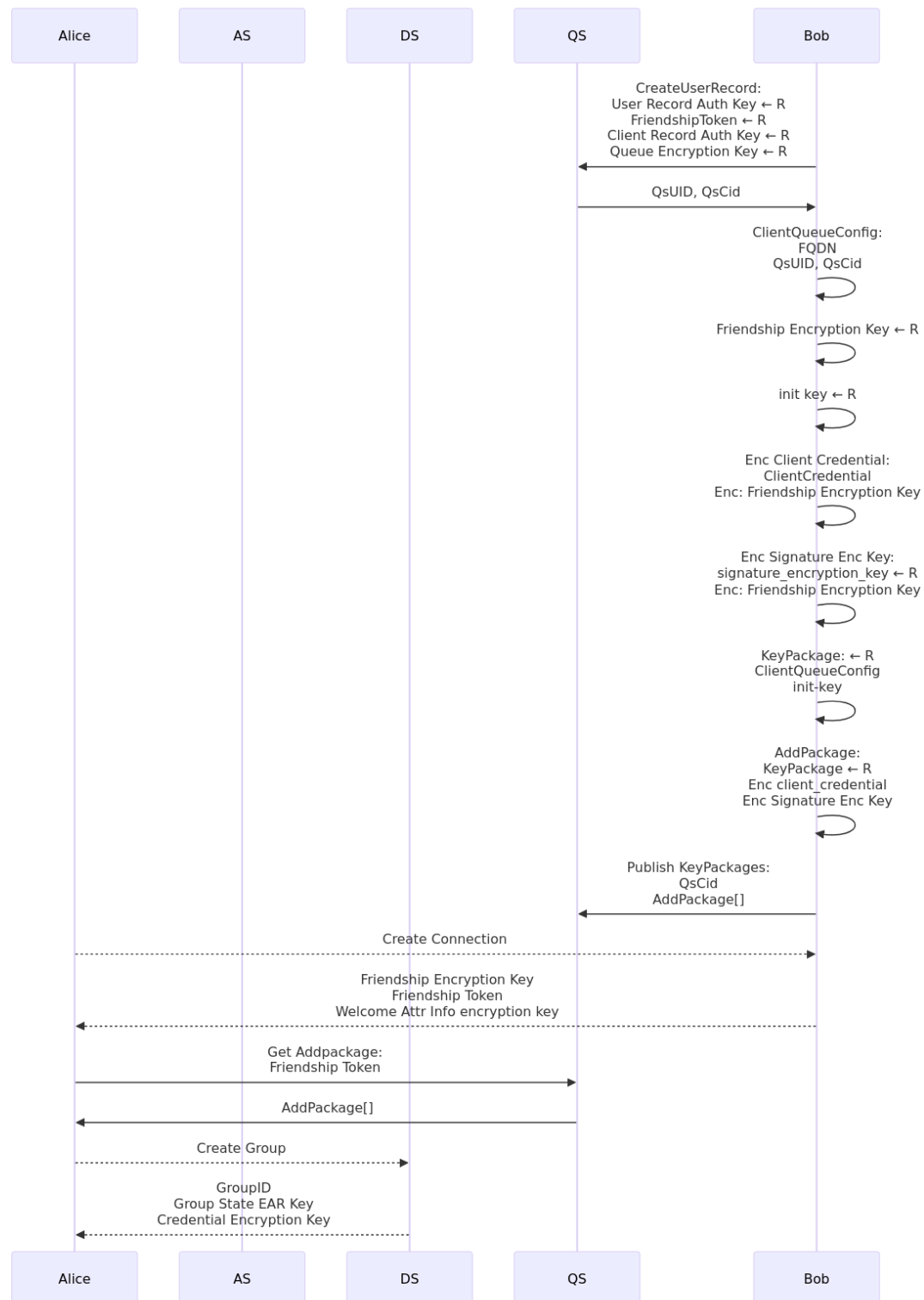


Figure 7. Adding a user to a group 1/2

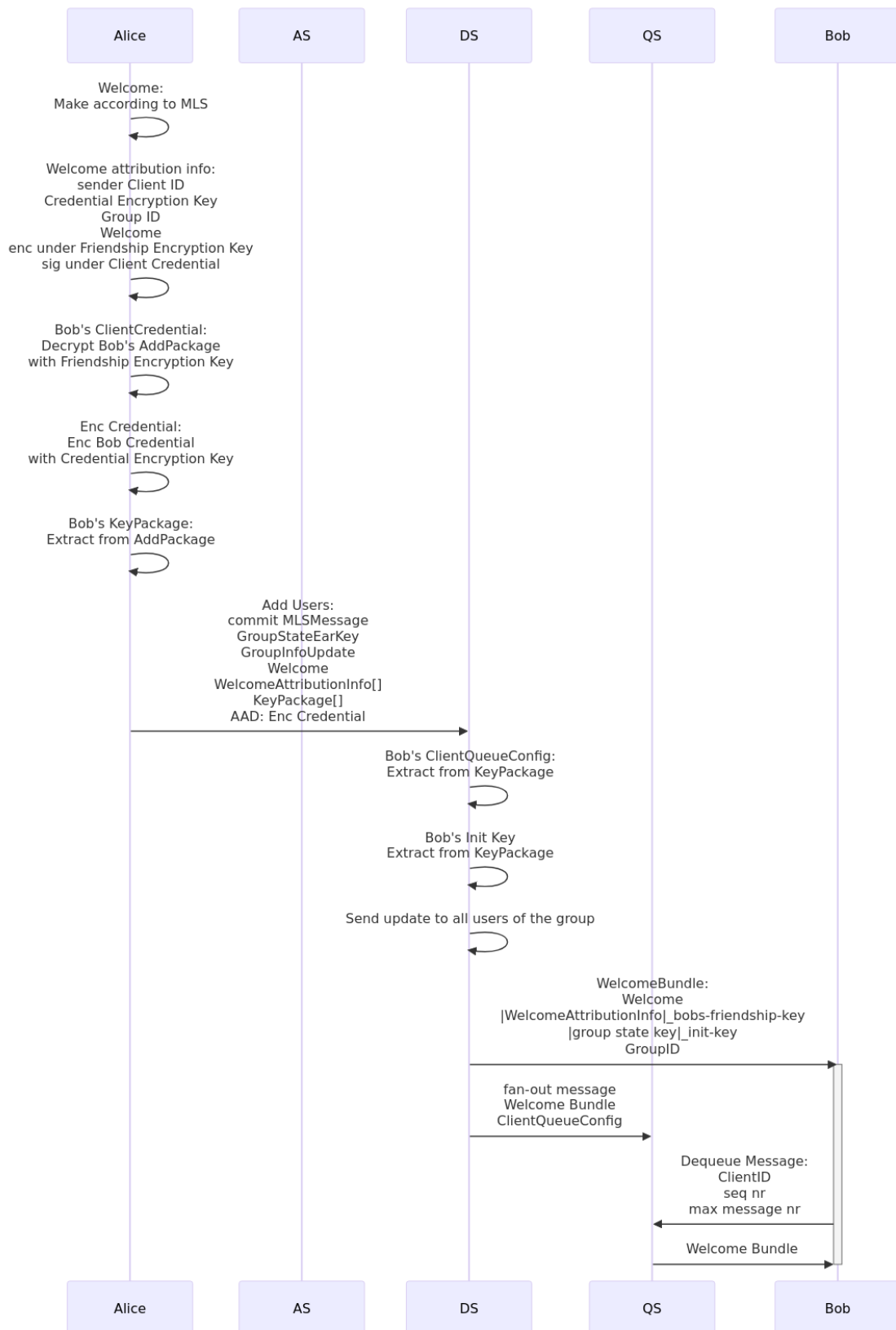


Figure 8. Adding a user to a group 2/2

5.6 Message fan-out

Finally, users want to send messages to each other. The DS accepts and distributes the messages.

