1 Introduction

1.1 Data privacy and protection

With GDPR becoming effective in 2018 the focus on data privacy is at its peak. Privacy violation is when sensitive data is exposed to unauthorized actors[3]. OWASP top ten ranks *sensitive data exposure* as 3rd biggest security threat[19].

Recent cases of data leakage has put more attention on data privacy and protection. Some cases are due to poor security measures and could arguably have been prevented. Examples of cases are:

- The infamous Facebook - Cambridge Analytica scandal. Third parties were able to collect data through Facebook Login API.
- Google Plus leak. 500.000 users private data was exposed to third parties through APIs[6].
- Medicaid leak. A medical assistant had accessed patients' health records and exchanged mails with another employee containing the patients' private data[11].

The cases above failed to achieve end-to-end security and the improper handling of sensitive data could have been prevented with appropriate security policies and enforcement technique that enforces these policies.

There is more awareness on how applications deal with data. This add extra concern to the programmer and the application about how sensitive data is handled and protected.

The well-known security enforcement techniques like access controls, firewalls and encryption are inadequate alone and does not ensure end-to-end security[23].

1.2 Information Flow Control

There exist useful security enforcement mechanisms for protecting confidential information such as firewalls, encryption and access control. However, these mechanisms each have their drawbacks.

- *Access* control prevents unauthorized access to information but once access is granted there is no guarantee how that confidential information is handled.

- *Firewall* limits communication from the outside hence isolate and protect information. Yet the firewall have no way of telling if the communication going through violates confidentiality.
- *Encryption* secures information on a channel with only the endpoints being able to access that information. However there is no assurance that once the data is decrypted that the confidentiality of that information is ensured.

The mechanisms mentioned above all have in common that they lack control of how the information flows. Information-flow security aims at protecting confidentiality and integrity of information by enforcing security policies. Information-Flow Control allows the programmer to define and enforce policies in a language-based way[23].

1.3 Matrix

Matrix is an open standard protocol for messaging over HTTP and synchronizing data. Matrix provides secure real-time communication over a decentralized federated network. Matrix secures data by providing end-to-end encryption.

Matrix cover use cases such as instant messaging, VoIP, Internet of Things communication and is generally applicable anywhere for subscribing and publishing data over standard HTTP API.

The fragmentation of IP communication is the problem Matrix essentially wants to solve. Making calls and messages between users needless of which app they use. However they define their longer term goal as *"to act as a generic HTTP messaging and data synchronisation protocol for the whole web"*[8].

1.4 The case study

The goal of the case study is to make secure implementation of a prototype using Information-Flow Control. The case study will use Matrix as the communication channel and strengthen the security at the endpoints using IFC.

1.4.1 Journal system

The prototype implements a journal system and is loosely based on the Danish E-journal system.

Medical privacy is a well-known issue[22]. Sensitive data about patients needs to be handled carefully. In Denmark patients have access to their medical records through E-journal[4]. A patient's journal on E-journal is available for up to 90.000 different medical employees[1].

There are clear policies about who and under what conditions should access a journal. It is legally required that an employee accessing the journal must have the patient in care and that the lookup must be relevant for the employee. Safety measures have been applied through logging and audit trails with random sampling checks however they do not prevent access to journals. Any medical employee can access the patient journal and even if prevention mechanism were established there would be no limitation to what a medical employee could see once access was granted [2][7].

The mechanisms in the current journal system might restrain malicious intend. However it does not guarantee prevention of unintentional access or disclosure of information[15]. What is missing is the enforcement of secure information flow policies. Unintentional access or disclosure of information can be prevented by enforcing policies that define secure information flow.

The prototype will model a simplified scenario of hospitals with different actors accessing a patient journal. The bulk of information on the journal system is extracted from newspaper articles hence there is a high uncertainty of how the system really works. Therefore many assumptions are made about the current system when programming the prototype.

1.4.2 Scope

The objective of the project is to do a secure implementation of the prototype described above. Secure exchange of patient journal is ensured using Matrix and the endpoints are secured using IFC.

A successful project is one that fulfills these criteria:

- Evaluation of Matrix security model
- Survey of IFC tools and selection of tool.
- Implement a prototype distributed system running on Matrix, using the chosen tools
- Demonstrate increased security guarantee with Matrix and IFC

1.4.3 Why Matrix?

In the Digital Strategy 2016-2020 the Danish Agency of Digitisation defines initiative 7.2 as *"Common standards for secure exchange of information"*. The large number of software systems in the Danish public sector has created a need for an uniform way of exchanging data across different application in a secure manner[24].

The initiative has similarities to the issue Matrix is trying to solve with fragmented IP communication. With Matrix security guarantees and their long term goal as a generic HTTP messaging protocol there is a strong case for using Matrix as a communication channel in this case study.

1.5 Method

1.6 Threat model

The threat model is defined in the context of confidentiality and integrity.

- The adversary has the ability to observe information sent over the network.
- The adversary can generate input to the system .
- The adversary can observe public output.

1.7 Contribution

The contributions to the field are the findings of secure implementation using Paragon and how they compare to similar findings from secure implementation with JIF.

The thesis also contributes with the interface created between Paragon and Matrix making it possible to develop other secure applications on top of secure communication channel Matrix provides.

1.8 Structure of thesis

Chapter 2 sets the foundation for the thesis and introduces relevant information and background. Chapters 3 analyzes the Matrix security model and survey IFC tools. Chapter **??** goes in depth with design of the solution. The results are then presented and discussed in Chapter **??**. The thesis is wrapped up in the conclusion section Chapter 6.

1.9 Summary

2 Background

This chapter builds the foundation for the thesis introducing relevant concepts and disciplines for the thesis.

Section 3.1 provides an evaluation of the Matrix security model. The security model is evaluated in the context of a secure messaging system. The paper *SoK: Secure Messaging* describes a evaluation framework for evaluating secure messaging systems. They define several security properties related to such systems [25] which will be presented.

All secure messaging systems with end-to-end encryption are based on the Double Ratchet algorithm from the Signal Protocol which will also be described.

Finally the architecture of Matrix and concepts related to Information-Flow Control will be introduced.

2.1 Information security

Information security is the discipline of protecting information. The key principles in information security are expressed through the CIA model. For a system to be secure these principles should be guaranteed [17].

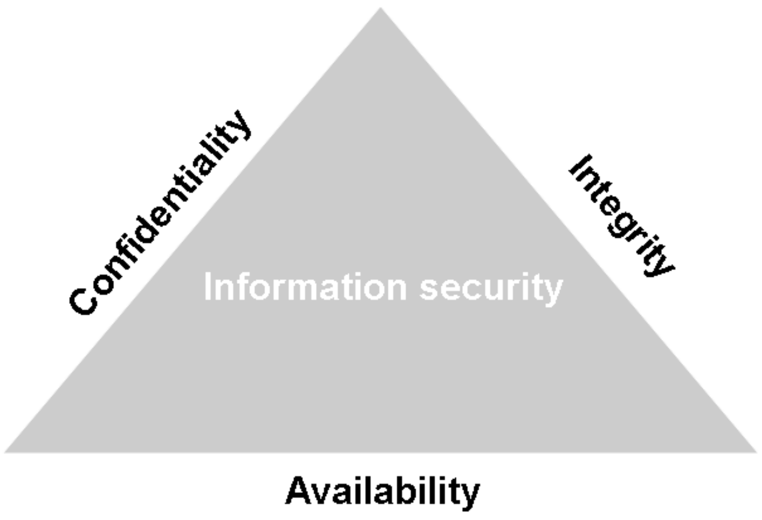


Figure 2.1: CIA triad

Confidentiality Confidentiality is keeping information secret from unauthorized people. This is a major goal in information security. Encryption and access control are common ways of ensuring confidentiality [17].

In a secure messaging system confidentiality would be guaranteed if the message being sent is only readable by the recipient and no one else [25].

Integrity Integrity is providing that information is unaltered and can only be changed by authorized people. If information is intercepted and changed during transit it would be a violation of integrity [17]. More specifically for a secure messaging it would mean that no altered message is accepted by the recipient [25].

Availability Making sure that information is accessible to authorized people is the goal of availability. Denial of Service attack [1] are common attacks targeting availability.

Availability is generally more related to the system being available where the information itself plays a minor role [17].

Depending on the type of system other properties must be satisfied as well.

2.1.1 Security properties

The goal in a secure messaging system is to protect the messages being sent. The following properties are related to protecting messages.

Authentication When a message is received the participant can verify that the message was sent from the actual sender. Furthermore a participant will receive evidence from a participant in a conversation that hey hold a known long-term secret.

Perfect Forward Secrecy If all keys are compromised than the decryption of any previously sent message should not be possible. Hence all previous messages would be secure however all future messages would be insecure

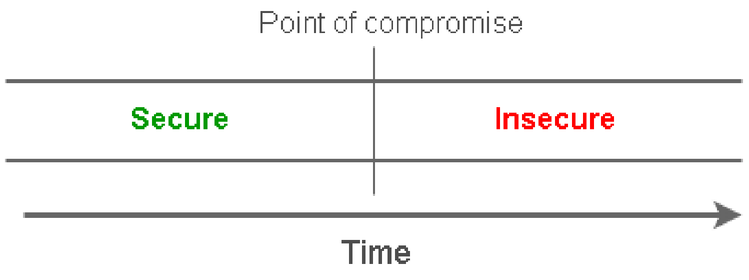


Figure 2.2: Forward secrecy

Backward secrecy If all keys are compromised than the decryption of *future* messages should be possible. This property also goes by the names *future secrecy* and *post compromise security*.

