SoK: Secure Messaging[1]

Nik Unger*, Sergej Dechand†
Joseph Bonneau‡§, Sascha Fahl¶, Henning Perl¶
Ian Goldberg*, Matthew Smith†
* University of Waterloo, † University of Bonn, ‡ Stanford University, § Electronic Frontier Foundation, ¶ Fraunhofer FKIE

***Abstract*—Motivated by recent revelations of widespread state surveillance of personal communication, many solutions now claim to offer secure and private messaging. This includes both a large number of new projects and many widely adopted tools that have added security features. The intense pressure in the past two years to deliver solutions quickly has resulted in varying threat models, incomplete objectives, dubious security claims, and a lack of broad perspective on the existing cryptographic literature on secure communication.**

In this paper, we evaluate and systematize current secure messaging solutions and propose an evaluation framework for their security, usability, and ease-of-adoption properties. We consider solutions from academia, but also identify innovative and promising approaches used "in-the-wild" that are not considered by the academic literature. We identify three key challenges and map the design landscape for each: trust establishment, conversation security, and transport privacy. Trust establishment approaches offering strong security and privacy features perform poorly from a usability and adoption perspective, whereas some hybrid approaches that have not been well studied in the academic literature might provide better trade-offs in practice. In contrast, once trust is established, conversation security can be achieved without any user involvement in most two-party conversations, though conversations between larger groups still lack a good solution. Finally, transport privacy appears to be the most difficult problem to solve without paying significant performance penalties.

I. Introduction

Most popular messaging tools used on the Internet do not offer end-to-end security. Even though protocols such as OpenPGP and S/MIME have been available for decades, they have failed to achieve widespread adoption and have been plagued by usability issues [2]–[5]. However, recent revelations about mass surveillance by intelligence services have highlighted the lack of security and privacy in messaging tools and spurred demand for better solutions. A recent Pew Research poll found that 80% of Americans are now concerned about government monitoring of their electronic communications. 68% of respondents reported feeling "not very secure" or "not at all secure" when using online chat and 57% felt similarly insecure using email [6]. Consequently, many new applications claiming to offer secure communication are being developed and adopted by end users.

Despite the publication of a large number of secure messaging protocols in the academic literature, tools are being released with new designs that fail to draw upon this knowledge, repeat known design mistakes, or use cryptography in insecure ways. However, as will become clear over the course of this paper, the academic research community is also failing to learn some lessons from tools in the wild.

Furthermore, there is a lack of coherent vision for the future of secure messaging. Most solutions focus on specific issues and have different goals and threat models. This is compounded by differing security vocabularies and the absence of a unified evaluation of prior work. Outside of academia, many products mislead users by advertising with grandiose claims of "military grade encryption" or by promising impossible features such as self-destructing messages [7]–[10]. The recent EFF Secure Messaging Scorecard evaluated tools for basic indicators of security and project health [11] and found many purportedly "secure" tools do not even attempt end-to-end encryption.

We are motivated to systematize knowledge on secure messaging due to the lack of a clear winner in the race for widespread deployment and the persistence of many lingering unsolved research problems. Our primary goal is to identify where problems lie and create a guide for the research community to help move forward on this important topic. A further goal in this work is to establish evaluation criteria for measuring security features of messaging systems, as well as their usability and adoption implications. We aim to provide a broad perspective on secure messaging and its challenges, as well as a comparative evaluation of existing approaches, in order to provide context that informs future efforts. Our primary contributions are: (1) establishing a set of common security and privacy feature definitions for secure messaging; (2) systematization of secure messaging approaches based both on academic work and "in-the-wild" projects; (3) comparative evaluation of these approaches; and (4) identification and discussion of current research challenges, indicating future research directions.

After defining terminology in Section II, we present our systematization methodology in Section III. In subsequent sections (Sections IV–VI), we evaluate each of the proposed problem areas (namely *trust establishment*, *conversation security* and *transport privacy*) in secure messaging. Our findings are discussed and concluded in Section VII.

II. Background and Definitions

Secure messaging systems vary widely in their goals and corresponding design decisions. Additionally, their target audiences often influence how they are defined. In this section, we define terminology to differentiate these designs and provide

[1]This is an extended version of our paper appearing in the 2015 IEEE Symposium on Security and Privacy [1].

a foundation for our discussion of secure messaging.

A. Types of specification

Secure messaging systems can be specified at three different broad levels of abstraction:

Chat protocols: At the most abstract level, chat protocols can be defined as sequence of values exchanged between participants. This mode of specification deals with high-level data flows and often omits details as significant as the choice of cryptographic protocols (e.g., key exchanges) to use. Academic publications typically specify protocols this way.

Wire protocols: Complete wire protocols aim to specify a binary-level representation of message formats. A wire protocol should be complete enough that multiple parties can implement it separately and interoperate successfully. Often these are specific enough that they have versions to ensure compatibility as changes are made. Implicitly, a wire protocol implements some higher-level chat protocol, though extracting it may be non-trivial.

Tools: Tools are concrete software implementations that can be used for secure messaging. Implicitly, a tool contains a wire protocol, though it may be difficult and error-prone to derive it, even from an open-source tool.

B. Synchronicity

A chat protocol can be *synchronous* or *asynchronous*. Synchronous protocols require all participants to be online and connected at the same time in order for messages to be transmitted. Systems with a peer-to-peer architecture, where the sender directly connects to the recipient for message transmission, are examples of synchronous protocols. Asynchronous protocols, such as SMS (text messaging) or email, do not require participants to be online when messages are sent, utilizing a third party to cache messages for later delivery.

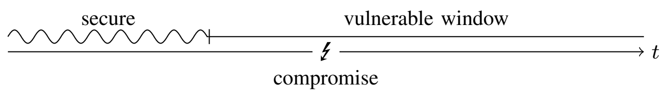
Due to social and technical constraints, such as switched-off devices, limited reception, and limited battery life, synchronous protocols are not feasible for many users. Mobile environments are also particularly prone to various transmission errors and network interruptions that preclude the use of synchronous protocols. Most popular instant messaging (IM) solutions today provide asynchronicity in these environments by using a *store-and-forward* model: a central server is used to buffer messages when the recipient is offline. Secure messaging protocols designed for these environments need to consider, and possibly extend, this store-and-forward model.

C. Deniability

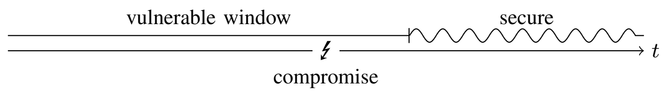
Deniability, also called *repudiability*, is a common goal for secure messaging systems. Consider a scenario where Bob accuses Alice of sending a specific message. Justin, a judge, must decide whether or not he believes that Alice actually did so. If Bob can provide evidence that Alice sent that message, such as a valid cryptographic signature of the message under Alice's long-term key, then we say that the action is *non-repudiable*. Otherwise, the action is *repudiable* or *deniable*. We can distinguish between *message repudiation*, in which Alice denies sending a specific message, and *participation repudiation* in which Alice denies communicating with Bob at all. The high-level goal of repudiable messaging systems is to achieve deniability similar to real-world conversations.

A fundamental problem of deniability is that Justin may simply trust Bob even with no technical evidence due to Bob's reputation or perceived indifference. In a group chat, this problem may be even worse as Alice may need to convince Justin that a number of accusers are all colluding to frame her. It is not possible to construct a messaging system that overcomes this fundamental social problem; the best that can be done is to provide no stronger evidence than the word of the accusers. Some technical systems clearly offer more evidence; for example, signed PGP emails offer strong evidence that Alice really was the sender.

The cryptographic literature has produced many definitions of "deniability" since deniable encryption was first formally proposed [12]. For example, we can draw a distinction between an offline and online judge: in the offline case, the accuser attempts to convince the judge of an event after the conversation has already concluded; in the online case, the judge exchanges private communications with the accuser while the conversation is still taking place. Existing work defines online repudiation in incompatible ways, and very few protocols attempt to achieve meaningful online repudiation [13], [14]. Thus, in this work we only consider the offline setting.



(a) Forward Secrecy



(b) Backward Secrecy

Fig. 1. Session keys are protected from long-term key compromise.

D. Forward/Backward Secrecy

In systems that use the same static keys for all messages, a key compromise allows an attacker to decrypt the entire message exchange. A protocol provides *forward secrecy* if the compromise of a long-term key does not allow ciphertexts encrypted with previous session keys to be decrypted (Figure 1a). If the compromise of a long-term key does not allow subsequent ciphertexts to be decrypted by passive attackers, then the protocol is said to have *backward secrecy* (Figure 1b). However, tools with *backward secrecy* are still vulnerable to active attackers that have compromised long-term keys. In this context, the "self-healing" aspect of *backward secrecy* has also been called *future secrecy*. The terms are controversial and vague in the literature [15]–[17].

III. Systematization Methodology

Over the years, hundreds of secure messaging systems have been proposed and developed in both academia and industry. An exhaustive analysis of all solutions is both infeasible and undesirable. Instead, we extract recurring secure messaging techniques from the literature and publicly available messaging tools, focusing on systematization and evaluation of the underlying concepts and the desirable secure messaging properties. In this section, we explain our precise methodology.

A. Problem Areas

While most secure messaging solutions try to deal with all possible security aspects, in our systematization, we divide secure messaging into three nearly orthogonal problem areas addressed in dedicated sections: the *trust establishment* problem (Section IV), ensuring the distribution of cryptographic long-term keys and proof of association with the owning entity; the *conversation security* problem (Section V), ensuring the protection of exchanged messages during conversations; and the *transport privacy* problem (Section VI), hiding the communication metadata.

While any concrete tool must decide on an approach for each problem area, abstractly defined protocols may only address some of them. Additionally, the distinction between these three problem areas is sometimes blurred since techniques used by secure messaging systems may be part of their approach for multiple problem areas.

B. Threat Model

When evaluating the security and privacy properties in secure messaging, we must consider a variety of adversaries. Our threat model includes the following attackers:

Local Adversary (active/passive): An attacker controlling local networks (e.g., owners of open wireless access points).

Global Adversary (active/passive): An attacker controlling large segments of the Internet, such as powerful nation states or large internet service providers.

Service providers: For messaging systems that require centralized infrastructure (e.g., public-key directories), the service operators should be considered as potential adversaries.

Note that our adversary classes are not necessarily exclusive. In some cases, adversaries of different types might collude. We also assume that all adversaries are participants in the messaging system, allowing them to start conversations, send messages, or perform other normal participant actions. We assume that the endpoints in a secure messaging system are secure (i.e., malware and hardware attacks are out of scope).

C. Systematization Structure

Sections IV–VI evaluate *trust establishment*, *conversation security*, and *transport privacy* approaches, respectively. For each problem area, we identify desirable properties divided into three main groups: *security and privacy features*, *usability features*, and *adoption considerations*. Each section starts by defining these properties, followed by the extraction of generic approaches used to address the problem area from existing secure messaging systems. Each section then defines and evaluates these approaches, as well as several possible variations, in terms of the already-defined properties. Concrete examples of protocols or tools making use of each approach are given whenever possible. The sections then conclude by discussing the implications of these evaluations.

In each section, we include a table (Tables I, II, and III) visualizing our evaluation of approaches within that problem area. Columns in the tables represent the identified properties, while rows represent the approaches. Groups of rows begin with a generic concept, specified as a combination of cryptographic protocols, followed by extension rows that add or modify components of the base concept. Whenever possible, rows include the name of a representative protocol or tool that uses the combination of concepts. Representatives may not achieve all of the features that are possible using the approach; they are merely included to indicate where approaches are used in practice. Each row is rated as providing or not providing the desired properties. In some cases, a row might only partially provide a property, which is explained in the associated description.

