Decentralized But Not Immune: Empirical and Formal Security Analysis of Messaging Networks

Hao Li[1], Rui Wang[1*], Hongfeng Sun[1], Zhenxiang Chen[2], Shanqing Guo[3]
[1]Shandong Women's University, China. [2]University of Jinan, China. [3]Shandong University, China.
202401111@sdwu.edu.cn

***Abstract*—As decentralized messaging networks (DMNs) evolve, there is an increasing need for automated methods to systematically profile their ecosystems and formally verify the security of their underlying protocols. In this study, we propose a modular and automated framework that combines large-scale data collection with formal security verification, enabling the abstraction, modeling, and analysis of security protocols across diverse DMN platforms. Leveraging this framework, we perform empirical measurements and protocol-level verification across several representative DMNs. Our analysis uncovers a range of security issues spanning both network infrastructure and protocol layers, including DMNs node IP address exposure, software supply chain weaknesses, and cryptographic flaws in handshake and encryption protocol that permit man-in-the-middle and impersonation attacks.**

***Index Terms*—Matrix protocol, security, formal verification**

I. Introduction

Traditional centralized communication platforms (e.g., Facebook, Twitter, WhatsApp, etc.) typically control information collaboration and data sovereignty centrally on a centralized server or platform [1]. This centralized architecture has a number of potential problems and challenges such as, data sovereignty, single point of failure, monitoring and privacy issues, and limited openness. With the development of Web3, decentralized communication protocols are becoming more and more popular [2], and most decentralized messaging protocols tend to rely on distributed systems, which can help combat government censorship of data and the monopoly of tech giants. And the architecture between centralized and decentralized messaging network as shown in Fig. 1. Several decentralized communication networks have gained notable adoption due to their privacy-preserving designs. Matrix [3], a leading protocol in this space, reports over 80 million registered accounts [4], spanning individuals, enterprises, governments, and academic institutions. Jami [5], a peer-to-peer communication platform with end-to-end encryption and perfect forward secrecy, is also widely used. Status [6] combines a decentralized messaging app with a Web3.0 wallet, offering metadata protection and private chats secured by end-to-end encryption. Berty [7] is a privacy-focused messaging application built on the Wesh protocol, featuring offline communication capabilities via Bluetooth Low Energy (BLE), making it suitable for connectivity-constrained environments.

Protocols such as Matrix, Status, Berty, and Jami exemplify the growing adoption of end-to-end encrypted communication in decentralized ecosystems, offering strong privacy guarantees and increased resilience against surveillance and centralized compromise. For example, Matrix implements a custom cryptographic ratchet protocol [8], [9] to provide session-level message security. Berty introduces the Wesh protocol [10], enabling secure and asynchronous communication even in offline scenarios, while Jami adopts the well-established X3DH key agreement [11] and Double Ratchet [12] algorithms to protect user communications.

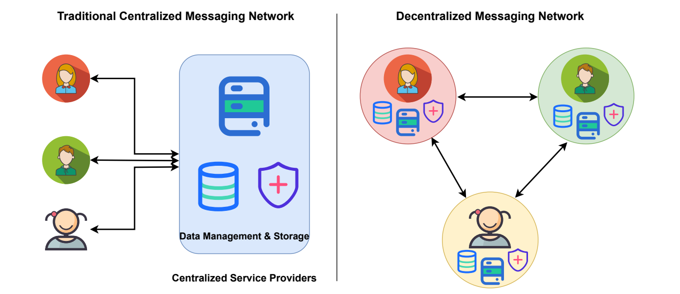

Traditional Centralized Messaging Network
Data Management & Storage
Centralized Service Providers
Decentralized Messaging Network

Fig. 1. The architecture between centralized and decentralized messaging network.

Motivation. Decentralized protocols introduce unique attack surfaces and architectural complexities. These challenges include key distribution without a central authority, and reliable multi-device synchronization in the presence of adversaries. Existing research often overlooks these critical aspects, placing greater emphasis on system functionality or scalability rather than comprehensive security evaluation [13], [14]. This work is motivated by the need to bridge that gap and systematically investigate the security of decentralized messaging networks (DMNs), specifically:

- To the best of our knowledge, there is a lack of comprehensive large-scale measurement studies of decentralized messaging networks, along with a shortage of automated formal verification and rigorous security analysis of their cryptographic protocols.
- In terms of ecosystems measurement, DMNs operate without centralized servers, with users (e.g., Alice and Bob) often on different home servers. Existing studies rarely address key aspects such as network composition, geographic and infrastructure distribution, metadata leakage, adversarial behavior, and long-term availability [13], [15].
- At the security protocols verification, handshake and ratchet

[*]Corresponding author.

mechanisms require modular, formalizable designs to support automated analysis and attack trace generation. Current work lacks formal models covering authentication, forward and backward secrecy, resistance to key compromise impersonation (KCI) security [14], [16].

What we have done and findings. To address the aforementioned challenges, this paper proposes a comprehensive solution, named as the D-MAP. This approach features automated data collection capabilities for decentralized messaging networks, employing distributed crawlers and passive monitoring techniques to efficiently gather critical ecological information such as network topology, geographic node distribution, traffic characteristics, and protocol interaction data. On the security protocol analysis side, D-MAP integrates automated formal verification tools based on ProVerif [17], constructing modular protocol models that encompass key subprotocols such as handshake processes, key agreement, and encrypted message exchange of Matrix, Berty, and Status. By automatically generating potential attack traces, the proposed method precisely identifies authentication flaws, key leakage risks, and multi-device synchronization vulnerabilities.

On one hand, we found that the decentralized messaging ecosystem is large and globally distributed, however, since the limited adoption of fully decentralized infrastructure, signs of infrastructure centralization still persist. Additionally, over 4.4% of node IPs exhibit malicious activity, and many nodes are affected by severe vulnerabilities, highlighting the significant impact of software supply chain risks on the overall network security (details in Sec. IV). On the other hand, after conducting the formal verification method that we designed and implemented, we uncovered security property violations within different DMN security protocols under diverse threat models, revealing distinct attack traces. Specifically, our analysis revealed that the handshake protocol in decentralized messaging networks is susceptible to man-in-the-middle attacks, particularly during the key handshake phase, which undermines the protocol's authenticity guarantees. Furthermore, the encryption protocol is vulnerable to impersonation attacks targeting compromised nodes or master servers. In such scenarios, an attacker can replace a victim's identity key to clandestinely forge communications with other parties, thereby compromising not only authenticity but also forward secrecy and post-compromise security. For a detailed exploration of these findings, refer to Sec. V.

Contributions. The main contributions are summarized as follows:

- This work presents the first large-scale measurement and formal security analysis of decentralized messaging networks. We propose D-MAP, an automated framework that combines distributed data collection with formal protocol verification, enabling active scanning, passive monitoring, and extraction of protocol-relevant metadata.
- Our measurement shows that decentralized messaging networks (DMNs) are globally distributed but still exhibit signs of infrastructural centralization, such as reliance on a few dominant servers or IP blocks. Additionally, over 4.4% of nodes show signs of malicious activity, including blacklisting and active exploitation.
- Using the proposed D-MAP method, our formal analysis uncovers critical protocol vulnerabilities, including man-in-the-middle and impersonation attacks that undermine authenticity, forward secrecy, and post-compromise security. To address these issues, we propose mitigations such as secure key verification, decentralized trust anchoring, and protocol refinements.
- We responsibly disclosed the discovered vulnerabilities to the DMN teams, who confirmed flaws in the encryption system leading to message leakage, caused by weaknesses in initial trust establishment. We have also open-sourced D-MAP, including raw datasets and formal analysis code, in our GitHub repository[1].

TABLE I
AN OVERVIEW ACROSS DECENTRALIZED MESSAGING NETWORKS.

System	Decentralized Mechanisms	Security Protocols	Measurement[1]	Protocol Analysis[2]
Matrix [3]	Homeserver	Olm, Megolm	✓	✓
Berty [7]	libp2p	Wesh Protocol	✓	✓
Status [6]	Waku	X3DH, Double Ratchet	✓	✓
Jami [5]	OpenDHT	TLS v1.3	✓	✗

[1] Measurement: Empirical measurements are conducted on all networks.
[2] Protocol Analysis: All but Jami undergo protocol-level analysis.

II. Background

Decentralized messaging networks (DMNs), including Matrix [3] which uses a federated architecture with strong encryption, Berty [7] which supports offline peer-to-peer communication, Status [6] which integrates messaging with a crypto wallet using the Waku protocol [18], and Jami [5] which enables serverless real-time communication, distribute data and control across multiple nodes. We list an overview of the decentralized messaging networks in Table I.

A. Overview of the Decentralized Messaging Networks

In this section, we explore the underlying principles, architectural designs, and broader ecosystems of these decentralized messaging platforms.

1) Matrix: Matrix is a federated system that mainly includes the homeservers, rooms, and users, and all communication between Matrix users takes place within the room, suppose a user (`Alice`) wants to communicate with others in the room, she first needs to send messages to her homeserver (e.g., *matrix.alice.com*), and the homeserver will then synchronize messages to other homeservers in the room. Eventually, the destination users' clients (`Bob` and `Charlie`) will receive the synchronized messages from their homeserver.

Besides, for privacy-minded customers, Matrix rooms also support communication with E2EE based on the Olm cryptographic ratchets protocol [9], [12]. Every registered user

[1]https://github.com/lihaoSDU/DecentralizedMesagers

in Matrix has a unique "user ID" (`MXID`) in the format `@user_id:domain`, and a user device (e.g., Android, iOS) that supports E2EE has its `DeviceKeys`, the public part of the DeviceKeys will be uploaded to their homeserver, and the private part will be retained in the local device. The DeviceKeys encompass an Ed25519 [19] fingerprint key pair for signing messages, a Curve25519 [20] identity key pair, and Curve25519 one-time keys respectively. In particular, long-term Curve25519 identity keys and Curve25519 one-time keys enable authentication and establishment of shared key within Olm [8] sessions.

2) Berty Messager: Berty is fundamentally grounded in a peer-to-peer (P2P) communication model built atop the libp2p [21] networking stack. This design entirely eliminates the reliance on centralized servers, and as the core transport and networking layer, libp2p handles essential networking tasks including peer discovery, secure connection negotiation, NAT traversal, and multiplexing, which allows multiple logical streams to share a single physical connection. To facilitate decentralized peer discovery and initial contact, Berty employs a "Rendezvous Point" mechanism supported by decentralized servers. This signaling system enables "Peers" to locate one another and initiate secure communications without centralized coordination, thereby enhancing both privacy and robustness. The Wesh protocol [10], which underpins Berty's peer discovery system, defines two types of rendezvous points:

- A public rendezvous point, used to receive contact requests, leverages the Account ID as the resource identifier. To maintain access control, users can rotate the associated seed, thereby invalidating previous discovery attempts by unauthorized parties.
- A group rendezvous point, used for group messaging, employs the Group ID as the resource identifier and uses a fixed, non-rotatable seed, ensuring consistent group-level communication.

Berty also supports Bluetooth Low Energy (BLE) as a communication mode for nearby, offline interactions. However, as BLE-related mechanisms and implementations are outside the scope of this paper, they are not discussed in detail.

3) Status: Status Messenger is an open-source, decentralized application that combines a secure messaging platform with a self-custodial cryptocurrency wallet. It enables users to communicate privately and manage digital assets without relying on centralized intermediaries. Within the Status network, every "Status Node" operates as a peer, though with varying capabilities. While some elements of the current implementation still reflect a client-server-like structure (e.g., Status presently runs the primary message relaying and storage nodes), the underlying architecture is a peer-to-peer (P2P) overlay network built atop the TCP/IP stack. This foundational layer constitutes a public, permissionless network, powered by the devp2p protocol[2], which supports decentralized peer discovery, connectivity, and message routing.

[2]https://github.com/ethereum/devp2p

Status defines various "Node Types" based on their supported capabilities. A node may serve one or more roles, such as sending and receiving messages, relaying traffic, storing historical messages, or bootstrapping new nodes into the network [6]. To establish connections, a Status node must either discover peers dynamically or reference a preconfigured list of known nodes. This is achieved through a combination of Discovery v5 [22], the Rendezvous protocol, and optionally, static nodes. Discovery v5 uses bootstrap nodes to initiate the peer discovery process, allowing nodes to dynamically join and participate in the decentralized messaging network.

4) Jami: Jami is a fully decentralized, end-to-end encrypted communication platform designed to ensure user privacy without relying on centralized servers for data relay. Instead, it uses a peer-to-peer architecture where communication occurs directly between users. At the core of its infrastructure is OpenDHT[3], a distributed in-memory key-value store based on the Kademlia DHT protocol. OpenDHT enables decentralized connectivity establishment and efficient message distribution, forming the backbone of Jami's serverless operation.