[1]https://en.wikipedia.org/wiki/Denial-of-service_attack

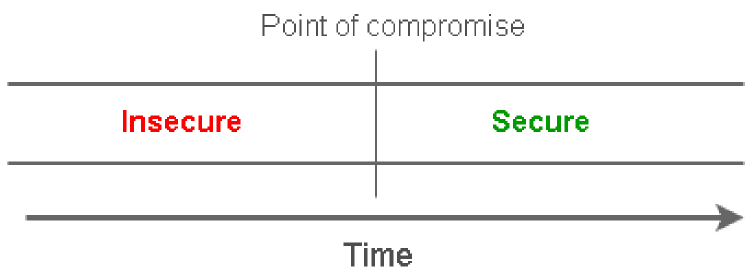


Figure 2.3: Backward secrecy

2.1.1.1 Other security properties

Participant Consistency Whenever a message is accepted by a participant all participants are guaranteed to have identical view of the participant list.

Destination Validation When a participant receives a message it can be verified that the participant was the intended recipient.

Anonymity Preserving The anonymity of the participants should be preserved and not linking any key identifiers.

Speaker Consistency There is consensus among the participants on the sequence of messages they receive by each participant. There might be a mechanism for checking consistency whenever a message is sent or after it has been received.

Causality Preserving Messages must not be displayed before the message that originally precedes it has been displayed.

Global Transcript A global order where all messages are viewed in the same order for all participants.

Deniability Deniability is a property where other participants cannot confirm that the message being sent was from the sender. Yet during the conversation there will be assurance for the recipient that the message being sent was authentic and sent by the sender [25].

- *Message Unlinkability:* A deniability property that gives no guarantees that if a participant sent a message that other messages was sent by that participant as well.
- *Message Repudiation:* It can not be proved that a message was authored by a participant given the conversation transcript and all cryptographic key material.
- *Participant Repudiation:* It can not be proved that a participant was in a group conversation without his conversation transcript and cryptographic key material.

The following properties are also defined in the paper *SoK: Secure Messaging* but are less relevant for security.

Group

- *Computational Equality:* The computational load is equal for all participants.
- *Trust Equality:* There is equal trust among all participants.
- *Subgroup messaging:* In the same conversation a participant can send messages to a subset of the participants.
- *Contractible Membership:* When a participant leaves a conversation the protocol does not need to restart.
- *Expandable Membership:* When a participant joins a conversation the protocol does not need to restart.

Adoption

- *Out-of-Order Resilient:* Messages received out-of-order should be accessible when received.
- *Dropped Message Resilient:* On a unreliable network messages might be dropped in transit however it should not prevent decryption of future messages.
- *Asynchronous:* Messages can be sent securely to recipients while they are offline.
- *Multi-Device Support:* A participant can have multiple devices in a conversation and each device must be synchronized and should have the same historical conversation view
- *No Additional Service:* There is no requirement of additional infrastructure being setup other than the participants.

2.1.2 Concepts

2.1.2.1 Diffie-Hellman Key Exchange

Diffie-Hellman is a key exchange protocol to establish a shared secret over an insecure channel. Public information is send over an insecure and using asymmetric keys two parties can derive the same shared key.

The first step is to agree on some public values. Either of the parties start the protocol by picking a large prime p and a integer g then the values are sent over the insecure channel.

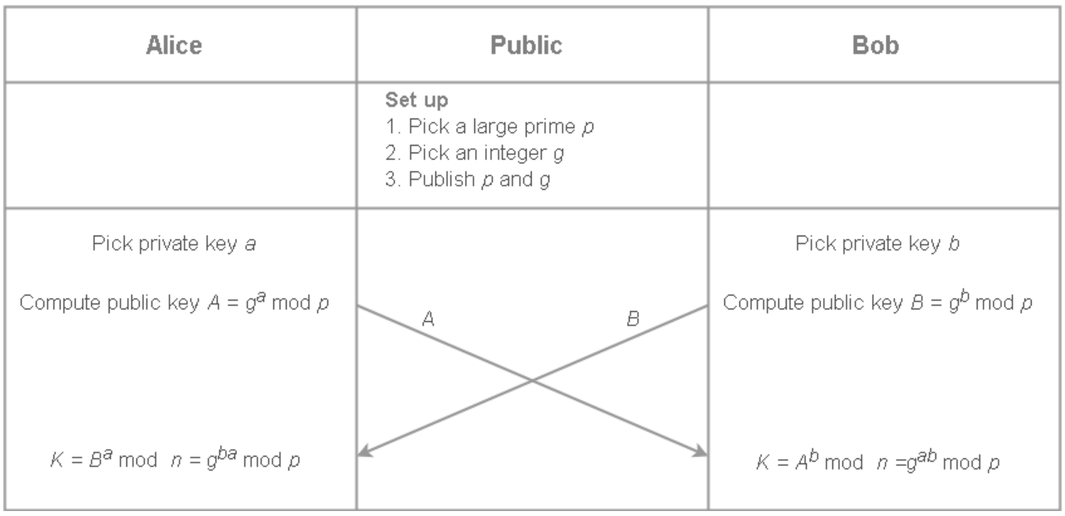

Alice
Public
Bob
Set up
1. Pick a large prime p
2. Pick an integer g
3. Publish p and g
Pick private key a
Compute public key $A = g^a \bmod p$
Pick private key b
Compute public key $B = g^b \bmod p$
A
B
$K = B^a \bmod n = g^{ba} \bmod p$
$K = A^b \bmod n = g^{ab} \bmod p$

Figure 2.4: Simple Diffie-Hellman Key Exchange

Alice and Bob will each pick a private key value and compute a public key value. The computed key value is sent over the insecure channel and Alice and Bob will both perform the same computation as previously [21].

The Diffie-Hellman is vulnerable to man in the middle attack since there is no authentication taking place. There exist a solution to this problem using asymmetric key pairs and signing the messages being sent [21].

2.1.2.2 Key Derivation function

Assume a secret key is established between two parties and is used to encrypt messages and exchange them over an insecure channel. An adversary listening might store all the messages being send even though he is not able to read them. However at some point he manages to compromise the secret key hence being able to decrypting every message ever sent.

To overcome the above scenario ephemeral keys are used. Such keys are short lived and are discarded after use.

New secret keys can be generated using a *Key Derivation Function* (KDF). A KDF is a one way function that derives one or more randomized secret keys based on a secret key (or multuple) and optionally some input value [21]. Figure 2.5 illustrates this.

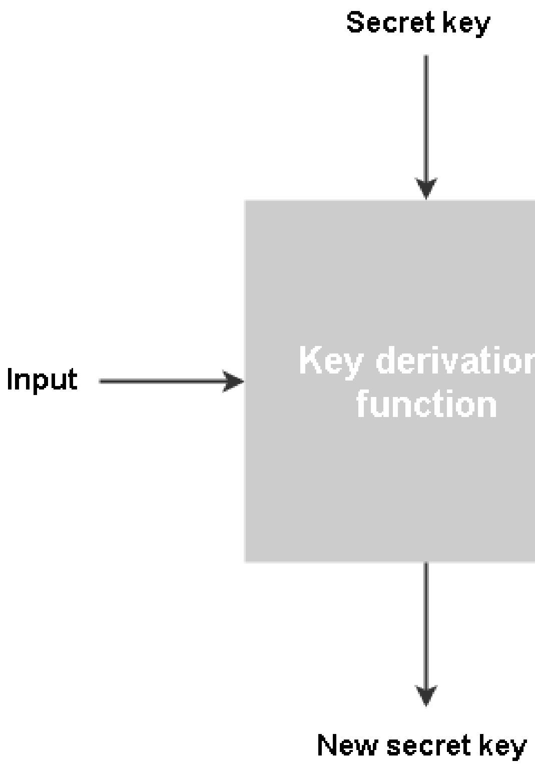


Figure 2.5: Key derivation function

A core concept in the Double Ratchet algorithm is KDF chains which build a chain of secret keys using KDF [20].

2.1.2.3 Signal Protocol

The Signal Protocol provides end-to-end encryption and was developed in 2013 and was first introduced in the app TextSecure[2].

The Signal Protocol consists of two parts; The *Triple Diffie-Hellman protocol* (TripleDH) and the Double Ratchet algorithm.

Triple Diffie-Hellman protocol

Before the Double Ratchet algorithm can be used the two parties communicating need to agree on a shared secret key. In the Signal protocol this is achieved with Triple Diffie-Hellman protocol *(TripleDH).*

The Triple Diffie-Hellman protocol is a *key agreement protocol.* It involves a server and two parties; Alice and Bob.

The TripleDH protocol is characterized by three phases:

1. *Publishing keys:* A identity key and several prekeys belonging to Bob is published by him to a server.

[2]https://en.wikipedia.org/wiki/TextSecure

2. *Sending initial message:* Alice sends an initial message to Bob. A prekey bundle is obtained by Alice from the server in order to send an initial message to Bob.

3. *Receiving initial message:* Alice's message is received and processed by Bob.