For each problem area, we identify desirable properties in three main categories:

1) Security and Privacy Properties: Most secure messaging systems are designed using standard cryptographic primitives such as hash functions, symmetric encryption ciphers, and digital signature schemes. When evaluating the security and privacy features of a scheme, we assume cryptographic primitives are securely chosen and correctly implemented. We do not attempt to audit for software exploits which may compromise users' security. However, if systems allow end users to misuse these cryptographic primitives, the scheme is penalized.

2) Usability Properties: Usability is crucial for the use and adoption of secure messaging services. Human end users need to understand how to use the system securely and the effort required to do so must be acceptable for the perceived benefits.

In previous research, various secure messaging tools have been evaluated and weaknesses in the HCI portion of their design have been revealed. The seminal paper "Why Johnny Can't Encrypt" [2] along with follow-up studies evaluating PGP tools [3], [4] and other messaging protocols [18]–[22] have also showed users encountering sever problems using encryption securely. However, these studies focused on UI issues unique to specific implementations. This approach results in few generic insights regarding secure messenger protocol and application design. Given the huge number of secure messaging implementations and academic approaches considered in our systematization, we opted to extract generic concepts. Because we focus on usability consequences imposed by generic concepts, our results hold for any tool that implements these concepts.

To evaluate the usability of secure messaging approaches, we examine the additional user effort (and decisions), security-related errors, and reduction in reliability and flexibility that they introduce. Our usability metrics compare this extra effort to a baseline approach with minimal security or privacy features. This is a challenging task and conventional user studies are not well suited to extract such high-level usability comparisons between disparate tools. We opted to employ expert reviews to measure these usability properties, which is consistent with previous systematization efforts for security schemes in other areas [23], [24]. To consider usability and adoption hurdles in practice, we combined these expert reviews with cognitive walkthroughs of actual implementations based on Nielsen's usability principles [25]–[27] and already known end-user issues discovered in previous work [2]–[5], [19]–[21], [28]. These usability results supplement our technical systematization and highlight potential trade-offs between security and usability.

3) Ease of Adoption: Adoption of secure messaging schemes is not only affected by their usability and security claims, but also by requirements imposed by the underlying technology. Protocols might introduce adoption issues by requiring additional resources or infrastructure from end users or service operators. When evaluating the adoption properties of an approach, we award a good score if the system does not exceed the resources or infrastructure requirements of a baseline approach that lacks any security or privacy features.

IV. TRUST ESTABLISHMENT

One of the most challenging aspects of messaging security is *trust establishment*, the process of users verifying that they are actually communicating with the parties they intend. *Long-term key exchange* refers to the process where users send cryptographic key material to each other. *Long-term key authentication* (also called *key validation* and *key verification*) is the mechanism allowing users to ensure that cryptographic long-term keys are associated with the correct real-world entities. We use *trust establishment* to refer to the combination of *long-term key exchange* and *long-term key authentication* in the remainder of this paper. After contact discovery (the process of locating contact details for friends using the messaging service), end users first have to perform trust establishment in order to enable secure communication.

A. Security and Privacy Features

A trust establishment protocol can provide the following security and privacy features:

Network MitM Prevention: Prevents Man-in-the-Middle (MitM) attacks by local and global network adversaries.

Operator MitM Prevention: Prevents MitM attacks executed by infrastructure operators.

Operator MitM Detection: Allows the detection of MitM attacks performed by operators after they have occurred.

Operator Accountability: It is possible to verify that operators behaved correctly during trust establishment.

Key Revocation Possible: Users can revoke and renew keys (e.g., to recover from key loss or compromise).

Privacy Preserving: The approach leaks no conversation metadata to other participants or even service operators.

B. Usability Properties

Most trust establishment schemes require key management: user agents must generate, exchange, and verify other participants' keys. For some approaches, users may be confronted with additional tasks, as well as possible warnings and errors, compared to classic tools without end-to-end security. If a concept requires little user effort and introduces no new error types, we award a mark for the property to denote good usability. We only consider the minimum user interaction required by the protocol instead of rating specific implementations.

Automatic Key Initialization: No additional user effort is required to create a long-term key pair.

Low Key Maintenance: Key maintenance encompasses recurring effort users have to invest into maintaining keys. Some systems require that users sign other keys or renew expired keys. Usable systems require no key maintenance tasks.

Easy Key Discovery: When new contacts are added, no additional effort is needed to retrieve key material.

Easy Key Recovery: When users lose long-term key material, it is easy to revoke old keys and initialize new keys (e.g., simply reinstalling the app or regenerating keys is sufficient).

In-band: No *out-of-band* channels are needed that require users to invest additional effort to establish.

No Shared Secrets: Shared secrets require existing social relationships. This limits the usability of a system, as not all communication partners are able to devise shared secrets.

Alert-less Key Renewal: If other participants renew their long-term keys, a user can proceed without errors or warnings.

Immediate Enrollment: When keys are (re-)initialized, other participants are able to verify and use them immediately.

Inattentive User Resistant: Users do not need to carefully inspect information (e.g., key fingerprints) to achieve security.

C. Adoption Properties

Multiple Key Support: Users should not have to invest additional effort if they or their conversation partners use multiple public keys, making the use of multiple devices with separate keys transparent. While it is always possible to share one key on all devices and synchronize the key between them, this can lead to usability problems.

No Service Provider Required: Trust establishment does not require additional infrastructure (e.g., key servers).

No Auditing Required: The approach does not require auditors to verify correct behavior of infrastructure operators.

No Name Squatting: Users can choose their names and can be prevented from reserving a large number of popular names.

Asynchronous: Trust establishment can occur asynchronously without all conversation participants online.

Scalable: Trust establishment is efficient, with resource requirements growing logarithmically (or smaller) with the the total number of participants in the system.

D. Evaluation

1) Opportunistic Encryption (Baseline): We consider *opportunistic encryption*, in which an encrypted session is established without any key verification, as a baseline. For

TABLE I

TRADE-OFFS FOR COMBINATIONS OF TRUST ESTABLISHMENT APPROACHES. SECURE APPROACHES OFTEN SACRIFICE USABILITY AND ADOPTION.

Scheme	Example	Security Features						Usability									Adoption					
		Network MitM Prevented	Operator MitM Prevented	Operator MitM Detected	Operator Accountability	Key Revocation Possible	Privacy Preserving	Automatic Key Initialization	Low Key Maintenance	Easy Key Discovery	Easy Key Recovery	In-Band	No Shared Secrets	Alert-less Key Renewal	Immediate Enrollment	Inattentive User Resistant	Multiple Key Support	No Service Provider	No Auditing Required	No Name Squatting	Asynchronous	Scalable
Opportunistic Encryption$^{\dagger *}$	TCPCrypt	-	-	-	-	-	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
+TOFU (Strict)†	-	◐	◐	◐	◐	-	●	●	●	●	-	●	●	-	●	●	-	●	●	●	●	●
+TOFU$^{\dagger *}$	TextSecure	◐	◐	◐	◐	-	●	●	●	●	●	●	●	-	●	-	-	●	●	●	●	●
Key Fingerprint Verification$^{\dagger *}$	Threema	●	●	●	●	◐	●	-	-	-	-	-	●	-	-	-	-	●	●	●	●	●
+Short Auth Strings (Out-of-Band)$^{\dagger *}$	SilentText	●	●	●	●	◐	●	-	-	-	-	-	●	-	-	-	-	-	●	●	-	●
+Short Auth Strings (In-Band/Voice/Video)$^{\dagger *}$	ZRTP	●	●	●	●	◐	●	-	-	-	-	◐	●	-	-	-	-	●	●	●	-	●
+Socialist Millionaire (SMP)$^{\dagger *}$	OTR	●	●	●	●	◐	●	-	-	-	-	●	-	-	-	-	-	●	●	●	-	●
+Mandatory Verification$^{\dagger *}$	SafeSlinger	●	●	●	●	◐	●	-	-	-	-	●	-	-	●	●	-	●	●	●	-	●
Key Directory$^{\dagger *}$	iMessage	●	-	-	-	●	-	●	●	●	●	●	●	●	●	●	●	-	●	●	◐	●
+Certificate Authority$^{\dagger *}$	S/MIME	●	-	-	-	-	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
+Transparency Log	-	●	-	◐	◐	-	-	●	●	●	●	●	●	●	◐	●	●	●	-	●	●	●
+Extended Transparency Log†	-	●	-	◐	●	●	-	●	●	●	●	●	●	●	◐	●	●	●	-	●	●	●
+Self-Auditable Log†	CONIKS	●	-	◐	●	●	●	●	●	●	●	●	●	●	●	●	●	●	◐	●	●	●
Web-of-Trust$^{\dagger *}$	PGP	●	●	●	◐	◐	-	-	-	◐	◐	-	-	-	-	-	●	●	●	●	●	●
+Trust Delegation$^{\dagger *}$	GnuNS	●	●	●	◐	◐	●	-	-	◐	◐	-	-	-	-	-	●	●	●	●	●	●
+Tracking*	Keybase	●	◐	◐	-	◐	-	◐	◐	◐	◐	-	-	-	●	-	●	-	●	◐	●	●
Pure IBC†	SIM-IBC-KMS	●	-	-	-	-	●	●	●	●	●	●	●	●	●	●	-	-	●	-	●	●
+Revocable IBC†	-	●	-	-	-	◐	●	●	●	●	●	●	●	●	●	●	-	-	●	-	●	●
Blockchains*	Namecoin	●	●	●	●	-	●	●	◐	●	-	●	●	●	-	●	●	●	-	-	●	-
Key Directory+TOFU+Optional Verification$^{\dagger *}$	TextSecure	◐	◐	◐	◐	●	-	●	●	●	●	●	●	-	●	-	◐	-	●	●	●	●
Opportunistic Encryption+SMP$^{\dagger *}$	OTR	◐	◐	◐	◐	●	●	-	●	-	-	◐	◐	-	●	-	●	●	●	●	-	●

● = provides property; ◐ = partially provides property; - = does not provide property; †has academic publication; *end-user tool available

instance, this could be an OTR encryption session without any authentication. The main goal of opportunistic encryption is to counter passive adversaries; active attackers can easily execute MitM attacks. From a usability perspective, this approach is the baseline since it neither places any burden on the user nor generates any new error or warning messages.

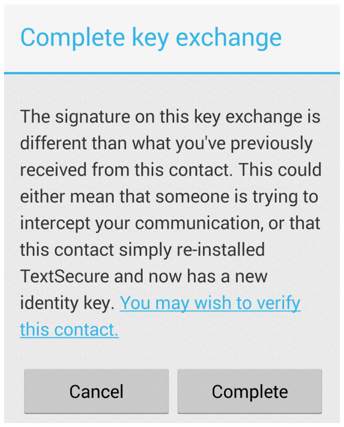
2) TOFU: Trust-On-First-Use (TOFU) extends opportunistic encryption by remembering previously seen key material [29]. The *network MitM prevented* and *infrastructure MitM prevented* properties are only partially provided due to the requirement that no attacker is present during the initial connection. TOFU *requires no service provider* since keys can be exchanged by the conversation participants directly. TOFU does not define a mechanism for *key revocation*. TOFU can be implemented in strict and non-strict forms. The strict form fails when the key changes, providing *inattentive user resilience* but preventing *easy key recovery*. The non-strict form prompts users to accept key changes, providing *easy key recovery* at the expense of *inattentive user resilience*.