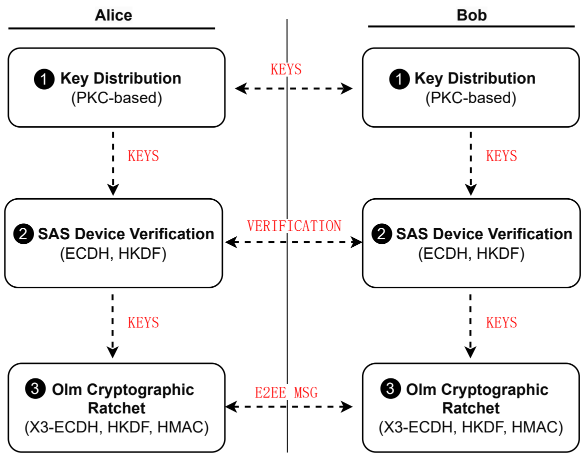
Anyone can join the OpenDHT network by connecting to an existing node, which then shares knowledge about other nodes in the system, which enabling seamless integration and peer discovery. To support robust connectivity, Jami clients cache known "Nodes" from previous sessions, allowing them to quickly reconnect to the network. When cached nodes are unavailable, Jami uses a configurable, stable "bootstrap node" to establish the initial connection. Users seeking complete autonomy can deploy their own OpenDHT node and designate it as a bootstrap node, enhancing not only their own independence but also the resilience and scalability of the broader OpenDHT ecosystem.

B. Secure Protocols in DMN

1) Matrix: Matrix implements a layered security architecture, featuring protocols designed to protect both the handshake phase and encrypted message sessions between users. As illustrated in Figure 2, the security protocols at the key handshake level primarily encompass key distribution protocols (❶) founded on public key cryptography (PKC) [23], involves key generation, upload, claim, query, etc., to guarantee the secure dissemination of keys. Besides, the Matrix security protocol also employs a Short Authentication String (SAS) based device verification protocol [24] (as depicted in ❷, Fig. 2) to authenticate user devices and ensure their authenticity. Regarding the encrypted message session level among users, Matrix also has devised a cryptographic ratchet protocol based on Olm [8] (❸ of the Fig. 2). This protocol ensures the confidentiality, PFS, and PCS of end-to-end encrypted message transmission within the system.

Key Handshake Protocol. The key handshake protocol in Matrix involves several core steps: generating key pairs using public-key cryptography, uploading identity and one-time public keys, claiming one-time keys, and querying other users'

[3]https://github.com/savoirfairelinux/opendht



Alice
Bob
1 Key Distribution (PKC-based)
1 Key Distribution (PKC-based)
KEYS
KEYS
KEYS
2 SAS Device Verification (ECDH, HKDF)
2 SAS Device Verification (ECDH, HKDF)
VERIFICATION
KEYS
KEYS
3 Olm Cryptographic Ratchet (X3-ECDH, HKDF, HMAC)
3 Olm Cryptographic Ratchet (X3-ECDH, HKDF, HMAC)
E2EE MSG

Fig. 2. Overview of the Matrix Security Protocols.

public keys for key agreement. Matrix clients use algorithms such as Ed25519 for digital signatures and Curve25519 for key agreement to generate identity and one-time key pairs. These public keys are then uploaded to the user's homeserver, which acts as a key distribution point for establishing secure sessions. To initiate an end-to-end encrypted session with another user (e.g., Bob), Alice queries Bob's homeserver through her own to retrieve Bob's public identity key and any available one-time keys. To ensure authenticity and resist impersonation attacks, Matrix supports device verification through a Short Authentication String (SAS) mechanism, which is inspired by Phil Zimmermann's ZRTP key agreement protocol [25]. In this process, both parties exchange and verify each other's public keys, derive a shared secret via a Diffie–Hellman key exchange, and confirm its authenticity through an out-of-band verification step using a human-readable SAS.

Message Encryption Protocol. To ensure the confidentiality of end-to-end encrypted messages and achieve both perfect forward secrecy (where the compromise of the current key does not impact the confidentiality of previously encrypted messages) and post-compromise security (where the compromise of the current key does not affect the confidentiality of subsequent encrypted messages), Matrix has introduced a cryptographic ratchet protocol based on Olm [8]. The core idea revolves around using the identity key and the one-time key to generate a shared key through multiple key agreements. Following this, the ratchet key and message encryption key are derived from this shared key, ensuring the security of end-to-end encryption. The inclusion of a one-time key adds an additional layer of assurance for both forward and backward confidentiality of encrypted messages. For a detailed explanation of the Olm protocol procedures, refer to Section V-B1. On the flip side, the Megolm cryptographic ratchet protocol [9], built on top of Olm [14], is designed to ensure the confidentiality of end-to-end encrypted group communication. In this study, we primarily focus on analyzing the security properties of the Olm cryptographic ratchet protocol.

2) Berty: Berty Messenger is a privacy-focused messaging application built atop the Wesh protocol [10], a custom-designed security framework for decentralized, end-to-end encrypted communication. The protocol draws from Scuttlebutt's handshake [26] and Signal's symmetric-key ratchet [12] mechanism to provide strong guarantees of confidentiality, authenticity, and message integrity. Participation in the Wesh network requires users to generate a Berty account, and all cryptographic key pairs in Wesh are derived using X25519 for secure key exchange and Ed25519 for digital signatures. Each peer is assigned a unique identity key pair, which serves as the cryptographic basis for secure and authenticated communication throughout the decentralized network.

Handshake Protocol. Berty's handshake protocol takes inspiration from Scuttlebutt's capability-based handshake [26], a secure key exchange framework built around the principle of cryptographic access capabilities. In this design, the server's public key serves as an implicit capability token, granting access to the system while simultaneously acting as its identifier. The handshake protocol includes a pre-authentication phase, in which the client is authenticated prior to any disclosure of sensitive information by the server, thereby reducing exposure to potentially untrusted or malicious peers. Following this initial step, the protocol proceeds with mutual authentication, involving a sequence of four verification steps to establish a secure and trusted communication channel.

Message Encryption Protocol. In the Wesh protocol, all communications are secured through end-to-end encryption based on a symmetric-key ratchet mechanism [12]. For each outbound message, a distinct Message Key (`MK`) is derived from the current Chain Key (`CK`) using the HMAC-based Key Derivation Function (`HKDF`). This derivation generates a unique encryption key for each message and simultaneously advances the Chain Key, ensuring forward secrecy so that compromising a single key does not compromise the security of past or future messages. The resulting Message Key is strictly single-use, never reused for multiple messages.

3) Status: Status employs the Waku protocol [18], a modular suite of protocols designed to support secure, censorship-resistant, and anonymous peer-to-peer communication. Waku builds upon the Whisper protocol [27] and refines its architecture to improve scalability, privacy, and modularity.

Security Protocol. Before establishing a one-to-one encrypted session, peers such as Alice and Bob must first perform mutual authentication. This is typically conducted through out-of-band methods, such as scanning QR codes face-to-face or matching Identicons derived from public key fingerprints. This authentication step serves two essential functions: confirming the identity of the peer and obtaining the initial public key material necessary to establish a secure communication session. Following authentication, asynchronous key exchange is conducted using the X3DH (Extended Triple Diffie-Hellman) protocol [11]. Status uses an elliptic-curve Diffie-Hellman (ECDH) scheme [28] based on the `secp256k1` curve for cryptographic operations. X3DH involves a combination of identity keys, ephemeral keys, and prekeys (signed and one-time) to derive a shared secret. Once a shared secret is established, Status employs the Double Ratchet algorithm [12] to encrypt subsequent message exchanges. This algorithm combines a Diffie-Hellman ratchet (to provide forward secrecy between sessions) with a symmetric-key ratchet (to secure message sequences within a session), ensuring that the compromise of one message key does not affect the security of past or future messages.

4) Jami: Jami implements end-to-end encryption (E2EE) using well-established cryptographic standards, without introducing custom encryption schemes. For media sessions, Jami uses Transport Layer Security version 1.3 during session initiation to negotiate symmetric session keys while providing perfect forward secrecy[4]. After the handshake, media streams are encrypted using the Secure Real-time Transport Protocol[5], which protects audio and video data from eavesdropping and tampering. Since Jami relies entirely on standardized and widely reviewed protocols such as TLS 1.3 and SRTP, this paper does not include a detailed security analysis of its encryption mechanisms.

The following section presents our methodology for measuring decentralized messaging networks and formally verifying their security protocols.

III. Proposed Method Design

To enable a comprehensive measurement and security analysis of decentralized messaging networks (DMNs), this study is guided by the following key research questions (RQs):

RQ1: What are the distribution characteristics of different DMN nodes? This question investigates the structural and geographical distribution of nodes within decentralized messaging networks, focusing on their roles in the overall network topology and their associations with higher-level applications.

RQ2: What security risks and vulnerabilities exist across different DMN nodes? This question focuses on evaluating the security posture of nodes within decentralized messaging networks. It involves analyzing network-layer risks such as exposure to malicious traffic, IP reputation issues, and centralized infrastructure dependencies.

RQ3: How can threat models and formal verification be systematically applied to DMN security protocols? This involves modeling protocol behavior, defining adversarial assumptions, identifying attack surfaces, and using formal tools to assess security properties such as authenticity and secrecy.

RQ4: What protocol-level vulnerabilities exist in the security protocols of different DMNs, and how can they be systematically identified and mitigated? This question focuses on analyzing the design and implementation of end-to-end encryption protocols, identifying cryptographic flaws, and proposing effective mitigation and protocol improvement strategies.

[4]https://docs.jami.net/en_US/user/faq.html#advanced
[5]https://datatracker.ietf.org/doc/html/rfc3711

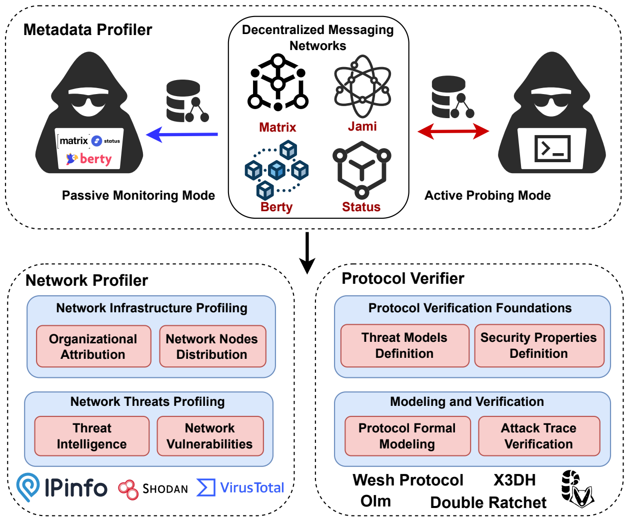


Fig. 3. The proposed D-MAP framework.

A. D-MAP

To address the above research questions, we introduce D-MAP (Decentralized Messaging Analysis Platform), a measurement and analysis framework that automatically captures communication traces from real-world decentralized messaging systems. D-MAP enables systematic analysis to uncover protocol-level inconsistencies, potential vulnerabilities, and deviations from expected security guarantees. Framework of the D-MAP as depicting in Fig. 3.

1) Metadata & Network Profiler: Studying decentralized messaging networks, including their node distribution, scale, and behavior, poses several challenges. For instance, privacy-focused designs limit peer visibility, node availability is often unstable due to mobile use, and protocol diversity across systems like Matrix, libp2p, and Waku hinders uniform analysis. To address this, D-MAP includes a Network Profiler module that automatically collects and analyzes node-level data from real-world deployments. We propose a multi-dimensional data acquisition strategy aimed at systematically capturing key characteristics of DMNs, including:

Passive Monitoring Mode. We deploy multiple measurement nodes that join the target network as regular peers without initiating active interactions. Instead, these nodes passively observe network and protocol events such as broadcasts, synchronization, and handshake processes. This approach allows us to gather data on peer join and leave behaviors, routing and relay paths, and gossip-based message propagation. For peer-to-peer networks based on protocols like libp2p or devp2p, we also capture low-level events including connection establishment and multiplexed stream communications.

Source Code Analysis and Key Path Instrumentation. By analyzing the source code of open-source clients and core modules within the target networks, we identify key protocol components and communication workflows such as identity authentication, key exchange, and message forwarding. We then insert logging or state-extraction hooks into these critical functions. For example, in Waku-based systems, we instrument the Relay module to capture metadata (e.g., ID, IP, hostname) on message flows and storage operations. In Matrix, we instrument the Synapse[6] server to monitor event processing and synchronization logic.

Active Probing and Layered Traversal. D-MAP configures measurement nodes to act as active endpoints within the network, initiating peer connections, exchanging messages, and joining groups. These nodes can recursively query neighboring peers or expose status interfaces, enabling hierarchical traversal (Breadth First Search) and comprehensive discovery of the network topology. We formalized as,

$$D = \bigcup_{k=0}^{d} D_k, D_k = \bigcup_{v \in L_k} data(v) \tag{1}$$

where D represents the complete set of data collected from all d layers, while $data(v)$ denotes the data obtained from node v, which belongs to layer L. For example, in the Matrix network, we utilize selected homeservers as entry points and iteratively join public rooms (via the `/publicRooms` API[7]) to discover and collect data on additional nodes. Furthermore, in networks employing Rendezvous [10] or Discovery protocols [22], such as Status or Berty, measurement nodes may use temporary identities to participate in peer discovery, facilitating the collection of seed nodes and their expansion paths.