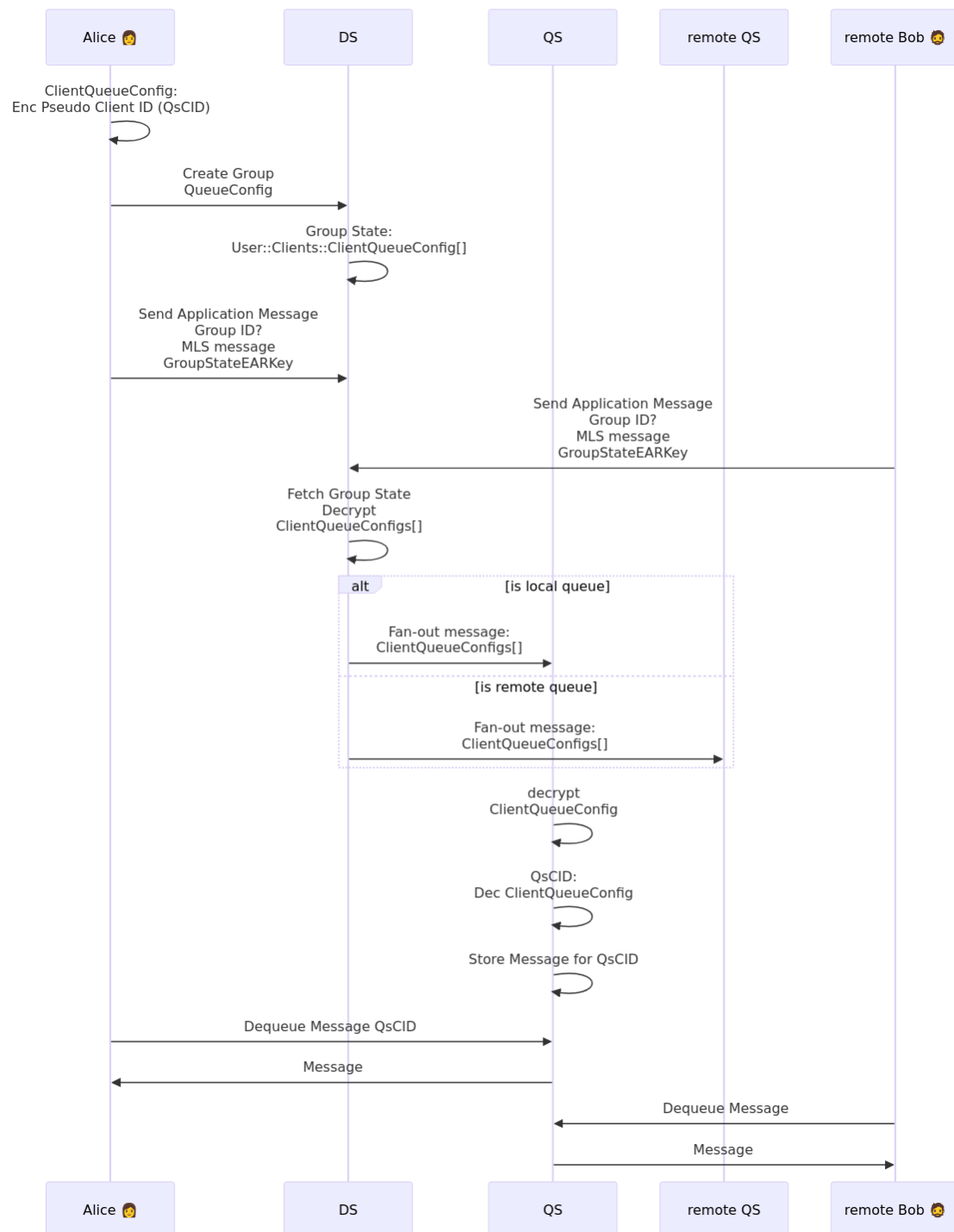





Figure 9. Message Fan-Out

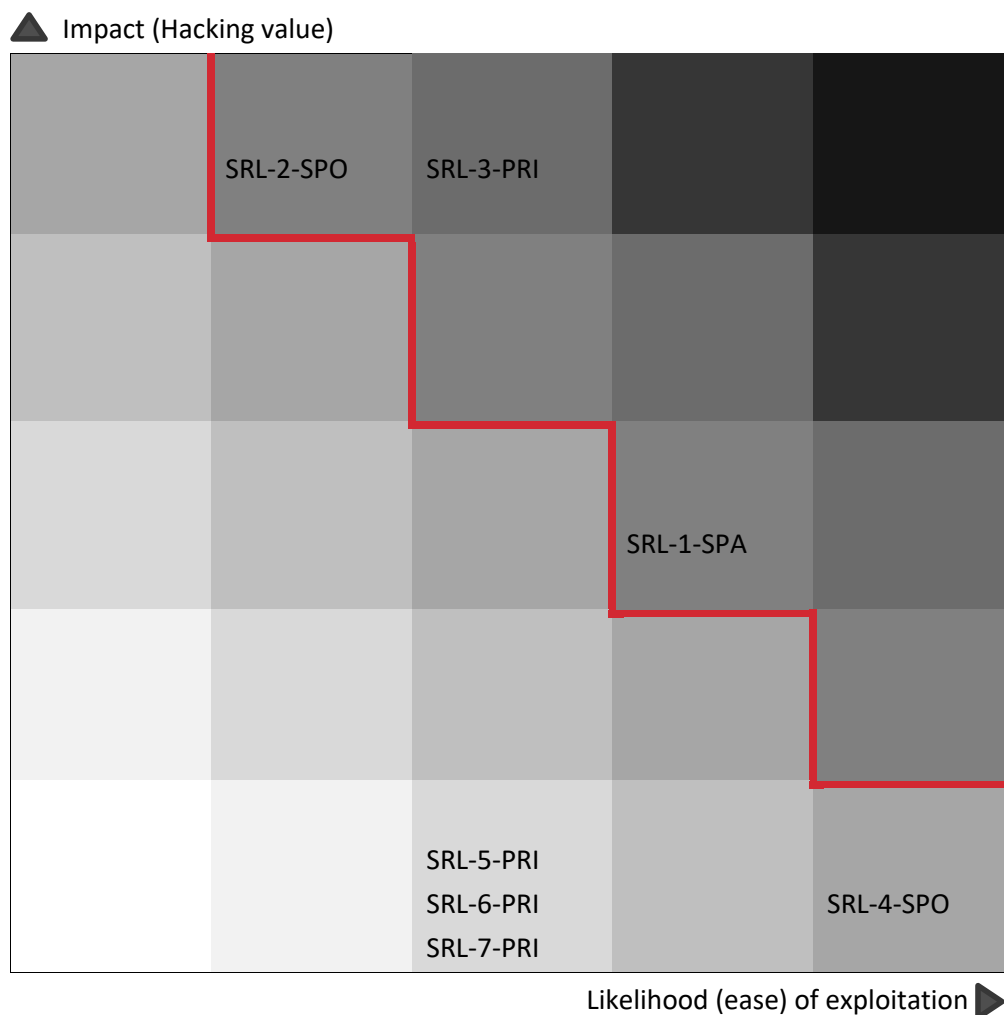
6 Findings summary

The analysis of the Phoenix protocol identified **seven** issues in line with the possible attack scenarios described in Section 4.3. In summary the findings amounted to zero critical severity, three high severity, one medium severity, three low severity, and zero info level issues were found.

Severity	Count
Critical	–
High	3 
Medium	1 
Low	3 
Informational	–
Total	7

6.1 Risk Profile

The table below contains the findings' identifiers according to the impact and likelihood of exploitation, increasing to the top right. The red margin separates high/critical issues from informational/low/medium ones.



6.2 Issue Summary

Issue	Severity	Status
SRL-1-SPA: Friendship tokens can be passed on to maliciously add users to a group	High	Mitigated
SRL-2-SPO: User identity confusion for client credential authenticated calls	High	Mitigated
SRL-3-PRI: Server can link users to Privacy Pass token redemption	High	Mitigated
SRL-4-SPO: Existing friendship tokens can be overwritten by other users	Medium	Mitigated
SRL-5-PRI: Queue sizes allow inferring the number and size of messages	Low	Accepted
SRL-6-PRI: FQDNs of federated servers visible to honest-but-curious QS	Low	Accepted
SRL-7-PRI: Insufficient anonymity set size in small instances	Low	Accepted

Table 5: Findings overview

7 Findings details

This section describes the findings of this audit in detail. Each finding is presented through an initial description of the underlying problem, its associated risk, and potential mitigation strategies. Additionally, each issue is assigned a severity level based on the likelihood of it taking place and its estimated impact.

7.1 SRL-1-SPA: Friendship tokens can be passed on to maliciously add users to a group

Scope category	SPAM
Reference	SRL-1-SPA
Severity	High
Status	Mitigated in PR 32 [6]

Issue description