Publishing keys Bob needs to register a *prekey bundle* to the server if he wants Alice to be able to send him messages. Alice will likewise have registered a prekey bundle so anyone can to anyone wants to start a message conversation with her. The prekey bundle exists of:

- Identity key IK_B. This key is only published once by Bob.
- Signed prekey SPK_B. This key is reuploaded again after some period of time (eg. after each week or each month).
- Prekey signature *Sig(IK_B, Encode(SPK_B))*. This key is also reuploaded again like the signed prekey.
- Set of one-time prekeys *($OPK_B{}^1$, $OPK_B{}^2$, $OPK_B{}^3$, ...)*. These keys are uploaded by Bob occasionally. Bob is informed by the server when there are few one-time prekeys left.

  To ensure forward secrecy the private key of the one-time prekeys are deleted once Bob received messages that uses them. The signed prekey is deleted as well. However Bob might hold on to it for some time to get the messages that was delayed.

Sending initial message Alice retrieves Bobs public keys from the server. She receives one of Bob's single one-time prekey. The server deletes the one-time prekey that was send. It might be the case that all the one-time prekeys at the server has been used [16].

Alice verifies the prekey signature if the verification fails the protocol is aborted or else the following public keys are provided to generate a shared secret:

- Identity key IK_A. Her own identity key.
- Ephemeral key EK_A. The public key from a generated ephemeral key pair.

To generate the shared secret the following calculations are made:

$$DH_1 = DH(IK_A, SPK_B)$$
$$DH_2 = DH(EK_A, IK_B)$$
$$DH_3 = DH(EK_A, SPK_B)$$
$$DH_4 = DH(EK_A, OPK_B)$$
$$SK = KDF(DH_1||DH_2||DH_3||DH_4)$$

There are performed atleast three Diffie-Hellman where DH_4 is optional depending on if the server had more one-time prekeys.

Authentication is provided by DH_1 and DH_2 while DH_3 and DH_4 provides forward secrecy.

The figure 2.1.2.3 illustrates the calculations.

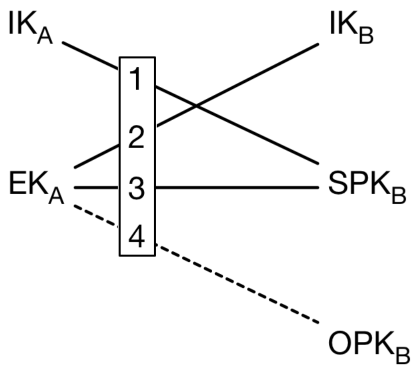


Figure 2.6: Calculations of DH1-DH4 [16].

Alice then deletes the DH values and her private key associated to the ephemeral key to uphold forward secrecy and sends the initial message to Bob which consists of [16]:

- Cypher text with AEAD encryption [3] with *associated data* using her own and Bob's identity keys.
- Her identity key IK_A.
- Her ephemeral key EK_A.
- Some identifiers to which of Bob's prekeys she used.

Receiving initial message When Bob receives Alice's initial message he performs the exact same calculations in phase two and derives the same shared secret key *SK*.

Bob then decrypts the cypher text with the shared key and *associated data* using his own and Alice's identity keys. If the decryption fails the protocol is aborted and *SK* is deleted. Otherwise the protocol is complete and Bob deletes the one-time private prekey. The shared secret key can then be used for the Double Ratchet algorithm [16].

Double Ratchet algorithm

After a shared secret key has been established the Double Ratchet algorithm can then be used to send and receive encrypted messages.

Each party has three chains; root chain, sender chain and receive chain. The chains are KDF chains and will take two keys as input (a KDF chain key and some other input key) and output new two keys (a new KDF chain key and some other output key). The KDF chain is illustrated in figure **??**.

The algorithm has a *Diffie-Hellman ratchet* step and *symmetric ratchet* step and the chains are used across both steps.

- *Diffie-Hellman ratchet:* The parties exchanges new Diffie-Hellman public keys with the messages being sent. New secrets are then derived using

[3]https://en.wikipedia.org/wiki/Authenticated_encryption

Diffie-Hellman (DH). The secret that DH outputs is used as input to the root chain. The root chain then output new chain keys for the receiving and sending chains.

- *Symmetric ratchet:* The sending and receiving chains uses the chain keys derived from the root chain and for each message sent and received the chains are advanced. The output from the receiving and sending chains are keys for encrypting or decrypting messages.

Symmetric ratchet The symmetric ratchet provides message key through the receiving and sending chains. A message key is used for encryption or decryption of a message.

In the symmetric ratchet a single ratchet step is the calculation of the next key chain and message key. The inputs are the current chain key and a constant. Figure 2.7 illustrates two steps in a symmetric ratchet.

Forward secrecy is provided since KDF is a one-way function and it is not possible to go backward and get the input chain key from the output chain key. However since the other input is simply a constant all future keys chain keys and message keys can be derived from an older chain key more specifically there is lack of backward secrecy.

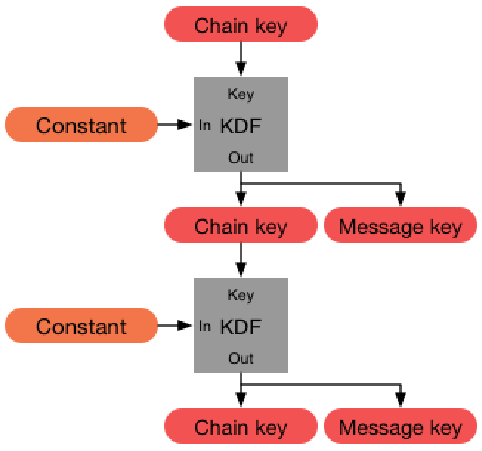


Figure 2.7: Symmetric key ratchet [20].

Diffie-Hellman ratchet Double ratchet provides backward secrecy by comining the Symmetric ratchet with a Diffie-Hellman ratchet hence the name *Double Ratchet*.

Every message from either party begins with a header which contains the sender's current ratchet public key. Whenever a new ratchet public key is received a new ratchet key pair is generated; a secret is derived through Diffie-Hellman with the input being the received ratchet public key and the ratchet private key from the new generated key pair.

Alice starts a conversation with Bob and uses his published public key as a ratchet public key. Alice then generates a new ratchet key pair and derives a shared secret key using Diffie-Hellman and would that as input to her *sending chain*. Alice then sends her new ratchet public key to Bob. At the receiving end Bob derives the same shared secret which would be the input to his *receiving chain*. Alice's sending chain and Bob's receiving chain share the same secret hence he can derive the message key and decrypt the message sent from Alice. When Bob sends a reply to Alice he would generate a new ratchet key pair and derive a new secret which would be input to his *sending chain*.

Figure 2.8 shows an ongoing message exchange with new secrets being derived and the sending and receiving chains being advanced.

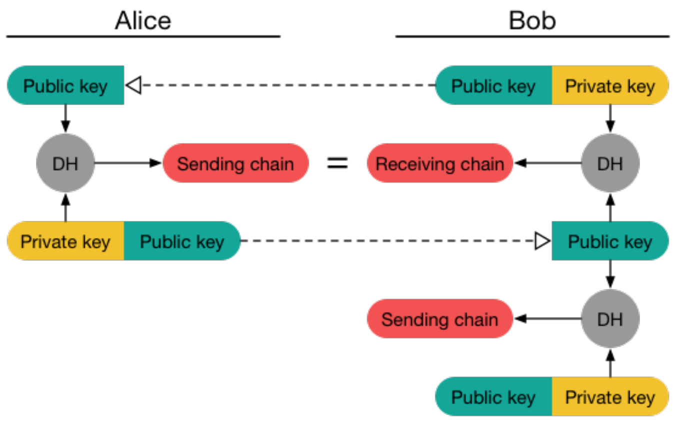


Figure 2.8: Diffie-Hellman ratchet [20].

When Alice receive the reply from Bob she would perform the exact same steps. This ultimately results in a continous loop of generating new ratchet key pairs and using Diffie-Hellman to derive the same shared secret key. A continuation is shown in figure 2.9

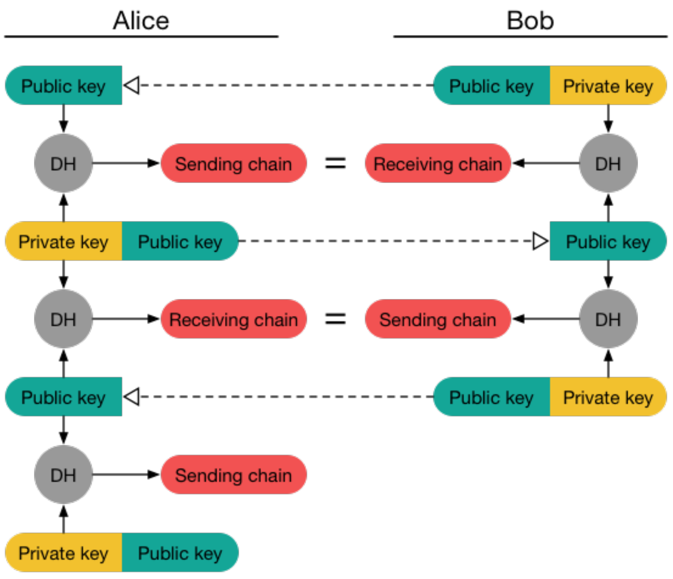


Figure 2.9: Continuation of Diffie-Hellman ratchet [20].

As mentioned in the beginning of the Double Ratchet section the Diffie-Hellman ratchet does have a root chain which would provide inputs to the sending and receiving chains. A more correct view of the process in Diffie-Hellman is shown in figure 2.10.

