Secure Contact Agreement Protocol For Messenger Services Through Randomized ID Assignments

Michael Maigwa M. Kangethe
School of Computing and Informtics
University of Nairobi
Nairobi, Kenya
email: mich01mk@gmail.com

Dr Elisha Odira Abade
School of Computing and Informtics
University of Nairobi
Nairobi, Kenya
e-mail: eabade@uonbi.ac.ke

***Abstract*—Messenger services' security and privacy are implemented through TLS/SSL and E2E Encryption. However, they expose users' conversations and networks through traffic metadata. The proposed solution uses a model similar to the TLS handshake with random ID assignments as a contact sharing agreement mechanism.**

Keywords*— CID, PKI Shared Secret, E-2-E End to End, D-2-D Device to Device, MQTT, Message Queuing Telemetry Transport, XMPP Extensible Messaging and Presence Protocol, Metadata, Protocol, Contact Configuration Message.

I. INTRODUCTION

Messenger apps have evolved to adapt with the use and advancement of IoT and mobile communications. D-2-D (Device to Device) communication has become the backbone architecture of most if not all messenger communication technologies. Recently technologies such as MQTT and XMPP have become popular amongst instant messaging services. This has led to the adoption of end-to-end (E-2-E) security measures that prevent interception of personal communications. End to End encryption has been fairly researched and implemented in current messenger Services however due to server/broker technologies monitoring and controlling traffic between clients, the need to develop new security solutions for clients' anonymity arises. The proposed solution implements anonymity and traffic abstraction through Identity obfuscation and dynamic client settings configuration.

II. MESSENGER PROTOCOLS

D-2-D communication protocols differ from conventional Client-Server communication protocols through distributed data processing, storage, and presentation. These technologies decentralize the stated functions to the endpoints and leave the server with the task of only routing communications and in some cases user's Key Management, and data backup operations. Three of the most popular technologies currently used are the XMPP and MQTT and AMQP protocols.

MQTT (Message Queuing Telemetry Transport) protocol developed in 1999 is a lightweight publish/subscribe messaging protocol designed for M2M (machine to machine) telemetry in low bandwidth environments. This protocol was initially implemented in popular messaging apps like WhatsApp and the earlier versions of Facebook Messenger since its packets were small represented in bytes, and didn't contain too much metadata, unlike string messages. Because MQTT clients don't have addresses like email addresses, phone numbers, etc. One didn't need to assign addresses to clients as you do with most messaging systems. This was a good security implementation as it enabled flexible source or clients' anonymizations.

XMPP(Extensible Messaging and Presence Protocol) originally named Jabber is an open communication protocol designed for instant messaging (IM), presence information, and contact list maintenance (3). Based on XML (Extensible Markup Language), it enables the near-real-time exchange of structured data between two or more network entities (4). Designed to be extensible, the protocol offers a multitude of applications beyond traditional IM in the broader realm of message-oriented middleware, including signaling for VoIP, video, file transfer, gaming, and other uses. AQMP (Advanced Message Queuing Protocol) is an open standard application layer protocol for message-oriented middleware. The defining features of AMQP are message orientation, queuing, routing including point-to-point and publish-and-subscribe, reliability and security (5).

III. SECURITY IMPLEMENTATIONS

Nearly all messenger protocols have the same operating principle as the previously discussed protocols MQTT, XMPP, and AQMP. For this, the security measures implemented are similar in most cases and only differ in minor details as explained below. Security implemented discussed in this paper focuses on the following guides with the assumption of all other factors being constant and device-dependent:

- The security of Communication: This is the prevention of the messages between parties being intercepted and read by any adversaries (MITM).
- End-user management Security: These are the security measures available either by default or as an option from the protocol technologies.
- Internal privacy policies implementation: These are the measures taken by the service providers to ensure all client communication is not only secure from adversaries but also those managing the services and who do have access to the servers.

Messenger service secure their communications by implementing End to End(E-2-E) encryption in their client applications.

A. End to End Encryption

The operating principle behind the end-to-end encryption is by having the client app generate secure symmetric keys and session keys that will be used to communicate with other clients using the same service provider. This prevents the traffic from being intercepted and exposed to those managing the service as it has become a necessary security implementation since government agencies have become involved in the direct interception of confidential communications. The exchange of these keys is secured by the use of the PKI in their initial connection.

Most services manage usage by requiring users to create accounts while also implementing extra security as 2FA/MFA ensuring no one user can register as another existing user. This is the most common security implementation in all services both public and private service providers.