This first issue has to do with friendship tokens that can be passed on to maliciously add users to a group. In our analysis, we could identify that a user, which we will call Bob, could have his friendship token and friendship encryption key passed on to a third party, allowing them to add Bob to a group and send him messages without his consent. Friendship tokens are static, and each connection partner receives the same token. This means that another user, like Alice can establish a connection with Bob, obtain his friendship token, and pass it on to a third malicious user, Eve. Eve can then add Bob to a group without having to establish a connection with Bob first.

Risk

The risk of this issue is that a user can be added to a group without the user's permission. The specification describes that users must have established a previous connection so that a user can add another to a group and send them messages. However, as shown in our example, Bob and Eve do not have a connection.

Mitigation

Our proposed mitigation is to ensure affected users ignore unsolicited group invitations or messages. This mitigation relies on the expectation that the affected user whose friendship token is shared is has knowledge of which other users received their friendship token. As such our mitigation suggestion is to ensure this best practice of ignoring unsolicited requests is reflected in through guidance and development changes:

- **Guidance:** Currently, the specification does not explain how users should deal with incoming groups by users who they have no connection with. This best practice should be explicitly included in the documentation to ensure users can validate incoming group requests.
- **Per-user friendship token:** Instead of a single symmetric, long-lived friendship token, the user can create per-connection friendship tokens to achieve the following:
 - keep a record of which other users received which Friendship Token, and
 - validate incoming messages' provenance to be from that user.

