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.
Email: 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, D-MAP, that combines large-scale data collection with formal security verification, enabling the abstraction, modeling, and analysis of security protocols across diverse DMN platforms. Leveraging D-MAP, we perform empirical measurements and protocol-level verification across several representative DMNs (namely, Matrix, Berty, Jami, and Status). Our analysis uncovers a range of security issues spanning both network infrastructure and protocol layers, including DMNs node IP address exposure (4.4% of 78K DMN nodes), software supply chain weaknesses, and cryptographic flaws in handshake and encryption protocol that permit man-in-the-middle and impersonation attacks.**

I. Introduction

Traditional centralized messaging platforms (e.g., Facebook, WhatsApp, etc.) typically control information collaboration and data sovereignty centrally on a centralized server or platform [45]. 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 Web3.0 [19], decentralized messaging protocols are becoming more and more popular [26], 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 [35], a leading protocol in this space, reports over 80 million registered accounts [16]. Jami [18], a peer-to-peer communication platform with end-to-end encryption and perfect forward secrecy, and Status [20] combines a decentralized messaging app with a Web3.0 wallet, offering metadata protection and private chats secured by end-to-end encryption. Berty [51] is a privacy-focused messaging application built on the Wesh protocol [50], 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, for example, Matrix implements a custom cryptographic ratchet protocol [33], [34] to provide session-level message security. Berty introduces the Wesh protocol [50], enabling secure and asynchronous

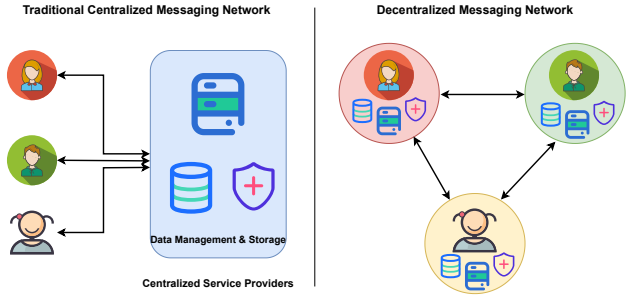

Fig. 1. The architecture between centralized and decentralized messaging network.

communication even in offline scenarios, while Status adopts the well-established X3DH key agreement [32] and Double Ratchet [42] algorithms to protect user communications.

Motivation. Decentralized protocols introduce unique attack surfaces and architectural complexities, yet existing research often prioritizes functionality and scalability over thorough security and formal verification [2], [31]. This work is motivated by the need to bridge that gap and systematically assess the security of decentralized messaging networks (DMNs), focusing specifically on:

- To the best of our knowledge, there is a lack of comprehensive large-scale measurement studies on decentralized messaging networks, as well as a shortage of automated formal verification and rigorous security analysis of their cryptographic protocols.
- In terms of ecosystem measurement of DMNs, existing studies seldom examine critical aspects such as network composition, geographic and infrastructure distribution, metadata leakage, and adversarial behavior [31], [56].
- In security protocol verification, handshake and encryption 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 security [2], [3].

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, and

protocol interaction data. In security protocols verification, D-MAP introduces an automated formal verification method using ProVerif [9], constructing modular protocol models that include key subprotocols such as handshake, key agreement, and message encryption in Matrix, Berty, and Status. By automatically generating potential attack traces, the proposed method can accurately identify vulnerabilities such as authentication flaws and key leakage risks.

On one hand, our analysis reveals that while the decentralized messaging ecosystem is large and globally distributed (78,000 nodes span 150+ countries), its underlying infrastructure remains partially centralized. Additionally, over 4.4% of DMN nodes exhibit malicious behavior, and a substantial number are affected by critical vulnerabilities, underscoring the significant impact of software supply chain risks on 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 an in-depth discussion of these results, refer to Sec. V.

Contributions. The main contributions about this research are summarized as follows:

- This study proposes D-MAP, an automated framework that integrates distributed data collection with formal protocol verification to conduct the first large-scale measurement and formal security analysis of the leading decentralized messaging networks.
- Our measurements reveal that decentralized messaging networks are globally distributed yet still show signs of infrastructural centralization, including dependence on a limited number of dominant servers or organizations. Furthermore, over 4.4% of nodes exhibit malicious behavior, such as blacklisting and active exploitation.
- Our formal verification uncovered critical protocol vulnerabilities, including man-in-the-middle and impersonation attacks that compromise authenticity, forward secrecy, and post-compromise security. We responsibly disclosed these findings to the respective DMN teams, and Matrix acknowledged a message leak caused by flaws in initial trust establishment, while the others did not provide a response.

II. Background

Prominent DMNs include Matrix [35], which employs a robust cryptographic federated architecture, Berty [51], supporting offline peer-to-peer communication, Status [20] which integrates messaging with crypto wallets using the Waku protocol [48], and Jami [18], enabling serverless real-time communication by distributing data and control across multiple nodes. Table I provides an overview of these networks.

TABLE I
AN OVERVIEW ACROSS DMNS.