2) Protocol Verifier: Verifying the security protocols of decentralized messaging networks poses significant challenges, including modeling diverse and complex protocols, defining appropriate threat models, and efficiently tracing potential attack vectors. For instance, the presence of multiple roles and distributed participants as seen in protocols like Berty's Wesh and Matrix's Olm adds layers of abstraction making it essential to develop threat models that capture the inherent complexity and heterogeneity of decentralized environments.

To address the diverse security challenges outlined above we perform a comprehensive analysis of the underlying security protocols and propose a formal verification framework. This framework supports defining multiple threat models such as external access server compromise and forced access as illustrated in Fig. 6 and enables automated verification of key security properties including authenticity perfect forward secrecy backward secrecy and post compromise security.

The Protocol Verifier abstracts and formalizes each target protocol including Matrix Olm, Berty Wesh, and Jami's X3DH and Double Ratchet protocols by modeling detailed interactions among participants such as key generation, key agreement, and message encryption. It then uses ProVerif [17], a widely adopted automated formal verification tool for cryptographic protocols, which enables the modeling and verification of security properties such as confidentiality, authenticity, and forward secrecy. For example, to verify authentication, the Protocol Verifier checks that whenever Bob completes a session, Alice must have initiated it: $\forall x, end_{Bob}(x) \Rightarrow begin_{Alice}(x)$, and formalized as:

```
event begin_Alice(x).
event end_Bob(x).
query ev:end_Bob(x) ==> ev:begin_Alice(x).
```

More details specification of the threat models and verification procedures are presented in Section V.

B. Experimental Setup and Data Overview

Experimental Setup. We deploy D-MAP with a hybrid architecture combining remote and local infrastructure. Remote Alibaba Cloud servers[8] in regions including Singapore, the United States, and Europe collect node level data such as IP addresses, domain names, URLs, and cryptographic elements from decentralized messaging networks. This data is periodically sent to local servers for centralized processing. The local infrastructure handles data preprocessing, node profiling, and protocol verification to support efficient measurement and security analysis.

The local server is equipped with dual Intel Xeon 4410Y processors, each running at 2.0 GHz with 12 cores and 24 threads, supported by 192 GB of DDR5 memory. This configuration provides ample computational power for formal analysis tasks, including running ProVerif and processing its outputs. On the remote side, three node acquisition instances are deployed, each featuring an Intel x86 architecture with four virtual CPUs and eight gigabytes of memory. These distributed network profilers are strategically located across different regions to enhance data acquisition coverage of decentralized messaging networks.

Data Overview. We conducted a six-month data collection campaign using D-MAP from January 3 to June 27, 2025, targeting multiple decentralized messaging networks. The system successfully identified and extracted a substantial volume of IP address data exceeding 78,000 entries. This includes 36,393 unique IPs associated with Matrix, corresponding to 69,284 distinct domains, 29,818 from Berty, 12,332 from the Jami network, and 1,190 from Status.

Additionally, by leveraging data from IPInfo [29], Shodan [30], and VirusTotal [31], we enrich the collected IP address datasets with comprehensive metadata. This includes autonomous system numbers (ASNs), organizational ownership, ISP details, geolocation information (country, region, city), hosting classifications (such as cloud infrastructure versus residential access), and threat intelligence indicators like reported malicious activity, blacklist status, and known malware signatures. These enriched features enable detailed analysis of node distribution, operational behavior, and potential security risks across various decentralized messaging networks. A thorough presentation of the extended dataset and its analysis can be found in Section IV. Furthermore, the protocol verification module systematically extracts and

[6]https://github.com/element-hq/synapse

[7]https://spec.matrix.org/v1.13/client-server-api/

[8]https://www.alibabacloud.com/en

formalizes the cryptographic specifications of the underlying security protocols used in each network. This involves granular component-level details from protocols such as Matrix Olm, Berty Wesh, and Status's X3DH and Double Ratchet. A comprehensive analysis of these cryptographic components and their security properties is provided in Section V.

Ethics. D-MAP's Network Profiler and Protocol Verifier run on self-hosted nodes, ensuring controlled resource use without impacting other participants or exposing private data such as authentication information. Protocol vulnerability assessments were performed in a controlled environment, preventing risks to real users. Therefore, we maintain that our data collection and security research adhered to ethical standards.

The proposed D-MAP framework, encompassing the full implementation of the Network and Protocol Profilers along with the collected datasets, has been open-sourced at https://github.com/lihaoSDU/DecentralizedMesagers.

IV. Measurement Study on Decentralized Messaging Networks

To address Research Questions 1 and 2, this section presents a large-scale empirical measurement study of decentralized messaging networks. We analyze both the physical and network-level geographic distribution of nodes to assess their deployability, and systematically evaluate the security risks and potential vulnerabilities associated with various types of decentralized messaging nodes.

A. Network Infrastructure Profiling

In this section, we conduct a comprehensive profiling of the network infrastructures underlying prominent DMNs to uncover critical insights into their global deployment characteristics, hosting environments, and associated organizational entities. Our analysis focuses on key infrastructure-level attributes, including geographic dispersion, autonomous system (AS) affiliations, the prevalence of cloud-based versus residential hosting, and overall network topology structures. By examining these dimensions, we aim to elucidate underlying operational patterns, deployment strategies, and potential vectors of centralization.

1) Network Nodes Distribution: D-MAP has identified and collected over 78,000 unique IP addresses associated with nodes in decentralized messaging networks (DMNs), encompassing deployments across six continents: North America, Europe, Asia, Oceania, Africa, and South America. A summary of the global distribution is provided in Table II. At finer granularity, the Matrix network exhibits presence in over 91 countries and 3,306 cities, Berty in 96 countries and 2,024 cities, Jami in 98 countries and 1,103 cities, and Status in 26 countries and 79 cities. The top 10 countries hosting the largest number of DMN-related IP addresses are detailed in Table III. The collected IP addresses span a wide range of IPv4 address space, including allocations at the /8, /16, and /24 prefix levels. For example, Matrix nodes are distributed across 202 distinct /8 and 7,688 /16 IPv4 prefixes, while Berty nodes span 207 /8 and 4,878 /16 prefixes. This broad geographic and network-layer dispersion highlights the extensive global reach and heterogeneous deployment patterns of decentralized messaging infrastructures.

TABLE II
Continent-wise distributions.

Continent	# Matrix	# Berty	# Jami	# Status
North America	13,158	5,093	512	17
Europe	21,274	7,848	1,676	207
Asia-Pacific	1,201	3,409	420	121
Oceania	574	139	208	0
South America	142	149	97	1
Africa	39	335	52	2

TABLE III
Country-wise distributions of Top-10.

# Matrix	# Berty	# Jami	# Status
(USA, 12,068)	(USA, 4,778)	(USA, 341)	(Germany, 156)
(Germany, 10,404)	(Russia, 2,879)	(Russia, 256)	(Indonesia, 64)
(France, 2,488)	(Germany, 1,999)	(France, 255)	(Hong Kong, 30)
(Netherlands, 1,724)	(UK, 855)	(Germany, 249)	(USA, 17)
(UK, 1,149)	(China, 797)	(Australia, 186)	(Netherlands, 17)
(Finland, 1,091)	(South Korea, 641)	(Canada, 133)	(Russia, 10)
(Canada, 1,057)	(Japan, 484)	(China, 131)	(UK, 9)
(Russia, 602)	(Singapore, 421)	(Italy, 121)	(India, 6)
(Switzerland, 568)	(France, 414)	(UK, 116)	(Singapore, 6)
(Australia, 498)	(Hong Kong, 359)	(India, 104)	(China, 5)

Ports. Investigating the open ports of decentralized messaging network nodes offers valuable insights into the network services accessible to users. For example, the presence of open ports such as 80 (HTTP) and 22 (SSH) reveals the types of services these nodes may support and their potential exposure to external access.

In our investigation, we observed that Matrix nodes exposed a total of 4,023 unique ports, Berty nodes exposed 1,419, Jami nodes exposed 1,539, and Status nodes exposed 79. Table IV provides a comprehensive overview of the most frequently exposed ports and their corresponding node counts. Interestingly, Berty nodes often did not utilize the default port 9000 specified in its official deployment documentation. Instead, they relied on dynamic port allocation strategies facilitated by the libp2p framework, which incorporates NAT traversal techniques such as AutoNAT and hole punching∼ []. In addition to TCP services, we identified a substantial number of exposed UDP based ports across decentralized messaging network nodes. For example, port 30303, which is commonly used by Ethereum and Waku for peer discovery and communication, was frequently observed.

2) Organizational Attribution of Nodes: Understanding node ownership, including organizational affiliation and metadata tags, is essential for analyzing the ecosystem of decentralized messaging networks. To support this analysis, we examined the autonomous systems (ASes), service associations, and descriptive tags associated with the nodes.

ASes and ISPs. The nodes of decentralized messaging networks are distributed across a wide range of Autonomous Systems (ASes), reflecting the global and heterogeneous nature of their deployment. Specifically, Matrix nodes are spread

TABLE IV
OPEN PORT LANDSCAPE ACROSS DIVERSE DEPLOYMENT PLATFORMS.

Index	# Matrix	# Berty	# Jami	# Status
1	(80, 17,315)	(22, 6,733)	(443, 103)	(22, 182)
2	(443, 16,682)	(80, 2,060)	(80, 82)	(3000, 176)
3	(22, 9,402)	(4001, 1,940)	(8443, 41)	(8080, 41)
4	(8080, 4,414)	(443, 1,276)	(22, 38)	(8545, 32)
5	(8443, 4,233)	(5443, 870)	(7547, 22)	(53, 30)
6	(2087, 3,886)	(5080, 868)	(8080, 20)	(30303, 29)
7	(8880, 3,823)	(8080, 699)	(7443, 19)	(80, 28)
8	(2083, 3,807)	(5001, 347)	(8089, 19)	(1234, 25)
9	(2082, 3,787)	(7547, 210)	(4000, 19)	(3001, 20)
10	(2086, 3,763)	(9000, 170)	(1701, 17)	(443, 15)

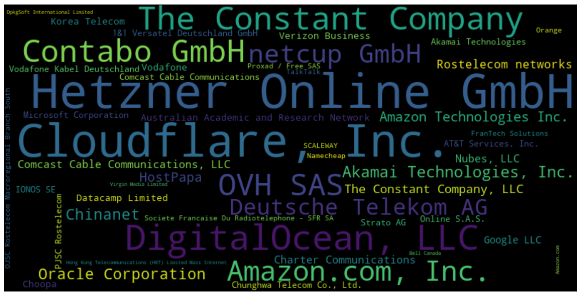


Fig. 4. ISP-based word cloud of decentralized messaging nodes.

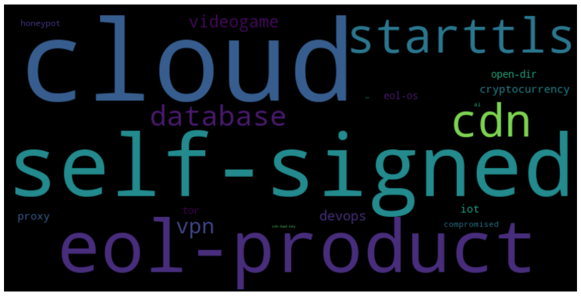


Fig. 5. Tag-based word cloud of decentralized messaging nodes.

across 2,477 ASes, Berty across 1,207 ASes, Jami across 764 ASes, and Status across 70 ASes. However, despite this apparent diversity, a substantial concentration of nodes exists within a small number of ASes, which may pose risks to the intended decentralization. For example, 4,523 node IP addresses are hosted within AS13335 (Cloudflare, Inc., ranked 69th globally by AS Rank[9]), 4,278 within AS24940 (Hetzner Online GmbH, ranked 650th), and 2,260 within AS20473 (The Constant Company, LLC, ranked 98th). In addition, the IP addresses associated with decentralized messaging networks are distributed across more than 5,500 distinct Internet Service Providers (ISPs), indicating a broad and diverse deployment landscape. Specifically, Matrix nodes span 3,244 ISPs, Berty nodes 1,603, Jami nodes 922, and Status nodes 85. Despite this diversity, a significant proportion of nodes are hosted by a small number of major infrastructure providers. We visualize the organizations exhibiting monopolistic characteristics using a word cloud representation in Fig. 4. For instance, 4,367 node IP addresses are attributed to Cloudflare, Inc., 4,179 to Hetzner Online GmbH, 1,405 to The Constant Company, LLC, and 1,163 to Contabo GmbH. This concentration suggests a degree of infrastructural dependency on dominant cloud providers, highlighting potential centralization risks within decentralized systems.

Tags and Service Association. In our measurement study, we identified over 78,000 decentralized messaging network nodes, which were categorized into more than 30 distinct tags based on their observed infrastructure and service characteristics. These tags were derived through the enrichment of raw IP data using threat intelligence platforms such as VirusTotal [31] and Shodan [30], combined with port fingerprinting and service banner analysis. The resulting classifications include self-signed (nodes using self-issued TLS certificates), cloud (hosted on known cloud provider IP ranges), starttls (supporting opportunistic TLS upgrades), CDN, VPN, proxy, cryptocurrency, and honeypot. A detailed distribution of these tags across Matrix, Berty, Jami, and Status networks is presented in Fig. 5. For instance, many nodes exposed common metadata patterns associated with commercial VPN providers (e.g., NordVPN[10]), and were flagged by VirusTotal based on historical scanning activity and threat feeds. Specifically, 340 IP addresses were tagged as VPN, and 25 as cryptocurrency-related, often linked to mining pools or blockchain nodes.