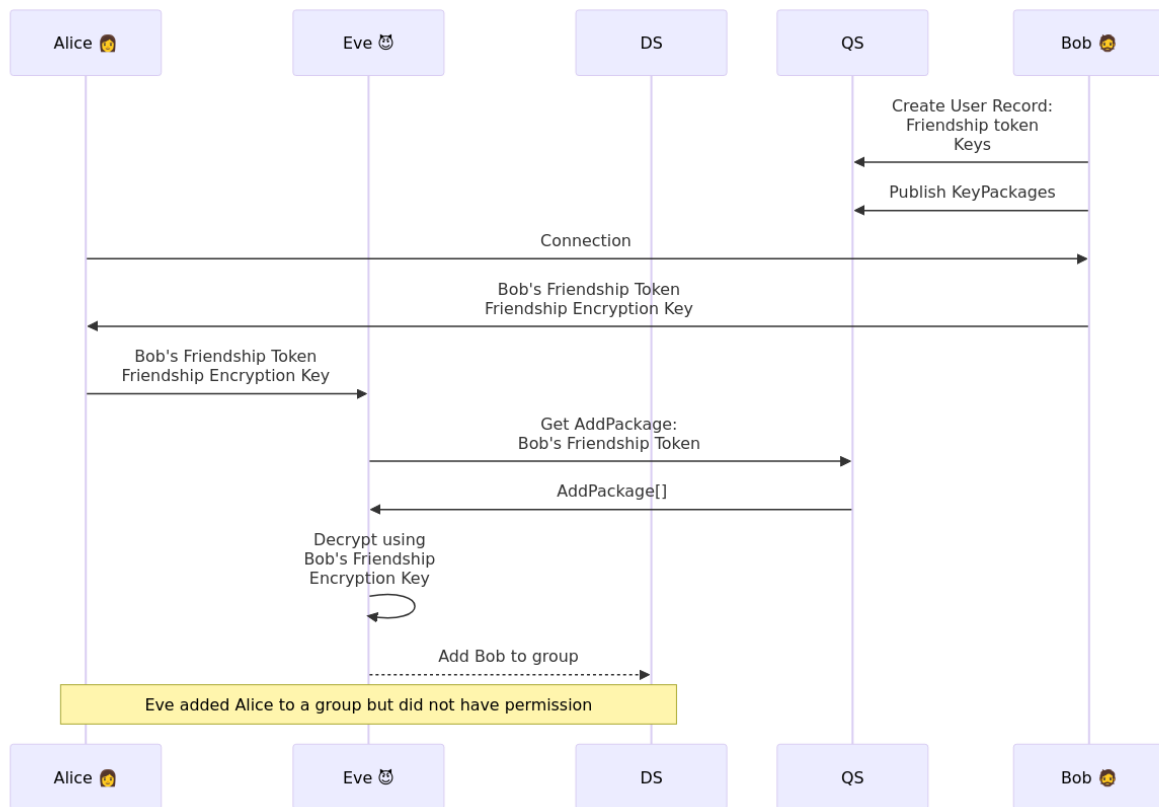


Figure 10. Friendship token can be passed on

7.2 SRL-2-SPO: User identity confusion for client credential authenticated calls

Scope category	Spoofing
Reference	SRL-2-SPO
Severity	High
Status	Mitigated in PR 26 [7]

Issue description

This issue is related to user identity mix-ups during certain authenticated calls. The protocol's documentation outlines that certain calls require two user IDs to be provided by the caller; one for the server to identify the entity on which to perform the action, and another for the authentication. The required authentication identity is called `ClientCredential` and it contains the calling user's ID. The server thus receives two user IDs. The specification does not provide guidance on how to handle mismatching user IDs.

The `ClientCredential` authentication is required for the following calls:

- Initiate 2FA
- Finish User Registration
- Update User Profile
- Upload UserID Packages
- Delete User
- Dequeue Messages

Risk

This issue has the associated risk that duplication of values, such as providing the same user ID through outer authentication and the inner payload, can lead to confusion about the actual user ID or unauthorized actions. For instance, the server could authorize a call based on outer authentication but then use the inner payload's value to determine the user to perform the action on. An example would be the `Update User Profile` call, which requires both a user ID and the `ClientCredential` authentication. While the server may authorize the update, it could potentially upload a new profile for a different user.

Mitigation

To mitigate this issue we suggest either removing the redundant information or its guiding implementors through the following steps:

- **Eliminate redundancy:** We suggest having a single source of truth when it comes to the information about users. This can be achieved through the decision to not require the user ID for calls that are authenticated via `ClientCredential`. Thus, either remove the user ID from those parameters and extract it from the `ClientCredential` or synthesize the `ClientCredential` when receiving a user ID as a parameter.
- **Explicitly require matching user IDs:** Another suggestion is to document the requirement of validating user IDs to mitigate the risk of implementors confusing the user IDs.