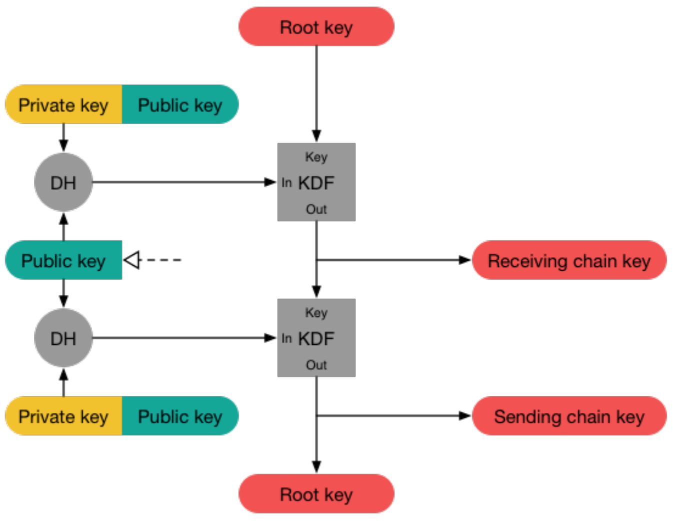


Figure 2.10: Diffie-Hellman ratchet 7 [20].

Double ratchet

When combining Diffie-Hellman ratchet and symmetric-key ratchet the result is the Double ratchet.

- When sending or receiving a message the corresponding message key is derived by performing a symmetric-key ratchet step.
- Upon receiving a new ratchet public key the Diffie-Hellman ratchet step performed right before the symmetric-key ratchet step with the goal of replacing old chain keys with new ones.

Assume that the message exchanged is a continuation from the TripleDH key exchange described in section 2.1.2.3. Alice had sent an initial message. The initial ratchet public key would be Bob's signed prekey SPK_B and the new ratchet key pair would be the Alice's ephemeral key pair that she generated. Alice calculated a shared secret which is the *root key*. She then generates a new ratchet key pair and takes the output from Diffie-Hellman and use it as input for the *root chain*. The root chain then outputs a new root key *RK* and a sending chain key CK.

The figure **??** depicts this with a view of Alice's chains.

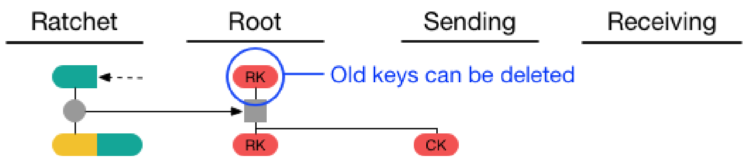


Figure 2.11: Double ratchet 1 [20].

When Alice then sends a message *A1* the symmetric-key ratchet step will return a new chain key and a message key. The message can then be encrypted with the message key.

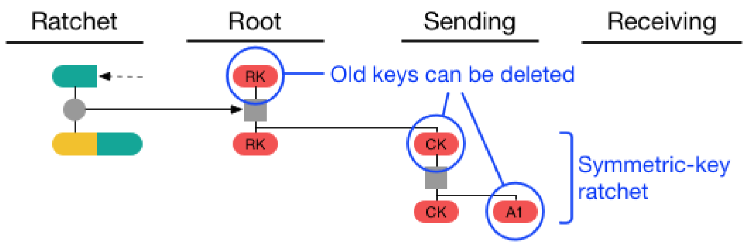


Figure 2.12: Double ratchet 2 [20].

Next Alice receives a message *B1* from Bob. The message header contains a new ratchet public key and a Diffie-Hellman ratchet step is performed. New sending and receiving chain keys are derived and followed by a symmetric-key ratchet step to derive the receiving message key to decrypt the message.

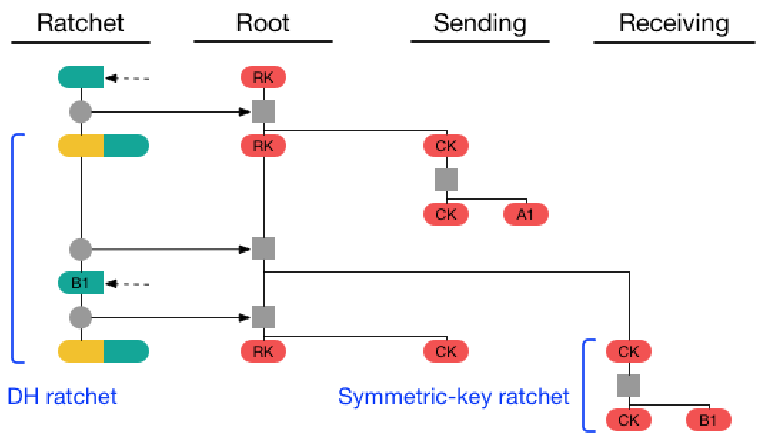


Figure 2.13: Double ratchet 3 [20].

2.2 Matrix

Matrix is a set of open APIs for open-federated Instant Messaging (IM), Voice over IP (VoIP) and Internet of Things (IoT) communication, designed to create and support a new global real-time communication ecosystem. The intention is to provide an open decentralised pubsub layer for the internet for securely persisting and publishing/subscribing JSON objects.

2.2.1 How does it work?

Here are three Matrix homeservers, each with one client connected. The clients are all participating in the same Matrix room, which is synchronised across the three participating servers.

Alice sends a JSON message to a room on her homeserver.

Alice's homeserver adds the JSON to its graph of history, linking it to the most recent unlinked object(s) in the graph.The server then signs the JSON including the signatures of the parent objects to calculate a tamper-resistent signature for the history.

The server then sends the signed JSON over HTTPS to any other servers which are participating in the room.

The destination servers perform a series of checks on the message:

- Validate the message signature to protect against tampering with history
- Validate the HTTP request's auth signature to protect against identity spoofing
- Validate whether Alice's historical permissions allow her to send this particular message

If these checks pass, the JSON is added to the destination servers' graphs.

Destination clients receive Alice's message with a long-lived GET request. (Clients are free to implement more efficient transports than polling as desired).

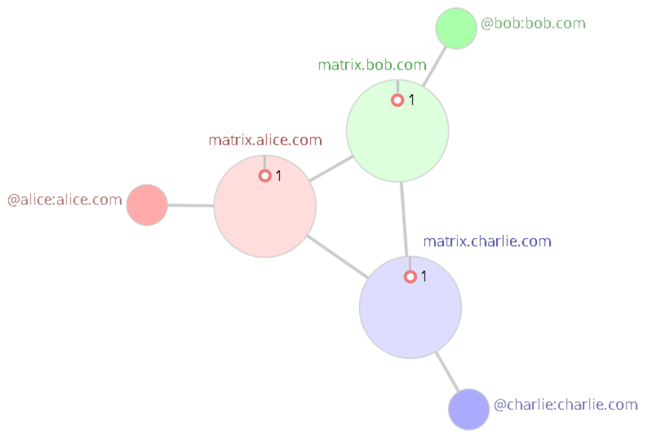


Figure 2.14: Matrix [9].

Bob sends a response to Alice's message, and his server adds his message into his copy of the room's history, linking it to the most recent unlinked object in the graph - Alice's last message.

Meanwhile, Charlie also responds to Alice's message - racing with Bob's message. Alice, Bob and Charlie's homeservers all have different views of the message history at this point - but Matrix is designed to handle this inconsistency.

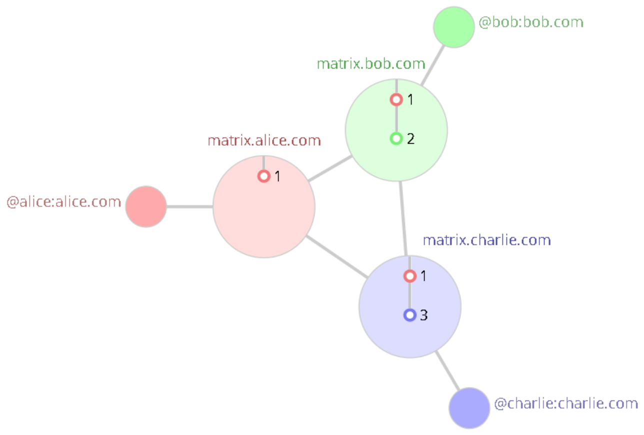


Figure 2.15: Matrix [9].

Bob's homeserver relays his message through to Alice and Charlie's servers, who accept it. At this point Alice and Bob are in sync, but Charlie's room history has split - both messages 2 and 3 follow on from message 1. This is not a problem; Charlie's client will be told about Bob's message and can handle it however it chooses.

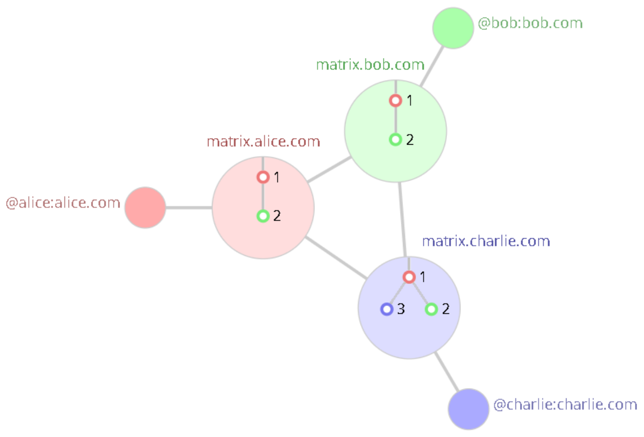


Figure 2.16: Matrix [9].

Charlie's homeserver relays his message through as well, at which point all 3 servers have a consistent view of history again (including the race between Bob and Charlie). All three clients have seen all three messages, and the room history is now back in sync across the participating servers.