In addition, our analysis reveals distinct differences in the software products associated with node IPs across decentralized messaging platforms. For instance, 52.4% of Matrix nodes are associated with nginx, and 29.2% with OpenSSH. In contrast, within the Berty network, 22.9% of nodes use OpenSSH, while only 7.1% utilize nginx. This discrepancy is largely attributed to the architectural differences: Matrix homeservers are typically deployed with domain name mappings to IP addresses, often involving web-facing components, whereas Berty, Jami, and Status nodes generally lack such domain-IP associations, resulting in lower exposure to typical web infrastructure products. Furthermore, similar to the distribution observed across ISPs, the cloud service providers hosting nodes of various decentralized messaging networks exhibit clear signs of infrastructural centralization. Specifically, 11,629 nodes are hosted by Google Cloud, 9,283 by DigitalOcean, and 9,027 by Vultr.

This concentration reveals two important insights. First, the heavy reliance on centralized infrastructure stands in contrast to the core principles of decentralization that these networks aim to uphold. Second, the narrow range of hosting providers suggests that alternative decentralized infrastructure platforms, such as Akash Network[11] and Golem[12], have yet to achieve widespread adoption.

[9]https://asrank.caida.org/

[10]https://nordvpn.com/cybersec-site/

[11]https://akash.network/

[12]https://www.golem.network/

B. Network Threats Profiling

To address Research Question 2, which pertains to the identification of cybersecurity threats within decentralized messaging networks, we performed a large-scale measurement study utilizing data collected from VirusTotal and Shodan. This analysis seeks to systematically uncover existing vulnerabilities and potential security risks associated with node IP assets across multiple platforms.

1) Threat Intelligence: In this section, we perform a comprehensive analysis leveraging threat intelligence data from VirusTotal to proactively identify and characterize potential security threats.

Malicious IPs. We identified more than 3,400 IP addresses of decentralized messaging network nodes, representing approximately 4.4% of the total, that are associated with malicious activity. For example, an IP address in the Matrix network, 209.*.*.9, was flagged by 18 different threat detection engines, while an IP in Berty, 154.*.*.35, was detected by 12 engines. To better understand the malicious activity within decentralized messaging networks, we manually analyzed a subset of 3,400 flagged IP addresses. This revealed that many function as command-and-control nodes for malware families, notably including Mirai botnet infrastructure and Venom RAT, both known for device exploitation and unauthorized access.

In addition, our analysis revealed that more than 9,200 node IP addresses across decentralized messaging networks were associated with malware-related communication activity, involving over 20,000 unique malicious files. For instance, one Matrix homeserver node (IP address 34.*.*.180) was found to have interacted with 674 distinct malware samples, strongly indicating its involvement in malicious infrastructure such as command and control communication or malware distribution. We further categorized the malware samples based on VirusTotal's aggregated threat classifications. The majority were labeled as *trojans* (78%), followed by *viruses* (4.8%), *adware* (3.3%), *phishing* (2.7%), and *worms* (1.5%). Notably, we also observed the presence of miner-related and *cryptojacking* malware within the communicated files, indicating potential abuse of decentralized nodes for unauthorized resource mining. Specifically, we identified 138 miner-associated samples in Matrix, 93 in Berty, and 14 in Jami. This suggests that certain nodes may have been compromised or misconfigured, enabling their exploitation for unauthorized computational tasks.

TABLE V
TOP-10 CVES ACROSS DECENTRALIZED MESSAGING NETWORKS.

Matrix	Berty	Jami	Status
CVE-2023-44487	CVE-2023-44487	CVE-2013-4365	CVE-2013-4365
CVE-2025-26465	CVE-2021-3618	CVE-2012-3526	CVE-2012-3526
CVE-2021-3618	CVE-2021-23017	CVE-2009-0796	CVE-2024-38476
CVE-2021-23017	CVE-2013-4365	CVE-2012-4001	CVE-2024-38477
CVE-2013-4365	CVE-2012-3526	CVE-2011-1176	CVE-2009-0796
CVE-2012-3526	CVE-2009-0796	CVE-2011-2688	CVE-2012-4001
CVE-2009-0796	CVE-2012-4001	CVE-2013-2765	CVE-2012-4360
CVE-2012-4001	CVE-2012-4360	CVE-2007-4723	CVE-2011-1176
CVE-2012-4360	CVE-2011-1176	CVE-2013-0941	CVE-2024-40898
CVE-2011-1176	CVE-2011-2688	CVE-2012-4360	CVE-2011-2688

2) Network Vulnerabilities: To assess the security risks and vulnerabilities present in nodes of decentralized messaging networks, we identified a substantial number of known vulnerabilities associated with node IP assets across multiple platforms. The top 10 vulnerabilities detected in each network are summarized in Table V. For example, CVE-2025-26465, a recently disclosed vulnerability in OpenSSH that is triggered when the `VerifyHostKeyDNS` option is enabled, was found in over 1,800 Matrix node IPs, posing a considerable security threat. Another critical vulnerability, CVE-2021-3618, which allows potential man in the middle attacks due to protocol mismatches among TLS servers using compatible certificates, was identified in 1,785 Matrix nodes and 531 Berty nodes.

In addition, several Common Weakness Enumeration (CWE) categories were identified in association with node IP assets across the analyzed decentralized messaging networks. For example, 84 Matrix nodes and 11 Jami nodes exhibited the CWE-79 [] vulnerability, which pertains to improper neutralization of input during web page generation, a flaw that can facilitate cross-site scripting (XSS) attacks. Furthermore, 21 nodes in the Berty network were found to be affected by CWE-400 [], representing uncontrolled resource consumption that could potentially be exploited to launch denial-of-service (DoS) attacks.

Beyond CWE classifications, we also revealed that the identified vulnerabilities affect a wide range of third-party software vendors and libraries commonly integrated into decentralized messaging infrastructures. Specifically, 769 vulnerabilities in Matrix nodes were associated with components from Oracle Corporation, notably outdated deployments of Java and MySQL. In the Berty network, 67 vulnerabilities were linked to the Apache Software Foundation, often stemming from misconfigured or unpatched Apache HTTP Server instances. The Jami network showed 43 instances of vulnerabilities tied to OpenSSL, reflecting potential weaknesses in cryptographic implementations. Matrix nodes also exhibited 66 JetBrains-related issues, likely due to publicly accessible development environments or misconfigured build tools. Additionally, 53 Status nodes were impacted by vulnerabilities in Red Hat software, commonly resulting from insecure default settings and outdated system packages.

Takeaway #1: The decentralized messaging ecosystem is global in scope, yet signs of infrastructure centralization remain due to limited use of decentralized hosting. In addition, over 4.4% of node IPs show malicious activity, with many affected by vulnerabilities such as XSS and DoS across multiple software supply chains.

V. FORMAL VERIFICATION OF SECURITY PROTOCOLS

To address Research Questions 3 and 4, this study proposes a formal verification methodology utilizing the ProVerif tool to rigorously analyze the security protocols deployed

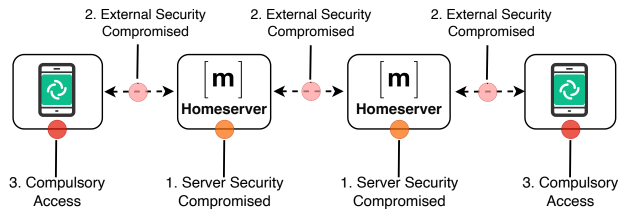

2. External Security Compromised
2. External Security Compromised
2. External Security Compromised
Homeserver
Homeserver
3. Compulsory Access
1. Server Security Compromised
1. Server Security Compromised
3. Compulsory Access

Fig. 6. Threat models of the DM security protocols.

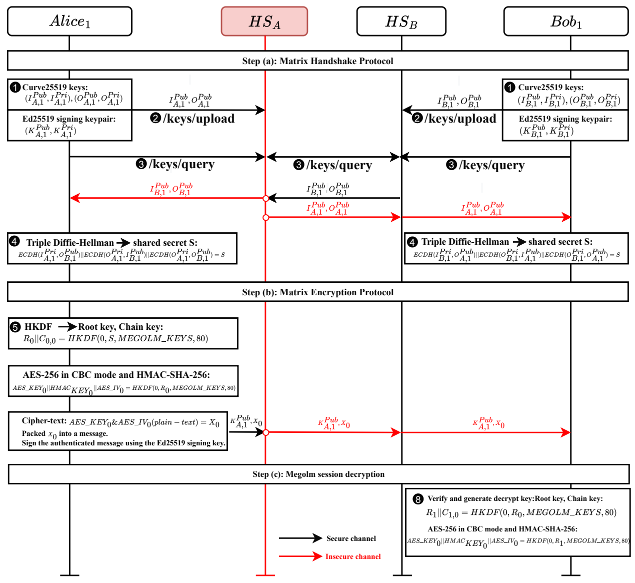

$Alice_1$
HS_A
HS_B
Bob_1
Step (a): Matrix Handshake Protocol
❶ Curve25519 keys: $(I^{Pub}_{A,1}, I^{Pri}_{A,1}), (O^{Pub}_{A,1}, O^{Pri}_{A,1})$
Ed25519 signing keypair: $(K^{Pub}_{A,1}, K^{Pri}_{A,1})$
$I^{Pub}_{A,1}, O^{Pub}_{A,1}$
❷ /keys/upload
$I^{Pub}_{B,1}, O^{Pub}_{B,1}$
❷ /keys/upload
❶ Curve25519 keys: $(I^{Pub}_{B,1}, I^{Pri}_{B,1}), (O^{Pub}_{B,1}, O^{Pri}_{B,1})$
Ed25519 signing keypair: $(K^{Pub}_{B,1}, K^{Pri}_{B,1})$
❸ /keys/query
❸ /keys/query
❸ /keys/query
$I^{Pub}_{B,1}, O^{Pub}_{B,1}$
$I^{Pub}_{B,1}, O^{Pub}_{B,1}$
$I^{Pub}_{A,1}, O^{Pub}_{A,1}$
$I^{Pub}_{A,1}, O^{Pub}_{A,1}$
❹ Triple Diffie-Hellman → shared secret S:
$ECDH(I^{Pri}_{A,1}, O^{Pub}_{B,1}) \| ECDH(O^{Pri}_{A,1}, I^{Pub}_{B,1}) \| ECDH(O^{Pri}_{A,1}, O^{Pub}_{B,1}) = S$
❹ Triple Diffie-Hellman → shared secret S:
$ECDH(I^{Pri}_{B,1}, O^{Pub}_{A,1}) \| ECDH(O^{Pri}_{B,1}, I^{Pub}_{A,1}) \| ECDH(O^{Pri}_{B,1}, O^{Pub}_{A,1}) = S$
Step (b): Matrix Encryption Protocol
❺ HKDF → Root key, Chain key:
$R_0 \| C_{0,0} = HKDF(0, S, MEGOLM\_KEYS, 80)$
AES-256 in CBC mode and HMAC-SHA-256:
$AES\_KEY_0 \| HMAC_{KEY_0} \| AES\_IV_0 = HKDF(0, R_0, MEGOLM\_KEYS, 80)$
Cipher-text: $AES\_KEY_0 \& AES\_IV_0(plain-text) = X_0$
Packed X_0 into a message.
Sign the authenticated message using the Ed25519 signing key.
$K^{Pub}_{A,1}, X_0$
$K^{Pub}_{A,1}, X_0$
$K^{Pub}_{A,1}, X_0$
Step (c): Megolm session decryption
❽ Verify and generate decrypt key:Root key, Chain key:
$R_1 \| C_{1,0} = HKDF(0, R_0, MEGOLM\_KEYS, 80)$
AES-256 in CBC mode and HMAC-SHA-256:
$AES\_KEY_0 \| HMAC_{KEY_0} \| AES\_IV_0 = HKDF(0, R_1, MEGOLM\_KEYS, 80)$
Secure channel
Insecure channel

Fig. 7. Protocol specifics of handshake, encryption, and decryption within the Matrix Olm ratchet.

in decentralized messaging systems. The analysis specifically targets the handshake and end-to-end encryption protocols implemented by Matrix, Berty, and Status, with the objective of systematically evaluating their cryptographic guarantees and uncovering potential protocol-level vulnerabilities.

A. Threat Models and Security Properties

In this section, a comprehensive threat modeling framework is developed to facilitate the formal verification of security protocols adopted by decentralized messaging networks.

Given the decentralized nature of the messaging security protocol, which involves multiple participants and roles, we have introduced three threat models in distinct scenarios based on the Dolev-Yao [32] model. As shown in Fig. 6, we present the "Server Security Compromised", "External Security Compromised", and "Compulsory Access" threat models in a detailed manner.