TOFU-based approaches, like the baseline, do not require any user interaction during the initial contact discovery. This yields good scores for all user-effort properties except for the *key revocation* property, which is not defined, and *alert-less key renewal*, since users cannot distinguish benign key changes from MitM attacks without additional verification methods. For instance, TextSecure shows a warning that a user's key has changed and the user must either confirm the new key or apply manual verification to proceed (shown in Figure 2). If the user chooses to accept the new key immediately, it is possible to perform the verification later. The motivation behind this approach is to provide more transparency for more experienced or high-risk users, while still offering an "acceptable" solution for novice end users. Critically, previous work in the related domain of TLS warnings has shown that frequent warning messages leads to higher click-through rates in dangerous situations, even with experienced users [30].

From an adoption perspective, TOFU performs similarly to the baseline, except for *key recovery* in the strict version and *multiple key support* in both versions. The multiple key support problem arises from the fact that if multiple keys are used, the protocol cannot distinguish between devices. An attacker can claim that a new device, with the attacker's key, is being used.

3) Key Fingerprint Verification: Manual verification requires users to compare some representation of a cryptographic hash of their partners' public keys out-of-band (e.g., in person or via a separate secure channel).

Assuming the fingerprint check is performed correctly by end users, manual verification provides all desirable security properties with the exception of only partial *key revocation* support, as this requires contacting each communication part-


Complete key exchange

The signature on this key exchange is different than what you've previously received from this contact. This could either mean that someone is trying to intercept your communication, or that this contact simply re-installed TextSecure and now has a new identity key. You may wish to verify this contact.

Fig. 2. TextSecure warning for key changes: the user must either accept the new key by selecting “complete”, or perform manual verification.

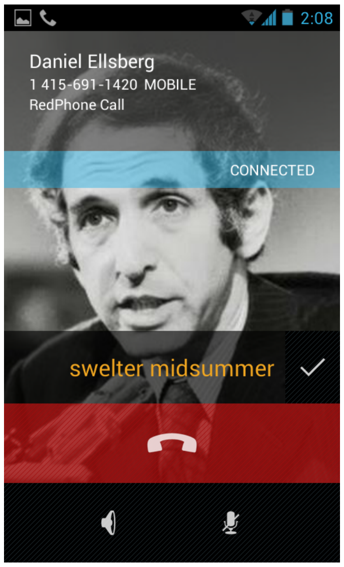


Fig. 3. Users read random words during SAS verification in RedPhone [36].

ner out-of-band. The approaches differ only in their usability and adoption features.

Fingerprint verification approaches introduce severe usability and adoption limitations: users have to perform manual verification before communicating with a new partner (and get them to do the same) to ensure strong authentication. Thus, manual verification does not offer *automatic key initialization*, *easy key discovery*, or *immediate enrollment*. In addition, new keys introduce an alert on key renewal, resulting in a *key maintenance* effort. Fingerprints complicate *multiple key support* since each device might use a different key.

While it is possible to improve the usability of key fingerprint verification by making it optional and combining it with other approaches, we postpone discussion of this strategy until the discussion.

4) Short Authentication String (SAS): To ease fingerprint verification, shorter strings can be provided to the users for comparison. A SAS is a truncated cryptographic hash (e.g., 20–30 bits long) of all public parts of the key exchange. It is often represented in a format aimed to be human-friendly, such as a short sequence of words. All participants compute the SAS based on the key exchange they observed, and then compare the resulting value with each other. The method used for comparison of the SAS must authenticate the entities using some underlying trust establishment mechanism.

Several encrypted voice channels, including the ZRTP protocol and applications like RedPhone, Signal, and SilentPhone, use the SAS method by requiring participants to read strings aloud [31], [32]. Figure 3 shows an example of SAS verification during establishment of a voice channel in RedPhone. For usability reasons, RedPhone and SilentPhone use random dictionary words to represent the hash. Because these tools require the user to end the established call manually if the verification fails, they are not *inattentive user resistant*.

SAS systems based on voice channels anchor trust in the ability of participants to recognize each other’s voices. Users who have never heard each other’s voices cannot authenticate using this method. Even for users that are familiar with each other, the security provided by voice identification has been the subject of controversy [33], [34]. Recent work [35] suggests that, even with a small number of samples of a target user’s speaking voice, audio samples can be synthesized which are indistinguishable from the genuine user’s voice with typical levels of background noise. We should expect that artificial voice synthesis will improve in cost and accuracy, while human auditory recognition will not improve.

For this reason, we consider voice-based SAS verification to be obsolescent from a security standpoint. In Table I, we assume that users verify the SAS with a method providing stronger security (e.g., using audio and video channels with careful inspection during the SAS verification). If the communication channel (e.g., text messaging) does not support a mechanism to establish trust, the SAS must be compared out of band (e.g., as recommended by SilentText).

The SAS approach sacrifices *asynchronicity*, since mutual authentication must be done with all users at the same time. Due to the short size of the SAS, the naive approach is vulnerable to a MitM attack by an adversary that attempts to select key exchange values that produce a hash collision for the two connections. To mitigate this problem, the attacker can be limited to a single guess by forcing them to reveal their chosen keys before observing the keys of the honest parties. This can be accomplished by requiring that the initiator of the key exchange release a commitment to their key, and then open the commitment after the other party reveals theirs.

5) Secret-based Zero-Knowledge Verification: The Socialist Millionaire Protocol (SMP) is a zero-knowledge proof of knowledge protocol that determines if secret values held by two parties are equal without revealing the value itself. This protocol is used in OTR as the recommended method for user verification [37], [38]. Alice poses a question based on shared knowledge to Bob *in-band* and secretly records her answer. After Bob answers the question, the two parties perform the

SMP to determine if their answers match, without revealing any additional information. Users are expected to choose secure questions with answers based on shared knowledge that attackers would be unable to know or guess.

The SMP used in OTR is performed on a cryptographic hash of the session identifier, the two parties' fingerprints, and their secret answers. This prevents MitM and replay attacks.

Since a MitM must perform an online attack and can only guess once, even low min-entropy secrets achieve strong security [38], [39]. However, use of the SMP sacrifices *asynchronicity* since all participants must be online during the verification. If the protocol fails, the end users do not know whether their answers did not match, or if a MitM attacker exists and has made an incorrect guess.

6) Mandatory Verification: The previously defined verification methods are prone to *inattentive users*. *Mandatory verification* approaches counter user negligence by requiring that users enter the correct fingerprint strings instead of merely confirming that they are correct. Of course, entering the fingerprints takes user effort. In practice, QR-Codes and NFC are popular methods to ease this process.

In SafeSlinger the user must choose the correct answer among three possibilities to proceed [40]. Physically co-located users form a group and exchange ephemeral keys. Each device hashes all received information and displays the hash as a sequence of three common words. Two additional sequences are randomly generated. The users communicate to determine the sequence that is common to all devices and select it to verify the ephemeral keys, preventing users from simply clicking an "OK" button. These keys are then used to exchange contact information within the group with security guarantees including confidentiality and authenticity.

Mandatory verification inherits the usability properties of the underlying scheme. Incorporating mandatory verification sacrifices *asynchronicity* to ensure *inattentive user resistance*.

7) Authority-based Trust: In authority-based trust schemes, public keys must be vouched for by one or more trusted authorities.

During key initialization, authorities can verify ownership of public keys by the claimed subjects through various means, such as password-based authentication or email validation. The authority then asserts the keys' validity to other users. Two well-known examples of authority-based trust establishment are public-key directories and certificate authority schemes.

A Certificate Authority (CA) may issue signed certificates of public keys to users, who can then present them directly to other users without needing to communicate further with the authority. This model has been widely deployed on the web with the X.509 Public Key Infrastructure (PKIX) for HTTPS [20]. While the S/MIME standard uses this model for secure email, it has seen less widespread deployment than PGP.

Alternatively, users may look up keys directly from an online *public-key directory* over a secure channel. This is common in several proprietary messaging applications such as Apple iMessage and BlackBerry Protected Messenger. In contrast to CA schemes, where the conversation partner directly provides an ownership assertion from the CA, the authority is directly asked for ownership assertions in key directory schemes.

From the security point of view, the two schemes only differ in *key revocation* and *privacy preservation*. While key updates in key directories imply the revocation of old keys, in the CA approach, certificates signed by the authority are trusted by default; revocation lists have to be maintained separately. However, CA-based revocation lists used in web browsers are known to have issues with effectiveness and practicality [24], [41], [42]. Since certificates may be exchanged by peers directly, the CA-based approach can be *privacy preserving*.

With either system, users are vulnerable to MitM attacks by the authority, which can vouch for, or be coerced to vouch for, false keys. This weakness has been highlighted by recent CA scandals [43], [44]. Both schemes can also be attacked if the authority does not verify keys before vouching for them. Authorities in messaging services often rely on insecure SMS or email verification, enabling potential attacks.

The two approaches both support good usability. Well-known systems using public-key directories, such as iMessage, work without any user involvement.

8) Transparency Logs: A major issue with trusted authorities is that they can vouch for fraudulent keys in an attack. The *Certificate Transparency* protocol [45] requires that all issued web certificates are included in a public log.

This append-only log is implemented using a signed Merkle tree with continual proofs of consistency [45]. Certificates are only trusted if they include cryptographic proof that they are present in the log. This ensures that any keys the authority vouches for will be visible in the log and evidence will exist that the authority singed keys used in an attack.

Certificate Transparency is a specific proposal for logging PKIX certificates for TLS, but the general idea can be applied to authority-based trust establishment in secure messaging. We refer to the general concept as *transparency logs* for the remainder of the paper. While there are no known deployments to date, Google plans to adapt *transparency logs* for user keys in End-to-End, its upcoming email encryption tool [46]. In the absence of a concrete definition, we evaluate *transparency logs* based on the certificate transparency protocol.

The main security improvement of the two schemes consists of *operator accountability* and the *detection of operator MitM attacks* after the fact. The remaining security features are inherited from authority-based trust systems.

However, these schemes introduce new and unresolved usability and adoption issues. For instance, the logs must be audited to ensure correctness, negating the *no auditing required* property. The auditing services require gossip protocols to synchronize the view between the monitors and prevent attack bubbles (e.g., where different views are presented to different geographical regions) [45]. Also, since only identity owners are in a position to verify the correctness of their long-term keys, they share responsibility for verifying correct behavior of the log. Previous research has shown that users often neglect such security responsibilities [30], so this task should be performed automatically by client applications. However, if a client detects a certificate in the log that differs from their version, it is not clear whether the authorities have performed an attack, an adversary has successfully impersonated the subject of the certificate to the authorities, or if the subject actually maintains multiple certificates (e.g., due to installing the app on a second device). Ultimately, end users have to cope with additional security warnings and errors, and it remains to be seen whether they can distinguish between benign and malicious log discrepancies without training. In addition, *transparency logs* might hamper *immediate enrollment* due to delays in log distribution.

Ryan [47] proposed extending the *transparency logs* concept using two logs: one of all certificates in chronological order of issuance, and one of currently valid certificates sorted lexicographically. This enables a form of revocation by making it efficient to query which certificates are currently valid for a given username. The scheme also supports a *proof-of-absence*, which guarantees the actual absence of an identifier or key in the log.

Melara et al. [48] proposed CONIKS, using a series of chained commitments to Merkle prefix trees to build a key directory that is self-auditing, that is, for which individual users can efficiently verify the consistency of their own entry in the directory without relying on a third party. This "self-auditing log" approach makes the system partially have *no auditing required* (as general auditing of non-equivocation is still required) and also enables the system to be *privacy preserving* as the entries in the directory need not be made public. This comes at a mild bandwidth cost not reflected in our table, estimated to be about 10 kilobytes per client per day for self-auditing.