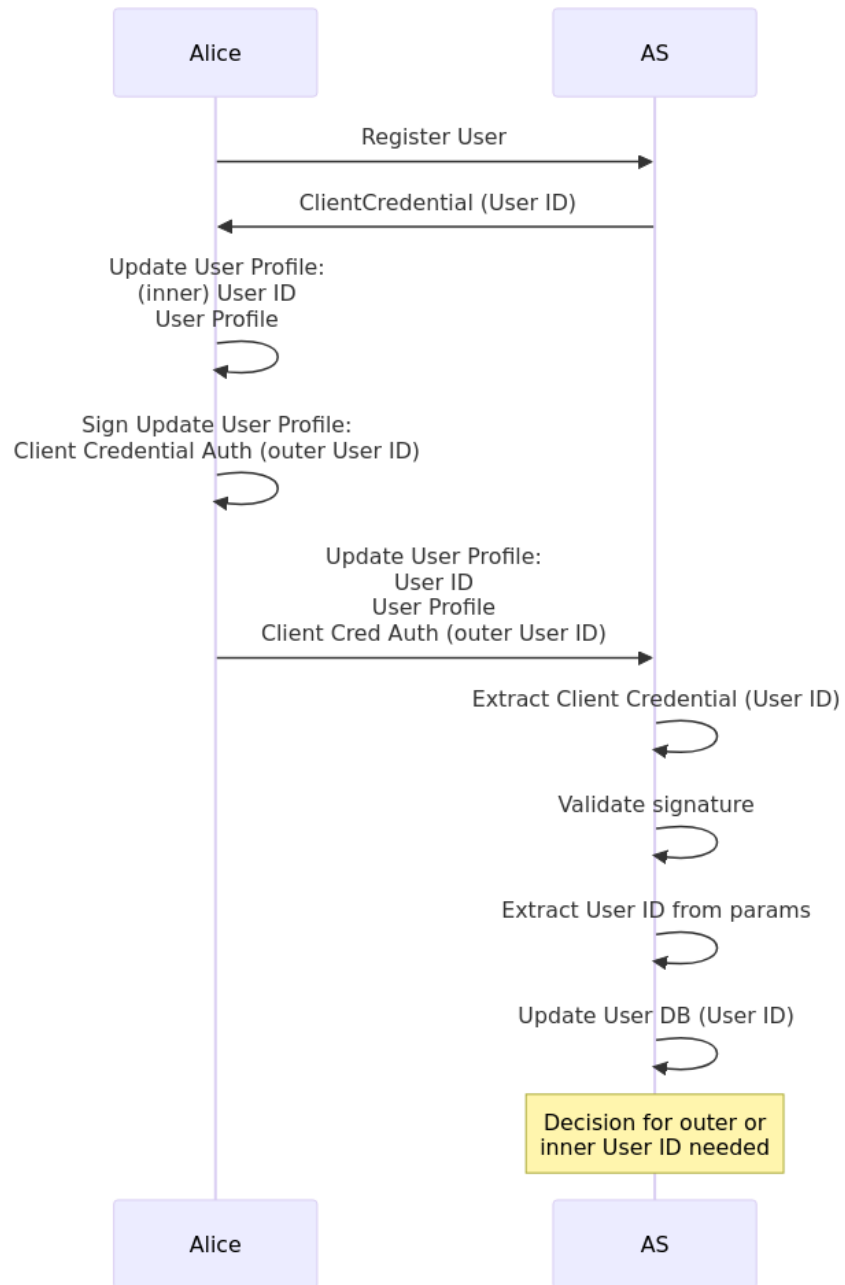


Figure 11 Identity confusion

7.3 SRL-3-PRI: Server can link users to Privacy Pass token redemption

Scope category	Privacy
Reference	SRL-3-PRI
Severity	High
Status	Mitigated in PR 25 [8]

Issue description

This issue is related to the protocol's design reliance on privacy pass tokens to provide unlinkable rate-limitation of clients. Because of this, the server could generate a new private key for each user asking for a token and record the key ID used for the user ID. During redemption, the server could then verify all private keys and find the corresponding user ID.

Risk

This issue rises privacy concerns, as a malicious server could potentially recognize the user redeeming a privacy pass token. The behavior of assigning a unique private key to a user, then recording that key's usage, and finally retrieving the user at the time of token redemption allows a server to link the token's applications to a particular user. This infringes on the user's privacy.

Mitigation

We propose the following approach to mitigate this risk:

- **Transparency:** Assuming the server rotates keys for each epoch, it could publish the key post each rotation. Clients can then validate that the tokens they receive can be verified using a single key. While this does not prevent possible attacks, detection may deter a potential server operator from carrying it out.

7.4 SRL-4-SPO: Existing friendship tokens can be overwritten by other users

Scope category	Spoofing
Reference	SRL-4-SPO
Severity	Medium
Status	Mitigated in PR 32 [6]

Issue description

This issue is related to the documented process in which users upload their KeyPackages to the QS under a freely chosen index, the friendship token. Other users fetch the KeyPackages by querying for the friendship token. The specification furthermore allows for the QS to overwrite an existing friendship token.

Risk

A malicious user (e.g. user Eve) could obtain another user's friendship token, (e.g. from user Alice), and utilize it for uploading a new KeyPackage. Another user, (e.g. user Bob) who established a connection with Alice and obtained her friendship token, could then query the QS for the KeyPackages under that token and expect Alice's KeyPackages in return for adding her to a new group. Instead, the user gets Eve's KeyPackage and adds Eve to the new group, without having established a connection with Alice beforehand.

Mitigation

The specification does not mention the case of a user overwriting an existing friendship token. For this reason, we suggest reducing this gap by documenting and providing guidance through the following principles:

- **Prevent:** Treat the upload of an existing friendship token as an error. Given that the friendship token is a random string chosen by the client, the likelihood of an attacker exhausting the key space and preventing key distribution is low.
- **Validate:** When fetching an AddPackage with a friendship token, the user's validation routine should include whether the AddPackage belongs to the intended party. Since the user knows which party a friendship token belongs to, the user can expect the party's ClientCredential when fetching the AddPackage. An AddPackage with a different party's ClientCredential should be rejected.
- **Delay:** After deleting a queue and the associated friendship token, do not allow populating an AddPackage for a certain time to prevent the squatting of tokens.