- The **Server Security Compromised** threat model unfolds in scenarios where the server undergoes compromise or functions as a Byzantine server. In such situations, attackers can attempt to infiltrate the server through various avenues, seeking unauthorized access and subsequently engaging in malicious activities, such as the substitution or alteration of user key information.
- The **External Security Compromised** threat model addresses scenarios where external transmissions are compromised, leading to potential vulnerabilities in the communication link within the system. Adversaries may attempt interception, tampering, or eavesdropping on sensitive information during data transmission.
- The **Compulsory Access** threat model envisions scenarios involving the brief takeover of an endpoint device (full-compromised) or compromise by malicious apps on the phone (semi-compromised). This takeover could happen during border inspections or situations where adversaries use aggressive techniques like brute force attacks, malware, or other forceful means to gain unauthorized access to the endpoint.

Security Properties. Threat models characterize the potential capabilities and attack vectors of adversaries that must be accounted for in the design, analysis, and deployment of cryptographic protocols. To ensure resilience against these threats, we define a set of critical security properties that decentralized messaging protocols should satisfy, including confidentiality, authenticity, perfect forward secrecy (PFS), and post-compromise security (PCS). In this paper, we conduct a formal analysis of decentralized messaging security protocols with an emphasis on verifying these core properties. By modeling the protocols and their associated guarantees, we systematically explore possible adversarial behaviors and attack traces that may compromise security.

B. Formal Modeling and Verification

Given the complexity of decentralized messaging security protocols, which involve multiple components and participants, formal verification becomes particularly challenging. To address this, we propose a modular formal verification methodology that supports flexible threat modeling and enables systematic evaluation of security properties across different protocols.

1) Matrix: In this section, we conduct a detailed analysis of the protocol flow of the Matrix Olm cryptographic ratchet protocol. Building on this analysis, we develop and implement a formal model using ProVerif [17], which has been instrumental in advancing research across various domains, including IoT security [33], [34].

Handshake Phase. In the handshake phase of the Olm protocol, we extract the protocol logic based on the Matrix specification [35] and present a detailed representation of the key distribution process, which forms a core component of the Matrix security sub-protocol, as illustrated in Figure 7. This process unfolds through several critical stages, namely, identity key generation, identity key upload, one-time key claim, and key query. Each stage is essential to establishing secure communication channels and contributes significantly to the overall robustness and resilience of the protocol's security architecture.

As depicted in Figure 7, the protocol involves four distinct participants: the sender, the receiver, the sender's homeserver, and the receiver's homeserver. Furthermore, the communication exchange among these participants adheres to our proposed External Security Compromised threat model. In the context of this model, every entity engages within an insecure transmission channel, underscoring the significance of addressing potential external security threats. The core of information exchange among participants is centered on the primary dissemination of public key components emanating from the generated key pairs, and denoted as `pk_A_I`, `pk_A_O`, `pk_B_I`, and `pk_B_O`. We comprehensively formalized the operations embedded within the protocol, capturing each distinct behavior in detail through precise modeling. Our primary objective is to rigorously assess whether the key distribution protocol adheres to the previously defined security properties. Such as,

```
// Authenticity in Handshake Phase
query userID: bitstring, pk_I: pkey, pk_E: pkey;
event(evRecieveBobKeys(userID, pk_I, pk_E)) ==>
    event(evUploadBobKey(pk_I)) && event(
    evClaimBobKey(pk_E)).
```

Encryption Phase. The Matrix Olm cryptographic ratchet protocol is engineered to ensure both forward secrecy and backward secrecy for end-to-end encrypted communication between users. When Alice initiates a secure session with Bob, the protocol begins a key negotiation process. Olm achieves this by performing three Elliptic-Curve Diffie–Hellman (ECDH) computations [], as outlined below:

$$S = ECDH(I_A, E_B)||ECDH(E_A, I_B)||ECDH(E_A, E_B) \quad (2)$$

This involves utilizing one's own identity key (`I_A, I_B`) and the private component of the ephemeral key (`E_A, E_B`), along with the other party's identity and the public component of the ephemeral key, ultimately generating the shared secret. Following this, both entities can employ the shared key to derive the root key (`RK`), chain key (`CK`), and message key (`MK`) for the cryptographic ratchet using the HKDF algorithm [].

Leveraging the message key, the HKDF function is applied to derive a set of encryption parameters, including the AES encryption key, HMAC key, and AES initialization vector (IV). To encrypt a new message, the client generates a fresh one-time key, enabling the negotiation of a new shared secret and the update of the ratchet chain key. This process preserves both forward secrecy and backward secrecy for end-to-end encrypted communications. When a message is received, it can be decrypted using the locally derived decryption key based on asymmetric key agreement. A critical component of the Olm protocol is the Triple-ECDH key agreement algorithm, which underpins the security of the initial key exchange. In our formal model, we explicitly capture the operational logic of this algorithm, carefully detailing each step of its computation and its integration into the protocol, as an example in:

```
letfun get_ecdhkey(sk_I_local:skey, pk_E_remote:
    pkey) = sca(sk_I_local, pk_E_remote).
```

This granular representation allows us to dissect and understand the algorithm's behavior at a detailed level. Drawing from the negotiated shared key, the subsequent step involves the computation of various keys essential for encryption. This encompasses the derivation of keys such as `rootKey`, `aesKey`, `hmacKey`, `aesIV`, and others, ensuring a comprehensive suite of cryptographic elements to facilitate secure communication.

$$R_i||C_{i,0} = HKDF(R_{i-1}, ECDH(T_{i-1}, T_i), INFO, 64) \quad (3)$$

where T_{i-1} is the previous ratchet key, and T_i is the current ratchet key, and we formal this in our verification model:

```
// (* root key, chain key, message key *)
let (rootKey: key, chainKey: key) = h1(rootKey,
    get_ecdhkey(pk_ratchetKey_B, ratchetKey_A),
    INFO, 64) in
let chainKey = HMAC_SHA256(chainKey,PADDING_1) in
let msgKey = HMAC_SHA256(chainKey,PADDING_2) in
```

To finalize the encryption process, we utilize the derived keys, as previously established, to employ the `senccbc` (AES-256 in CBC mode):

```
// encrypt plain-text with the message key
let (aesKey: key, hmacKey: key, aesIV: iv) = h2(
    SALT, msgKey, INFO, 80) in
let cipher_text_A = senccbc(plain_text_A, aesKey,
     aesIV) in
```

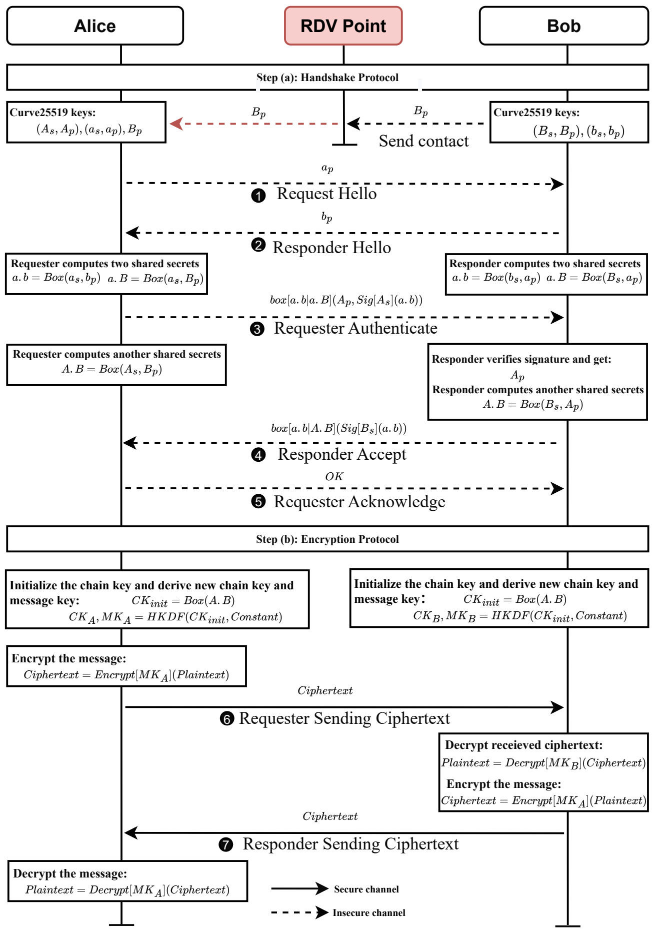
This step ensures the secure transformation of the plaintext message into a ciphertext, safeguarding its confidentiality during transmission. On the receiving end, decryption is orchestrated using the `sdeccbc`. This cryptographic operation ensures the secure retrieval and reconstruction of the original plaintext message from the received ciphertext.

In order to verify various security properties, encompassing confidentiality, authenticity, PFS, and PCS, we incorporate a series of query statements within the Olm cryptographic ratchet protocol. These queries function as a pivotal mechanism for gauging the protocol's alignment with and fulfillment of the predefined security attributes. This execution step serves as the conclusive phase in our comprehensive evaluation of the Olm cryptographic ratchet protocol, ensuring the thorough examination of its adherence to the specified security properties.

```
// Authenticity and Confidentiality properties
query ...; event(evClientAliceReceiveCiphertext(
    cipher_text)) ==> event(
    evClientBobSendCiphertext(cipher_text)).
query attacker(plain_text).
```

2) Berty: The Berty Wesh protocol provides end-to-end encryption and guarantees perfect forward secrecy for all exchanged messages. In this section, we formally model both the handshake and end-to-end encryption phases of the Wesh protocol. The complete protocol flow is depicted in Fig. 8.

Modeling Handshake Phase. We first model the handshake phase, which is largely inspired by Scuttlebutt's capability-based handshake mechanism []. This phase begins with the requester initiating the process by sending a `Hello` message to the responder. This message includes the requester's ephemeral public key, denoted as `pk_A_E`. Upon receiving the `Hello`, the responder replies with a `Hello` message of their own, which contains both their identity public key `pk_B_I` and



Alice
RDV Point
Bob
Step (a): Handshake Protocol
Curve25519 keys: $(A_s, A_p), (a_s, a_p), B_p$
B_p
B_p
Send contact
Curve25519 keys: $(B_s, B_p), (b_s, b_p)$
a_p
❶ Request Hello
b_p
❷ Responder Hello
Requester computes two shared secrets $a.b = Box(a_s, b_p)$ $a.B = Box(a_s, B_p)$
Responder computes two shared secrets $a.b = Box(b_s, a_p)$ $a.B = Box(B_s, a_p)$
$box[a.b|a.B](A_p, Sig[A_s](a.b))$
❸ Requester Authenticate
Requester computes another shared secrets $A.B = Box(A_s, B_p)$
Responder verifies signature and get: A_p
Responder computes another shared secrets $A.B = Box(B_s, A_p)$
$box[a.b|A.B](Sig[B_s](a.b))$
❹ Responder Accept
OK
❺ Requester Acknowledge
Step (b): Encryption Protocol
Initialize the chain key and derive new chain key and message key: $CK_{init} = Box(A.B)$ $CK_A, MK_A = HKDF(CK_{init}, Constant)$
Initialize the chain key and derive new chain key and message key: $CK_{init} = Box(A.B)$ $CK_B, MK_B = HKDF(CK_{init}, Constant)$
Encrypt the message: $Ciphertext = Encrypt[MK_A](Plaintext)$
$Ciphertext$
❻ Requester Sending Ciphertext
Decrypt receieved ciphertext: $Plaintext = Decrypt[MK_B](Ciphertext)$
Encrypt the message: $Ciphertext = Encrypt[MK_A](Plaintext)$
$Ciphertext$
❼ Responder Sending Ciphertext
Decrypt the message: $Plaintext = Decrypt[MK_A](Ciphertext)$
Secure channel
Insecure channel

Fig. 8. The handshake and end-to-end encryption protocols within the Berty.

their ephemeral public key pk_B_E. As a result, both the requester and the responder can use the other party's public key together with their own private key to compute two shared secrets, denoted as a.b and a.B.

```
let ecdhkey_ab_Responder = calc_ecdhkey(skey_B_E,
      pk_A_E) in
let ecdhkey_aB_Responder = calc_ecdhkey(skey_B_I,
      pk_A_E) in
```

The requester sends a keybox containing a digital signature and their identity public key to authenticate themselves to the responder. This keybox is encrypted using the previously derived shared secrets, a.b and a.B (as illustrated in Step ❸ of Figure 8), thereby implicitly proving the requester's possession of the responder's identity key. Following this exchange, both parties can compute an additional shared secret, denoted A.B, which strengthens the session's cryptographic binding.

To rigorously model this multi-step key derivation and authentication process in ProVerif, we decomposed the protocol into distinct, modular components: ephemeral key exchange, identity validation, and shared key computation. Confidentiality of intermediate keys was preserved through the use of private channels, while event annotations were employed to track identity commitments and enforce injective agreement, ensuring both parties' participation in the same session instance, an shown in evResponderCalcShareSecret and evRequesterCalcShareSecret.