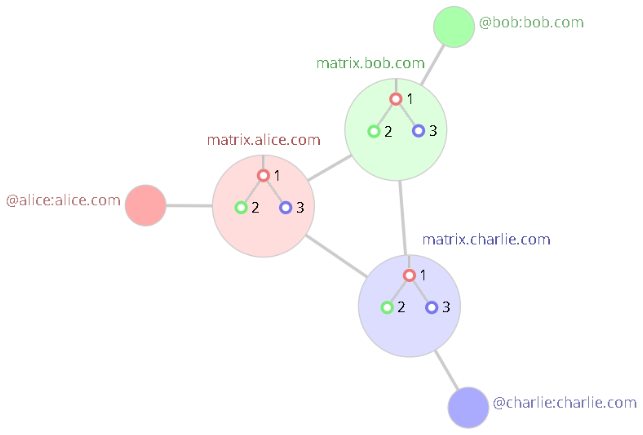


Figure 2.17: Matrix [9].

Later on, Alice sends another message - her homeserver adds it to her history, and links it to the most recent unlinked objects in the graph: Bob and Charlie's messages. This effectively merges the split in history and asserts the integrity of the room (or at least her view of it).

Alice's message is then relayed to the other participating servers, which accept it and update their own history with the same rules, ensuring eventual consistency and integrity of the distributed room.

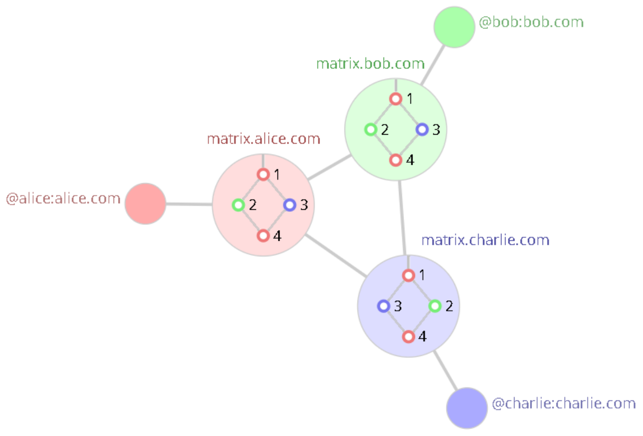


Figure 2.18: Matrix [9].

2.2.2 Architecture

Matrix defines APIs for synchronising extensible JSON objects known as "events" between compatible clients, servers and services. Clients are typically messaging/-VoIP applications or IoT devices/hubs and communicate by synchronising communication history with their "homeserver" using the "Client-Server API". Each homeserver stores the communication history and account information for all of its clients, and shares data with the wider Matrix ecosystem by synchronising communication history with other homeservers and their clients.

Clients typically communicate with each other by emitting events in the context of a virtual "room". Room data is replicated across all of the homeservers whose users are participating in a given room. As such, no single homeserver has control or ownership over a given room. Homeservers model communication history as a partially ordered graph of events known as the room's "event graph", which is synchronised with eventual consistency between the participating servers using the "Server-Server API". This process of synchronising shared conversation history between homeservers run by different parties is called "Federation". Matrix optimises for the Availability and Partitioned properties of CAP theorem at the expense of Consistency.

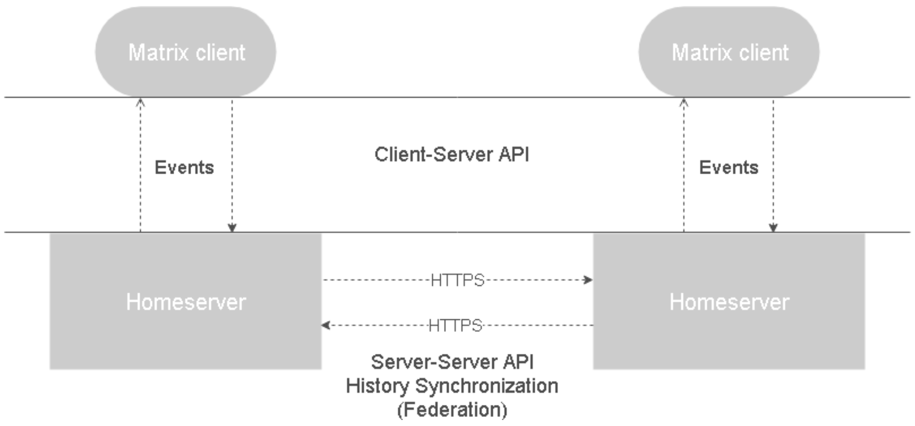


Figure 2.19: Matrix conceptual architecture [10].

2.2.2.1 Event

All data exchanged over Matrix is expressed as an "event". Typically each client action (e.g. sending a message) correlates with exactly one event. Each event has a type which is used to differentiate different kinds of data. type values MUST be uniquely globally namespaced following Java's package naming conventions, e.g. com.example.myapp.event. The special top-level namespace m. is reserved for events defined in the Matrix specification - for instance m.room.message is the event type for instant messages. Events are usually sent in the context of a "Room".

Events exchanged in the context of a room are stored in a directed acyclic graph (DAG) called an "event graph". The partial ordering of this graph gives the chronological ordering of events within the room. Each event in the graph has a list of zero or more "parent" events, which refer to any preceding events which have no chronological successor from the perspective of the homeserver which created the event.

Typically an event has a single parent: the most recent message in the room at the point it was sent. However, homeservers may legitimately race with each other when sending messages, resulting in a single event having multiple successors. The next event added to the graph thus will have multiple parents. Every event graph has a single root event with no parent.

2.2.2.2 Room

A room is a conceptual place where users can send and receive events. Events are sent to a room, and all participants in that room with sufficient access will receive the event. Rooms are uniquely identified internally via "Room IDs", which have the form:

$$!r00m1d : matrix.org$$

There is exactly one room ID for each room. Whilst the room ID does contain a domain, it is simply for globally namespacing room IDs. The room does NOT reside on the domain specified.

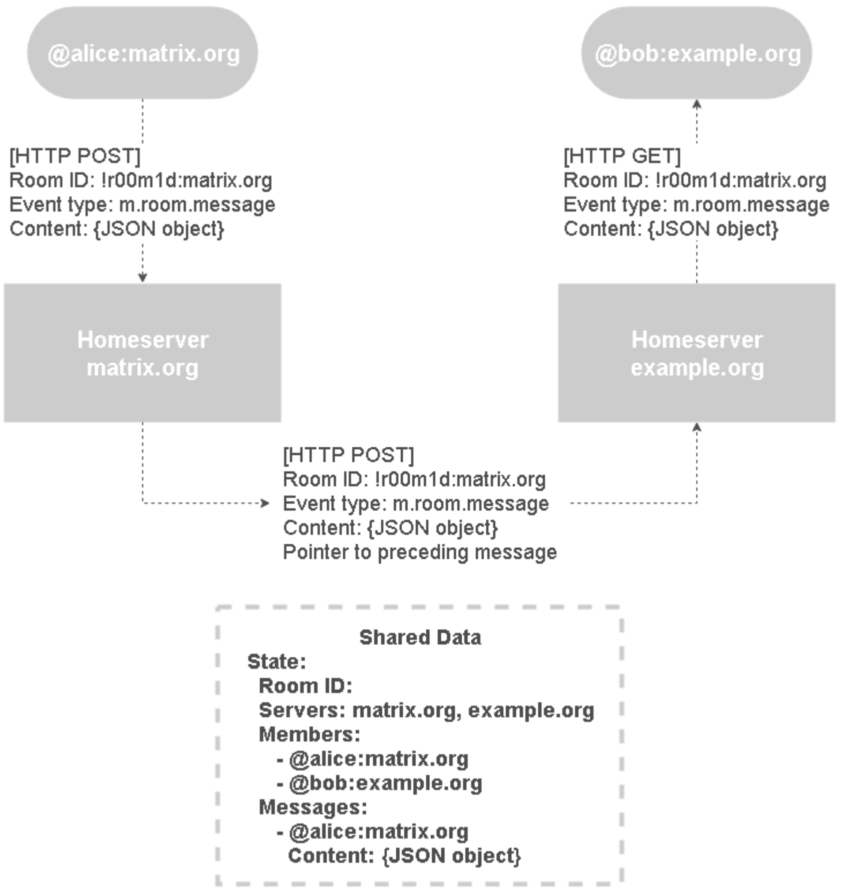


Figure 2.20: Matrix conceptual architecture [10].

Federation maintains shared data structures per-room between multiple homeservers. The data is split into message events and state events.

Message events These describe transient 'once-off' activity in a room such as an instant messages, VoIP call setups, file transfers, etc. They generally describe communication activity.

State events These describe updates to a given piece of persistent information ("state") related to a room, such as the room's name, topic, membership, participating servers, etc. State is modelled as a lookup table of key/value pairs per room, with each key being a tuple of state_key and event type. Each state event updates the value of a given key. The state of the room at a given point is calculated by considering all events preceding and including a given event in the graph. Where events describe the same state, a merge conflict algorithm is applied. The state resolution algorithm is transitive and does not depend on server state, as it must consistently select the same event irrespective of the server or the order the events were received in. Events are signed by the originating server (the signature includes the parent relations, type, depth and payload hash) and are pushed over federation to the participating servers in a room, currently using full mesh topology. Servers may also request backfill of events over federation from the other servers participating in a room.