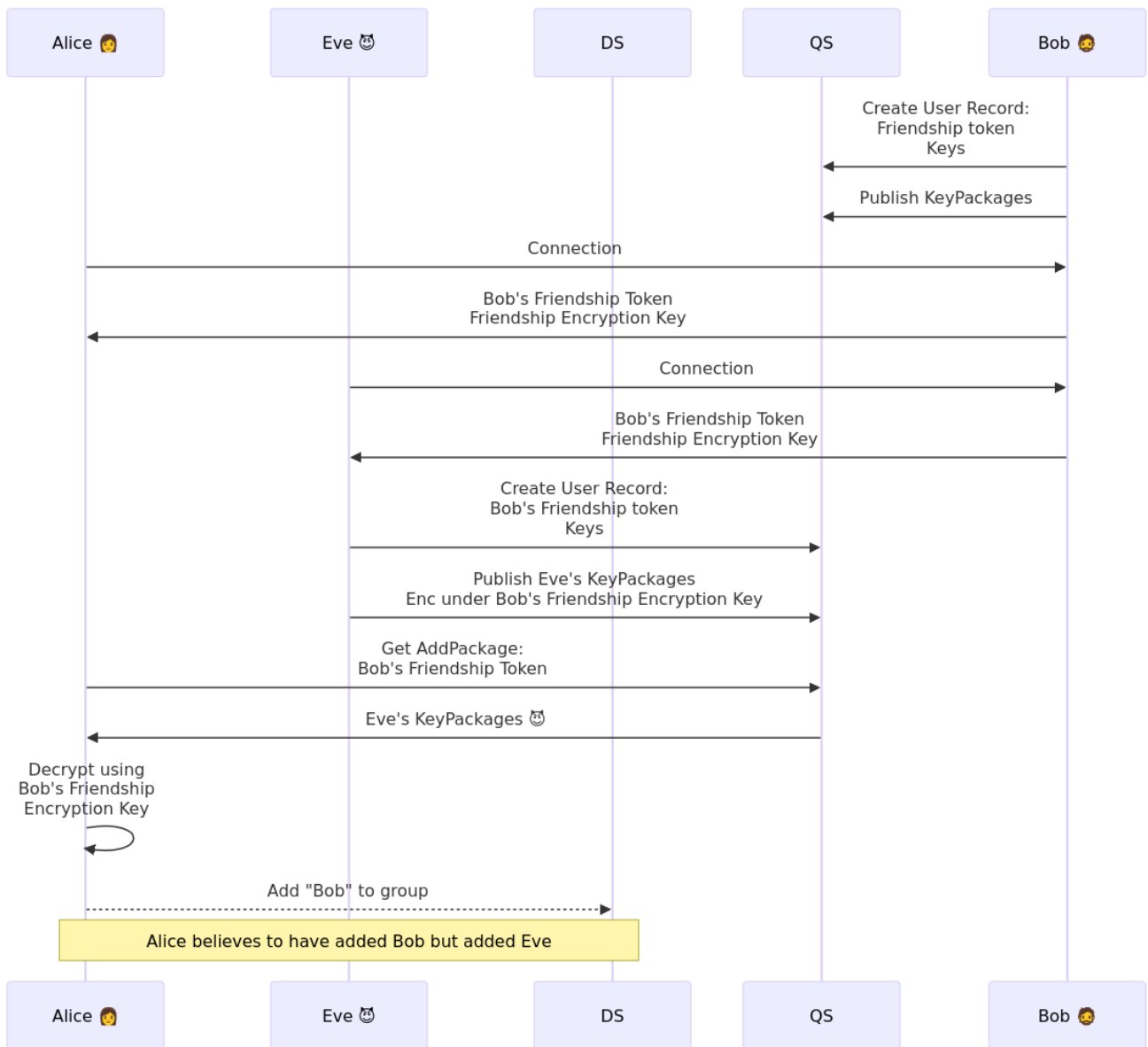


Figure 12 Overwriting friendship token

7.5 SRL-5-PRI: Queue sizes allow inferring the number and size of messages

Scope category	Privacy
Reference	SRL-5-PRI
Severity	Low
Status	Accepted in PR 32 [6]

Issue description

This issue is related to unpadded messages which are added to queues increasing the queue size with each message. A snapshot attacker has the capability to learn and analyze the sizes of the queues within the messaging server. This knowledge can be leveraged to infer information about message traffic.

Risk

This issue can allow a snapshot attacker to infer the number or size of messages being sent to a particular user or an alias. While the use of aliases provides a layer of obfuscation against direct correlation with individual users, the visibility of queue sizes can still lead to unintended exposure of user behavior. An adversary in the search for a busy group could deduce which queue is used by the group by looking at the size of it. Furthermore, this could help organize targeted DoS attacks on specific groups.

Mitigation

We suggest the following mitigation strategies to address this risk:

- **Padding:** In addition to storing the messages in the queues, the queues can be inflated up to a certain number or size of messages to yield a more uniform encrypted size across all queues. For example, all queues could be padded up to the closest multiple of 1000.
- **Grouping:** Multiple queues can be combined so that separate keys decrypt only the relevant queue. By combining multiple queues, the anonymity set of the affected users is bigger, thus making it more difficult to learn something about a particular client or alias.
- **ORAM:** Advanced cryptographic schemes, such as Oblivious RAM, allow for operating over data without the server learning which data is accessed and which operations are performed on them.
- **Updating the protocol specification:** As this issue is hard to mitigate and has a low severity, another potential mitigation is to explain this issue and its potential mitigations at the implementation level.

7.6 SRL-6-PRI: FQDNs of federated servers visible to honest-but-curious QS

Scope category	Privacy
Reference	SRL-6-PRI
Severity	Low
Status	Accepted in PR 32 [6]

Issue description

This issue is related to the federated nature of the protocol. In a federated setup, the use of FQDN to route messages across different servers (e.g., `phnx.shady-company.de` and `phnx.parliament.de`) could lead to the correlation of user data. The QS can access the user's home server FQDN through `ClientQueueConfig` when a message is sent.

Risk

An active observer, as well as an honest-but-curious home server, can log the outgoing messages' FQDN to infer information about the server's users. This information could be leveraged to reveal sensitive information about a user's interests and affiliations, even if the actual content of the messages is encrypted. This has a more severe effect on smaller servers (as in they are not having many registered users), as correlating a few users to outgoing domains may reveal the nature of the interests of such users.

Mitigation

This issue is hard to mitigate as it comes by design with a federated setup. However, we recommend improving the documentation in the following ways:

- **Document risks:** State the risks different adversaries pose, the data the attackers have access to, and the knowledge they can infer.
- **Transfer risks:** Another route to partially mitigate the risk is enforcing a Tor [9] onion route for each FQDN as well as federating only with `.onion` FQDNs, which rotate keys (and associated domains) regularly. For better usability, we recommend documenting this as an optional (opt-in) configuration for servers.
- **Updating the protocol specification:** As this issue is hard to mitigate and has a low severity, another potential mitigation is to explain this issue and its potential mitigations at the implementation level.

7.7 SRL-7-PRI: Insufficient anonymity set size in small instances

Scope category	Privacy
Reference	SRL-7-PRI
Severity	Low
Status	Accepted in PR 32 [6]

Issue description

In home servers with a small number of users, the correlation between the user activities and the stored data, e.g., groupid, userid, queues, and timings, is much easier than in larger instances. The small anonymity set size results in an attacker needing less additional information to attribute a usage pattern to an individual user. Therefore, users relying on smaller servers can have insufficient anonymity protections.

Risk

The risk of a snapshot attacker attributing a particular usage of the service to an actual user is higher in small instances since there are fewer users that could be causing the usage. The usage can be attributed by correlating a user ID with an alias, group ID, queues, message frequency, data volume, and recipients. This information can be leveraged to infer and leak sensitive user information such as how many groups a user is participating in, or to launch targeted denial-of-service (DoS) attacks against specific users. The possibility of attributing a particular use of the service to a user is breaching the privacy promises made by the service.

Mitigation

Small anonymity set sizes inherently reduce the protection an individual user can enjoy. Finding technical measures to increase the protection for a small number of users conflicts with providing a modern user experience to be able to compete in the messaging market as many academic attempts have shown. We suggest documenting the resulting risk and finding technical solutions that do not compromise user convenience:

- **Quantify degree of anonymity:** Establish metrics to quantify privacy risks in relation to the number of users on the home server. For example, an instance with a single user allows for trivial attribution of the observed usage.
- **Minimum user threshold:** Implement a minimum threshold for the number of users on a home server before starting operation. Before an acceptable number of users has been reached, the server could let users during sign-up know that operation will start soon.
- **Padding:** Letting both the server and users generate fake data to inflate the anonymity set size could increase the user's protection. A snapshot attacker cannot distinguish artificial data from actual usage. The protection is less strong under an honest-but-curious server operator since they can mark artificial and actual data.
- **Updating the protocol specification:** As this issue is hard to mitigate and has a low severity, another potential mitigation is to explain this issue and its potential mitigations at the implementation level.

8 Detailed recommendations

In addition to the high-level recommendations outlined above, we propose a series of more specific and actionable steps to address some key areas of concern. These detailed recommendations are designed to provide further clarity on how to strengthen the development processes.

We group our suggestions into three categories: The attacker, the protocol, and the defense.