The responder transmits a sealed message, referred to as a secret box, that contains a digital signature over the previously derived shared key a.b and a.B, specifically formatted as,

$$box[a.b|a.B](A, sig[A](a.b)) \tag{4}$$

This secret box is encrypted using both the ephemeral shared secret a.b and the newly derived key a.B, thereby demonstrating that the responder has successfully received, decrypted, and validated the requester's earlier message.

```
// Requester sends: box[a.b|a.B](A,sig[A](a.b))
let boxKey = boxSeal(ecdhkey_ab, ecdhkey_aB,
      pk_A_I, sign(skey_A_I, ecdhkey_ab)) in
let ecdhkey_AB=calc_ecdhkey(skey_A_I,pk_B_I) in
```

This response not only confirms the responder's possession of the correct key material but also provides implicit authentication of the session. To formally model this behavior in ProVerif, the responder's operations are decomposed into structured processes, each corresponding to distinct protocol phases such as key derivation, message construction, and signature verification. These events capture the requester's successful receipt and verification of the responder's authenticated response and the correct derivation of the shared secret A.B.

$$box(A_s, B_p) = A.B = box(A_p, B_s) \tag{5}$$

For confidentiality and authenticity guarantees, we apply private channels to simulate secure communication and use correspondence assertions to verify that event sequences are logically consistent (e.g., evRequesterReceiveboxKey and evRequesterAuthenticateCalcSharedKey).

Encryption Phase. In the Wesh protocol, all message exchanges are secured using symmetric-key ratchet encryption, following a design similar to the Signal protocol [12]. When a user initiates message transmission, a fresh message key is (MK) derived from the current chain key (CK) using the HMAC-based Key Derivation Function (HKDF). This derivation process yields two outputs: the next message key for encrypting the message and an updated chain key for subsequent derivations, ensuring that each message is encrypted with a unique, non-reused key and maintaining forward secrecy.

To formally verify this encryption mechanism, we construct a dedicated Encryption Phase module in our ProVerif model. Within this module, we abstract and formalize key operations including key evolution (deriveNextKeys) and authenticated encryption (secretBoxSeal). Specifically, we simulate the HKDF-based derivation using ProVerif's cryptographic primitives to capture the one-way nature and unlinkability of chain keys. Ciphertexts are modeled using authenticated encryption constructors, capturing both confidentiality and integrity guarantees.

```
let (nextCK_Requester:chainKey, MK:msgKey) =
      deriveNextKeys(init_CK_Requester,Constant) in
let cipherTextRequester = secretBoxSeal(
      plainTextRequester, Constant, MK) in
```

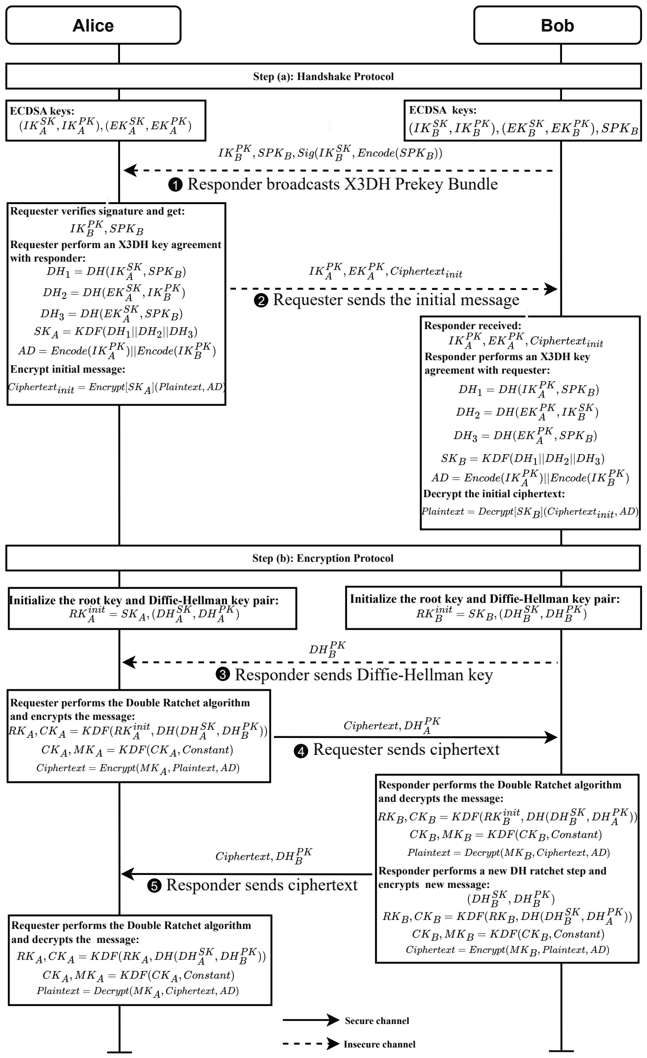

Alice
Bob
Step (a): Handshake Protocol
ECDSA keys: $(IK_A^{SK}, IK_A^{PK}), (EK_A^{SK}, EK_A^{PK})$
ECDSA keys: $(IK_B^{SK}, IK_B^{PK}), (EK_B^{SK}, EK_B^{PK}), SPK_B$
$IK_B^{PK}, SPK_B, Sig(IK_B^{SK}, Encode(SPK_B))$
❶ Responder broadcasts X3DH Prekey Bundle
Requester verifies signature and get: IK_B^{PK}, SPK_B
Requester perform an X3DH key agreement with responder:
$DH_1 = DH(IK_A^{SK}, SPK_B)$
$DH_2 = DH(EK_A^{SK}, IK_B^{PK})$
$DH_3 = DH(EK_A^{SK}, SPK_B)$
$SK_A = KDF(DH_1||DH_2||DH_3)$
$AD = Encode(IK_A^{PK})||Encode(IK_B^{PK})$
Encrypt initial message:
$Ciphertext_{init} = Encrypt[SK_A](Plaintext, AD)$
$IK_A^{PK}, EK_A^{PK}, Ciphertext_{init}$
❷ Requester sends the initial message
Responder received: $IK_A^{PK}, EK_A^{PK}, Ciphertext_{init}$
Responder performs an X3DH key agreement with requester:
$DH_1 = DH(IK_A^{PK}, SPK_B)$
$DH_2 = DH(EK_A^{PK}, IK_B^{SK})$
$DH_3 = DH(EK_A^{PK}, SPK_B)$
$SK_B = KDF(DH_1||DH_2||DH_3)$
$AD = Encode(IK_A^{PK})||Encode(IK_B^{PK})$
Decrypt the initial ciphertext:
$Plaintext = Decrypt[SK_B](Ciphertext_{init}, AD)$
Step (b): Encryption Protocol
Initialize the root key and Diffie-Hellman key pair: $RK_A^{init} = SK_A, (DH_A^{SK}, DH_A^{PK})$
Initialize the root key and Diffie-Hellman key pair: $RK_B^{init} = SK_B, (DH_B^{SK}, DH_B^{PK})$
DH_B^{PK}
❸ Responder sends Diffie-Hellman key
Requester performs the Double Ratchet algorithm and encrypts the message:
$RK_A, CK_A = KDF(RK_A^{init}, DH(DH_A^{SK}, DH_B^{PK}))$
$CK_A, MK_A = KDF(CK_A, Constant)$
$Ciphertext = Encrypt(MK_A, Plaintext, AD)$
$Ciphertext, DH_A^{PK}$
❹ Requester sends ciphertext
Responder performs the Double Ratchet algorithm and decrypts the message:
$RK_B, CK_B = KDF(RK_B^{init}, DH(DH_B^{SK}, DH_A^{PK}))$
$CK_B, MK_B = KDF(CK_B, Constant)$
$Plaintext = Decrypt(MK_B, Ciphertext, AD)$
Responder performs a new DH ratchet step and encrypts new message:
(DH_B^{SK}, DH_B^{PK})
$RK_B, CK_B = KDF(RK_B, DH(DH_B^{SK}, DH_A^{PK}))$
$CK_B, MK_B = KDF(CK_B, Constant)$
$Ciphertext = Encrypt(MK_B, Plaintext, AD)$
$Ciphertext, DH_B^{PK}$
❺ Responder sends ciphertext
Requester performs the Double Ratchet algorithm and decrypts the message:
$RK_A, CK_A = KDF(RK_A, DH(DH_A^{SK}, DH_B^{PK}))$
$CK_A, MK_A = KDF(CK_A, Constant)$
$Plaintext = Decrypt(MK_A, Ciphertext, AD)$
Secure channel
Insecure channel

Fig. 9. Protocol specifics for handshake and encryption in Status.

To verify the authenticity property of the ciphertexts, we introduce a `evResponderReceiveCipherText` event, which is triggered upon successful decryption and validation of the message by the receiver. This event is paired with a corresponding sender-side commitment, allowing ProVerif to check injective agreement and ensure that the ciphertext originated from the expected sender under a fresh session context.

3) Status: The Waku protocol used by Status employs the X3DH [11] and Double Ratchet [12] mechanisms to establish end-to-end encryption, offering robust privacy and security guarantees. It ensures critical security properties such as confidentiality, authentication, and forward secrecy throughout the communication process. The overall protocol workflow, including the handshake and message encryption phases, is illustrated in Fig. 9.

Handshake Phase. In Status Waku, each node periodically broadcasts its pre-key bundle on a content topic derived from its public key []. To initiate communication, a sender derives the recipient's topic from their public key, which is typically exchanged through an out-of-band channel. During the key handshake phase, Status Waku employs the Extended Triple Diffie–Hellman (X3DH) protocol to establish an initial shared secret. The process unfolds as follows: the recipient (responder) publishes a pre-key bundle to the server, comprising the identity public key (`IK`), and a long-term signed pre-key (`SPK`) along with its signature. The sender (requester) retrieves this bundle and performs four Diffie–Hellman computations using their own identity key and the responder's public keys to derive three shared key components:

$$\begin{aligned} DH1 &= DH(IK_requester, SPK_responder), \\ DH2 &= DH(EK_requester, IK_responder), \\ DH3 &= DH(EK_requester, SPK_responder) \end{aligned} \tag{6}$$

And our formalization expresses this mechanism as:

```
let DH1_A = calc_ECDH_Key(skey_IK_A, SPK_B) in
let DH2_A = calc_ECDH_Key(skey_EK_A, pk_IK_B) in
let DH3_A = calc_ECDH_Key(skey_EK_A, SPK_B) in
```

These shared keys are subsequently input into a key derivation function, such as HKDF, to derive the final session key (shared secret) used for securing future encrypted communication. As the Waku protocol does not utilize one-time prekeys [], the sender includes its public identity key within the initialization message, enabling the receiver to authenticate the sender and derive the identical shared secret. In our formal model, the progression of key agreement states is tracked using event annotations, which facilitate the formal verification of both identity agreement (via injective agreement) and the confidentiality of the established session key.

Encryption Phase. Upon completing the X3DH-based key exchange, the Status Waku protocol initiates the Double Ratchet algorithm to enable secure message encryption and continuous key evolution. The Double Ratchet algorithm comprises two fundamental components: a symmetric-key ratchet and an asymmetric (Diffie–Hellman) ratchet. Together, these mechanisms provide strong cryptographic guarantees, including forward secrecy and post-compromise security, by ensuring that each message is encrypted with a unique key and that the compromise of a single key does not affect past or future communications.

In the symmetric-key ratchet, each message encryption key is derived from the current chain key (`CK`) using a key derivation function (`HKDF`). After encryption, the message key (`MK`) is immediately discarded to enforce single-use semantics. The chain key is simultaneously updated with each derivation, thereby forming a forward-evolving key chain as follows:

$$CK_{i+1}, MK_i = HKDF(CK_i) \tag{7}$$

where CK_i denotes the chain key at the i-th step, and MK_i represents the corresponding message key derived from CK_i. Each message is encrypted using MK_i through symmetric encryption, typically implemented via a secretbox primitive such as `secretBoxSeal(plaintext, MK)`. In this study, the key derivation and encryption process is abstracted using the `deriveNextKeys(CK)` function to generate MK_i, followed by `encryptMessage(MK, plaintext)` to represent the actual ciphertext generation. This formalization enables precise tracking of key evolution and supports the verification of critical security properties, including confidentiality and forward secrecy. This is represented as follows:

```
let (RK_1_A: rootKey, CK_0_A: chainKey) = kdfRK(
    init_RK_A, skey_DH_A, pk_DH_B) in
let (CK_1_A: chainKey, MK_1_A: msgKey) = kdfCK(
    CK_0_A, Constant) in
let (encKey_1_A: msgKey, authKey_1_A: authKey, iv
    : nonce) = deriveEncKeys(MK_1_A) in
let cipherTextRequester = ENCRYPT(encKey_1_A,
    plainTextRequester, iv) in
```

In the Diffie–Hellman ratchet mechanism, when either communicating party detects the arrival of a new public key from the other side, referred to as a DH ratchet public key, the protocol initiates an asymmetric ratchet step, which resets the symmetric key chain to ensure forward and backward secrecy. Each ratchet round involves three main steps, specifically, generating a new ephemeral key pair `(sk, pk)`, exchanging the new public keys between parties, and performing a Diffie–Hellman computation using the received public key and the local private key to derive a new shared secret. This secret is then used as input to a key derivation function that updates the root key and initializes a new symmetric-key chain for encrypting subsequent messages.

$$DH_i = DH(SK_{requester}, PK_{responder}) \tag{8}$$

The Diffie–Hellman output serves as the key seed and is combined with the current state as input to the key derivation function, which produces a new root key and a new chain key.