System	Decentralized Mechanisms	Security Protocols	Measurement[1]	Protocol Analysis[2]
Matrix [35]	Homeserver	Olm, Megolm	✓	✓
Berty [51]	libp2p	Wesh Protocol	✓	✓
Status [20]	Waku	X3DH, Double Ratchet	✓	✓
Jami [18]	OpenDHT	TLS v1.3	✓	✗

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

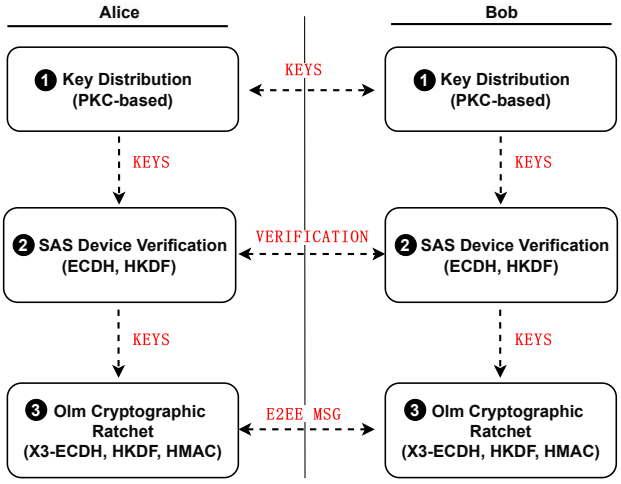

Fig. 2. Overview of the Matrix Security Protocols.

A. 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) [36], 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 [38] (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 [33] (❸ of the Fig. 2).

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

Message Encryption Protocol. Matrix has also introduced a cryptographic ratchet protocol based on Olm [33], and 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 (`RK`) and message encryption key (`MK`) are derived from this shared key, ensuring the security of end-to-end encryption. For a detailed explanation of the Olm protocol procedures, refer to Section V-B1. On the flip side, the Megolm cryptographic ratchet protocol [34], built on top of Olm [2], 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 in one-to-one communication.

2) Berty: Berty Messenger is a privacy-focused messaging application built atop the Wesh protocol [50], a custom-designed security framework for decentralized, end-to-end encrypted communication. The protocol draws from Scuttlebutt's handshake [49] and Signal's symmetric-key ratchet [42] 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.

Handshake Protocol. Berty's handshake protocol takes inspiration from Scuttlebutt's capability-based handshake [49], 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 [50].

Message Encryption Protocol. In the Wesh protocol, all communications are secured through end-to-end encryption based on a symmetric-key ratchet mechanism [42]. 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 [48], a modular suite of protocols designed to support secure, censorship-resistant, and anonymous peer-to-peer communication. Waku builds upon the Whisper protocol [52] 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 [32]. Status uses an elliptic-curve Diffie-Hellman (ECDH) scheme [24] 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 [42] 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 [18] 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[1]. After the handshake, media streams are encrypted using the Secure Real-time Transport Protocol[2], 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.

B. Research Questions in this Study

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 DMN nodes? Focuses on the structural and geographical distribution of nodes, their roles within the network topology, and associations with higher-level applications.

RQ2: What security risks and vulnerabilities exist across different DMN nodes? Evaluates node-level security, including exposure to malicious traffic, poor IP reputation, and dependence on centralized infrastructure.

RQ3: How to systematically apply threat modeling and formal verification to DMN security protocols? Involves modeling protocol behavior, defining adversarial assumptions, identifying attack surfaces, and applying formal methods to verify properties like authenticity and secrecy.

RQ4: What protocol-level vulnerabilities exist, and how can they be systematically identified and mitigated? Addresses

[1]https://docs.jami.net/en_US/user/faq.html#advanced

[2]https://datatracker.ietf.org/doc/html/rfc3711

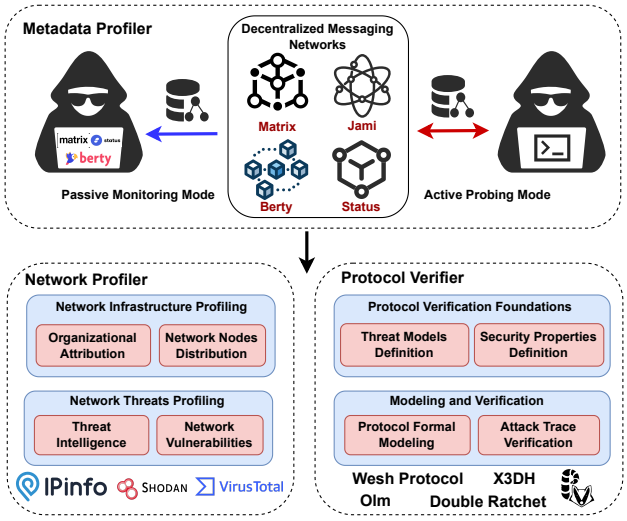

Fig. 3. The proposed D-MAP framework.

flaws in the design and implementation of end-to-end encryption protocols, uncovering cryptographic weaknesses and proposing mitigation and improvement strategies.

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

III. Proposed Method Design

A. D-MAP

In this section, we introduce D-MAP (Decentralized Messaging Analysis Platform), a measurement and security analysis framework illustrated in Fig. 3. D-MAP can automatically capture communication traces from real-world decentralized messaging systems, enabling systematic analysis to uncover protocol-level inconsistencies, potential vulnerabilities, and deviations from expected security guarantees.

1) Metadata & Network Profiler: Studying decentralized messaging networks in terms of node distribution, scale, and behavior is challenging. Privacy-preserving designs limit peer visibility, and protocol diversity across Matrix, libp2p, and Waku hinders unified analysis. To address this, D-MAP integrates a Network Profiler that automatically collects and analyzes node-level data from real-world deployments. We adopt a multi-dimensional data acquisition strategy to systematically capture key DMN characteristics, including:

Passive Monitoring Mode. We deploy multiple measurement nodes globally that join target networks as regular peers without actively initiating interactions. Instead, they passively observe network and protocol events such as broadcasts, synchronization, and handshakes. This approach allows us to capture data on peer churn, routing and relay paths, and gossip-based message propagation.