2.2.3 Matrix specification

Matrix defines a set of open APIs for decentralised communication, suitable for securely publishing, persisting and subscribing to data over a global open federation of servers with no single point of control. Uses include Instant Messaging (IM), Voice over IP (VoIP) signalling, Internet of Things (IoT) communication, and bridging together existing communication silos - providing the basis of a new open real-time communication ecosystem.

Matrix two main API specification; Client/Server API and Federated API. Any Matrix SDK implements the API defined at the Client/Server API specification. If a custom homeserver was to be developed from scratch it would have to conform to the Federated API to be able to be a part of Matrix.

Client/Server API The client-server API provides a simple lightweight API to let clients send messages, control rooms and synchronise conversation history. It is designed to support both lightweight clients which store no state and lazy-load data from the server as required - as well as heavyweight clients which maintain a full local persistent copy of server state.

2.2.4 End-to-end Encryption

Privacy is particularly critical for a decentralized network like Matrix, where user data is replicated over all the servers participating in a room and so must be protected. Matrix provides state of the art end-to-end encryption in beta using the Olm and Megolm cryptographic ratchets, and ensuring that only explicitly authorized devices can participate in a conversation.

Olm and Megolm are examined in section 3.1.3.

The end-to-end encryption is based on the Double Ratcet algorithm.

- Provides 1:1 encrypted channels between pairs of devices via the Olm double ratchet
- Supports group chats with hundreds of devices via the new Megolm shared ratchet, with either perfect forward secrecy or the ability to decrypt conversation history on new devices.
- Supports multiple devices, tracking verification trust per-device to help spot interception.

Currently any apps built on matrix-js-sdk, matrix-ios-sdk or matrix-android-sdk support encryption, which is in beta as of March 2017. The libolm library itself is considered stable however.

2.3 Information Flow Control

2.3.1 The Lattice Model

adsadsad

2.3.2 Noninterference

dasdsa

2.3.3 Static policies

sadsad

2.3.4 Dynamic policies

sadsadsad

2.3.5 Declassification

Taking some specific information and changing it to a lower security classification.

Identify: What to classify, who declassifies, where the declassification happens and when the declassification happens

2.4 Summary

3 Analysis

This chapter consists of two parts. The first part will provide an evaluation of the Matrix security model and relies on the paper *SoK: Secure Messaging* [25] and *The Olm Cryptographic Review* by NCC Group [18].

The second part provides a preliminary analysis of the IFC tools, the selection of Paragon and the rationale behind it, and a further analysis of the selected tool Paragon.

3.1 Evaluation of Matrix security model

The security of matrix will be evaluated in the context of secure messaging. An evaluation framework has been proposed in the paper *SoK: Secure messaging* which the evaluation will be loosely based on.

The evaluation framework covers several areas with *conversation security* being the most relevant for this evaluation. The area *conversation security* describes three categories; *Security and Privacy*, *Adoption*, and *Group Chat*. Obviously the most relevant category for the evaluation is *Security and Privacy*

3.1.1 Threat model

For secure messaging the evaluation framework defines a threat model with three types of adversaries. Note that an adversary can be of several types:

- *Local adversary:* The adversary is in control of the local network.
- *Global adversary:* The adversary is in control of great portions of the Internet
- *Service providers:* A potential adversary for messaging systems with centralized infrastructure.

In the messaging system the adversary may be a participant with the following properties:

- An adversary can start a conversation.
- An adversary can send messages.
- An adversary can perform any other action that a participant is capable of.

Furthermore it is assumed that the system's endpoints are secure [25].
This evaluation will inherit the described threat model.

3.1.2 The Signal Protocol

Matrix provides end-to-end encryption by using the Olm and Megolm library with the former being an implementation of the Double Ratchet algorithm also known as the Signal Protocol, and the latter being the algorithm used for group chat.

Olm is used for securely exchanging message keys/session keys during group chat and is vital part of the end-to-end encryption in Matrix.

Before the Matrix protocol is evaluated the Signal Protocol will be considered. The Signal Protocol is described in section xx.

Section xx provides a list of security properties relevant for *conversation security*. These security properties is used for evaluating a secure messaging protocol such as the Signal Protocol.

The table below shows an evaluation of the Signal Protocol (previously known as TextSecure) [25].

Scheme	Example	Security and Privacy														Adoption					Group Chat				
		Confidentiality	Integrity	Authentication	Participant Consistency	Destination Validation	Forward Secrecy	Backward Secrecy	Anonymity Preserving	Speaker Consistency	Causality Preserving	Global Transcript	Message Unlinkability	Message Repudiation	Particip. Repudiation	Out-of-Order Resilient	Dropped Message Resilient	Asynchronicity	Multi-Device Support	No Additional Service	Computational Equality	Trust Equality	Subgroup Messaging	Contractable	Expandable
+Double Ratchet+3DH AKE+Prekeys[†*]	TextSecure	●	●	●	●	●	●	●	-	◐	●	-	●	●	●	◐	◐	●	-	-					

Figure 3.1: Evaluation of Signal (TextSecure) [25].

Confidentiality When a message is sent using the Signal Protocol then only the intended recipient can read the message. The senders sending ratchet and receivers receiving ratchet will derive the same message key hence only the two parties will be able to encrypt the messages.

Integrity The receiver will only accept a message if it is successfully decrypted hence if in transit a message was modified then the message would be rejected.

Authentication The decryption of a message also gives authentication guarantees since only the intended recipient could compute the message key.

Forward secrecy The symmetric ratchet ensures forward secrecy. If a chain session key is compromised then the previous keys can not be generated since the ratchet is one way cryptographic hash function hence secrecy is provided for all previous send messages.

Backward secrecy Diffie-Hellman ratchet have the self-healing property and will generate a new chain session key for the symmetric ratchet hence if a chain key is compromised then secrecy for future messages is still provided because a new chain ratchet key will be generated.

Anonymity preserving Anonymity preservation is lost in the Signal Protocol since the initial key agreement requires long-term public keys hence making them observable during Triple-DH. However ***participant consistency*** is provided by Triple-DH [25].

Speaker consistency This property is partially provided through the key evolution of the ratchets. If a message is dropped then it is not possible to generate message keys for future messages. This also makes the protocol have the property ***Causality Preserving*** and partially have the property ***Dropped message resilence***. It will also not go unnoticed if a message is received out of order since this will result in the message's key being an unexpected key. Hence the recipient have to store expired keys to decrypt delayed messages. This makes the property ***Out-of-order resilient*** only partially provided [25].

Global transcript In an asynchrounous messaging protocol there is no global transcript. Both participants have to be online to receive messages hence the participants will not have all the messages if one of them is offline. This is a result of having the ***Asynchronicity*** property.

Deniability properties Since the ratchet session keys are used for encrypting messages and not the long-term public keys the properties ***Message unlinkability*** and ***Message repudiation*** are provided.

Other properties

- ***Participant repudiation***. Triple-DH achieves full participant repudiation since anyone can forge a transcript between any two participants [25].
- ***Destination validation***. The Deffie-Hellman ratchet provides this property since the recipients public key is used to generate the chain key [25].

The evaluation shows that several security properties are provided with the important ones being confidentiality, integrity, authentication, forward secrecy, backward secrecy.

Furthermore a formal analysis have been made on the Signal Protocol that proves the protocol is free from any major flaws and it satisfy the following security properties; confidentiality, authentication and secrecy [13].

Application variants

The Signal Protocol is a secure messaging protocol and have been extensively studied including proof that the standard security properties are assured.

The Olm library used by Matrix is a variant of the Signal Protocol. There is no implementation analysis of the Olm library hence there is no guarantee that all the security properties defined in xx is inherited by Olm. Nevertheless it is assumed that Olm inherits the above properties.

The further evaluation relies upon the the security assessment of Matrix.

3.1.3 Matrix protocol

As described in section xx *rooms* are a fundamental part of Matrix' architecture. There can be multiple participants in a room hence the support for secure group conversation is required.

Olm (and the Signal Protocol it is based on) is ideally meant for two party communication. Group conversation could be supported with a näive variant of Olm. In a group with N participants each participant would establish a secure Olm session with every other participant. When a message is send each message would then have to be encrypted N times. This solution would scale poorly if N was a large number. This was the motivation for introducing Megolm.

Megolm

Megolm is a multicast encryption solution [25].Each sender has a sender ratchet (Megolm Ratchet). Each recipient has a corresponding receiving ratchet for each sender. So if there are N participants in a group then each participant will have N-1 receiving ratchets. Figure 3.2 illustrates the setup with three participants.