Internal privacy policies implementation is always subject to debate since those who provide the services do have access to the communication and can without the user's permission and notice decide on what to intercept or log from their users communications and data. Privacy policies are mere promissory agreements between the service providers and the users of their services that are not entirely verifiable by the users which have led to some cases of data being leaked out and analyzed by either marketing companies or government agencies.

B. *End to End Confidentiality*

To ensure End to end confidentiality the following should be evident:

1. No client data is stored in the server
2. Keys and identities are managed by the clients
3. Client CIDs are created and verified by other clients and not the server.
4. Client IDs are verified using shared secrets.

IV. Security Gaps and Threats

A. *Traffic and Connection Metadata Interception*

Messenger technologies have been built with the end users' privacy in mind. However recent events indicate that the contrary is true. Several companies providing "free" messenger services and apps to the public have been shown to deliberately violate users' privacy by collecting communications metadata that they own then offer the metadata for sale to 3rd parties. Recent exposures of privacy violations by Facebook's WhatsApp are a clear example of how companies and service providers will stop at nothing in their effort to acquire as much intelligible data about their users as possible. These privacy violations of client communication have led to lawsuits specifically in the case where Facebook-owned WhatsApp was fined a record 225 million euros ($267 million) by Ireland's data watchdog for breaching EU data privacy rules. In an FAQ on its website, WhatsApp states that it shares phone numbers, transaction data, business interactions, mobile device information, IP addresses, and other information with Facebook. It says it does not share personal conversations, location data, or call logs (10). This information might sound trivial to any layman but it is sufficient for the use of advanced analytics tools and techniques used in the mapping of both user relationships and communication patterns.

E-2-E encryption, Strict User 2FA, and MFA are desirable measures in securing communications between users, however, an existing trend has re-emerged that exposes not only user-specific data by service providers but also communication patterns and users' networks and relationships. Several cases have been exposed revealing the ability to acquire useable information from targets without the need of revealing the actual communication such as the Facebook Cambridge Analytica Scandal, where in the 2010s, personal data belonging to millions of Facebook users was collected without their consent by British consulting firm Cambridge Analytica, predominantly to be used for political advertising (7). Facebook has since acquired WhatsApp and changed its privacy policies which allowed them to share certain user data, like phone numbers, with the parent company Facebook (8).

B. *Metadata exposure*

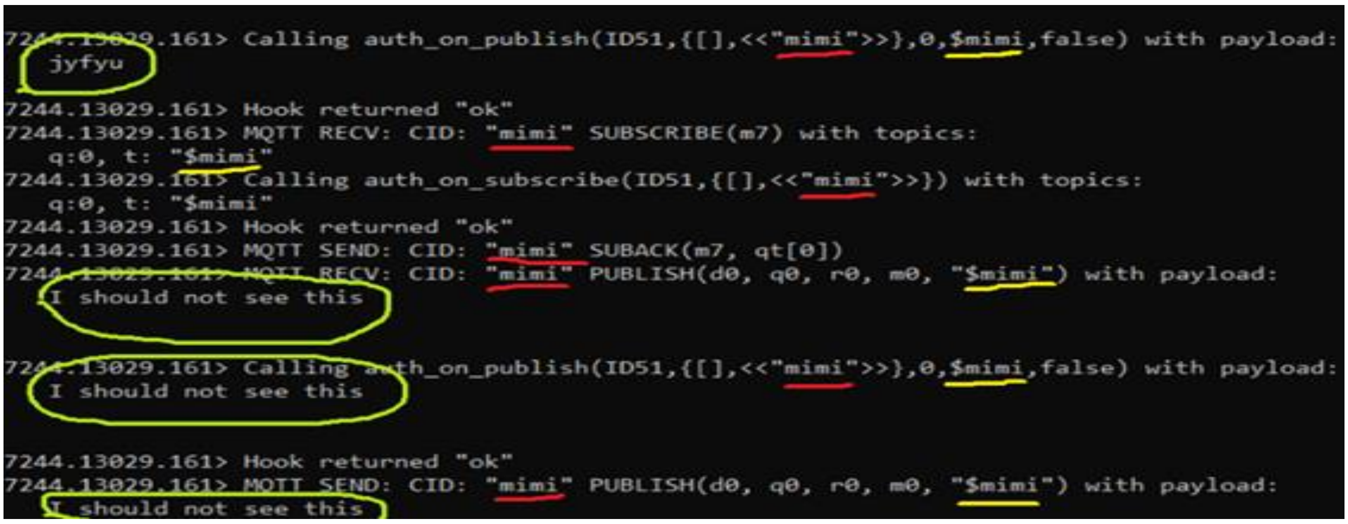
Messenger applications expose people's communication in two main ways.