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. D-MAP 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[3] 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 chat groups. These nodes can recursively query neighboring peers or expose status interfaces, enabling hierarchical traversal (Breadth First Search[4]) and comprehensive discovery of the network topology. We formalized as,

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

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[5]) to discover and collect data on additional nodes. Furthermore, in networks that employ the Rendezvous protocol [50], as used in Berty, measurement nodes can adopt temporary identities to participate in peer discovery, enabling 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.

In D-MAP, we conduct a comprehensive analysis of the underlying security protocols and introduce a formal verification method called the Protocol Verifier. As shown in Fig. 6, the Protocol Verifier supports the definition of multiple threat models, such as external server compromise and forced access, and enables automated verification of key security properties, including authenticity, perfect forward secrecy, backward secrecy, and post-compromise security. It abstracts and formalizes each target protocol, including Matrix Olm, Berty Wesh, and Jami implementations of X3DH and the Double Ratchet, by modeling detailed interactions among participants such as key generation, key agreement, and message encryption. Then we uses ProVerif [9] to enable 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,

[3]https://github.com/element-hq/synapse
[4]https://en.wikipedia.org/wiki/Breadth-first_search
[5]https://spec.matrix.org/v1.13/client-server-api/

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).
```

Detailed specifications of the proposed threat models and verification method are provided in Section V.

B. Data Overview

We conducted a seven-month data collection campaign using D-MAP from January 3 to July 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 [22], Shodan [46], and VirusTotal [53], 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. A thorough presentation of the extended dataset and its analysis can be found in Section IV. Furthermore, the protocol verification module systematically extracts and 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.

The proposed D-MAP framework together with the complete implementation of the Network and Protocol Profilers and the accompanying datasets, has been released as open source at https://github.com/dsec-lab/DecentralizedMesagers.

C. Ethics Considerations

D-MAP's Network Profiler and Protocol Verifier operate on self-hosted nodes, ensuring controlled resource usage without affecting other participants or exposing sensitive information such as authentication data. All protocol vulnerability assessments were conducted in a controlled environment to avoid any risk to real users. Accordingly, our data collection and security analysis adhere to established ethical standards.

IV. Measurement Study on DMNs

To address RQ1 and RQ2, this section presents a large-scale empirical study of decentralized messaging networks. We analyze node distribution at both physical and network levels, and systematically assess security risks and vulnerabilities across different types of nodes.

A. Network Infrastructure Profiling

In this section, we profile the network infrastructures of major DMNs to reveal their global deployment, hosting environments, and associated organizations. The analysis covers key attributes such as geographic distribution, AS affiliations, cloud versus residential hosting, and network topology.

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	13158	5093	512	17
Europe	21274	7848	1676	207
Asia-Pacific	1201	3409	420	121
Oceania	574	139	208	0
South America	142	149	97	1
Africa	39	335	52	2

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[6]. In addition to TCP services, we identified a substantial number of exposed UDP based ports across decentralized messaging

[6]https://docs.libp2p.io/concepts/nat/autonat/

TABLE III
COUNTRY-WISE DISTRIBUTIONS OF TOP-10.

# Matrix	# Berty	# Jami	# Status
(USA, 12068)	(USA, 4778)	(USA, 341)	(Germany, 156)
(Germany, 10404)	(Russia, 2879)	(Russia, 256)	(Indonesia, 64)
(France, 2488)	(Germany, 1999)	(France, 255)	(Hong Kong, 30)
(Netherlands, 1724)	(UK, 855)	(Germany, 249)	(USA, 17)
(UK, 1149)	(China, 797)	(Australia, 186)	(Netherlands, 17)
(Finland, 1091)	(South Korea, 641)	(Canada, 133)	(Russia, 10)
(Canada, 1057)	(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)

TABLE IV
OPEN PORTS ACROSS DIVERSE DEPLOYMENT PLATFORMS.

Index	# Matrix	# Berty	# Jami	# Status
1	(80, 17315)	(22, 6733)	(443, 103)	(22, 182)
2	(443, 16682)	(80, 2060)	(80, 82)	(3000, 176)
3	(22, 9402)	(4001, 1940)	(8443, 41)	(8080, 41)
4	(8080, 4414)	(443, 1276)	(22, 38)	(8545, 32)
5	(8443, 4233)	(5443, 870)	(7547, 22)	(53, 30)
6	(2087, 3886)	(5080, 868)	(8080, 20)	(30303, 29)
7	(8880, 3823)	(8080, 699)	(7443, 19)	(80, 28)
8	(2083, 3807)	(5001, 347)	(8089, 19)	(1234, 25)
9	(2082, 3787)	(7547, 210)	(4000, 19)	(3001, 20)
10	(2086, 3763)	(9000, 170)	(1701, 17)	(443, 15)