Both Ryan's Extended Certificate Transparency and CONIKS also supports a *proof-of-absence*, which guarantees the absence of an identifier or key in the log.

9) Web of Trust: In a *web of trust* scheme, users verify each other's keys using manual verification and, once they are satisfied that a public key is truly owned by its claimed owner, they sign the key to certify this. These certification signatures might be uploaded to key servers. If Alice has verified Bob's key, and Bob certifies that he has verified Carol's key, Alice can then choose to trust Carol's key based on this assertion from Bob. Ideally, Alice will have multiple certification paths to Carol's key to increase her confidence in the key's authenticity.

The user interface for web of trust schemes tends to be relatively complex and has never been fully standardized. The scheme also requires a well-connected social graph, hence the motivation for "key-signing parties" to encourage users to form many links within a common social context.

Assuming that the web of trust model performs correctly, MitM attacks by network and operator adversaries are limited due to distribution of trust. However, since key revocations and new keys might be withheld by key servers, the model offers only partial *operator accountability* and *key revocation*. Since the web of trust model produces a public social graph, it is not *privacy preserving*.

The key-initialization phase requires users to get their keys signed by other keys, so the system does not offer *automatic key initialization*, *alert-less key renewal*, or *immediate enrollment*, and is not *inattentive user resistant*. Because users must participate in key-signing parties to create many paths for trust establishment, users have a high *key maintenance* overhead and a need for an *out-of-band* channel. Even worse, users must understand the details of the PKI and be able to decide whether to trust a key.

PGP typically uses a web of trust for email encryption and signing. In practice, the PGP web of trust consists of one strongly connected component and many unsigned keys or small connected components, making it difficult for those outside the strongly connected component to verify keys [49].

A simplification of the general web of trust framework is SDSI [50] (Simple Distributed Security Infrastructure) later standardized as SPKI [51], [52] (Simple Public Key Infrastructure). With SDSI/SPKI, Bob can assert that a certain key belongs to "Carol" and, if Alice has verified Bob's key as belonging to "Bob," that key will be displayed to Alice as "Bob's Carol" until Alice manually verifies Carol's key herself (which she can then give any name she wants, such as "Carol N."). We refer to these approaches as *trust delegation*. A modern implementation is the GNU Name System (GNS) [53], [54], which implements SDSI/SPKI-like semantics with a key server built using a distributed hash table to *preserve privacy*.

10) Keybase: Keybase is a trust establishment scheme allowing users to find public keys associated with social network accounts. It is designed to be easily integrated into other software to provide username-based trust establishment. If a user knows a social network username associated with a potential conversation partner, they can use Keybase to find the partner's public key.

During key initialization, all users register for accounts with the Keybase server. They then upload a public key and proof that they own the associated private key. Next, the user can associate accounts on social networks or other services with their Keybase account. Each external service is used to post a signature proving that the account is bound to the named Keybase account.

When looking up the key associated with a given user, the Keybase server returns the public key, a list of associated accounts, and web addresses for the external proofs. The client software requests the proofs from the external services and verifies the links. The user is then prompted to verify that the key belongs to the expected individual, based on the verified social network usernames. To avoid checking these proofs for every cryptographic operation, the user can sign the set of accounts owned by their partner. This signature is stored by the Keybase server so that all devices owned by the user can avoid verifying the external proofs again. This process is known as *tracking*. Tracking signatures created by other users are also accessible, providing evidence of account age. Old tracking signatures provide confidence that a user's accounts have not been recently compromised, but does not protect against infrastructure operator attacks.

Keybase provides partial *operator MitM protection* since attacks require collusion between multiple operators. The scheme also provides easier *key initialization* and *key maintenance* than web-of-trust methods.

11) Identity-Based Cryptography: In identity-based cryptography (IBC), plaintext identifiers (such as email or IP addresses) are mapped to public keys. A trusted third party, the Private Key Generator (PKG), publishes a PKG public key that is distributed to all users of the system. Public keys for an identifier are computed using a combination of the identifier and the PKG public key. The owner of the identity requests the private key for that identity from the PKG while providing proof that they own the identity. The advantage of this system is that users do not need to contact any other entity in order to retrieve the public key of a target user, since the public key is derived from the identifier.

There are two main problems with basic IBC schemes: they lack any *operator MitM prevention* and *key revocation* is not possible. Since the PKG can generate private keys for any user, the operator of the PKG can break the security properties of all conversations. While this fundamental problem cannot be overcome, key revocation support can be added. Revocable IBC approaches [55]–[57] add timestamps to the public key derivation process, regularly refreshing key material.

IBC schemes are normally deployed in situations where the trustworthiness of the PKG operator is assumed, such as in enterprise settings. Few pure-IBC schemes have been proposed for end-user messaging [58], [59].

12) Blockchains: The Bitcoin cryptocurrency utilizes a novel distributed consensus mechanism using pseudonymous "miners" to maintain an append-only log [60]. Voting power is distributed in proportion to computational resources by using a probabilistic proof-of-work puzzle. For the currency application, this log records every transaction to prevent double-spending. Miners are rewarded (and incentivized to behave honestly) by receiving money in proportion to the amount of computation they have performed. The success of Bitcoin's consensus protocol has led to enthusiasm that similar approaches could maintain global consensus on other types of data, such as a mapping of human-readable usernames to keys.

Namecoin, the first fork of Bitcoin, allows users to claim identifiers, add arbitrary data (e.g., public keys) as records for those identifiers, and even sell control of their identifiers to others [61]. Namecoin and similar name-mapping blockchains are denoted by the *blockchain* entry in Table I. Unlike most other schemes, Namecoin is strictly "first-come, first-served", with any user able to purchase ownership of any number of unclaimed names for a small, fixed fee per name. This price is paid in Namecoins — units of currency that are an inherent part of the system. A small maintenance fee is required to maintain control of names, and small fees may be charged by miners to update data or transfer ownership of names.

From the security perspective, blockchain schemes achieve similar results to manual verification, except that instead of exchanging keys, the trust relies on the username only. Once users have securely exchanged usernames, they can reliably fetch the correct keys.

However, various shortcomings arise from a usability and adoption perspective. The primary usability limitation is that if users ever lose the private key used to register their name (which is not the same as the communication key bound to that name), they will permanently lose control over that name (i.e., *key recovery* is not possible). Similarly, if the key is compromised, the name can be permanently and irrevocably hijacked. Thus, the system requires significant *key management* effort and burdens users with high responsibility. If users rely on a web-based service to manage private keys for them, as many do with Bitcoin in practice, the system is no longer truly end-to-end. The system requires users to pay to reserve and maintain names, sacrificing *low key maintenance* and *automatic key initialization*. Users also cannot instantly issue new keys for their identifiers (i.e., there is no *immediate enrollment*) but are required to wait for a new block to be published and confirmed. In practice, this can take 10–60 minutes depending on the desired security level.

On the adoption side, for the system to be completely trustless, users must store the entire blockchain locally and track its progress. Experience from Bitcoin shows that the vast majority of users will not do this due to the communication and storage requirements and will instead trust some other party to track the blockchain for them. This trusted party cannot easily insert spurious records, but can provide stale information without detection. In any case, the system is not highly *scalable* since the required amount of storage and traffic consumption increases linearly with the number of users.

Finally, there are serious issues with *name squatting*, which have plagued early attempts to use the system. Because anybody can register as many names as they can afford, a number of squatters have preemptively claimed short and common names. Given the decentralized nature of blockchains, this is hard to address without raising the registration fees, which increases the burden on all users of the system.

E. Discussion

As Table I makes evident, no trust establishment approach is perfect. While it is common knowledge that usability and security are often at odds, our results show exactly where the trade-offs lie. Approaches either sacrifice security and provide a nearly ideal user experience, or sacrifice user experience to achieve nearly ideal security scores. Authority-based trust (whether in the form of a single authority or multiple providers) and TOFU schemes are the most usable and well-adopted, but only offer basic security properties. Not surprisingly, authority-based trust (particularly app-specific key directories) is predominant among recently developed apps in the wild, as well as among apps with the largest userbases (e.g., iMessage, BlackBerry Protected, TextSecure, and Wickr). By contrast, no approach requiring specific user action to manage keys, such as web-of-trust, Keybase, GNS, or blockchains, has seen significant adoption among nontechnically-minded users.

In practice, we may be faced with the constraint that *none* of the usability properties can be sacrificed in a system that will achieve mass adoption. Higher-security schemes may be useful within organizations or niche communities, but defending against mass surveillance requires a communication system that virtually all users can successfully use. Thus, it may be wise to start from the basic user experience of today's widely deployed communication apps and try to add as much security as possible, rather than start from a desired security level and attempt to make it as simple to use as possible.

There appears to be considerable room for security improvements over authoritative key directories even without changes to the user experience. Transparency logs might provide more accountability with no interaction from most users. Because this approach has not yet been deployed, it remains to be seen how much security is gained in practice. The insertion of new keys in the log does not provide public evidence of malicious behavior if insecure user authentication methods (e.g., passwords) are used to authorize key changes, as we fully expect will be the case. Still, the possible loss of reputation may be enough to keep the server honest.

Another promising strategy is a layered design, with basic security provided by a central key directory, additional trust establishment methods for more experienced users (e.g., visual fingerprint verification or QR-codes), and TOFU warning messages whenever contacts' keys have changed. TextSecure and Threema, among others, take such a layered approach (represented by the second-to-last row in Table I). In contrast, OTR uses opportunistic encryption with the ability to perform the SMP to ensure trust (represented by the last row in Table I).

Conversely, the approaches with good security properties should focus on improving usability. There has been little academic work studying the usability of trust establishment. Further research focusing on end-users' mental models and perception for trust establishment could help to develop more sophisticated and understandable approaches.

V. Conversation Security

After *trust establishment* has been achieved, a *conversation security* protocol protects the security and privacy of the exchanged messages. This encompasses how messages are encrypted, what data is attached to them, and what cryptographic protocols (e.g., ephemeral key exchanges) are performed. A conversation security scheme doesn't specify a trust establishment scheme nor define how transmitted data reaches the recipient.

In Table II, we compare the features of existing approaches for conversation security. Rows without circles in the "group features" columns can only be used in a two-party setting.

A. Security and Privacy Features

Confidentiality: Only the intended recipients are able to read a message. Specifically, the message must not be readable by a server operator that is not a conversation participant.

Integrity: No honest party will accept a message that has been modified in transit.

Authentication: Each participant in the conversation receives proof of possession of a known long-term secret from all other participants that they believe to be participating in the conversation. In addition, each participant is able to verify that a message was sent from the claimed source.

Participant Consistency: At any point when a message is accepted by an honest party, all honest parties are guaranteed to have the same view of the participant list.

Destination Validation: When a message is accepted by an honest party, they can verify that they were included in the set of intended recipients for the message.

Forward Secrecy: Compromising all key material does not enable decryption of previously encrypted data.

Backward Secrecy: Compromising all key material does not enable decryption of succeeding encrypted data. This property is also often called *future secrecy* [17]. The terms are controversial and vague in literature [15], [16], [62].

Anonymity Preserving: Any anonymity features provided by the underlying transport privacy architecture are not undermined (e.g., if the transport privacy system provides anonymity, the conversation security level does not deanonymize users by linking key identifiers).

Speaker Consistency: All participants agree on the sequence of messages sent by each participant. A protocol might perform consistency checks on blocks of messages during the protocol, or after every message is sent.

Causality Preserving: Implementations can avoid displaying a message before messages that causally precede it.

Global Transcript: All participants see all messages in the same order. Note that this implies speaker consistency.