- Through Traffic metadata. This is where the traffic from the app to the service provider is interceptable. This however doesn't pose a great risk since the communication and the sender-receiver information is encrypted using E-2-E and also the application of SSL/TLS Security.
- The second form of security is from the inside server. The metadata information retrievable between users include:
- Sender -this is the person sending a message.
- Recipient: Person receiving the message
- Time: message and presence timestamp
- Type of Information (File/Text message)
- Reset Keys and triggers (These are functional messages between messenger apps used to perform a particular background function e.g., Update user Private Key, etc.)

Different Messenger protocols represent data differently but since the overall structure and operating principles are similar this paper will focus on only two.

For MQTT the example where an administrator can monitor a client is through setting the logs as shown in the pictorial example detailing sspecific ttechnologies and commands used in this example as shown below.

- The server used is VerneMQ Version 1.2.0 however this still works with version 1.12.1.
- The command used to log the client's communication is vmq-admin trace client client-id="mimi" where mimi is the CID being monitored



```
7244.13029.161> Calling auth_on_publish(ID51,{[],<<"mimi">>},0,$mimi,false) with payload:
  jyfyu

7244.13029.161> Hook returned "ok"
7244.13029.161> MQTT RECV: CID: "mimi" SUBSCRIBE(m7) with topics:
   q:0, t: "$mimi"
7244.13029.161> Calling auth_on_subscribe(ID51,{[],<<"mimi">>}) with topics:
   q:0, t: "$mimi"
7244.13029.161> Hook returned "ok"
7244.13029.161> MQTT SEND: CID: "mimi" SUBACK(m7, qt[0])
7244.13029.161> MQTT RECV: CID: "mimi" PUBLISH(d0, q0, r0, m0, "$mimi") with payload:
  I should not see this

7244.13029.161> Calling auth_on_publish(ID51,{[],<<"mimi">>},0,$mimi,false) with payload:
  I should not see this

7244.13029.161> Hook returned "ok"
7244.13029.161> MQTT SEND: CID: "mimi" PUBLISH(d0, q0, r0, m0, "$mimi") with payload:
  I should not see this
```

Fig1. MQTT Message Server Log

The above picture shows the logging done by the broker administrator of clients connected. Since the actual message (Circled in green) can be encrypted as should, the other client's metadata such as the message time, CID (highlighted in red), and the topics the client is listening to (Highlighted in yellow), will be retrievable enabling the owner of the service to map all their client's communication patterns and relationship networks using automated analysis tools.

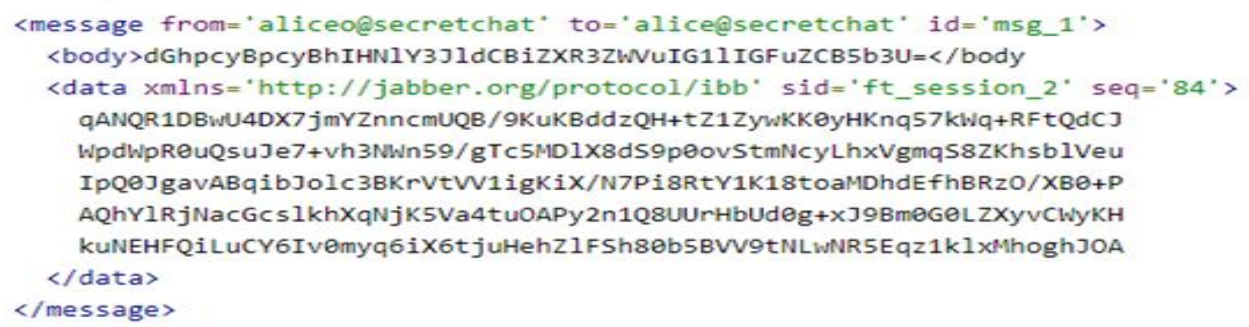
For XMPP this can be done more easily than in MQTT this is because of the appended metadata information appended to a message as shown below.

D. *Privacy Gaps Observations*

In as much as people rightfully demand and require privacy in their communications, it is noted that some systems can violate their privacy policies without the end-users knowing or being able to do anything about it. This is noted when either the service provider has been compromised by an adversary or is compliant in the violation of privacy.