network nodes. For example, port 30303, which is commonly used by Ethereum and Waku [48] 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, 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 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[7]), 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

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

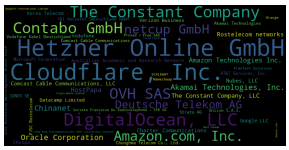

Fig. 4. ISP-based word cloud of DMN nodes.

Fig. 5. Tag-based word cloud of DMN nodes.

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 [53] and Shodan [46], 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[8]), 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.

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[9] and Golem[10], have yet to achieve widespread adoption.

B. Network Threats Profiling

To address RQ2, we conducted a large-scale measurement study using data from VirusTotal [53] and Shodan [46] to identify vulnerabilities and security risks associated with node IP assets across 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.

[8]https://nordvpn.com/cybersec-site/
[9]https://akash.network/
[10]https://www.golem.network/

TABLE V
TOP-10 CVES ACROSS DMNS.

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

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 Berty network, 209.*.*.9, was flagged by 18 different threat detection engines. To understand malicious activity in DMNs, we manually analyzed 3,400 flagged IPs and found many acting as command-and-control nodes for malware such as Mirai and Venom RAT, 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. 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.

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 [41] 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 [40], 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.

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 RQ3 and RQ4, we propose a formal verification methodology based on ProVerif [9] to analyze the handshake and end-to-end encryption protocols of Matrix, Berty, and Status, thereby enabling evaluation cryptographic guarantees and the detection of potential design 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 [14] model.

As shown in Fig. 6, we present the "Server Security Compromised" model considers scenarios where the server is compromised or behaves maliciously. Attackers may gain unauthorized access and manipulate user key information through substitution or alteration. The "External Security Compromised" model considers threats to communication links, where adversaries may intercept, tamper with, or eavesdrop on sensitive data during transmission. The "Compulsory Access" model involves brief endpoint compromise, either fully (e.g., device seizure) or partially (e.g., malicious apps). Adversaries can gain access through methods such as border inspections, brute-force attacks, or malware.

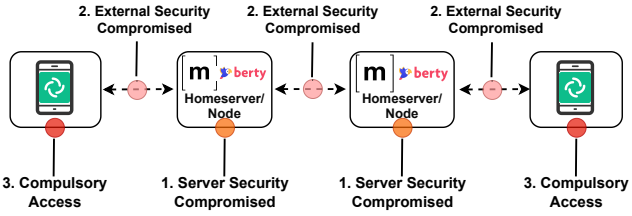

Fig. 6. Threat models of the DM security protocols.

Besides, 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

To tackle the formal verification challenges of decentralized messaging protocols with multiple components and participants, we propose a modular approach that enables flexible threat modeling and systematic evaluation of security properties across various protocols (Matrix Olm in Sec. V-B1, Berty Wesh in Sec. V-B2, and Status security protocols in our full version [29]). The complete formal verification code and corresponding results are available in GitHub repository[11].

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, D-MAP develops and implements a formal model using ProVerif [9], which has been instrumental in advancing research across various domains, including IoT security [54], [55].

Handshake Phase. In the handshake phase, we extract the protocol logic based on the Matrix specification [37] 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

[11] https://github.com/dsec-lab/DecentralizedMesagers/tree/main/DMCatcher/ProVerif

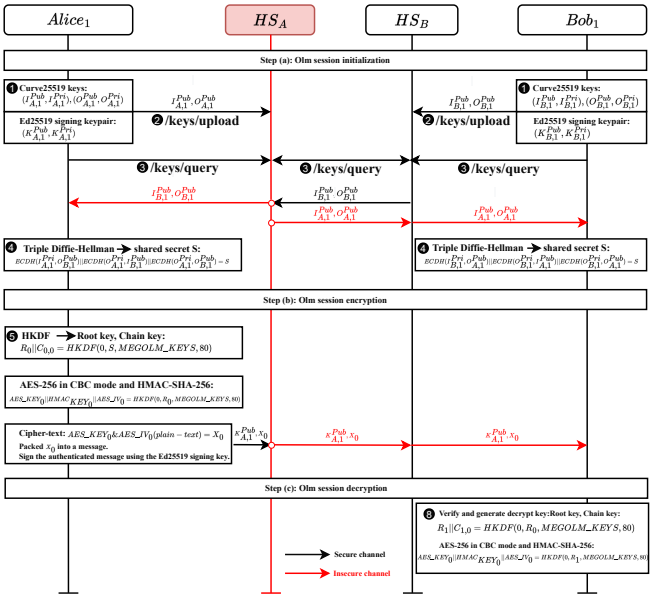

Fig. 7. Protocol specifics of handshake, encryption, and decryption within the Matrix Olm ratchet.

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 identity and one-time key pairs, and denoted as `pk_A_I`, `pk_A_O`, `pk_B_I`, and `pk_B_O`. Our primary objective is to rigorously assess whether the key distribution protocol adheres to the previously defined security properties. Specifically,

```
// Authenticity in Handshake Phase
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 [21], 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 [27].

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, `get_ecdhkey_1`, `get_ecdhkey_2`, and `get_ecdhkey_3`.

Besides, drawing from the negotiated shared key, the subsequent step involves the HKDF 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.

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

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

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 [50] 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, and the complete protocol flow is depicted in Fig. 8.