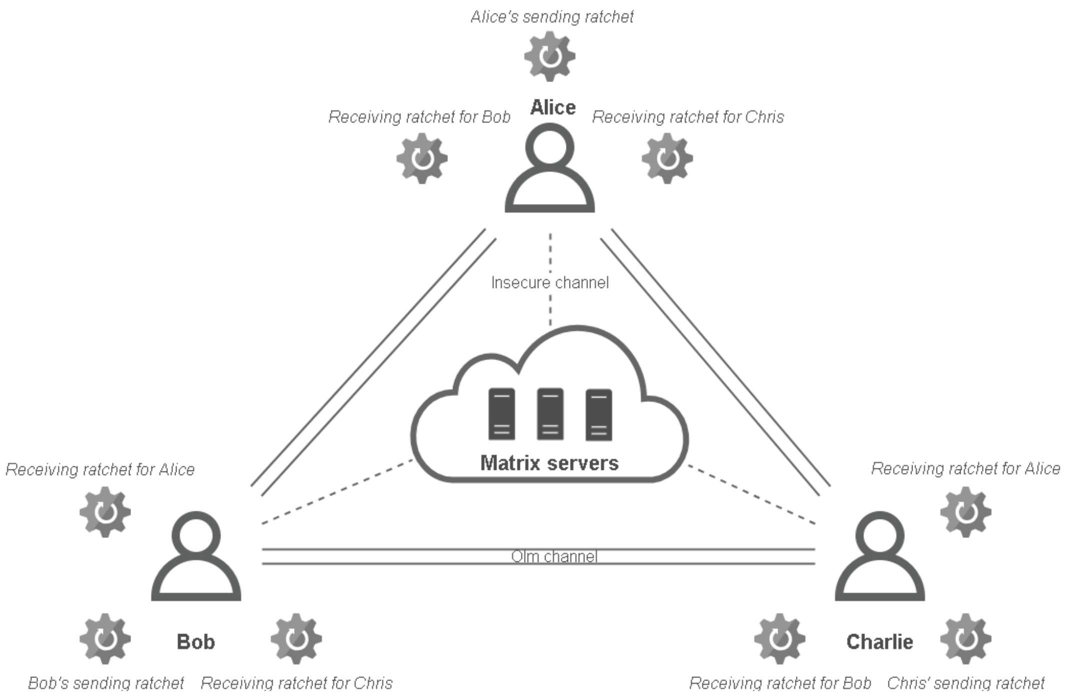

Alice's sending ratchet
Receiving ratchet for Bob
Alice
Receiving ratchet for Chris
Insecure channel
Matrix servers
Receiving ratchet for Alice
Receiving ratchet for Alice
Olm channel
Bob
Charlie
Bob's sending ratchet
Receiving ratchet for Chris
Receiving ratchet for Bob
Chris' sending ratchet

Figure 3.2: Conceptual model of Megolm with three participants.

When a session is started a sender will send his initial ratchet key to each recipient, so that the sender ratchet and each recipients ratchet are in sync. This key exchange happens over a secure communication channel (Olm). Furthermore there is send N-1 initial messages when a session is initiated. Until a new session is started no further session keys are exchanged and the corresponding message keys are generated by incrementing the ratchet.

When a sender sends a message a message key is generated from the ratchet key and the message is encrypted using that message key. The message will be signed so the recipient will know which sender the message is from and which ratchet to utilize. The message is then send to the server which relays the message to all recipients over an insecure channel. When they receive the message the same message key is generated using the corresponding receiver ratchet and the message is decrypted.

When a new participant joins the latest ratchet key would then be shared by each participant over Olm (or an earlier one if he should have access to historical conversation).

When a participant leaves a new session would be initiated yielding in refreshing the ratchet keys hence not making it possible for that ex-participant to decrypt any further messages.

The Matrix Protocol will be evaluated in the context of Megolm. The evaluation of the Matrix protocol heavily relies on the security assessment by NCC.

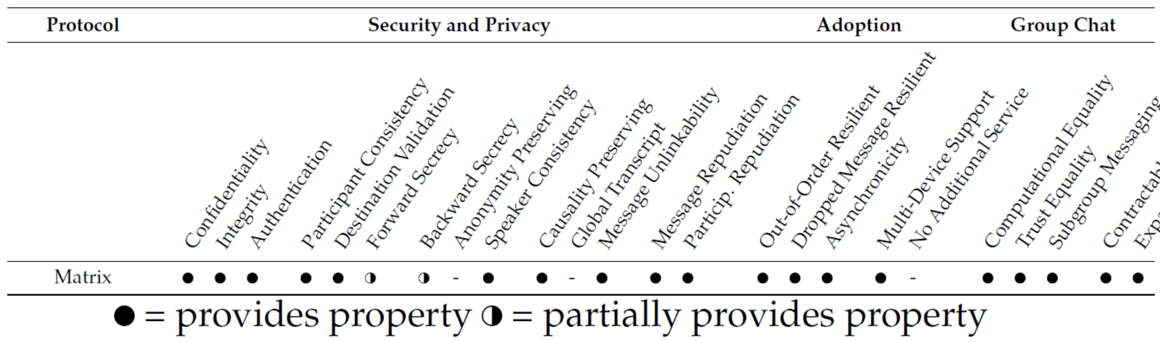
3.1.3.1 Evaluation

The Matrix Protocol provides several security properties shown in the table xx.

It is worth mentioning that there is a trade-off between security and usability which must be decided at application layer. The most secure configuration would come at the cost of usability and performance.

- *Usability.* From a users point of view it would be nice to have the possibility to load historical conversation instead of having to keep full history locally. Matrix supports multiple devices and if a participant adds another device at some later point it makes sense to load the participants historical conversation into the device. From a security perspective this would mean that the *initial ratchet state* is stored and is send to the new device so every message key can be generated. This certainly goes against the principle of forward secrecy. The most secure configuration would not store the *initial ratchet state* hence satisfy forward secrecy thus disable the described usability feature [18] [**?**].

- *Performance.* When a megolm session is initialized there is an initial burst of messages to exchange the initial ratchet key which is then stored in a *initial ratchet state* value at each recipient. If this key is compromised then any future key can be generated for that session. To satisfy backward secrecy this would mean initiating a new session for each message which would trigger a burst of messages to exchange the ratchet key [18] [**?**]. This would scale poorly for a large group or when sending large-sized messages.



Protocol	Security and Privacy														Adoption					Group Chat				
	Confidentiality	Integrity	Authentication	Participant Consistency	Destination Validation	Forward Secrecy	Backward Secrecy	Anonymity Preserving	Speaker Consistency	Causality Preserving	Global Transcript	Message Unlinkability	Message Repudiation	Particip. Repudiation	Out-of-Order Resilient	Dropped Message Resilient	Asynchronicity	Multi-Device Support	No Additional Service	Computational Equality	Trust Equality	Subgroup Messaging	Contractable	Expandable
Matrix	●	●	●	●	●	◑	◑	-	●	●	-	●	●	●	●	●	●	●	-	●	●	●	●	●

● = provides property ◑ = partially provides property

Figure 3.3: Evaluation of Matrix Security.

Some of the security properties in the table are briefly examined.

Confidentiality When a message is send it is encrypted and can only be decrypted by the intended recipients who has the corresponding ratchet session key received over an Olm channel.

Integrity The receiver will only accept a message if it is successfully decrypted hence if in transit a message was modified then the message would be rejected.

Forward secrecy Each participant keeps a *initial ratchet state* which holds the earliest ratchet session key for a session. This clearly violates forward secrecy since every message can be decrypted if the *initial ratchet state* value is compromised. However it is a deliberate trade-off for usability to enable historical conversation and storing the value is optional. Since this is an optional feature the forward secrecy is partially provided [18].

Backward secrecy If a ratchet key is compromised then an adversary can generate every message key from that point on hence intercept any message that sender sends to the group. This can be prevented strictly by starting a new session with every send message however it would not be possible to keep conversation history (only locally when data is encrypted). Hence the property is only partially provided [18].

Speaker consistency There is no guarantee for speaker consistency. A well known problem of multi-cast encryption group chat is transcript inconsistency. A sender may send different messages to different recipients. However it requires that the server is in collusion with the sender. This also applies to ***Causality preserving*** [18].

Other properties

The multi-cast encryption design does not provide *participant consistency* [25].

The properties *Dropped message resilence* and *Out-of-order resilient* are provided by keeping track of ratchet indices.

Several properties are inherited from the secure key exchanging channel provided by Olm while other properties are inherited because of asynchronicity of the Megolm protocol.

- *Authentication* is provided by Olm since the ratchet session key is send to the recipient through an Olm channel or else the message key could not be derived.
- *Destination validation.* The ratchet session key is exchanged over a secure Olm channel hence only the intended recipient could decrypt it.
- *Anonymity preserving* is not provided since Olm requires the long-term public key in the initial key exchange.
- *Global transcript* is also not provided because of the asynchronous nature of the Megolm protocol.
- *Asynchronicity* is obviously provided.
- *Deniability* properties are inherited from Olm as well.

All properties related to group chat are also provided. Although they are additional features and not related to security.

Other findings

Message Replays Matrix allows decryption of a message multiple times hence it is vulnerable to replay attacks. Replay attacks are handled at the application layer. Whenever a message is decrypted a message index is generated and stored. If the exact message is decrypted again the same message index will be generated and can be compared to the stored message index making the replayed message invalid.

Unknown key-share attack The *Unknown key-share attack*[1] is a vulnerability found with a high risk in Megolm. The vulnerability is inherited from Olm and occurs after the initial message in Triple-DH.

The vulnerability has been mitigated at the application layer by providing a unique identifier for the sender and receiver into each message and then checking the values when decrypted [18].

Recent research

The way backward secrecy would be provided in Matrix is computationally expensive. Recent research has proposed solutions with early implementations for these problems with IETF leading the research on the standard on *Messaging Layer Security.* Matrix has expressed awareness of the protocol and a possibility of adaption in the future.

[1] https://en.wikipedia.org/wiki/Unknown_key-share_attack

3.1.4 End-to-end security

Section xx describes Matrix long-term goal as being a generic HTTP messaging API. It could be utilized for any kind of data exchange in a system or between multiple systems.