Not all security and privacy features are completely independent. If a protocol does not authenticate participants, then it offers participation repudiation (since no proof of participation is ever provided to anyone). Similarly, no authentication of message origin implies message repudiation as well as message unlinkability. Note that the implications are only one-way: repudiation properties might be achieved together with authentication. Additionally, a global transcript order implies both speaker consistency and causality preservation since all transcripts are identical.

Conversation security protocols may provide several different forms of deniability. Based on the definitions from Section II-C, we define the following deniability-related features:

Message Unlinkability: If a judge is convinced that a participant authored one message in the conversation, this does not provide evidence that they authored other messages.

Message Repudiation: Given a conversation transcript and all cryptographic keys, there is no evidence that a given message was authored by any particular user. We assume that the accuser has access to the session keys because it is trivial to deny writing a plaintext message when the accuser cannot demonstrate that the ciphertext corresponds to this plaintext. We also assume that the accuser does not have access to the accused participant's long-term secret keys because then it is

TABLE II
CONVERSATION SECURITY PROTOCOLS AND THEIR USABILITY AND ADOPTION IMPLICATIONS. NO APPROACH REQUIRES ADDITIONAL USER EFFORT.

Scheme	Example	Security and Privacy														Adoption					Group Chat				
		Confidentiality	Integrity	Authentication	Participant Consistency	Destination Validation	Forward Secrecy	Backward Secrecy	Anonymity Preserving	Speaker Consistency	Causality Preserving	Global Transcript	Message Unlinkability	Message Repudiation	Particip. Repudiation	Out-of-Order Resilient	Dropped Message Resilient	Asynchronicity	Multi-Device Support	No Additional Service	Computational Equality	Trust Equality	Subgroup Messaging	Contractable	Expandable
TLS+Trusted Server†*	Skype	-	-	-	-	-	-	-	-	-	-	-	●	●	●	●	●	●	●	-	●	●	●	◐	◐
Static Asymmetric Crypto†*	OpenPGP, S/MIME	●	●	●	-	-	-	-	●	-	-	-	-	-	-	●	●	●	●	●					
+IBE†	Wang et al.	-	●	●	-	-	-	-	●	-	-	-	-	-	-	●	●	●	●	-					
+Short Lifetime Keys	OpenPGP Draft	●	●	●	-	-	◐	◐	●	-	-	-	-	-	-	●	●	●	●	-					
+Non-Interactive IBE†	Canetti et al.	●	●	●	-	-	●	-	●	-	-	-	-	-	-	◐	●	●	●	●					
+Puncturable Encryption†	Green and Miers	●	●	●	-	-	●	-	●	-	-	-	-	-	-	●	●	●	●	●					
Key Directory+Short Lifetime Keys†	IMKE	●	●	◐	-	●	◐	◐	-	-	-	-	●	●	●	●	●	-	-	-					
+Long-Term Keys†	SIMPP	●	●	◐	-	●	◐	◐	-	-	-	-	●	●	-	●	●	-	-	-					
Authenticated DH†*	TLS-EDH-MA	●	●	●	●	●	◐	◐	●	-	-	-	●	●	◐	●	●	-	-	●					
+Naïve KDF Ratchet*	SCIMP	●	●	●	●	●	●	◐	●	◐	-	-	●	●	◐	◐	◐	-	-	●					
+DH Ratchet†*	OTR	●	●	●	●	●	◐	●	●	◐	◐	-	●	●	◐	◐	◐	-	-	●					
+Double Ratchet†*	Axolotl	●	●	●	●	●	●	●	●	◐	◐	-	●	●	◐	●	◐	-	-	●					
+Double Ratchet+3DH AKE†*	-	●	●	●	●	●	●	●	◐	◐	◐	-	●	●	●	●	◐	-	-	●					
+Double Ratchet+3DH AKE+Prekeys†*	TextSecure	●	●	●	●	●	●	●	-	◐	●	-	●	●	●	◐	◐	●	-	-					
Key Directory+Static DH+Key Transport†	Kikuchi et al.	●	●	-	-	●	◐	◐	-	-	-	-	●	●	-	●	●	●	-	-	-	-	-	●	●
+Authenticated EDH+Group MAC†	GROK	●	●	◐	-	●	◐	◐	●	-	-	-	●	●	-	●	●	●	-	-	-	-	-	●	●
GKA+Signed Messages+Parent IDs†	OldBlue	●	●	●	●	●	◐	◐	●	●	●	-	-	-	-	●	●	◐	-	●	●	●	-	-	-
Authenticated MP DH+Causal Blocks†*	KleeQ	●	●	◐	◐	◐	●	●	●	◐	◐	●	-	●	●	●	●	◐	-	●	●	●	-	-	●
OTR Network+Star Topology†	GOTR (2007)	●	●	-	-	-	◐	●	-	-	-	-	●	●	●	◐	●	◐	-	●	-	-	-	●	●
+Pairwise Topology†		●	●	●	●	●	◐	●	●	-	-	-	●	●	●	◐	●	◐	-	●	●	●	●	●	●
+Pairwise Axolotl+Multicast Encryption*	TextSecure	●	●	●	-	●	●	●	-	●	●	-	●	●	●	●	●	●	-	-	●	●	●	●	●
DGKE+Shutdown Consistency Check†	mpOTR	●	●	●	●	●	◐	◐	●	◐	-	-	-	●	●	●	●	-	-	●	●	●	-	-	-
Circle Keys+Message Consistency Check†	GOTR (2013)	●	●	●	●	●	◐	◐	●	●	●	●	●	●	●	●	-	-	-	◐	●	●	-	●	●

● = provides property; ◐ = partially provides property; - = does not provide property; †has academic publication; *end-user tool available

simple for the accuser to forge the transcript (and thus any messages are repudiable).

Participation Repudiation: Given a conversation transcript and all cryptographic key material for all but one accused participant, there is no evidence that the honest participant was in a conversation with any of the other participants.

Several additional features are only meaningful for group protocols (i.e., protocols supporting chats between three or more participants):

Computational Equality: All chat participants share an equal computational load.

Trust Equality: No participant is more trusted or takes on more responsibility than any other.

Subgroup messaging: Messages can be sent to a subset of participants without forming a new conversation.

Contractible Membership: After the conversation begins, participants can leave without restarting the protocol.

Expandable Membership: After the conversation begins, participants can join without restarting the protocol.

When a participant joins a secure group conversation, it is desirable for the protocol to compute new cryptographic keys so that the participant cannot decrypt previously sent messages. Likewise, keys should be changed when a participant leaves so that they cannot read new messages. This is trivial to implement by simply restarting the protocol, but this approach is often computationally expensive. Protocols with *expandable / contractible membership* achieve this without restarts.

There are many higher-level security and privacy design issues for secure group chat protocols. For example, the mechanisms for inviting participants to chats, kicking users out of sessions, and chat room moderation are all important choices that are influenced by the intended use cases. We do not cover these features here because they are implemented at a higher level than the secure messaging protocol layer.

B. Usability and Adoption

In classic messaging tools, users must only reason about two simple tasks: sending and receiving messages. However, in secure communication, additional tasks might be added. In old secure messaging systems, often based on OpenPGP, users could manually decide whether to encrypt and/or sign messages. Many studies have shown that this caused usability problems [2]–[5], [21]. However, during our evaluation, we found that most recent secure messenger apps secure all messages by default without user interaction. Since all implementations can operate securely once the trust establishment is complete, we omit the user-effort columns in Table II. However, we take other usability and adoption factors, such as resilience properties, into account:

Out-of-Order Resilient: If a message is delayed in transit, but eventually arrives, its contents are accessible upon arrival.

Dropped Message Resilient: Messages can be decrypted without receipt of all previous messages. This is desirable for asynchronous and unreliable network services.

Asynchronous: Messages can be sent securely to disconnected recipients and received upon their next connection.

Multi-Device Support: A user can participate in the conversation using multiple devices at once. Each device must be able to send and receive messages. Ideally, all devices have identical views of the conversation. The devices might use a synchronized long-term key or distinct keys.

No Additional Service: The protocol does not require any infrastructure other than the protocol participants. Specifically, the protocol must not require additional servers for relaying messages or storing any kind of key material.

C. Two-party Chat Evaluation

1) Trusted central servers (baseline): The most basic conversation security features that a secure chat protocol can provide are confidentiality and integrity. This can be easily implemented without adversely affecting usability and adoption properties by using a central server to relay messages and securing connections from clients to the central server using a transport-layer protocol like TLS. This also allows the central server to provide presence information. Since this approach does not negatively affect usability, it is no surprise that this architecture has been adopted by some of the most popular messaging systems today (e.g., Skype, Facebook Chat, Google Hangouts) [63]–[67]. We do not consider these protocols further because they do not meet our stronger end-to-end definition of *confidentiality* — that messages cannot be read by anyone except the intended recipient(s). We include this approach as a baseline in Table II in order to evaluate the effects of various designs.

Note that the baseline protocols provide all *repudiation* features, since there is no cryptographic proof of any activity. Additionally, these protocols are highly resilient to errors since there are no cryptographic mechanisms that could cause problems when messages are lost. The use of a trusted central server makes *asynchronicity* and *multi-device support* trivial.

2) Static Asymmetric Cryptography: Another simple approach is to use participants' static long-term asymmetric keypairs for signing and encrypting.

OpenPGP and S/MIME are two well-known and widely implemented standards for message protection, mostly used for email but also in XMPP-based tools [63], [68]–[70].

While this approach provides *confidentiality*, message *authentication*, and *integrity*, it causes a loss of all forms of *repudiation*. Additionally, care must be taken to ensure that *destination validation* and *participant consistency* checks are performed. Without destination validation, *surreptitious forwarding* attacks are possible [71]. Without participant consistency, *identity misbinding* attacks might be possible [62]. Defenses against replay attacks should also be included. These considerations are particularly relevant since the OpenPGP and S/MIME standards do not specify how to provide these features, and thus most implementations remain vulnerable to all of these attacks [68], [69].

To simplify key distribution, several authors have proposed the use of identity-based cryptography in the same setting. The SIM-IBC-KMS protocol acts as an overlay on the MSN chat network with a third-party server acting as the PKG [59]. Messages are encrypted directly using identity-based encryption. The protocol from Wang et al. [72] operates similarly, but distributes the PKG function across many servers with a non-collusion assumption in order to limit the impact of a malicious PKG. These protocols partially sacrifice *confidentiality* since an attacker with access to the PKG private key could surreptitiously decrypt communications.

A second issue with naive asymmetric cryptography is the lack of *forward* or *backward secrecy*. One way to address this issue is to use keys with very short lifetimes (e.g., changing the key every day). Brown et al. propose several extensions to OpenPGP based on this principle [73]. In the most extreme proposal, conversations are started using long-term keys, but each message includes an ephemeral public key to be used for replies. This method provides *forward* and *backward secrecy* for all messages except those used to start a conversation.

From a usability and adoption perspective, static key approaches achieve the same properties as the baseline. Apart from the non-transparent trust establishment, iMessage is a prominent example of how static asymmetric cryptography can achieve end-to-end conversation security with no changes to the user experience. Since the same long-term keys are used for all messages, *message order resilience*, *dropped message resilience*, *asynchronicity*, and *multi-device-support* are provided. *No additional services* are required.