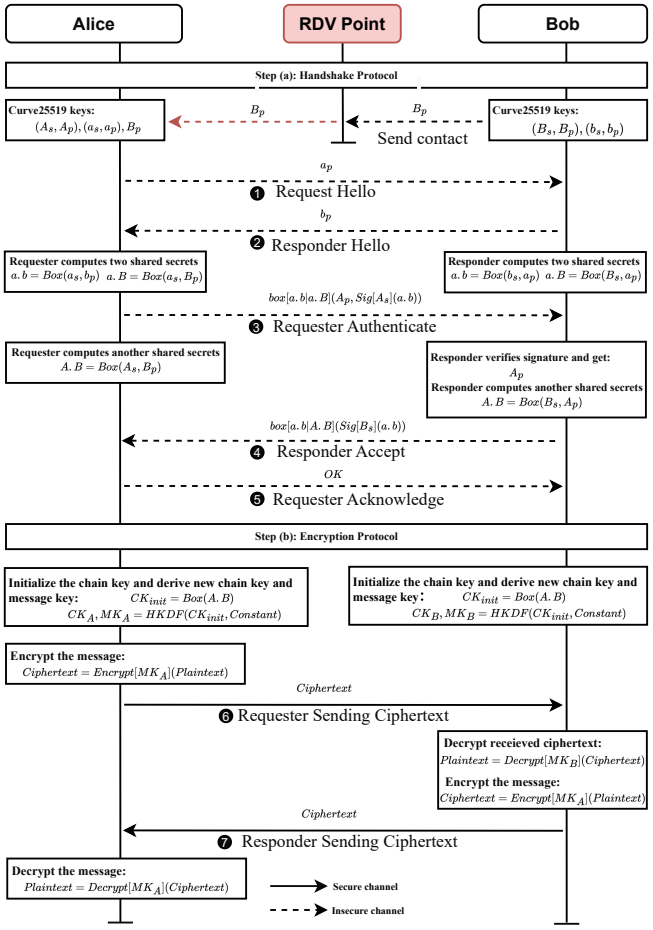

Fig. 8. The handshake and end-to-end encryption protocols within the Berty.

Modeling Handshake Phase. We first model the Berty handshake phase, which is inspired by Scuttlebutt's capability-based handshake mechanism [49]. This sub-protocol begins with the requester initiating the process by sending a `Hello` message to the responder (Step ❶ in Fig. 8). 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 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, `a.b` and `a.B`.

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, namely, ephemeral key ex-

change, 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` events.

The responder transmits a sealed message (Step ❹), 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
```

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 [42]. 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, D-MAP constructs a dedicated Encryption Phase module in our Protocol Verifier. 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 (CK_Requester:chainKey, MK:msgKey) =
    deriveKeys(init_CK_Requester,Constant) in
let cipherTextRequester = secretBoxSeal(
    plainTextRequester, Constant, MK) in
```

To verify the authenticity of the ciphertexts (Steps ❻ and ❼), we introduce a `evReceiveCipherText` 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.

C. Results and Findings

Table VI summarizes the formal verification results for each decentralized messaging protocol across various security properties. The following subsections detail the specific properties violated under different threat models and present the corresponding attack traces identified through our analysis, specifically, Matrix Olm and Berty Wesh in Sec. V-C2 and Sec. V-C1, and Status in [29].

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 security compromised 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.
```

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

Compromised Ratchet Key in Matrix Olm. The security of end-to-end encrypted communication in Matrix is based on the Olm cryptographic ratchet protocol [33]. Olm uses both one-time and identity keys, and 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. 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 compromised shared secret.

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 `evResponderAuthCalcSharedKey` without a matching `evRequesterSendboxKey` event. In particular, the proposed model is designed to verify the authenticity property. 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, which the responder then uses to compute the final shared key:

$$S = sca(skey\_B\_I, unboxSeal(...)) \tag{6}$$

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(evResponderAuthCalcSharedKey(SK))
==> event(evRequesterSendboxKey(BK)) 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
```

Takeaway #2: The DMN handshake protocol is vulnerable to man-in-the-middle attacks during the key request phase, while the encryption protocol risks impersonation under node or server compromise. An attacker can replace the victim's identity key to secretly establish sessions, undermining authenticity, perfect forward secrecy, and post-compromise security.

VI. Related Work

Decentralized Messaging Networks Analysis. Several studies [2], [23], [31], [45] on decentralized messaging networks have explored various dimensions, including network characteristics, operational ecosystems, and application-layer protocol security. For instance, [45] proposed a forensic approach to dissecting the Riot.im (Element [15]) application and the Matrix protocol to fill the knowledge gap in forensics and analysis for the Matrix community of investigators. Hao et al. [31] conducted not only theoretical analyses but also a comprehensive empirical study. They examined the Matrix ecosystem from multiple perspectives, with a particular focus on the network landscape, security threats, and implementation-level vulnerabilities. Martin et al. [2], [3] reported several exploitable cryptographic vulnerabilities in the Matrix protocol used for federated communication and its flagship client Element and prototype implementations. Besides, Kee et al. [25] 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. [47] analyzed Briar's cryptographic design and protocol security. This report provides a detailed account of Briar's cryptographic mechanisms and protocol stack. While prior studies have examined decentralized messaging networks from different perspectives, they often lack a comprehensive and systematic analysis that thoroughly integrates both ecosystem characteristics and security protocol evaluation.

Formal Analysis Tools. In the panorama of prior academic investigations, a range of sophisticated tools, including Proverif [8], [9], Tamarin [12], [13], [39], Deepsec [10], and others [6], [7]. As an illustration, [54] presents a ProVerif-based formal model of Bluetooth protocols covering key sharing and data transmission, enabling automated detection of design flaws. [10] introduces the DEEPSEC verification tool, providing new complexity results for static equivalence, trace equivalence, and marked similarity. This work further presents a decision-making procedure for determining these equivalences in scenarios involving a limited number of sessions. Furthermore, in [12] and [13] research works, Tamarin Prover was used to analyze Signal's Sesame protocol, revealing cases where post-compromise security 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.