To accurately capture the dynamic updates of key states and the timing of related events, we model the Double Ratchet key process as a state transition system, employing event annotations to record key state changes. And our functions are employed to model the Diffie-Hellman ratchet updates, preserving the assumption of key irreversibility which enables systematic verification of critical security properties.

```
query pk_IK: publicKey, pk_EK: publicKey, pk_DH:
    publicKey, MK: msgKey; event(
    evResponderCalcMessagerKey(MK)) ==> event(
    evRequesterSendDHKey(pk_DH)) || event(
    evRequesterSendKeys(pk_IK, pk_EK)).
```

C. Results and Findings

This section presents a formal analysis of the design and security guarantees of end-to-end encryption protocols employed in decentralized messaging systems, including Matrix Olm, Berty Wesh, and Status X3DH with Double Ratchet. The objective is to identify protocol-level vulnerabilities by evaluating whether these cryptographic protocols uphold key security properties. Table VI summarizes the formal verification outcomes for each decentralized messaging protocol across a range of security properties. In the following subsections, we detail the specific security properties that are violated under different threat models and outline the corresponding attack traces uncovered through our analysis.

1) Matrix: Our approach uncovers multiple vulnerabilities in Matrix's handshake and encryption protocols, including authenticity and confidentiality violations resulting from man-in-the-middle attacks.

Authenticity Violation in Handshake Protocol. Through automated model verification, we identified a critical authenticity violation in the Matrix key distribution protocol, specifically, the event indicating the receipt of Alice's keys (`evReceiveAliceKeys`) and shared secret derivation on Alice's part (`evClientAliceCalcShareSecret`) does not correlate with any genuine key publication (`evClientBobSendKeys`).

We further analyzed the attack trace generated by our proposed formal method, which reveals a potential vulnerability wherein an adversary can compromise the authenticity of the Matrix handshake protocol through a specific sequence of actions. Under the external attacker threat model, the adversary can intercept critical transmission elements, including the user ID, the public identity key, and the one-time prekey, represented as `attacker(userID, pk_I, pk_E)`. Subsequently, when client Bob initiates a request for Alice's key material, the attacker is able to inject this forged $(userID, pk_I, pk_E)$ tuple, misleading Bob into establishing a session under false pretenses. In this attack scenario, the protocol lacks cryptographic mechanisms such as digital signatures or MACs to authenticate and bind the received public keys to their legitimate sender. As a result, an adversary can impersonate the responder by injecting attacker-controlled values for pk_I and pk_E, causing Alice to establish a session key with a malicious party. This allows man-in-the-middle attacks and identity spoofing due to the lack of proper identity binding during the protocol's key confirmation phase.

```
RESULT event(evClientAliceCalcShareSecret(
    SharedSecret)) ==> event(evClientBobSendKeys(
    pk_I,pk_E)) is false.
```

Compromised Ratchet Key in Matrix Olm. The security of end-to-end encrypted communication in Matrix is based on the Olm cryptographic ratchet protocol [8]. Olm uses both one-time keys and long-term keys, and it builds on the triple elliptic curve Diffie Hellman key agreement to establish secure two-party sessions. By generating fresh encryption keys such as `aesKey`, `hmacKey`, and `aesIV`, Olm ensures that messages exchanged between users remain confidential and protected.

To assess key security properties, we formally modeled the Olm ratchet protocol under a server-compromise threat model. The analysis examines message authenticity and resistance to decryption by a compromised homeserver, which can impersonate users and violate confidentiality, authenticity, forward secrecy, and post-compromise security. The attack trace produced by the formal verification model reveals that the event `evClientAliceReceiveRatchetKey` is reached without a corresponding `evClientBobSendRatchetKey` event. This indicates that Alice receives and accepts an unverified ratchet public key from the communication channel, highlighting a lack of authenticity checks in the key update process.

TABLE VI

AN OVERVIEW OF THE FORMAL VERIFICATION RESULTS OF THE DECENTRALIZED MESSAGING NETWORK SECURITY PROTOCOL UNDER DIFFERENT SECURITY PROPERTIES. ✗ INDICATES THAT A SECURITY PROPERTY HAS BEEN COMPROMISED, ✓ INDICATES NOT.

#	DMN Security Protocols	Security Properties				Attacks
		Authenticity	Confidentiality	PFS	PCS	
1	Matrix Olm Cryptographic Ratchet	✗	✓	✗	✗	Impersonation & MitM Attack
2	Berty Wesh Protocol	✗	✓	✗	✗	Impersonation & MitM Attack
3	Status Waku X3DH & Double Ratchet	✗	✓	✗	✗	Impersonation & MitM Attack

During the key setup phase, Alice, acting as the requester, generates both an identity key pair and a temporary key pair `(skey_A_I, skey_A_E)`, and transmits their corresponding public keys `(pk(skey_A_I), pk(skey_A_E))` over a public channel. As the responder, Alice receives a pair `(pk_I, pk_E)`, which she assumes to be Bob's identity and ephemeral public keys. Then she derives the shared secret using the following computation method:

```
let SharedSecret_AB = concat3(ecdhkey_AB_1,
    ecdhkey_AB_2, ecdhkey_AB_3) in
```

During the ratchet key and ciphertext generation process, Alice generates a ratchet public key and encrypts the message using an AES key and initialization vector (IV), both derived via the HKDF key derivation function. She then transmits the ratchet public key along with the ciphertext. The vulnerability arises during the reception phase, where Alice accepts a received ratchet public key `ratchetKey` from the communication channel without proper verification. Our formal model validates the existence of this flaw through the following analysis. And the `evClientBobSendRatchetKey(ratchetKey)` event indicates that the ratchet key `ratchetKey` received by the client may have been injected by an adversary. The protocol does not incorporate cryptographic binding mechanisms such as digital signatures or message authentication codes to authenticate the ratchet key, which allows an attacker to impersonate a legitimate peer and introduce a forged ratchet key into the session.

```
RESULT event(evClientAliceReceiveRatchetKey(.))
==> event(evClientBobSendRatchetKey(.)) is false.
```

2) Berty: The formal verification model developed in this study reveals critical authenticity vulnerabilities in the Berty Wesh protocol, affecting both the generation of shared keys during the handshake phase and the encryption of messages in subsequent communications.

Authentication Violation in Key Confirmation. In the Wesh protocol, the responder computes the shared secret S without verifying that the requester has correctly provided the corresponding keybox. This represents a failure in authentication, allowing an attacker to craft a message that deceives the responder into accepting it as originating from a legitimate requester. Consequently, the responder derives the shared key based on forged input. Specifically, the attacker can fabricate the keybox, leading to the execution of the event `evResponderAuthenticateCalcSharedKey` without a matching `evRequesterSendboxKey` event. In particular, the proposed model is designed to verify the authenticity property:

```
event(evResponderAuthenticateCalcSharedKey(k))
    ==> event(evRequesterSendboxKey(k))
```

This property asserts that the responder should compute the authenticated shared key S only if a corresponding box key has been sent by the requester. However, the formal verification model produced a counterexample trace, demonstrating that this condition can be violated.

Specifically, the attacker injects a crafted ciphertext, which the responder successfully decrypts using two previously established shared secrets, `sca(skey_B_E, pk)` and `sca(skey_B_I, pk)`. The resulting plaintext is interpreted as the requester's identity public key. Based on this interpretation, the responder proceeds to compute the final shared key:

$$sca(skey\_B\_I, unboxSeal(...)) \tag{9}$$

where sca denotes the function used to compute the shared secret. This confirms that the protocol lacks adequate authentication binding between the contents of the keybox and the sender's identity. As a result, an adversary can exploit this weakness to impersonate a legitimate requester, derive a shared secret with the responder, and potentially launch impersonation or man-in-the-middle attacks.

```
RESULT event(evResponderAuthenticateCalcSharedKey
    (SharedSecret_AB)) ==> event(
    evRequesterSendboxKey(boxKey)) is false.
```

Validation Vulnerabilities in Secure Messaging. Our formal verification model also identified a critical authenticity flaw in the message exchange phase of the Wesh protocol, namely, the requester accepts a ciphertext without a corresponding legitimate transmission from the responder. This section presents a detailed attack trace where the adversary exploits knowledge of the responder's identity public key `PK_B_I` and manipulates the message derivation sequence to compute a valid shared secret using `sca(skey_A_I, PK_B_I)`. This enables the attacker to forge a ciphertext that successfully passes decryption and integrity verification on the requester's side. Notably, the `secretBoxSeal` function is executed with a message key derived from a compromised or replayed chain key, allowing ciphertext creation without requiring the responder to send the corresponding plaintext.

Specifically, the adversary intercepts or replays the `boxSeal` message containing the responder's public key and its associated signature. Using this forged key exchange, the attacker can derive the shared secret $sca(skey_A^I, PK_B^I)$. Subsequently, a ciphertext is generated via `secretBoxSeal` and transmitted to the requester. Upon receipt, the requester decrypts the message using the derived message key, which triggers the event `evRequesterReceiveCipherText`. This attack trace demonstrates that the requester accepts a message that was never genuinely transmitted by the responder, thereby compromising the protocol's authenticity guarantee.

```
RESULT event(evRequesterReceiveCipherText(.)) ==>
     event(evResponderSendCipherText(.)) is false
```

3) Status: Status Waku employs the X3DH and Double Ratchet protocols to provide end-to-end encryption. However, its X3DH implementation omits one-time prekeys and exhibits authentication vulnerabilities, notably allowing attackers to inject forged key material due to the absence of cryptographic binding between identity and ephemeral keys during the handshake. Consequently, responders may be deceived into deriving shared keys with malicious actors, enabling man-in-the-middle attacks that compromise both authenticity and confidentiality within the defined threat model.

Authenticity Violation in X3DH Handshake. In the formal verification of Status Waku's X3DH handshake phase, the responder derives the session key from public key inputs received via an unverified channel, without validating their legitimate origin from the initiator. This absence of cryptographic confirmation enables adversaries to inject forged key material, resulting in the following security violation:

```
RESULT event(evResponderCalcSharedKey(SK)) ==>
event(evRequesterSendKeys(pk_IK,pk_EK)) is false.
```

The proposed attack trace reveals a critical vulnerability whereby an adversary can inject malicious key material, causing the responder to derive a session key under the incorrect assumption that it originates from a legitimate peer. This behavior compromises the authentication guarantees of the key agreement protocol and enables potential identity spoofing or man-in-the-middle (MitM) attacks. Concretely, the responder completes the computation of the session key using input values `(sk, pk, P)` obtained over an unverified channel, without confirming their provenance through a corresponding `evRequesterSendKeys` event. Specifically, the responder first correctly initializes its identity key `skey_IK_B` and signed prekey `sk_SPK_B`, publishing both to the public channel alongside a valid signature. However, due to the absence of a prior verification step, namely, the observation of a corresponding `evRequesterSendKeys` event, an adversary can exploit this gap by injecting forged values for (sk, pk, P). Due to insufficient authentication checks, the responder accepts attacker-controlled inputs and proceeds with the Diffie–Hellman key derivation. As a result, the responder derives a session key under the erroneous assumption that it was established with a legitimate peer, thereby compromising the protocol's authenticity guarantees.

Breaking Encryption Guarantees in Waku. Our formal model further reveals that the attack trace allows the responder to derive a message key without obtaining valid key material from the legitimate sender, thereby violating the protocol's authentication guarantees. Initially, the responder broadcasts its signed prekey and identity key to the network. An adversary then injects a forged ciphertext accompanied by manipulated Diffie–Hellman public keys and associated metadata to impersonate a legitimate peer. As a result, the responder computes the session key using:

$$KDF(DH(sk_{SPK_B}, pk_{IK_A}), DH(sk_{IK_B}, pk_{IK_A}), DH(sk_{SPK_B}, pk_{EK_A})) \tag{10}$$

Furthermore, the attacker is able to decrypt the injected ciphertext using AEAD, leveraging attacker-controlled associated data. Notably, this occurs without the triggering of `event(evRequesterSendDHKey)` or `event(evRequesterSendKeys)`, which are intended to mark the receipt of authenticated inputs from a legitimate sender. In this scenario, the responder derives a valid session and message key based entirely on attacker-supplied inputs, without performing any identity verification. This vulnerability enables an attacker to impersonate a legitimate sender or launch a man-in-the-middle attack, thereby violating the protocol's guarantees of confidentiality and post-compromise security.

```
RESULT event(evResponderCalcMessagerKey(MK)) ==>
     event(evRequesterSendDHKey(pk_DH)) || event(
     evRequesterSendKeys(pk_IK,pk_EK)) is false.
```

Takeaway #2: The handshake protocol of the DMN is susceptible to man-in-the-middle attacks, particularly during the key request phase. Additionally, the encryption protocol is also vulnerable to impersonation attacks in scenarios involving compromised nodes or master servers. In such cases, an attacker can substitute the victim's identity key and covertly establish communication with other parties, thereby compromising authenticity, PFS, and PCS.

The complete formal verification code and corresponding attack traces are available in our GitHub repository[13].

VI. Lessons Learned

A. Security Insights Across the Ecosystems

Our empirical analysis revealed that a significant number of nodes across different decentralized messaging networks were exploited or misused for malicious activities in Section IV,

[13]https://github.com/lihaoSDU/DecentralizedMesagers/tree/main/DMCatcher/ProVerif

such as spam dissemination and C2 (Command and Control) communication. These abuses often stem from permissive node configurations, weak identity vetting, and a lack of shared reputation or abuse feedback mechanisms. To mitigate such threats, we propose the following technical countermeasures.