From the research done the sensitive information leaked is:

1. *The Client ID(CID):* This will enable the interceptors to identify individuals communicating.



```
<message from='aliceo@secretchat' to='alice@secretchat' id='msg_1'>
  <body>dGhpcyBpcyBhIHNlY3JldCBiZXR3ZWVuIG1lIGFuZCB5b3U=</body
  <data xmlns='http://jabber.org/protocol/ibb' sid='ft_session_2' seq='84'>
    qANQR1DBwU4DX7jmYZnncmUQB/9KuKBddzQH+tZ1ZywKK0yHKnq57kWq+RFtQdCJ
    WpdWpR0uQsuJe7+vh3NWn59/gTc5MD1X8dS9p0ovStmNcyLhxVgmqS8ZKhsblVeu
    IpQ0JgavABqibJolc3BKrVtVV1igKiX/N7Pi8RtY1K18toaMDhdEfhBRzO/XB0+P
    AQhYlRjNacGcslkhXqNjK5Va4tuOAPy2n1Q8UUrHbUd0g+xJ9Bm0G0LZXyvCWyKH
    kuNEHFQiLuCY6Iv0myq6iX6tjuHehZlFSh80b5BVV9tNLwNR5Eqz1klxMhoghJOA
  </data>
</message>
```

Fig 2. XMPP Encrypted Message Stanza

From the above message stanza example, it is intuitively easier to know who is communicating with whom and at what time. When intercepting messages and communication traffic all the analyst needs to do is extract the from, to, id, seq, and time to map out the communication patterns between individuals.

C. *Traffic Analysis*

From the discussed privacy gaps discussed earlier, it is possible to create automated traffic and pattern analysis tools that can analyze traffic from the service and extract previously anonymous relationships and communication patterns. One of the several ways for extracting such information was by the use of weighted graph networks which computed the level of associations between individuals communicating using GSM Message Services and Calls (6).

Since most service providers require individuals to provide personal information to use their services, it becomes a greater risk if the clients using the service require stringent levels of privacy and confidentiality either due to the nature of their work or associations. This involves the Sender and Receiver IDs.

2. *Time:* This is the time any client is online or communicating. This information can be retrieved from the service by mapping the times from the logs to the ClientID data.
3. *Message Type:* it can be a file of text but the size of messages can reveal a lot about clients' communications and the XMPP Protocol makes it easier to identify the type since each type has its stanza.

All the above can be mapped into a database for analysis to expose users' communication/relationships and even in some cases habits by the service providers.

V. Proposed Solutions

Due to the earlier security gaps discussed, the objective of the proposed solution is to prevent the analysis of secured communication through metadata extraction especially by the service providers.

It is imperative to propose and implement solutions that prevent the discussed exposure of metadata. Proposed solutions are categorized into four problem areas:

- On-demand reconfiguration of client services such as the scanning of server configurations using QR Codes.
- Randomized generation and assignment of user IDs and listener topics for Communication protocols such as MQTT.
- Message Ambiguity through ubiquity, the type of message should not be easily identifiable. One should not easily tell whether the message is a simple text, file shared, or functional message.
- Nontransferable Contact between people, one cannot share contact information without the contact either knowing or approving

A. *Randomized Contact Generation and Agreement*

The majority of the messenger apps require one to register either their phone number or email to use their services. This centralizes the client's registration data which also makes it hard to obfuscate identities. Some go to the extent of having the keys also generated from the server. For security-conscious entities that rely on extreme confidentiality. It is necessary to hide this information. We propose the introduction of Randomized IDs and Contact information. Randomized IDs are user IDs that are generated at random and abstracted from the user in a way that will allow seamless communication between individuals while abstracting and hiding their real Identities. The proposed protocol is detailed below.

B. *Obscurity Levels*

Contacts ID randomizations in MQTT and XMPP servers are possible if configured to allow anonymized users. With this feature, public use for messenger apps becomes easier since creating users will be a client process.

These levels of contact randomizations between messenger clients enhance the security of communications privacy by generating new random identities at random intervals.

1) *Level 1: Fixed ContactID, Fixed ChatID*

At this level, the client will maintain two IDs one for contact exchange and the other for current and all communications. The process will be achieved where a client will have one contact they can share with the public and once the contact has been added by another client, they will communicate using different ID generated randomly during the initial configuration setup of the messenger app.

2) *Level 2: Fixed original ID, Changeable ChatIDs*

At this level, the client will maintain two IDs one for long-term contact exchange and the other for current and all communications. The process will be achieved where a client will have one contact they can share with the public and once the contact has been added by another client, they will communicate using different randomized changeable IDs.