VII. Conclusions

This study presents the first systematic measurement and automated formal verification of the ecosystem and security protocols in decentralized messaging networks. For various DMNs, 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. We responsibly disclosed the vulnerabilities to the DMN teams. Matrix acknowledged a message leakage resulting from flaws in the initial trust establishment process, whereas the others did not provide a response.

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 [34] 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.

REFERENCES

[1] AbuseIPDB. Abuseipdb, making the internet safer, one ip at a time. https://www.abuseipdb.com/, 2025.

[2] Martin R Albrecht, Sofía Celi, Benjamin Dowling, and Daniel Jones. Practically-exploitable cryptographic vulnerabilities in matrix. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 164–181. IEEE, 2023.

[3] Martin R Albrecht, Benjamin Dowling, and Daniel Jones. Device-oriented group messaging: a formal cryptographic analysis of matrix'core. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 2666–1685. IEEE, 2024.

[4] D. J. Bernstein. A state-of-the-art Diffie-Hellman function. https://cr.yp.to/ecdh.html, Access in 2025.

[5] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. Ed25519: high-speed high-security signatures. https://ed25519.cr.yp.to/, Access in 2023.

[6] Karthikeyan Bhargavan, Abhishek Bichhawat, Quoc Huy Do, Pedram Hosseyni, Ralf Küsters, Guido Schmitz, and Tim 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*, pages 523–542. IEEE, 2021.

[7] Bruno Blanchet. Cryptoverif: Computationally sound mechanized prover for cryptographic protocols. In *Dagstuhl seminar "Formal Protocol Verification Applied*, volume 117, page 156, 2007.

[8] Bruno Blanchet, Vincent Cheval, and Véronique 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*, pages 69–86. IEEE, 2022.

[9] Vincent Cheval Bruno Blanchet. ProVerif: Cryptographic protocol verifier in the formal model. https://bblanche.gitlabpages.inria.fr/proverif/, Access in 2025.

[10] Vincent Cheval, Steve Kremer, and Itsaka 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*, pages 529–546. IEEE Computer Society, 2018.

[11] Corey Petty, Oskar Thorén, and Samuel Hawksby-Robinson. X3dh prekey bundle creation. https://specs.status.im/spec/2#x3dh-prekey-bundle-creation, 2025.

[12] Cas Cremers, Jaiden Fairoze, Benjamin Kiesl, and Aurora Naska. Clone detection in secure messaging: Improving post-compromise security in practice. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS'20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1481–1495. ACM, 2020.

[13] Cas Cremers, Charlie Jacomme, and Aurora Naska. Formal analysis of Session-Handling in secure messaging: Lifting security from sessions to conversations. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1235–1252, Anaheim, CA, August 2023. USENIX Association.

[14] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

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

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

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

[18] Savoir faire Linux. Share, freely and privately. https://jami.net/, 2025.

[19] Wensheng Gan, Zhenqiang Ye, Shicheng Wan, and Philip S Yu. Web 3.0: The future of internet. In *Companion Proceedings of the ACM Web Conference 2023*, pages 1266–1275, 2023.

[20] Status Research & Development GmbH. Make the jump to web3. https://status.app/, 2025.

[21] Rakel Haakegaard and Joanna Lang. The elliptic curve diffie-hellman (ecdh). *Online at https://koclab. cs. ucsb. edu/teaching/ecc/project/2015Projects/Haakegaard+ Lang. pdf*, 2015.

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

[23] Florian Jacob, Jan Grashöfer, and Hannes Hartenstein. A glimpse of the Matrix (extended version): Scalability issues of a new message-oriented data synchronization middleware. *CoRR*, abs/1910.06295, 2019.

[24] Charlie Jacomme, Elise Klein, Steve Kremer, and Maïwenn Racouchot. A comprehensive, formal and automated analysis of the {EDHOC} protocol. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5881–5898, 2023.

[25] Kee Jefferys, Maxim Shishmarev, and Simon Harman. Session: A model for end-to-end encrypted conversations with minimal metadata leakage. *arXiv preprint arXiv:2002.04609*, page 4, 2020.

[26] Gaurish Korpal and Drew Scott. Decentralization and web3 technologies. https://www.techrxiv.org/articles/preprint/Decentralization_and_web3_technologies/19727734, 2022.

[27] Hugo Krawczyk. Cryptographic extraction and key derivation: The hkdf scheme. In *Annual Cryptology Conference*, pages 631–648. Springer, 2010.

[28] Protocol Labs. The peer-to-peer network stack. https://libp2p.io/, 2025.

[29] Hao Li. Decentralized But Not Immune: Empirical and Formal Security Analysis of Messaging Networks. https://github.com/dsec-lab/DecentralizedMesagers, Access in 2025.

[30] Hao Li, Xianghang Mi, Yanzhi Dou, and Shanqing Guo. An empirical study of storj dcs: Ecosystem, performance, and security. In *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2023.

[31] Hao Li, Yanbo Wu, Ronghong Huang, Xianghang Mi, Chengyu HU, and Shanqing 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.

[32] Moxie Marlinspike and Trevor Perrin. The x3dh key agreement protocol. https://signal.org/docs/specifications/x3dh/, 2016.

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

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

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

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

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

[38] Matrix.org. Short Authentication String (SAS) verification. https://spec.matrix.org/v1.13/client-server-api/#short-authentication-string-sas-verification, 2024.

[39] Simon Meier, Benedikt Schmidt, Cas Cremers, and David 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*, pages 696–701. Springer, 2013.