3) FS-IBE: In traditional PKI cryptography, *forward secrecy* is achieved by exchanging ephemeral session keys or by changing keypairs frequently. The use of key agreement protocols makes *asynchronicity* difficult, whereas frequently changing keypairs requires expensive key distribution. Forward Secure Identity Based Encryption (FS-IBE) allows keypairs to be changed frequently with a low distribution cost. Unlike traditional identity-based encryption schemes, the private key generators (PKG) in FS-IBE are operated by the end users and not by a server. Initially, each participant generates a PKG for an identity-based cryptosystem. Participants generate N private keys (SK_i), one for each time period i, by using their PKG, and then immediately destroy the PKG. Each private key SK_i is stored encrypted by the previous private key SK_{i-1} [15], [74]. The participant then distributes the public key of the PKG. Messages sent to the participant are encrypted for the private key corresponding to the current time period. When a time period concludes, the next secret key is decrypted and the expired key is deleted. Thus, if intermediate keys are compromised, the attacker can only retrieve corresponding future private keys; *forward secrecy*, but not *backward secrecy*, is provided. In contrast to generating key pairs for each time period, which requires distribution of N keys, only a single public master key is published; however, the generation still needs to be repeated after all time periods expire.

Canetti, Halevi and Katz were the first to construct a non-interactive forward security scheme based on hierarchical IBE with logarithmic generation and storage costs [74]. In addition, they showed how their scheme can be extended to an unbounded number of periods (i.e., the private keys do not have to be generated in advance), removing the need for *additional services* to distribute new keys at the cost of increasing computational requirements over time. This scheme provides non-interactive *asynchronous forward secrecy* without relying on *additional services*. However, if messages arrive out of order, their corresponding private keys might have already been deleted. As a mitigation, expired keys might be briefly retained, providing partial *out-of-order resilience*.

Green and Miers proposed *puncturable encryption*, a modification of attribute-based encryption (built using a hierarchical IBE scheme) in which each message is encrypted with a randomly chosen "tag" and the recipient can update their private key to no longer be able to decrypt messages with that tag after receipt [75]. This approach provides arbitrary *out-of-order resilience*, although to make the scheme efficient in practice requires periodically changing keys.

Computational costs and storage costs increase over time for both systems, introducing *scalability* concerns. To our knowledge, neither scheme has been deployed and they thus merit further development.

4) Short lifetime key directories: Several protocols make use of a central server for facilitating chat session establishment. In these systems, users authenticate to the central server and upload public keys with short lifetimes. The server acts as a key directory for these ephemeral public keys. Conversations are initiated by performing key exchanges authenticated with the short-term keys vouched for by the key directory. Messages are then encrypted and authenticated using a MAC. IMKE [76] is a protocol of this type where the server authenticates users through the use of a password. SIMPP [77]–[79] operates similarly, but uses long-term keys to authenticate instead.

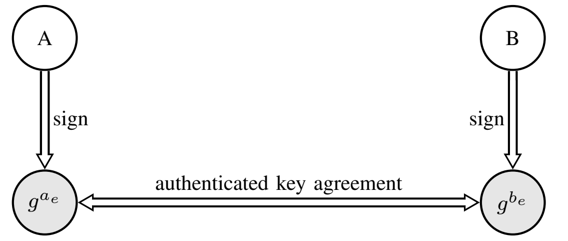
These protocols achieve *confidentiality* and *integrity*, but lack *authentication* of participants since the central server can vouch for malicious short-term keys. Since session keys are exchanged on a per-conversation basis, these protocols achieve *forward* and *backward secrecy* between conversations. Since SIMPP uses signatures during the login procedure, it loses *participation repudiability*; the accuser cannot forge their response to the server's challenge.

5) Authenticated Diffie-Hellman: While the use of central servers for presence information and central authentication is fundamental to systems such as IMKE and SIMPP, there is an alternative class of solutions that instead performs end-to-end authenticated Diffie-Hellman (DH) key exchanges. By default, the authenticated DH key agreement does not rely on central servers. In an authenticated key exchange (AKE) such as authenticated DH, the participants generate an ephemeral session key and authenticate the exchange using their long-term keys. The resulting session key is used to derive symmetric encryption and MAC keys, which then protect messages using an encrypt-then-MAC approach. This basic design provides *confidentiality*, *integrity*, and *authentication*. TLS with an ephemeral DH cipher suite and mutual authentication (TLS-EDH-MA) is a well-known example of this approach. Note that further protections are required during key exchange to protect against *identity misbinding* attacks violating *participant consistency*, such as those provided by the SIGMA protocol [38], [62].

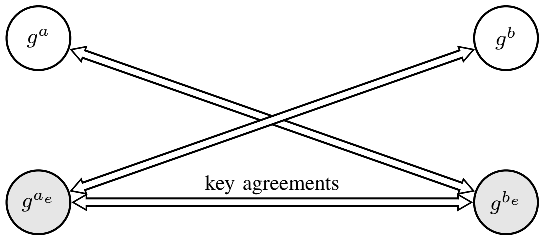
The use of ephemeral session keys provides *forward* and *backward secrecy* between conversations. *Message unlinkability* and *message repudiation* are provided since messages are authenticated with shared MAC keys rather than being signed with long-term keys. At a minimum, messages can be forged by any chat participants. Some protocols, such as OTR, take additional measures, such as publication of MAC keys and the use of malleable encryption, to expand the set of possible message forgers [81]. If the participants simply sign all AKE parameters, then this approach does not provide *participation repudiation*. However, if participants only sign their own ephemeral keys, these signatures can be reused by their conversation partners in forged transcripts. Figure 4a shows the authenticated key exchange used by OTRv1 (more recent versions use a SIGMA key exchange). Conversation partners are able to reuse ephemeral keys signed by the other party in forged transcripts, thereby providing partial *participation repudiation*. OTR users can increase the number of possible forgers by publishing previously signed ephemeral keys in a public location, thereby improving their *participation repudiation*.

Once the AKE has been performed, the encrypt-then-MAC approach allows messages to be exchanged asynchronously with *out-of-order* and *dropped message resilience*. However, since a traditional AKE requires a complete handshake before actual messages can be encrypted, this basic approach requires *synchronicity* during conversation initialization. Additionally, since key agreements can only be performed with connected devices, there is no trivial *multi-device support*.

6) Key Evolution: A desirable property is *forward secrecy* for individual messages rather than for entire conversations. This is especially useful in settings where conversations can last for the lifetime of a device. To achieve this, the session key from the initial key agreement can be evolved over time through the use of a *session key ratchet* [17]. A simple approach is to use key derivation functions (KDFs) to compute future message keys from past keys. This naive approach, as used in SCIMP [82], provides *forward secrecy*. However, it does not provide *backward secrecy* within conversations; if a key is compromised, all future keys can be derived using the KDF. *Speaker consistency* is partially obtained since messages cannot be surreptitiously dropped by an adversary without also dropping all future messages (otherwise, recipients would not be able to decrypt succeeding messages). If messages are dropped or arrive out of order, the recipient will notice since the messages are encrypted with an unexpected key. To handle this, the recipient must store expired keys so that delayed or



(a) OTRv1 DH handshake. The session key is derived from the key agreement based on signed ephemeral keys: $s = \mathrm{DH}(g^{a_e}, g^{b_e})$



(b) 3-DH handshake. The session key is a combination of all key agreements: $s = \mathrm{KDF}(\mathrm{DH}(g^{a_e}, g^{b_e})||\mathrm{DH}(g^{a_e}, g^b)||\mathrm{DH}(g^a, g^{b_e}))$

Fig. 4. TLS/OTRv1 handshake vs. 3-DH handshake (figures derived from [80]). Gray nodes represent ephemeral keys, white nodes represent long-term keys.

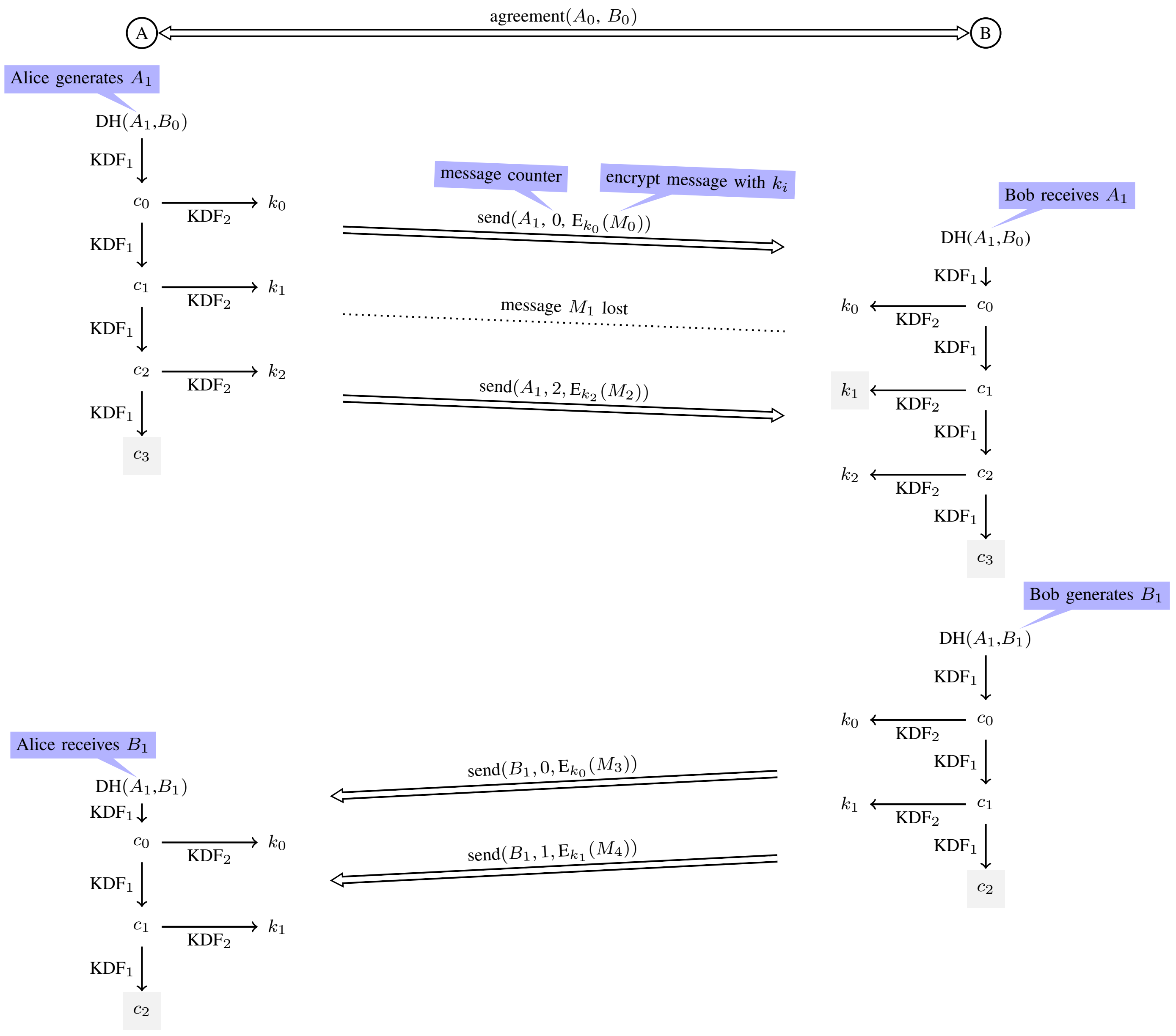


Fig. 5. Simplified version of Axolotl: c_i denote chain keys, k_i message keys, KDF_i arbitrary key derivation functions, E_{k_i} an encryption function using k_i, and $A_i = g^{a_i}$ and $B_i = g^{b_i}$ as public DH values. Gray key nodes denote keys held in memory after Alice receives message M_4.

re-transmitted messages can still be decrypted, leaving a larger window of compromise than necessary. Thus, *out-of-order* and *dropped message resilience* are only partially provided.