3) *Level 3: Single Randomized ClientIDs*

At this level, the client will maintain One for all communications. In this setup, the client will first register a user profile during the initial app configuration. From there the app will randomly generate new IDs and reassign them to the ClientID configuration settings and continue communicating using the new ID.

In both Level 2 and 3, every time the app reassigns itself a new ID it broadcasts it to all its contacts using the PKI infrastructure. This will ensure that whenever their clients communicate, their traffic metadata will be nearly impossible to associate over a while.

C. *New Contact sharing*

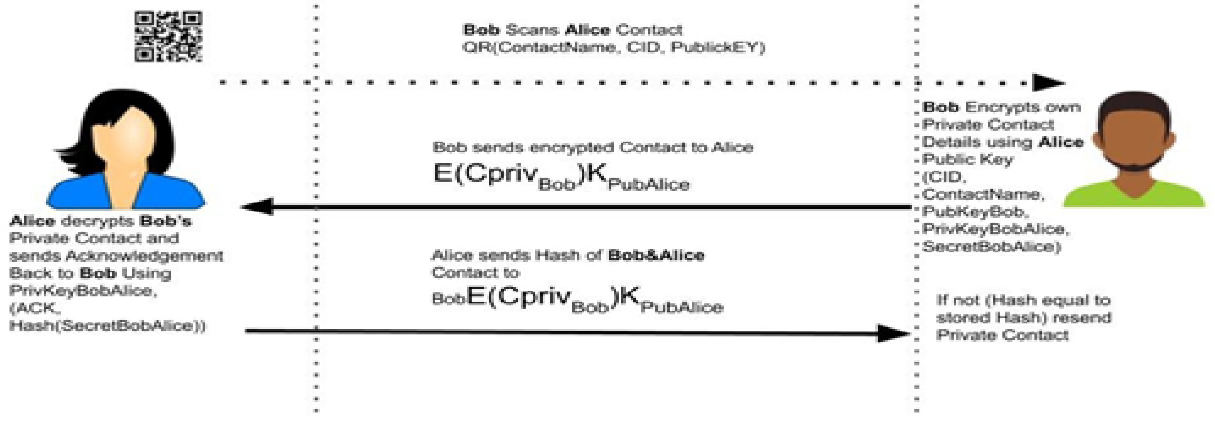

Bob Scans Alice Contact
QR(ContactName, CID, PublickEY)
Bob sends encrypted Contact to Alice
$E(Cpriv_{Bob})K_{PubAlice}$
Bob Encrypts own Private Contact Details using Alice Public Key (CID, ContactName, PubKeyBob, PrivKeyBobAlice, SecretBobAlice)
Alice decrypts Bob's Private Contact and sends Acknowledgement Back to Bob Using PrivKeyBobAlice, (ACK, Hash(SecretBobAlice))
Alice sends Hash of Bob&Alice Contact to
$_{Bob}E(Cpriv_{Bob})K_{PubAlice}$
If not (Hash equal to stored Hash) resend Private Contact

Fig 3. First Contact Agreement Protocol

The above protocol process is executed when a new contact is created or added. The initial contact variables are in plaintext the security purpose is to allow any users with a clear intention of initiating communication will be able to create private contacts and share secure private Contact, verification, and session keys with each other.

```
Start Bob:
{
ScanAliceContactQR(CIDAlice,PubKeyAlice,ContactNameAlice)
GenerateUsingRandom(BobAliceSharedSecret,BobAlicePrivateKey)
EncryptUsingAlicePublicKey(BobAliceSharedSecret,BobAlicePrivateKey, ContactNameBob,CIDBob,PubKeyBob)
=BobAlicePrivateContact
SendToAlice(BobPrivateContact)
}
Alice
{
Receive(BobAlicePrivateContact)
Decrypt(BobAlicePrivateContact)
=(BobAliceSharedSecret,BobAlicePrivateKey,
ContactNameBob,CIDBob,PubKeyBob)
Store(BobAliceSharedSecret,BobAlicePrivateKey,
ContactNameBob,CIDBob,PubKeyBob)
SecretHash= Hash(BobAliceSharedSecret)
SendToBob(CIDAlice, SecretHash)
}
BobGetFormAlice(CIDAlice,SecretHash)
Compare(Hash(BobAliceSharedSecret)= SecretHash)
If equal:
SendConfirm(Bob)
Else:
SendFail:
RepeatContactProtocol
End
```