[40] MITRE. CWE-400: Uncontrolled Resource Consumption. https://cwe.mitre.org/data/definitions/400.html, 2025.

[41] MITRE. CWE-79: Improper Neutralization of Input During Web Page Generation. https://cwe.mitre.org/data/definitions/79.html, 2025.

[42] Trevor Perrin and Moxie Marlinspike. The Double Ratchet Algorithm. https://signal.org/docs/specifications/doubleratchet/, 2016.

[43] Drummond Reed, Manu Sporny, Dave Longley, Christopher Allen, Ryan Grant, Markus Sabadello, and Jonathan Holt. Decentralized identifiers (dids) v1. 0. *Draft Community Group Report*, 2020.

[44] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 98–107, 2002.

[45] Guido Cornelis Schipper, Rudy Seelt, and Nhien-An Le-Khac. Forensic analysis of Matrix protocol and riot.im application. *Digit. Investig.*, 36 Supplement:301118, 2021.

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

[47] Yuanming Song. Cryptography in the wild: Briar. 2023.

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

[49] Dominic Tarr, Erick Lavoie, Aljoscha Meyer, and Christian Tschudin. Secure scuttlebutt: An identity-centric protocol for subjective and decentralized applications. In *Proceedings of the 6th ACM conference on information-centric networking*, pages 1–11, 2019.
[50] Berty Technologies. Wesh protocol. https://berty.tech/docs/protocol, 2023.
[51] Berty Technologies. Unstoppable p2p communication. https://berty.tech/, 2025.
[52] Dion Van Dam. Analysing the signal protocol. *A manual and automated analysis of the Signal Protocol*, 2019.
[53] VirusTotal. https://developers.virustotal.com/reference/overview, 2025.
[54] Jianliang Wu, Ruoyu Wu, Dongyan Xu, Dave Jing Tian, and Antonio 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*, pages 2285–2303. IEEE, 2022.
[55] Tsu-Yang Wu, Liyang Wang, Xinglan Guo, Yeh-Cheng Chen, and Shu-Chuan Chu. Sakap: Sgx-based authentication key agreement protocol in iot-enabled cloud computing. *Sustainability*, 14(17):11054, 2022.
[56] Mason Yeung. Sendingnetwork: Advancing the future of decentralized messaging networks. *arXiv preprint arXiv:2401.09102*, 2024.

VIII. Appendix

A. Overview of the DMNs

In this supplement section, we will further 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 [37]. Besides, for privacy-minded customers, Matrix rooms also support communication with E2EE based on the Olm cryptographic ratchets protocol [34], [42]. Every registered user in Matrix has a unique "user ID" 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 [5] fingerprint key pair for signing messages, a Curve25519 [4] 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 [33] sessions.

2) Berty Messager: Berty [51] is fundamentally grounded in a peer-to-peer (P2P) communication model built atop the libp2p [28] networking stack. 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 [50], 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.
- 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 [20] is an open-source decentralized app that combines private messaging with a self-custodial crypto wallet, removing the need for centralized intermediaries. Within the Status network, every "Status Node" operates as a peer, though with varying capabilities. This foundational layer constitutes a public, permissionless network, powered by the devp2p protocol[12], which supports decentralized peer discovery, connectivity, and message routing.

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. 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 [17], 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 [18] is a 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[13], a distributed in-memory key-value store based on the Kademlia[14] 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.

B. Experimental Setup

We deploy D-MAP with a hybrid architecture combining remote and local infrastructure. Remote Alibaba Cloud servers[15] in regions including Singapore, the United States, and Europe Germany 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 in D-MAP is equipped with dual Intel Xeon 4410Y processors, each running at 2.0 GHz with 12 cores

[12]https://github.com/ethereum/devp2p
[13]https://github.com/savoirfairelinux/opendht
[14]https://en.wikipedia.org/wiki/Kademlia
[15]https://www.alibabacloud.com/en

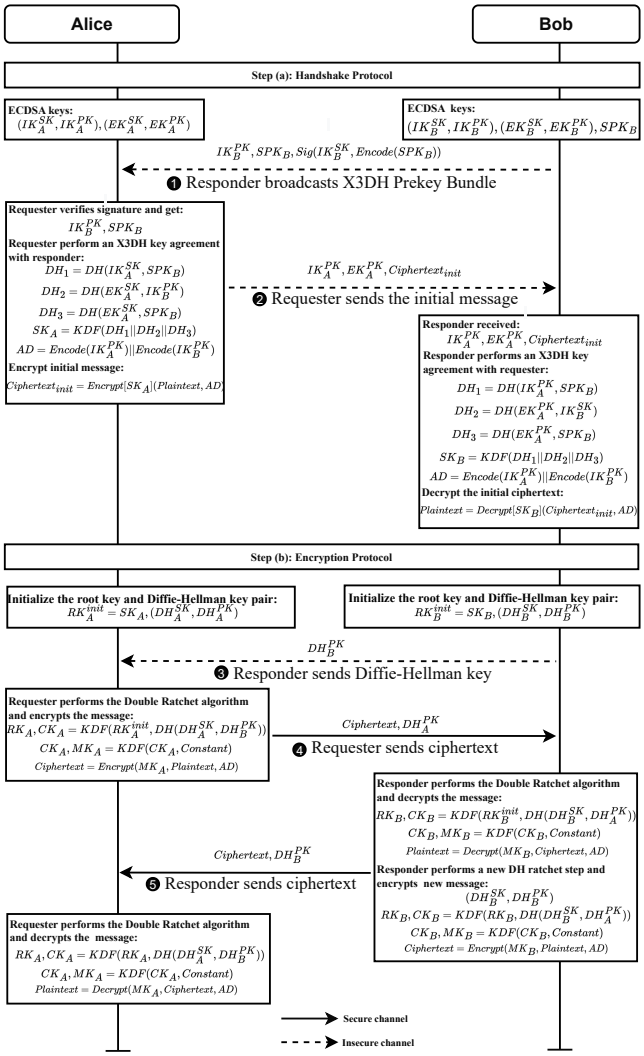

Fig. 9. Protocol specifics for handshake and encryption in Status Waku.