8.1 Attacker

Adversary: Design around an honest-but-curious attacker

The Phoenix protocol is designed to primarily defend against the so-called snapshot attacker. The snapshot attacker can read data at a certain point in time but not continuously monitor the operations. We recommend making the honest-but-curious attacker the primary adversary to defend against privacy-related attacks. The honest-but-curious attacker can extend logging at their discretion and keep historical records of data. We believe that this attacker model better reflects the risks for Phoenix as it requires less trust in the server operator and is therefore better suited to gain the users' trust.

Data: Analyze data accessible by each attacker in a worst-case scenario

We recommend defining and enumerating the data that could be obtained or inferred by each type of attacker in a worst-case scenario. We further suggest performing this enumeration in the form of a list that outlines potential vulnerabilities and the specific data that could be exposed based on the attacker's access level (e.g., malicious client, compromised server, or external attacker). By explicitly detailing these risks, the protocol can better address mitigation strategies and ensure that any potential data leaks are minimized and well understood by developers and users alike. We recommend conducting this exercise from the perspective of each defined attacker.

In particular, the honest-but-curious server operator mentioned before is interesting because the Phoenix server operator could be coerced into changing their behavior in ways that are not visible to an outsider, such as extending the logging or correlating data from the AS, DS, and QS. The projected userbase is interested in an analysis of the security guarantees based on both the protocol and the operator's behavior. By making explicit which data can be accessed or inferred from an honest-but-curious operator, users gain trust in the product and leave less room for speculations or insinuations.

DoS vectors: Quantify computationally expensive functions and potential DoS vectors

We recommend explicitly clarifying which party—client, server, or other—is responsible for the estimated computational, network, and I/O overhead in the provided calculations. While it is currently implied that both the client and the home server share the computational effort, the distinction between their respective workloads should be more clearly defined. This is particularly important for identifying potential DoS vectors. Understanding which entity bears the bulk of the effort will highlight areas of vulnerability. For example, computationally heavy operations like group joining (estimated at $5n-10n$ asymmetric operations) could be exploited if this cost is predominantly borne by the server. Similarly, fan-out messaging in federated environments might create network amplification risks, which need mitigation strategies, such as rate limiting. By clearly attributing these efforts, it will be easier to analyze performance and security risks.

8.2 Protocol description

Client-side: Describe the protocol around core use cases

The specification describes the services provided by the home server and intends to guide implementors to create such services. However, it predominantly takes the perspective of the server side. We recommend to also include the client's perspective by describing core processes, such as registration, connection establishment, group creation, or messaging. Currently, the specification does not provide an overview of the messages sent or API calls required to achieve a certain goal. Describing how and in which order the API calls are meant to be used helps both implementors and reviewers to understand the protocol.

Stored data: Explain the use of all types of persisted data

We recommend explaining the reasons behind data storage for each item of the persisted state. Currently, the specification lists the data the AS, DS, and QS persist. For example, the AS requires user entries with OPAQUE records, the user's encrypted profile, and their last activity timestamp. For most items, the specification either mentions what the data is used for or links to further information, e.g., the OPAQUE entry exists because it is a "protocol artifact that allows the user to authenticate itself via its password in queries to the AS".

However, some items such as the activity timestamp do not have an associated explanation for their persistence. The semantic of the data is explained as follows: "Timestamp indicating the last time a client has fetched messages from the queue". However, there is no explanation as to why the server needs it for operating the services.

As a general practice, the server should store as little data as possible, and each stored item should be motivated by a technical necessity for operating the service.

Single client: Clarify the constraints for multiple clients per user

The current protocol specification assumes that each user has a single client. However, this assumption is inconsistent throughout the specification, as some sections still consider the possibility of multiple clients per user. This inconsistency stems from the fact that a previous version of the protocol supported multiclient functionality. Phoenix has indicated that future updates may reintroduce multiclient support through potential enhancements to the MLS standard, such as "virtual clients" [10]. We recommend maintaining consistency in the specification by enforcing the one-client-per-user model and clarifying that multiclient support may be added in future updates.

Additionally, the STRIDE assessment showed that users can retrieve the set of clients without restriction. In a future multiclient scenario where the server allows this without any access controls, an attacker could repeatedly query this information, mapping out the user's devices or identities, and potentially leading to a privacy breach. This functionality would expose a user's set of clients to unauthorized parties, revealing the number of devices in use. An attacker could exploit this to profile users, track device usage, or link anonymous accounts to a known individual. This would pose a significant privacy risk, especially in environments where anonymity is crucial.

Key transparency: Reflect other products' developments

We suggest revising the "Transparency" column in the messaging apps comparison table, reflecting recent updates for WhatsApp and iMessage. Specifically, WhatsApp announced the implementation of key transparency in an engineering post in April 2023 [11]. Similarly, Apple announced iMessage contact key verification [12], signaling a shift towards key transparency. It would be prudent to amend the table entries for both apps to depict their latest security improvements accurately. These adjustments will ensure a more current and comprehensive comparison.

8.3 Security posture

Post-compromise security: Let users deactivate their accounts

Messaging platforms often face threats from state actors that may force users to surrender their credentials or access to an authenticated client, compromising the account's security. A possible solution could be a mechanism to deactivate accounts, which can be activated after such events. This would secure the account further without compromising message confidentiality or privacy. Advanced alternatives like pre-made tokens for account disabling, designated revocation users, or a revocation list would further enhance user protection.

Secret keys: Guide the storage of and analyze the leak of private or secret key material

The specification references various private and secret keys including the AS credential key material, the OPAQUE key material, and the privacy pass key material. For some of these keys, like the AS credential key material, it suggests protection measures such as keeping them "offline or in an HSM". However, for others like the privacy pass key material, no guidance is provided. Recommendations should include a risk assessment of the potential consequences of unprotected keys. For instance, if privacy pass keys are compromised, it simplifies an attacker's ability to make calls to the server, thereby easing enumeration.

Patents: Choose cryptographic primitives and schemes by patent-free algorithms

We suggest incorporating an additional criterion in the list of requirements to ensure the cryptographic primitive and scheme are patent-free. Such a strategy will guarantee unrestricted implementation, distribution, and future development of cryptographic algorithms or schemes. By clarifying this aspect early on, potential legal or financial roadblocks can be avoided, thereby promoting wider protocol adoption.