The Variables used in this protocol are as follows:

- CPub: Public Contact (Plaintext)
- C_{BA}: Private Contact Between Bob and Alice
- UC_{BA}: Updated Private Contact Between Bob and Alice.
- C: CID
- KP: Pub Key
- KPR: Private Key
- S: Shared Secret
- HS: Hashed Shared Secret

$$CPub_A \leftarrow (CID_A \mid KP_A) \tag{1}$$

$$CPub_B \leftarrow (CID_B \mid KP_B) \tag{2}$$

$$C_{AB} \leftarrow E\ (C_B \mid KP_B \mid KPR_{BA} \mid S_{BA})\ KPA \tag{3}$$

$$UC_{AB} \leftarrow E\ (C_B \mid KP_B \mid KPR_{BA} \mid S_{BA} \mid HS_{BA})\ KPA \tag{4}$$

Where E(Plaintext)KP is the public Key encryption using either ECDH or RSA.

To prevent traffic analysis by automated signals intelligence tools and algorithms, the IDs of the end-users will change at random moments not manipulatable or predictable by the user. That way when systems are tracking a conversation between two parties, the identities can change without informing both parties or signaling the change in a predictable and identifiable way to the interceptors.

D. Contact Update

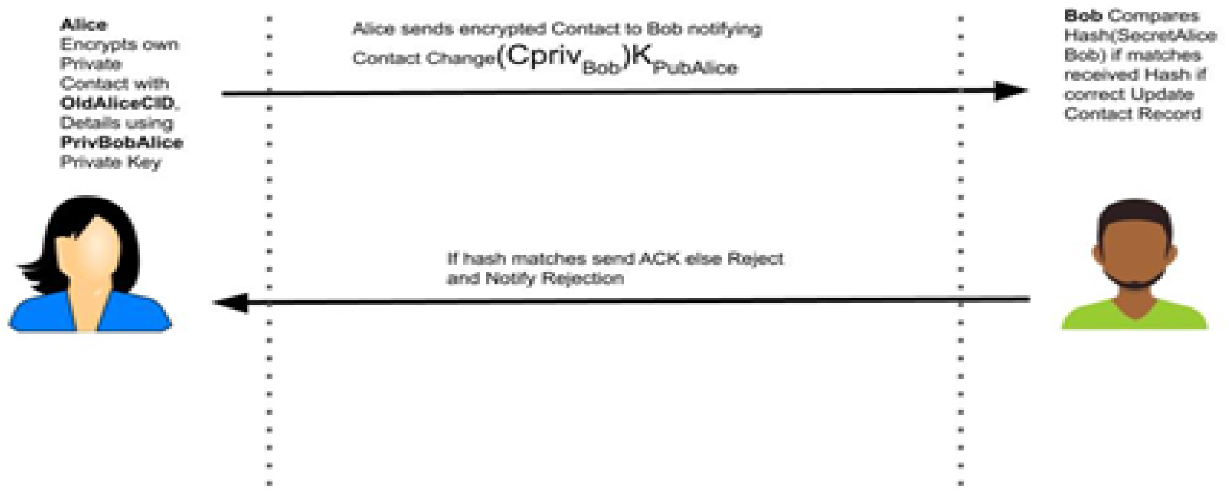

Alice Encrypts own Private Contact with **OldAliceCID**, Details using **PrivBobAlice** Private Key
Alice sends encrypted Contact to Bob notifying Contact Change$(Cpriv_{Bob})K_{PubAlice}$
Bob Compares Hash(SecretAliceBob) if matches received Hash if correct Update Contact Record
If hash matches send ACK else Reject and Notify Rejection

Fig 4: Contact Update

The mode of operation will be implemented as follows:

```
Alice
{
AT Time (X_i)
Generate NewAliceCID
SecretHash= Hash(BobAliceSharedSecret)
EncryptUsingAliceBobPrivKey(NewAliceCID, SecretHash)
=AliceNewContact
SendToBob AliceNewContact)
$X_{i+1} = TimeX_{(i+(Rand()))}$
SetTriggerFor time($X_{i+1}$)
}
Bob
{
GetMessagefromALICE(AliceNewContact)
If(SenderID Not in SavedContact):
Decrypt(AliceNewContact)AliceBobPrivKey
Extract: NewAliceCID, SecretHash
Compare(Hash(BobAliceSharedSecret), SecretHash)
If equal:
SendConfirm(Alice)
Else:
Reject
Else:
Decrypt(AliceNewContact)AliceBobPrivKey
Extract: NewAliceCID, SecretHash
Compare(Hash(BobAliceSharedSecret), SecretHash)
If equal:
SendConfirm(Alice)
Else:
Reject:
NotifyBob
```