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 8 GB of memory. These distributed network profilers are strategically located across different regions to enhance data acquisition coverage of decentralized messaging networks.

C. Supplement to Formal Verification

1) Formal Modeling of Status: The Waku protocol used by Status employs the X3DH [32] and Double Ratchet [42] 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[16]. To initiate communication (Step ❶ and ❷ in Fig. 9), 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 [32] to establish an initial shared secret. The process unfolds as follows, namely, the recipient (responder) publishes a pre-key bundle to the server, which includes their identity public key (`IK`), a long-term signed pre-key (`SPK`), and the corresponding signature. The sender (requester) retrieves this bundle and performs three Diffie–Hellman computations using their own identity key and the responder's public keys to derive three shared key components:

$$\begin{aligned} DH1 &= DH(IK\_requester, SPK\_responder), \\ DH2 &= DH(EK\_requester, IK\_responder), \\ DH3 &= DH(EK\_requester, SPK\_responder) \end{aligned} \tag{7}$$

This mechanism is formalized in our model as follows:

```
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 [11], 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 (`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 [42] 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. 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:

$$CK_{i+1}, MK_i = HKDF(CK_i) \tag{8}$$

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

[16]https://specs.status.im/spec/2#x3dh-prekey-bundles

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.

$$DH_i = DH(SK_{requester}, PK_{responder}) \tag{9}$$

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 capture key state updates and event timing, we model the Double Ratchet as a state transition system with annotated events. Our functions represent DH ratchet updates while preserving key irreversibility, enabling systematic verification of security properties.

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

2) Formal Verification of Status: Status Waku uses X3DH and Double Ratchet protocols for end-to-end encryption, but its X3DH omits one-time prekeys and lacks cryptographic binding between identity and ephemeral keys. This allows attackers to inject forged keys, leading to man-in-the-middle attacks that break authenticity and perfect forward secrecy.

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(IK,EK)) is false.
```

The attack trace reveals a vulnerability where an adversary injects forged key material, tricking the responder into deriving a session key under false assumptions. This compromises authentication and enables identity spoofing or MitM attacks, as the responder uses unverified inputs `(sk, pk, P)` without a matching `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 inadequate authentication checks, the responder accepts attacker-controlled inputs and performs the Diffie–Hellman key derivation, ultimately generating a session key under the false assumption of communicating with a legitimate peer. This behavior undermines the protocol's authenticity guarantees.

Breaking Encryption Guarantees in Waku. Our formal model shows that the responder can derive a message key without receiving valid key material from the legitimate sender, violating authentication guarantees. The attacker exploits this by injecting forged ciphertext and manipulated Diffie–Hellman keys to impersonate a peer. As a result, the responder computes the session key using:

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

Furthermore, the attacker is able to decrypt the injected ciphertext using authenticated encryption with associated data (AEAD) [44], leveraging attacker-controlled associated data. Notably, this occurs without the triggering of `evRequesterSendDHKey` or `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 authenticity and post-compromise security.

```
RESULT event(evResponderCalcMessagerKey(MK))
==> event(evRequesterSendDHKey(DH)) ||
event(evRequesterSendKeys(IK,EK)) is false.
```

D. Lessons Learned

1) 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 (Measurement Study on Decentralized Messaging Networks), such as spam dissemination and C2 (Command and Control) communication. These abuses often result from permissive node configurations, weak identity vetting, and absence of reputation or

abuse feedback mechanisms, which we address through the following technical countermeasures.

Avoidance of Abused Infrastructure. Decentralized messaging network 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 [1] or VirusTotal [53]). 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 removes reliance on centralized trust authorities, networks can adopt decentralized reputation systems such as Storj's [30] to identify and mitigate malicious or unreliable nodes. Gossip-based overlays can disseminate cryptographically signed feedback on metrics such as uptime, message integrity, and abuse. Reputation scores can inform peer selection, and privacy-preserving layers may support anonymous reporting. Nodes with poor reputations can be deprioritized, rate limited, or excluded by peers.

2) Remediation of Protocol-Level Threats: To address the protocol level vulnerabilities identified in decentralized messaging systems, as described in Section V, we propose the following categories of protocol-level remediation strategies to address vulnerabilities where weak identity binding allows attackers to hijack session keys during the handshake, compromising core security guarantees.

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) [43] (`did:method:identifier`) 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. Despite the widespread deployment of decentralized messaging systems, their underlying cryptographic protocols often lack rigorous formal validation. To enhance the security assurances of such systems, we advocate for the systematic integration of automated formal verification tools, such as ProVerif [9] and Tamarin [39], into the protocol design and development lifecycle. Each protocol revision should undergo formal analysis prior to release, thereby establishing a continuous feedback loop that iteratively refines protocol specifications through formal verification.