7) Diffie-Hellman Ratchet: A different ratcheting approach, introduced by OTR, is to attach new DH contributions to messages [81]. With each sent message, the sender advertises a new DH value. Message keys are then computed from the latest acknowledged DH values. This design introduces *backward secrecy* within conversations since a compromised key will regularly be replaced with new key material. *Causality preservation* is partially achieved since messages implicitly reference their causal predecessors based on which keys they use. The same level of *speaker consistency* as the naive KDF solution can be provided by adding a per-speaker monotonic counter to messages. A disadvantage of the DH ratchet is that session keys might not be renewed for every message (i.e., *forward secrecy* is only partially provided). Like the KDF-based ratchet, the DH ratchet lacks *out-of-order resilience*; if a message arrives after a newly advertised key is accepted, then the necessary decryption key was already deleted.

8) Double-Ratchet (Axolotl): To improve the forward secrecy of a DH ratchet, both ratchet approaches can be combined: session keys produced by DH ratchets are used to seed per-speaker KDF ratchets. Messages are then encrypted using keys produced by the KDF ratchets, frequently refreshed by the DH ratchet on message responses. The resulting *double ratchet*, as implemented by Axolotl [83], provides *forward secrecy* across messages due to the KDF ratchets, but also *backward secrecy* since compromised KDF keys will eventually be replaced by new seeds. To achieve *out-of-order resilience*, the Axolotl ratchet makes use of a second derivation function within its KDF ratchets. While the KDF ratchets are advanced normally, the KDF keys are passed through a second distinct derivation function before being used for encryption.

Figure 5 depicts the double ratchet used in Axolotl. The secondary KDF, denoted as KDF_2, allows the chain keys (c_i) to be advanced without sacrificing forward secrecy; each c_i is deleted immediately after being used to derive the corresponding message key (k_i) for encryption. If messages arrive out of order, this system provides a mechanism for decrypting the messages without compromising forward secrecy. For example, if Bob is expecting message M_1 and is storing c_1 in memory, but then receives M_2 instead, he uses c_1 to compute k_1, c_2, k_2, and c_3. Bob uses k_2 to decrypt the newly received message, and then he deletes c_1 and c_2 from memory, leaving only k_1 and c_3. When the missing M_1 eventually arrives, Bob can use k_1 to decrypt it directly. However, if an attacker compromises Bob's system at this moment, they will be unable to derive k_2 to decrypt M_2. A similar situation is depicted in Figure 5, where gray key nodes denote keys held in memory after Alice was able to receive M_4.

Axolotl also simplifies the use of its outer DH ratchet. In OTR, a chain of trust, allowing trust in new DH key exchanges to be traced back to the original AKE, is provided through the use of DH key advertisements and acknowledgments. To speed up this process, Axolotl instead derives a *root key* from the initial AKE in addition to the initial DH keys. Each subsequent DH secret is derived by using the sender's latest DH key, the latest DH key received from the other participant, and the current root key. Each time the DH ratchet is advanced, a new root key is derived in addition to a new chain key. Since deriving the chain keys requires knowledge of the current root key, newly received DH keys can be trusted immediately without first sending an acknowledgment. Despite these improvements, the double ratchet still requires *synchronicity* for the initial AKE.

9) 3-DH Handshake: A triple DH (3-DH) handshake is a different AKE scheme that provides stronger *participation repudiation*. Specifically, transcripts of conversations between any two participants can be forged by anyone knowing nothing more than the long-term public keys of the participants. Figure 4b depicts a 3-DH AKE. Triple DH is an implicitly authenticated key agreement protocol — a category that has been extensively examined in the literature [84]–[90]. Note that in this simplified version, if an attacker's ephemeral key was used (possible since ephemeral keys are not signed), the attacker would be able to calculate the session key retrospectively assuming the delayed possession of the corresponding long-term key. Thus, in practice the key exchange requires further protection mechanisms against ephemeral keys chosen by an attacker. Assuming that Alice and Bob both have long-term DH keys g^a and g^b and ephemeral keys g^{a_e} and g^{b_e}, the 3-DH shared secret s is computed as $s = \text{KDF}(\text{DH}(g^{a_e}, g^{b_e})||\text{DH}(g^a, g^{b_e})||\text{DH}(g^{a_e}, g^b))$ [83]. If a secure key derivation function is used, a MitM attacker must either know a and a_e, or b and b_e. Kudla et al. have shown that the 3-DH key exchange provides the same *authentication* level as achieved with the authenticated versions of DH key agreements [91]. 3-DH achieves full *participation repudiation* since anybody is able to forge a transcript between any two parties by generating both a_e and b_e and performing DH key exchanges with a and b. Assuming that Mallory uses g^m as her long-term DH value and g^{m_e} as her ephemeral key agreement value, and that she knows Alice's long-term DH value g^a, she is able to forge a transcript by calculating $s = \text{KDF}(\text{DH}(g^{a_e}, g^{m_e})||\text{DH}(g^a, g^{m_e})||\text{DH}(g^{a_e}, g^m))$ as the common HMAC and encryption secrets. Mallory can do this without ever actually interacting with Alice. Since the secret is partially derived from the long-term public keys, 3-DH also provides *participant consistency* without the need to explicitly exchange identities after a secure channel has been established. Unfortunately, this also causes a partial loss of *anonymity preservation* since long-term public keys are always observable during the initial key agreement (although future exchanges can be protected by using past secrets to encrypt these identities). It is possible to regain *anonymity preservation* by encrypting key identifiers with the given ephemeral keys.

10) Prekeys: While a *double ratchet* does not provide *asynchronicity* by itself, it can be combined with a *prekey* scheme to create an asynchronous version of the protocol. Prekeys are one-time ephemeral public DH contributions that have been uploaded in advance to a central server. This allows clients to complete a DH key exchange with a message recipient by requesting their next prekey from the server. When combined with a 3-DH exchange, this is sufficient to complete an asynchronous AKE as part of the first message. In comparison to time-window based FS-IBE approaches (cf. Section V-C3), this approach requires the pre-computation of a number of ephemeral keys, otherwise *forward secrecy* is weakened. However, this scheme also permits the destruction of the private ephemeral values immediately after receiving a message using them, instead of keeping a key until a time window is expired.

TextSecure [36] is a popular Android app that combines Axolotl, prekeys, and 3-DH to provide an *asynchronous* user experience while sacrificing the *no additional service* property. It has gained considerable attention recently after being incorporated into WhatsApp [92], [93]. Assuming Axolotl is used on two devices, the key material can evolve independently for each device. However, if one of those devices remains offline for a long time, a key compromise on that device is problematic: if the device can use its outdated keys to read messages that were sent when it was offline, then this compromise defeats *forward secrecy*; if the device cannot read the old messages, then the protocol does not achieve complete *multi-device support*. Deciding how long a device may be offline before it can no longer read buffered messages is an adoption consideration requiring further study of user behavior.

Frosch et al. analyzed the security of TextSecure [92]. With minor modifications, they prove that the protocol provides *authenticity* and *confidentiality*. They identify two primary issues: weakened HMAC security due to space-saving tag truncations, and an unknown key-share attack that enables surreptitious forwarding. Both attacks are theoretical in nature and can be addressed with minor protocol tweaks.

D. Group Chat Evaluation

1) Trusted central servers (baseline): The baseline protocol described in Section V-C1, where clients simply connect to a trusted central server using TLS, can trivially support group chats. While it is easy to add and remove group participants in this system, the only thing preventing participants from reading messages sent before or after they are part of the group is the trustworthiness of the server. This fact is indicated by half circles for *expandable / contractible membership*. SILC [94] in its default mode is an example of a protocol using this design. While SILC's architecture involves a network of trusted servers similar to the IRC protocol, for analysis purposes this network can be considered as one trusted entity.

To improve the security and privacy of these systems, participants can simply encrypt and authenticate messages before sending them to the server by using a pre-shared secret key for the group. This approach is useful because it can be applied as a layer on top of any existing infrastructure. SILC has built-in support for this method in its "private mode"; users can provide a password for a channel that is used to derive a pre-shared key unknown to the server. While this design provides *confidentiality* and *integrity*, it does not provide *authentication*.

2) Key transport: Rather than relying on users to exchange a secret password out-of-band, it is far better to automatically exchange a new secret for each conversation. A simple proposed method for doing this is to have one participant generate a session key and securely send it to the other participants. These systems begin by establishing secure channels between participants. The conversation initiator then generates a group key and sends it to the other participants using the pairwise channels. This design provides *forward* and *backward secrecy* since a new group key is randomly generated for each conversation. Due to the use of a group leader, *computational* and *trust equality* are also lost. However, groups are easily *expandable* and *contractible* by having the initiator generate and distribute a new group key.

An early design of this type, proposed by Kikuchi et al. [95], suggests using a key directory to store static DH public keys for users. When group chats are formed, these keys and are used to derive pairwise session keys for the participants. A modified DH exchange is used in order to allow the server to reduce the required computation for the clients. *Participation repudiation* is lost due to the design of the key exchange mechanism, whose security properties have not been rigorously verified. An improvement, used in the GROK protocol [96], is to use standard DH exchanges for the pairwise channels, authenticated using long-term public keys stored in the key directory. This improvement provides *authentication* and *anonymity preservation*, but still suffers from the inherent inequality of key transport approaches.

3) Causality preservation: One issue that is rarely addressed in the design of conversation security protocols is causality preservation. The user interface of the chat application must make design choices such as whether to display messages immediately when they are received, or to buffer them until causal predecessors have been received. However, the conversation security protocol must provide causality information in order to allow the interface to make these choices.

OldBlue [97] is a protocol that provides *speaker consistency* and *causality preservation*. An authenticated group key agreement (GKA) protocol is executed at the start of the conversation. Messages are encrypted with the group key and then signed with long-term asymmetric keys. This approach to signatures eliminates *message repudiation*. To preserve causality, messages include a list of identifiers of messages that causally precede them. In the OldBlue protocol, it is conservatively assumed that any message received by a user might influence the messages they subsequently send. Therefore, all received messages are considered to causally precede subsequently transmitted messages. Message identifiers are hashes of the sender, the list of preceding identifiers, and the message contents. When a message has been lost, the client continuously issues resend requests to the other clients.

A different approach is employed by KleeQ [98], a protocol designed for use by multiple trusted participants with tenuous connectivity. An authenticated multi-party DH exchange is performed to initiate the protocol. By authenticating the parameters in a manner similar to OTR, *participation repudiation* can be provided. The group can be easily *expanded* by incorporating the DH contribution of a new member into the multi-party DH exchange, deriving a new group key. However, the group is not *contractible* without restarting the protocol. When two conversation participants can establish a connection, they exchange the messages that the other is missing using a patching algorithm. All messages are encrypted and authenticated with a MAC using keys derived from the group secret, providing *message repudiation*. Messages are sealed into blocks, which are sequences of messages having the property that no messages could possibly be missing. After each block is sealed, rekeying is performed using the previous keys and the block contents. A mechanism is provided to seal blocks even if some users are inactive in the conversation. *Speaker consistency* is not guaranteed until the messages have been sealed in a block. While participants are *authenticated* during group formation, message contents are not authenticated until after they have been sealed into a block. The block sealing mechanism indirectly provides *participant consistency* and *destination validation*. If malicious participants send differing messages to others, this will be uncovered during the block sealing phase. Full accountability requires manual resolution.