A system using the Matrix Protocol for exchanging data would benefit from the security properties found in the evaluation yet the system-wide security or end-to-end security would be incomplete an further measures must be taken. Such system might demand confidentiality and integrity throughout the system yet the system as a whole would have a different threat model than the one described for a secure messaging system hence no guarantee of confidentiality or integrity beyond the endpoints in end-to-end encryption.

The following figure depicts how end-to-end encryption might be inadequate in such system.

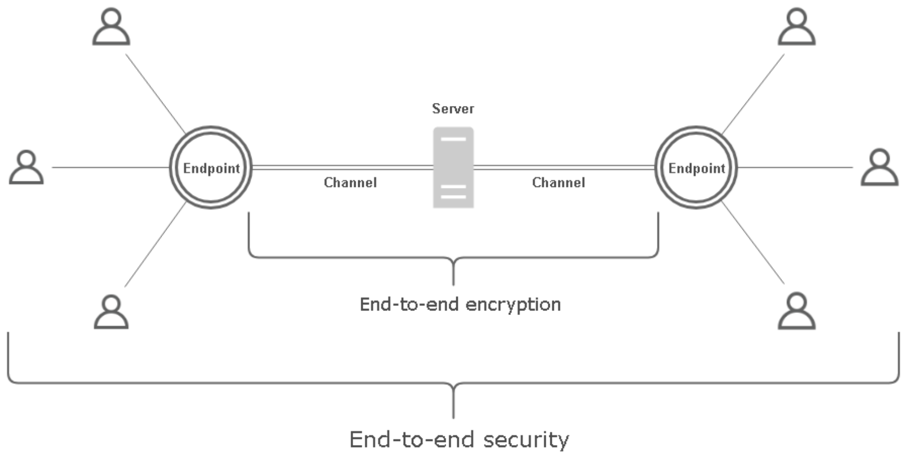


Figure 3.4: End-to-end security.

As the system depicts there might be several principals accessing the endpoint. Each principal could retrieve some information possibly protected with access control. Assume that the information resting at the endpoint is of confidential nature; access could still be granted with no respect of the confidentially of that information. There clearly lack a mechanism of specifying what information is confidential or public and where it may flow under what conditions.

Matrix identifies IoT as another use case. A person can have several devices for health tracking, entertainment and so on. The data from the devices are send to vendors - a device might send data to several vendors. Ultimately this give a fragmentation of the person's own data with it being placed at several vendors data back-end. Matrix proposes a solution where all the device data for a person is synchronized and persisted on Matrix. Vendors would be connected to Matrix. This is depicted in figure 3.5 below.

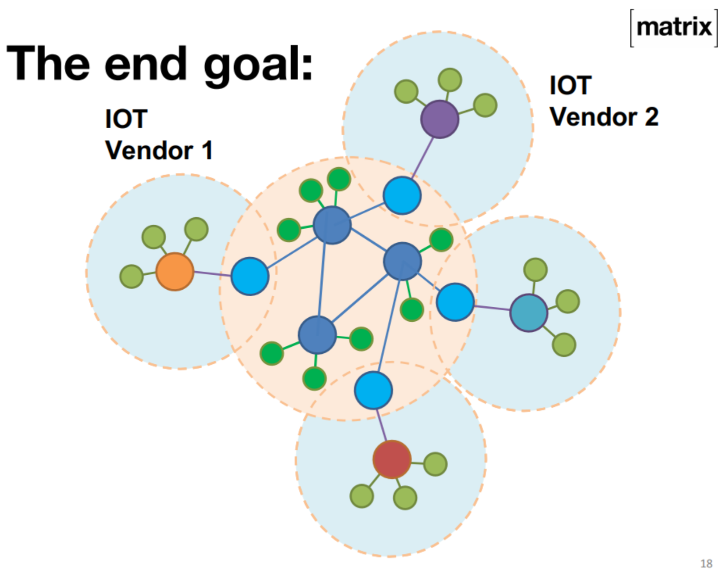


Figure 3.5: End goal for Matrix IoT

The data flowing from the sensors to Matrix might be of sensitive nature or the owner might only allow some data to flow to some vendors under specific conditions. This issue is not addressed by Matrix.

3.1.5 Summary

In this section an evaluation of Matrix security was presented. Several security properties are a part of Matrix security model with forward secrecy and backward secrecy being provided depending on the Matrix configuration.

End-to-end encryption is not the end of security. Other security measures must be taken to provide confidentiality and integrity. Information Flow Control is such measure and the next section will present a survey on Information Flow Control.

3.2 Survey of IFC Tools

Bibliography

[1] Op imod 90.000 ansatte kan kigge i din journal - Indland. URL https://jyllands-posten.dk/indland/ECE6715461/op-imod-90000-ansatte-kan-kigge-i-din-journal/.

[2] Adgang til sundhedsdata - sundhed.dk. URL https://www.sundhed.dk/borger/service/om-sundheddk/om-portalen/datasikkerhed/andres-dataadgang/adgang-til-sundhedsdata/.

[3] CWE - CWE-359: Exposure of Private Information ('Privacy Violation') (3.2). URL https://cwe.mitre.org/data/definitions/359.html.

[4] Journal fra sygehus. URL https://www.sundhed.dk/borger/min-side/min-sundhedsjournal/journal-fra-sygehus/.

[5] Principles for a More Informed Exceptional Access Debate - Lawfare. URL https://www.lawfareblog.com/principles-more-informed-exceptional-access-debate.

[6] Google Plus Will Be Shut Down After User Information Was Exposed - The New York Times. URL https://www.nytimes.com/2018/10/08/technology/google-plus-security-disclosure.html.

[7] Kontrol af opslag - sundhed.dk. URL https://www.sundhed.dk/borger/service/om-sundheddk/om-portalen/datasikkerhed/portalens-beskyttelse-af-data/kontrol-af-opslag-e-journal/.

[8] FAQ | Matrix.org, . URL https://matrix.org/docs/guides/faq.

[9] Home | Matrix.org, . URL https://matrix.org/.

[10] Matrix Specification, . URL https://matrix.org/docs/spec/{#}architecture.

[11] 91,000 state Medicaid clients warned of data breach | The Seattle Times. URL https://www.seattletimes.com/seattle-news/health/91000-state-medicaid-clients-warned-of-data-breach/.

[12] Matt Bishop. *Introduction to computer security*. Addison-Wesley, Boston, 2005. ISBN 978-0321247445.

[13] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A Formal Security Analysis of the Signal Messaging Protocol. *Proceedings - 2nd IEEE European Symposium on Security and Privacy, EuroS and P 2017*, (November):451–466, 2017. ISSN 13484214. doi: 10.1109/EuroSP.2017.27.

[14] Tilman Frosch, Christian Mainka, Christoph Bader, Florian Bergsma, Jörg Schwenk, and Thorsten Holz. How Secure is TextSecure? Technical report. URL `https://cryptome.org/2014/11/textsecure-secure.pdf`.

[15] Laurinda B. Harman, Cathy A. Flite, and Kesa Bond. Electronic Health Records: Privacy, Confidentiality, and Security. *Virtual Mentor*, 14(9):712–719, sep 2012. ISSN 1937-7010. doi: 10.1001/virtualmentor.2012.14.9.stas1-1209. URL `http://virtualmentor.ama-assn.org/2012/09/stas1-1209.html`.

[16] Moxie Marlinspike and Trevor Perrin. The X3DH Key Agreement Protocol. Technical report, 2016. URL `https://signal.org/docs/specifications/x3dh/x3dh.pdf`.

[17] Katina Michael. *The Basics of Information Security: Understanding the Fundamentals of InfoSec in Theory and Practice*, volume 31. 2012. ISBN 9781597496537. doi: 10.1016/j.cose.2012.03.005. URL `http://linkinghub.elsevier.com/retrieve/pii/S0167404812000557`.

[18] NCC Group. Olm Cryptographic Review. (November):1–27, 2016. ISSN 0737-4038.

[19] P. D. Pacey and J. H. Purnell. OWASP Top 10 - 2017. *International Journal of Chemical Kinetics*, 4(6):657–666, 1972. ISSN 10974601. doi: 10.1002/kin.550040606.

[20] Trevor Perrin and Moxie Marlinspike. The Double Ratchet Algorithm. Technical report, 2016. URL `https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf`.

[21] Ralph Spencer Poore. Crypto' 101. *Information Systems Security*, (2), 2017. ISSN 1065-898X. doi: 10.1201/1086/43305.8.2.19990601/31062.6. URL `http://www.tandfonline.com/doi/full/10.1201/1086/43305.8.2.19990601/31062.6`.

[22] Fiza Abdul Rahim, Zuraini Ismail, and Ganthan Narayana Samy. Information privacy concerns in electronic healthcare records: A systematic literature review. In *2013 International Conference on Research and Innovation in Information Systems (ICRIIS)*, pages 504–509. IEEE, nov 2013. ISBN 978-1-4799-2487-5. doi: 10.1109/ICRIIS.2013.6716760. URL `http://ieeexplore.ieee.org/document/6716760/`.

[23] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003. ISSN 07338716. doi: 10.1109/JSAC.2002.806121.

[24] The Government, Local Government Denmark, and Danish Regions. *The Digital Strategy - A stronger and more secure digital Denmark.* Agency for Digitisation, 2016. ISBN 9789400769250 | 9400769245 | 9789400769243. doi: 10.1007/978-94-007-6925-0_9. URL `https://en.digst.dk/media/14143/ds{_}singlepage{_}uk{_}web.pdf`.

[25] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. SoK: Secure Messaging. pages 232–249, 2015. doi: 10.1109/SP.2015.22.