9 Bibliography

- [1] Phoenix, "Protocol specification," [Online].
Available: <https://docs.phnx.im/>.
- [2] IETF, "RFC9420," [Online].
Available: <https://www.rfc-editor.org/rfc/rfc9420.html>.
- [3] O. t. fund, "Open tech fund," [Online].
Available: <https://www.opentech.fund/>.
- [4] Phoenix, "Implementation of Phoenix home-server," [Online].
Available: <https://github.com/phnx-im/infra/>.
- [5] Wikipedia, "STRIDE Model," [Online].
Available: https://en.wikipedia.org/wiki/STRIDE_model.
- [6] Phoenix, "PR 32 - Phoenix documentation," [Online].
Available: <https://github.com/phnx-im/docs/pull/32>.
- [7] Phoenix, "PR 26 - Phoenix documentation," [Online].
Available: <https://github.com/phnx-im/docs/pull/26>.
- [8] Phoenix, "PR 25 - Phoenix documentation," [Online].
Available: <https://github.com/phnx-im/docs/pull/25>.
- [9] Tor, "Tor project," [Online].
Available: <https://www.torproject.org/>.
- [10] IETF, "Memo Virtual clients MLS," [Online].
Available: <https://www.ietf.org/archive/id/draft-kohbrok-mls-virtual-clients-00.html>.
- [11] Whatsapp, "Key transparency engineering post," [Online].
Available: <https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/>.
- [12] Apple, "Contact key verification engineering post," [Online].
Available: <https://security.apple.com/blog/imessage-contact-key-verification>.

10 Appendix A: SRLabs technical services

Security Research Labs delivers extensive technical expertise to meet your security needs. Our comprehensive services include software and hardware evaluation, penetration testing, red team testing, incident response, and reverse engineering. We aim to equip your organization with the security knowledge essential for achieving your objectives.

SOFTWARE EVALUATION We provide assessments of application, system, and mobile code, drawing on our employees' decades of experience in developing and securing a wide variety of applications. Our work includes design and architecture reviews, data flow and threat modeling, and code analysis with targeted fuzzing to find exploitable issues.

BLOCKCHAIN SECURITY ASSESSMENTS We offer specialized security assessments for blockchain technologies, focusing on the unique challenges posed by decentralized systems. Our services include smart contract audits, consensus mechanism evaluations, and vulnerability assessments specific to blockchain infrastructure. Leveraging our deep understanding of blockchain technology, we ensure your decentralized applications and networks are secure and robust.

POLKADOT ECOSYSTEM SECURITY We provide comprehensive security services tailored to the Polkadot ecosystem, including parachains, relay chains, and cross-chain communication protocols. Our expertise covers runtime misconfiguration detection, benchmarking validation, cryptographic implementation reviews, and XCM exploitation prevention. Our goal is to help you maintain a secure and resilient Polkadot environment, safeguarding your network against potential threats.

TELCO SECURITY We deliver specialized security assessments for telecommunications networks, addressing the unique challenges of securing large-scale and critical communication infrastructures. Our services encompass vulnerability assessments, secure network architecture reviews, and protocol analysis. With a deep understanding of telco environments, we ensure robust protection against cyber threats, helping maintain the integrity and availability of your telecommunications services.

DEVICE TESTING Our comprehensive device testing services cover a wide range of hardware, from IoT devices and embedded systems to consumer electronics and industrial controls. We perform rigorous security evaluations, including firmware analysis, penetration testing, and hardware-level assessments, to identify vulnerabilities and ensure your devices meet the highest security standards. Our goal is to safeguard your hardware against potential attacks and operational failures.

CODE AUDITING We provide in-depth code auditing services to identify and mitigate security vulnerabilities within your software. Our approach includes thorough manual reviews, automated static analysis, and targeted fuzzing to uncover critical issues such as logic flaws, insecure coding practices, and exploitable vulnerabilities. By leveraging our expertise in secure software development, we help you enhance the security and reliability of your codebase, ensuring robust protection against potential threats.

PENETRATION & RED TEAM TESTING We perform high-end penetration tests that mimic the work of sophisticated attackers. We follow a formal penetration testing methodology that emphasizes repeatable, actionable results that give your team a sense of the overall security posture of your organization.

SOURCE CODE-ASSISTED SECURITY EVALUATIONS We conduct security evaluations and penetration tests based on our code-assisted methodology, allowing us to find deeper vulnerabilities, logic flaws, and fuzzing targets than a black-box test would reveal. This gives your team a stronger assurance that the significant security-impacting flaws have been found and corrected.

SECURITY DEVELOPMENT LIFECYCLE CONSULTING We guide organizations through the Security Development Lifecycle to integrate security at every phase of software development. Our services include secure coding training, threat modelling, security design reviews, and automated security testing implementation. By embedding security practices into your development processes, we help you proactively identify and mitigate vulnerabilities, ensuring robust and secure software delivery from inception to deployment.

REVERSE ENGINEERING We assist clients with reverse engineering efforts not associated with malware or incident response. We also provide expertise in investigations and litigation by acting as experts in cases of suspected intellectual property theft.

HARDWARE EVALUATION We evaluate new hardware devices ranging from novel microprocessor designs, embedded systems, and mobile devices via consumer-facing end products over to core networking equipment that powers Internet backbones.

VULNERABILITY PRIORITIZATION We streamline vulnerability information processing by consolidating data from compliance checks, audit findings, penetration tests, and red team insights. Our prioritization and automation strategies ensure that the most critical vulnerabilities are addressed promptly, enhancing your organization's security posture. By systematically categorizing and prioritizing risks, we help you focus on the most impactful threats, ensuring efficient and effective remediation efforts.

SECURITY MATURITY REVIEW We conduct comprehensive security maturity reviews to evaluate your organization's current security practices and identify areas for improvement. Our assessments cover a wide range of criteria, including policy development, risk management, incident response, and security awareness. By benchmarking against industry standards and best practices, we provide actionable insights and recommendations to enhance your overall security posture and guide your organization toward achieving higher levels of security maturity.

SECURITY TEAM INCUBATION We provide comprehensive support for building security teams for new, large-scale IT ventures. From Day 1, our ramp-up program offers essential security advisory and assurance, helping you establish a robust security foundation. With our proven track record in securing billion-dollar investments and launching secure telco networks globally, we ensure your new enterprise is protected against cyber threats from the start.

HACKING INCIDENT SUPPORT We offer immediate and comprehensive support in the event of a hacking incident, providing expert analysis, containment, and remediation. Our services include detailed forensics, malware analysis, and root cause determination, along with actionable recommendations to prevent future incidents. With our rapid response and deep expertise, we help you mitigate damage, recover swiftly, and strengthen your defenses against potential threats.