Avoidance of Abused Infrastructure. DMN nodes should proactively avoid deploying services on IP addresses, IP ranges, or DNS subdomains with a known history of abuse (e.g., flagged by public blacklists like AbuseIPDB or VirusTotal). This can be achieved by integrating abuse-check APIs during node initialization or deployment, and periodically revalidating the infrastructure's reputation. Additionally, node bootstrap lists (e.g., for peer discovery) should exclude IPs/-subdomains with poor reputational standing.

Reputation System Enhancement. While decentralization reduces reliance on centralized trust authorities, networks can still implement decentralized reputation mechanisms, as demonstrated by the Storj reputation system [], to identify and mitigate malicious or low-quality nodes. Such mechanisms can use distributed overlay networks, such as gossip-based protocols, to collect and disseminate cryptographically signed feedback on metrics including uptime, message integrity, and abuse reports. Reputation scores can guide routing and peer selection to favor more trustworthy nodes. Additionally, a privacy-preserving reputation layer can enable anonymous abuse reporting. Nodes accumulating negative feedback may be locally deprioritized, rate limited, or excluded by peers.

B. Remediation of Protocol-Level Threats

To address the protocol level vulnerabilities identified in decentralized messaging systems, as described in Section V, we suggest the following categories of protocol level remediation strategies. These vulnerabilities include cases where attackers can hijack session keys during the handshake phase due to weak identity binding, which compromises core security properties.

Implement Verifiable Key Distribution Mechanisms. In protocols such as Matrix and Berty, public keys or prekeys are typically distributed through servers or relay nodes without global verifiability, introducing opportunities for key substitution attacks. To address this, we suggest the adoption of tamper-resistant key publication mechanisms. One approach is to leverage blockchain or Decentralized Identifiers (DIDs) [] to store identity-bound public keys, thereby preventing undetected key replacement. This allows verifiers to ensure consistency between identity claims and blockchain-published keys, effectively mitigating man-in-the-middle threats.

Establish a Formal Verification and Feedback Loop. Most real-world deployments of decentralized messaging systems lack rigorous formal validation of their underlying cryptographic protocols. To strengthen protocol security, we recommend the integration of automated formal verification tools, such as ProVerif [] and Tamarin [], into the protocol design and development lifecycle. Each new protocol version should undergo formal validation prior to deployment, establishing a continuous feedback loop between design and verification.

C. Limitations and Future Work

Although one-to-one messaging can be regarded as a subset of group messaging, this study deliberately focuses exclusively on the former, omitting group communication scenarios. For instance, formal modeling of the Megolm protocol [] for group end-to-end encryption is excluded from Section V-B1 as our analysis centers on Olm, the foundational protocol of Megolm. Furthermore, this paper does not address multi-device scenarios, in which a single user operates across multiple devices. These limitations highlight important avenues for future research, including the formal verification of group encryption protocols such as Megolm, and a comprehensive examination of multi-device protocol behavior to more accurately reflect real-world security threats.

VII. Related Work

Decentralized Messaging Networks Analysis. Several studies [1], [13], [14], [36] on decentralized messaging networks have explored various dimensions, including network characteristics, operational ecosystems, and application-layer protocol security. For instance, [1] proposed a forensic approach to dissecting the Riot.im (Element [37]) application and the Matrix protocol to fill the knowledge gap in forensics and analysis for the Matrix community of investigators. Hao et al. [13] have not only conducted theoretical analyses but have also implemented a comprehensive study. They explore the Matrix ecosystem from various viewpoints, specifically focusing on the landscape, network threats, and implementation vulnerabilities. Martin et al. [14], [16] report several exploitable cryptographic vulnerabilities in the Matrix protocol used for federated communication and its flagship client Element and prototype implementations. Besides, Kee et al. [38] highlighted that the Session application integrates privacy preserving technologies such as the Session protocol, onion routing, and decentralized message storage to deliver a secure messaging experience with robust encryption while significantly reducing metadata leakage. Song et al. [39] analyze Briar's cryptographic design and protocol security. This report provides a detailed account of Briar's cryptographic mechanisms and protocol stack. Prior studies have explored decentralized messaging networks from various perspectives, however, their limited scope highlights the absence of a comprehensive and systematic analysis encompassing both ecosystem characteristics and security protocol evaluation.

Formal Analysis Tools. In the panorama of prior academic investigations, a range of sophisticated tools, including Proverif [17], [40], Tamarin [41]–[43], Deepsec [44], and others [45], [46]. For instance, [34] presents a ProVerif-based formal model of Bluetooth protocols covering key sharing and data transmission, enabling automated detection of design flaws. [44] introduces the DEEPSEC verification tool, providing new complexity results for static equivalence, trace equivalence, and marked similarity. The paper also offers a decision-making procedure for these equivalences in scenarios with a limited number of sessions. Furthermore, in Papers [42] and [43], Tamarin Prover was used to analyze Signal's Sesame protocol, revealing cases where Post-Compromise Security (PCS) can still be broken despite using the double-ratchet protocol.

We implemented ProVerif primarily because of our familiarity with its capabilities and its ability to generate clear and interpretable outputs, which facilitate the analysis and understanding of protocol behavior and potential security vulnerabilities.

VIII. Conclusions

This study presents the first systematic measurement and automated formal verification of the ecosystem and security protocols in decentralized messaging networks. For various decentralized messaging platforms, we propose D-MAP, a comprehensive data acquisition framework, along with a formal verification methodology for analyzing protocol security. Our findings show that despite global distribution, infrastructure centralization remains due to limited use of decentralized hosting. Moreover, over 4.4% of node IPs exhibit malicious behavior, including vulnerabilities such as XSS and DoS that span software supply chains. Complementing this, our formal verification analysis identifies critical protocol-level vulnerabilities, including potential threats such as man-in-the-middle and impersonation attacks.

References

[1] G. C. Schipper, R. Seelt, and N. Le-Khac, "Forensic analysis of Matrix protocol and riot.im application," *Digit. Investig.*, vol. 36 Supplement, p. 301118, 2021.

[2] G. Korpal and D. Scott, "Decentralization and web3 technologies," 2022. [Online]. Available: https://www.techrxiv.org/articles/preprint/Decentralization_and_web3_technologies/19727734

[3] Matrix.org, "An open network for secure, decentralized communication." https://matrix.org/, 2023.

[4] Element.io, "Customers," https://element.io/customers, 2024.

[5] S. faire Linux, "Share, freely and privately," 2025. [Online]. Available: https://jami.net/

[6] S. R. . D. GmbH, "Make the jump to web3," 2025. [Online]. Available: https://status.app/

[7] B. Technologies, "Unstoppable p2p communication," 2025. [Online]. Available: https://berty.tech/

[8] Matrix.org, "Olm: A cryptographic ratchet," https://gitlab.matrix.org/matrix-org/olm/-/blob/master/docs/olm.md, 2019.

[9] ——, "Megolm group ratchet," https://gitlab.matrix.org/matrix-org/olm/-/blob/master/docs/megolm.md, 2022.

[10] B. Technologies, "Wesh protocol," 2023. [Online]. Available: https://berty.tech/docs/protocol

[11] M. Marlinspike and T. Perrin, "The x3dh key agreement protocol," https://signal.org/docs/specifications/x3dh/, 2016.

[12] T. Perrin and M. Marlinspike, "The Double Ratchet Algorithm," https://signal.org/docs/specifications/doubleratchet/, 2016.

[13] H. Li, Y. Wu, R. Huang, X. Mi, C. HU, and S. Guo, "Demystifying decentralized matrix communication network: Ecosystem and security," in *29th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2023, Haikou, China, December 17-21, 2023*. IEEE, 2023.

[14] M. R. Albrecht, S. Celi, B. Dowling, and D. Jones, "Practically-exploitable cryptographic vulnerabilities in matrix," Cryptology ePrint Archive, Paper 2023/485, 2023, https://eprint.iacr.org/2023/485. [Online]. Available: https://eprint.iacr.org/2023/485

[15] M. Yeung, "Sendingnetwork: Advancing the future of decentralized messaging networks," *arXiv preprint arXiv:2401.09102*, 2024.

[16] M. R. Albrecht, B. Dowling, and D. Jones, "Device-oriented group messaging: a formal cryptographic analysis of matrix'core," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 2666–1685.

[17] V. C. Bruno Blanchet, "ProVerif: Cryptographic protocol verifier in the formal model," https://bblanche.gitlabpages.inria.fr/proverif/, 2023.

[18] Status, "Waku is Uncompromising Peer-to-Peer Communication at Scale," https://waku.org/, 2025.

[19] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "Ed25519: high-speed high-security signatures," https://ed25519.cr.yp.to/, Access in 2023.

[20] D. J. Bernstein, "A state-of-the-art Diffie-Hellman function," https://cr.yp.to/ecdh.html, Access in 2023.

[21] P. Labs, "The peer-to-peer network stack," https://libp2p.io/, 2025.

[22] ethereum.org, "Node Discovery Protocol v5," https://github.com/ethereum/devp2p/blob/master/discv5/discv5.md, 2020.

[23] Matrix.org, "Key Distribution," https://spec.matrix.org/v1.13/client-server-api/#key-distribution, 2024.

[24] ——, "Short Authentication String (SAS) verification," https://spec.matrix.org/v1.9/client-server-api/#short-authentication-string-sas-verification, 2024.

[25] P. Zimmermann, "ZRTP: Media Path Key Agreement for Unicast Secure RTP," https://philzimmermann.com/docs/zrtp/ietf/rfc6189.html, 2011.

[26] D. Tarr, E. Lavoie, A. Meyer, and C. Tschudin, "Secure scuttlebutt: An identity-centric protocol for subjective and decentralized applications," in *Proceedings of the 6th ACM conference on information-centric networking*, 2019, pp. 1–11.

[27] D. Van Dam, "Analysing the signal protocol," *A manual and automated analysis of the Signal Protocol*, 2019.

[28] C. Jacomme, E. Klein, S. Kremer, and M. Racouchot, "A comprehensive, formal and automated analysis of the {EDHOC} protocol," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 5881–5898.

[29] ipinfo, "The trusted source for ip address data." https://ipinfo.io/, 2025.

[30] Shodan.io, "Search engine for the internet of everything," https://www.shodan.io/, 2025.

[31] VirusTotal, https://developers.virustotal.com/reference/overview, 2025.

[32] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983.

[33] T.-Y. Wu, L. Wang, X. Guo, Y.-C. Chen, and S.-C. Chu, "Sakap: Sgx-based authentication key agreement protocol in iot-enabled cloud computing," *Sustainability*, vol. 14, no. 17, p. 11054, 2022.

[34] J. Wu, R. Wu, D. Xu, D. J. Tian, and A. Bianchi, "Formal model-driven discovery of bluetooth protocol design vulnerabilities," in *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 2285–2303. [Online]. Available: https://doi.org/10.1109/SP46214.2022.9833777

[35] Matrix.org, "Matrix Specification," https://spec.matrix.org/v1.13/, 2024.

[36] F. Jacob, J. Grashöfer, and H. Hartenstein, "A glimpse of the Matrix (extended version): Scalability issues of a new message-oriented data synchronization middleware," *CoRR*, vol. abs/1910.06295, 2019. [Online]. Available: http://arxiv.org/abs/1910.06295

[37] Element.io, https://element.io/, 2023.

[38] K. Jefferys, M. Shishmarev, and S. Harman, "Session: A model for end-to-end encrypted conversations with minimal metadata leakage," *arXiv preprint arXiv:2002.04609*, p. 4, 2020.

[39] Y. Song, "Cryptography in the wild: Briar," 2023.

[40] B. Blanchet, V. Cheval, and V. Cortier, "Proverif with lemmas, induction, fast subsumption, and much more," in *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 69–86. [Online]. Available: https://doi.org/10.1109/SP46214.2022.9833653

[41] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols," in *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25*. Springer, 2013, pp. 696–701.

[42] C. Cremers, J. Fairoze, B. Kiesl, and A. Naska, "Clone detection in secure messaging: Improving post-compromise security in practice," in *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM, 2020, pp. 1481–1495. [Online]. Available: https://doi.org/10.1145/3372297.3423354

[43] C. Cremers, C. Jacomme, and A. Naska, "Formal analysis of Session-Handling in secure messaging: Lifting security from sessions to conversations," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 1235–1252. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/cremers-session-handling

[44] V. Cheval, S. Kremer, and I. Rakotonirina, "DEEPSEC: deciding equivalence properties in security protocols theory and practice," in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 2018, pp. 529–546. [Online]. Available: https://doi.org/10.1109/SP.2018.00033

[45] B. Blanchet, "Cryptoverif: Computationally sound mechanized prover for cryptographic protocols," in *Dagstuhl seminar "Formal Protocol Verification Applied*, vol. 117, 2007, p. 156.

[46] K. Bhargavan, A. Bichhawat, Q. H. Do, P. Hosseyni, R. Küsters, G. Schmitz, and T. Würtele, "Dy*: A modular symbolic verification framework for executable cryptographic protocol code," in *IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*. IEEE, 2021, pp. 523–542. [Online]. Available: https://doi.org/10.1109/EuroSP51992.2021.00042