1) *Configuration Message*

The plaintext Contact Configuration Message (CCM) between client applications is detailed in the example below:

```
{
"Data":"UpdateContact",
"OCID":"mich01",
"PubKey":"923u21u12fndjf?FDf?Fdskflksdf;;sdfkdsjdshsdjhd",
"CName":"Michael Maigwa",
"NewCID":"CN7N",
"Secret":"7e071fd9b023ed8f18458a73613a0834f6220bd5cc50357ba3493c6040a9ea8c",
"NewSecret":"anLzYaBYZ5SREUyT",
"StegKey":"0000",
"Alg":"229T"
}
```

The parameters necessary for the exchange and updating of client's contacts are:

- "Data":"UpdateContact": This will ensure the message isn't handled like a normal message or displayed rather the app will update its contact Database using the values given.
- "OCID":"mich01": This is the sender's original Contact ID which the recipient will use to compare it with all contacts in its Database. If it already exists then the app will update the records else it will create a new contact record and notify the sender and the user.
- "PubKey":"923u21u12fndjf?FDf?Fdskflksdf;;sdfkdsjdshsdjhd": this is the sender's public encryption key that the recipient should use to send its Updated contact in case of reinstallation or ID regeneration.
- "NewCID":"CN7N": This is the sender's new Identity. That should be updated to ensure the communications and records remain updated and seamless.
- "Secret":"7e071fd9b023ed8f18458a73613a0834f6220bd5cc50357ba3493c6040a9ea8c": This is the Hash of the partial secret or secret exchanged between the sender and receiver. The purpose of the secret is to ensure any contact update request that comes from any sender can be verified and validated to have come from the true sender and not a masqueraded or spoofed sender. Once initial contact has been added the app will use the new contacts public Key to encrypt the private contact details which include the Secret. Then the process for updating the new contact will be done by first comparing the hash of the secret associated with the stored Original ContactID in the Database with the received Secret Hash and if they Match it will update the contact else the message will be rejected.
- "NewSecret":"anLzYaBYZ5SREUyT": only necessary if the sender wants to update the contact's secret

This Contact JSON message will be encrypted using the recipient's Public Key. This way if the sender's ID changes even after the recipient's ID changes.

To note for a client to generate a new ID their data can remain intact and all that needs to be sent over to the peers is a hash of the IDs and if another hash exists then the client will generate a new ID and send its hash over the network to be verified.

2) *Random Topics Generators and Listeners*

By using insecure channels such as MQTT the traffic can be hidden in a way it will be difficult to map out by the use of interchanging topics. It is also advisable to use hidden topics to prevent interceptors from analyzing the traffic by using wildcard listeners.

3) *ContactID Reuse and Conflicts*

Due to the sporadic random ID generation, assignment, and the limited character sizes for each ID the probability of reusing an already created ID increases with the number of App users. This initially would become a challenge if there was no E2E Encryption implementation as messages would end up at the wrong recipient. However, this issue is already resolved in current apps and the contact sharing protocol as only the recipient with the valid shared key between the recipient and sender will be able to.

4) *Reconfigurable Messenger Apps*

Reconfigurable apps are any applications whose server and client settings can be dynamically changed on demand. Unlike most current messenger apps like WhatsApp, Facebook, and Telegram whose server settings are hardcoded, the proposed solution is to have a messenger app whose server settings can be changed by a simple process as scanning a QR Code. This ensures that if by any chance a server/Service is compromised, new settings can be sent directly to a client or posted publicly to ensure the end-users will not need to reinstall the app thus maintaining the data within the device and not the service providers.

5) *Encrypted Backups and Keys*

All apps should be able to encrypt their internal databases backups. This ensures complete access to uses data is only through the app and by using correct keys and app passwords.

E. Proof of Concept (POC) Implementation

The working proof of concept was achieved by the use of the following technologies and their versions

TABLE I. TECHNOLOGIES USED

Technology	***Version Used***
Android Studio IDE	2020.3.1 Patch 2
OPEN JDK	11.0
Ubuntu	20.04
Ejabberd	20.03
Android	7,8,11
Gajim A GTK XMPP client	1.3.2
Spark	2.9.4

Fig 5. Technologies Used During the Research Process

VI. SECURITY AND EXPECTED STRENGTH

To understand and appreciate the level of security implemented through this protocol we must first identify the parameters we use and quantify their complexity in terms of exposure to cryptanalysis attacks specifically brute force.

- *The secret:* the secret is the shared secret between the sender and receiver when creating a connection between two users. The security metric will be based on the number of characters and character space per character.
- *The secret Hash:* This is the computed hash that is shared between contacts when verifying and update requests from a client. The metric in this is the likelihood to find the actual secret from the secret Hash.
- *Contact Agreement exchange exposure:* This is where the app has been reversed and the adversary monitors the app's traffic locally. The adversary will only be able to view traffic to and from the device. Also, manipulation will only be between the device and its contacts.

VII. ADVANTAGES AND LIMITATIONS

We summarize the advantages and limitations of the proposed protocol

A. Advantages

- •Randomized ID/Topic Assignments: This prevents the possibility of the target's communication being traced A graph within a graph is an "inset", not an "insert". The word alternatively is preferred to the word "alternately" (unless you really mean something that alternates).
- The simplicity of the Algorithm: Uses only three verifiable steps for new contacts and two for existing contacts

B. Performance

- •Minimal traffic is exchanged over the network.
- •Pure E2E Exchange security only the sharing contacts can know who's contact belongs to whom .

C. Security

- Based on the TLS/SSL Key exchange protocol.
- Resistant to packet injection and replay attacks

VIII.CONCLUSION

E2E Encrypted communication in messenger services may not be a full-proof solution to much-needed privacy since the communication is hosted by a provider who has access to the messenger traffic, however, if the proposed protocol is implemented correctly, the ability to analyze the traffic and identify patterns reduces significantly with time.

Unlike the Diffie Heilman secret Key exchange which uses multiplicative recursive functions to compute a shared key the proposed protocol doesn't. it simplifies the key exchange by generating a random value then after the contact exchange the way to compute the secret is only done by comparing the hash sent. With the hash of the secret stored in the contact's record.

Another personal measure applicable to all messenger apps is to disable the detailed notifications because other apps might be listening to notifications and can leak out private messages to those apps.

REFERENCES

[1] Wanda, P. and J. Jie, H. (2018). "Efficient Data Security for Mobile Instant Messenger". TELKOMNIKA (Telecommunication Computing Electronics and Control), 16(3), p.1426.

[2] Johansson, Leif (April 18, 2005). "XMPP as MOM - Greater NOrdic MIddleware Symposium (GNOMIS)" (PDF). Oslo: University of Stockholm. Archived from the original (PDF) on May 10, 2011.

[3] Saint-Andre, P. (March 2011). "Extensible Messaging and Presence Protocol (XMPP)": Core. IETF. doi:10.17487/RFC6120. RFC 6120. Retrieved May 4, 2014.

[4] O'Hara, J. (2007). "Toward a commodity enterprise middleware" (PDF). ACM Queue. 5 (4): 48–55. doi:10.1145/1255421.1255424.

[5] Michael M Kangethe, Robert Oboko. "Associations Rankings Model for Cellular Surveillance Analysis". Journal of Computer Sciences and Applications. Vol. 8, No. 2, 2020, pp 40-45.

[6] 6. Chan, Rosalie. "The Cambridge Analytica whistleblower explains how the firm used Facebook data to sway elections". Business Insider. Retrieved May 7, 2020.

[7] 7. Hay Newman, L., 2021. "WhatsApp's New Privacy Policy Just Kicked In. Here's What You Need to Know". [online] Wired. Available at: <https://www.wired.com/story/whatsapp-privacy-policy-facebook-data-sharing/> [Accessed 21 July 2021].

[8] "Openfire: Plugin Developer Guide", Download.igniterealtime.org, 2021. [Online]. Available: http://download.igniterealtime.org/openfire/docs/latest/documentation/plugin-dev-guide.html. [Accessed: 21- Jul- 2021]

[9] E. Team, "The 5 Reasons WhatsApp Could be a National Security Risk", Groupsense.io, 2021. [Online]. Available: https://www.groupsense.io/resources/the-5-reasons-whatsapp-could-be-a-national-security-risk. [Accessed: 21- Jul- 2021].

[10] 10. Sam Shead, "WhatsApp is fined $267 million for breaching EU privacy rules", [Accessed 3-Sep-21], https://www.cnbc.com/2021/09/02/whatsapp-has-been-fined-267-million-for-breaching-eu-privacy-rules.html