4) OTR networks: Since OTR [81] provides desirable features for two-party conversations, it is natural to extend it to a group setting by using OTR to secure individual links in a network. A basic strategy is to enlist a trusted entity to relay messages and then secure client links to this entity using OTR. This is the approach taken by the GOTR protocol released in 2007 (we write the year to distinguish it from a different protocol with the same name from 2013). GOTR (2007) [99] selects a participant to act as the relay, forming a star topology of pairwise connections with the selected participant acting as the hub. All *authentication* properties, *speaker consistency*, and *causality preservation* are lost because they do not persist across the relay node. Since the relay server can buffer messages, *asynchronicity* is provided as long as the relay node remains online. All other properties are inherited from OTR. Groups can be *expanded* and *contracted* simply by establishing new OTR connections to the relay.

Instead of using a star topology, pairwise OTR connections between all participants can be established. This approach restores *authentication* and *anonymity preservation*, as well as *equal trust* between members. It is also possible to send messages to *subgroups* by only transmitting the message across selected OTR links. The downside of this approach is that it does not *preserve causality* or provide *speaker consistency*; participants can send different messages to different people. This design also incurs significant *computational overhead*. It would be desirable to achieve these security properties without this level of additional cost.

5) OTR for groups: Several protocols have been proposed to achieve OTR-like *repudiation* properties for group conversations. The TextSecure protocol can be naturally extended to groups by sending messages to each recipient using the two-party TextSecure protocol [100]. Multicast encryption is used for performance: a single encrypted message is sent to a central server for relaying to recipients while the decryption key for the message is sent pairwise using TextSecure. In practice, the wrapped decryption keys are attached to the same message for broadcasting. It is also possible to accomplish this task using one of the many existing broadcast encryption schemes [101]. This design does not provide any guarantees of *participant consistency*, but it inherits the *asynchronicity* of the two-party TextSecure protocol. *Speaker consistency* and *causality preservation* are achieved by attaching preceding message identifiers to messages. A message identifier is a hash of the sender, the list of preceding identifiers, and the message contents.

A *repudiable* group chat scheme can also be designed by utilizing a deniable group key exchange (DGKE) protocol, as in the mpOTR protocol [102], [103]. When completed, the DGKE provides each participant with a shared secret group key and individual ephemeral signing keys. This information is authenticated with long-term keys in a manner providing *participation repudiation* while still *authenticating* participants — participants receive proof of each other's identities, but this proof cannot be used to convince outsiders. All parties must be online to complete the DGKE, so the protocol does not support *asynchronicity*. Messages are encrypted with the shared group key and signed with the ephemeral keys. The ephemeral signatures provide proof of authorship to others in the group but, because outsiders cannot be certain that these ephemeral signing keys correspond to specific long-term keys, *message repudiation* is preserved. However, since all messages from an individual are signed with the same (ephemeral) key, the protocol does not have *message unlinkability*. When the conversation has concluded, each participant hashes all messages received from each other participant. The hashes are then compared to ensure that everyone received the same set of messages, providing *speaker consistency*. If this check fails, messages must be individually compared to uncover discrepancies. This approach, where a consistency check is performed only once at the conclusion of the conversation, does not work if a network adversary disconnects users from the conversation before the consistency check can be completed. In this worst-case scenario, the only information received by users is that something went wrong at some point during the protocol, but nothing more specific. Unfortunately, in many scenarios it is unclear how users should respond to this limited information. In this scheme, *subgroup messaging* is not possible since all messages share a single encryption key. The group is also not *expandable* or *contractible* without performing a new DGKE.

A completely different approach is taken by the GOTR (2013) protocol. GOTR (2013) [14] is built using a "hot-pluggable" group key agreement (GKA) protocol, allowing members to join and drop out of the conversation with little overhead. This system involves the use of "circle keys": sets of public keys having the property that a shared secret key can be computed by anyone with a private key matching a public key in the set. The key exchange mechanism in this protocol is relatively complex; we refer the interested reader to the original publication for details [14]. Pairwise secure channels are set up between participants to send consistency check messages. These consistency channels have the effect of providing *global transcript order*, but all participants are required to be online to receive messages. The system otherwise provides features similar to mpOTR but with *flexible group membership* and *message unlinkability*.

E. Discussion

Similar to our study of trust establishment, Table II makes immediately clear that no conversation security protocol provides all desired properties. Since most of the properties in the table are not mutually exclusive, however, there is significant room for improvement by combining protocol designs and this should be seen as a tangible and important call to action for the research community.

Sadly, the most widely adopted solutions also have the worst security and privacy properties, with most non-security-focused applications providing only basic static asymmetric cryptography. This does not appear to be due to the usability drawbacks of the more secure protocols: once the trust establishment has been done, all of the conversation security approaches we studied can be automated without any additional effort for the user. An exception is enabling *asynchronous* communication while still providing *forward* and *backward secrecy*; the only solution for this problem that appears to have any significant deployment in practice is the prekeys approach implemented by TextSecure. This requires relatively complicated infrastructure compared to a simple key server, introduces problems for *multi-device support*, and is prone to denial-of-service attacks if it is used in anonymous communication. This approach is poorly studied in the academic literature. The FS-IBE scheme discussed in Section V-C3 promises to resolve the issues of server complexity and denial of service, but introduces new challenges such as scalability and performance issues [74]. Unlike prekeys (Section V-C10), this scheme has received a considerable amount of follow-up research and academic citations, but we are unaware of any practical tool implementing it. In addition, a time-window based FS-IBE scheme requires holding the ephemeral keys for a certain amount of time to allow decryption of delayed messages. One possible mitigation is to rely on an additional server maintaining window counters where every window number is used once, analogous to the prekeys approach. Improving the practicality of FS-IBE and puncturable encryption schemes warrants further research.

Another outstanding concern that limits adoption of secure conversation security protocols is the limited support for multiple devices. Despite a vast number of users owning multiple devices, only the most insecure protocols support this property without requiring users to perform pairing procedures. Device pairing has proved extremely difficult for users in practice [104], [105] and allowing users to register multiple devices with distinct keys is a major usability improvement. Although extremely difficult, implementing usable device pairing is not necessarily an insurmountable problem. Additional work in this area is needed.

When it comes to group chat properties, we can identify several areas for improvement in Table II. Classic protocols often do not provide *participant consistency* or *destination validation*, making them potentially vulnerable to surreptitious forwarding or identity misbinding attacks. However, these are sometimes addressed in concrete implementations. The double ratchet used in Axolotl improves *forward secrecy* with low cost in performance, implementation complexity, and resilience, but it has not yet been thoroughly evaluated in an academic context. Additionally, decentralized group chat systems inherently permit a participant to send different messages to different people. Due to network conditions, users can also end up observing significantly different transcripts. Despite these intrinsic weaknesses, surprisingly few protocols explicitly consider *speaker consistency* or *causality preservation*. The recently proposed (n+1)sec protocol [106] is an example of new work in this area. (n+1)sec builds off of Abdalla et al.'s flexible group key exchange protocol [107] to provide a DGKE and checks for transcript consistency.

Existing solutions achieve mixed results concerning *repudiation*. It is often debated whether repudiation is a desirable feature and, if it is, whether or not it is worth pursuing. There are situations in which the mere suspicion of authoring a given message is potentially harmful to an author; in these cases, repudiation is not useful, and participants should make use of a protocol providing anonymity instead. Even in scenarios where repudiation is traditionally thought to be useful, such as during criminal trials in societies requiring proof of authorship "beyond a reasonable doubt", there are no prominent examples of repudiable messaging properties influencing legal decisions. Nonetheless, if it is possible to maintain repudiation within a secure messaging protocol without substantial cost, we believe that it remains a desirable property. Users typically think of real-world conversations as "off-the-record", so it is natural to desire (or expect) this property from a secure messaging protocol. For the definitions of *participation repudiation* and *message repudiation* used in this work, the two-party protocols based on authenticated DH key exchanges and the OTR-like group protocols provide inexpensive solutions.

There are also additional adoption constraints imposed by many modern secure group chat protocols. Group protocols often choose to employ either a trusted participant or an additional service to improve protocol performance, which can lead to security concerns or introduce additional costs for deployment. Very few group protocols support *subgroup messaging*, and just as few support changing group membership after the conversation has started without incurring the substantial costs of a new protocol run. Additionally, many proposed designs require synchronicity in order to simplify their protocols, which largely precludes their use on current mobile devices.

TABLE III

TRANSPORT PRIVACY SCHEMES. EVERY PRIVACY-ENHANCING APPROACH CARRIES USABILITY AND/OR ADOPTION COSTS.

Scheme	**Example**	**Privacy**					**Usability**					**Adoption**							
		Sender Anonymity	**Recipient Anonymity**	**Particip. Anonymity**	**Unlinkability**	**Global Adv. Resistant**	**Contact Discovery**	**No Message Delays**	**No Message Drops**	**Easy Initialization**	**No Fees Required**	**Topology Independent**	**No Additional Service**	**Spam/Flood Resistant**	**Low Storage**	**Low Bandwidth**	**Low Computation**	**Asynchronous**	**Scalable**
Store-and-Forward$^{\dagger *}$	Email/XMPP	-	-	-	-	-	●	◐	●	●	●	●	-	-	●	●	●	●	●
+DHT Lookup$^{\dagger *}$	Kademlia	◐	◐	-	-	-	●	◐	●	●	●	●	●	◐	●	◐	●	●	●
Onion Routing+Message Padding$^{\dagger *}$	Tor	●	-	●	●	-	-	◐	●	●	●	●	◐	-	●	●	●	-	●
+Hidden Services*	Ricochet	●	●	●	◐	-	-	◐	●	●	●	●	◐	-	●	●	●	-	●
+Inbox Servers†	-	●	-	●	●	-	-	◐	●	●	●	●	-	-	●	●	●	●	●
+Random Delays$^{\dagger *}$	Mixminion	●	-	●	●	◐	-	-	●	●	●	●	-	-	◐	●	●	●	●
+Hidden Services+Delays+Inboxes+ZKGP*	Pond	●	-	●	●	◐	-	-	●	●	●	●	-	●	◐	●	●	●	●
DC-Nets$^{\dagger *}$	-	●	●	-	-	●	-	-	●	●	●	-	●	-	●	●	●	-	-
+Silent Rounds†	Anonycaster	●	●	-	-	●	-	-	●	●	●	-	●	◐	●	●	●	-	-
+Shuffle-Based DC-Net+Leader†	Dissent	●	●	-	-	●	-	-	●	●	●	-	●	●	●	●	●	-	-
+Shuffle-Based DC-Net+Anytrust Servers†	Verdict	●	●	-	-	●	-	-	●	●	●	-	-	●	●	●	●	-	◐
Message Broadcast†	-	-	●	●	●	●	●	●	●	●	●	●	●	-	-	-	◐	-	-
+Blockchain	-	◐	●	●	●	●	●	-	-	●	-	●	◐	●	-	-	-	●	-
PIR*	Pynchon Gate	-	●	●	●	●	●	-	●	◐	●	●	-	-	-	◐	◐	●	◐

● = provides property; ◐ = partially provides property; - = does not provide property; †has academic publication; *end-user tool available

VI. TRANSPORT PRIVACY

The transport privacy layer defines how messages are exchanged, with the goal of hiding message metadata such as the sender, receiver, and conversation to which the message belongs. Some transport privacy architectures impose topological structures on the conversation security layer, while others merely add privacy to data links between entities. The transport privacy schemes may also be used for privacy-preserving contact discovery. In this section, we compare approaches for transport privacy in terms of the privacy features that they provide, as well as usability concerns and other factors that limit their adoption. Table III compares the various schemes.

A. Privacy Features

We make the distinction between *chat messages*, which are the user-generated payloads for the messaging protocol to exchange, and *protocol messages*, which are the underlying data transmissions dictated by the upper protocol layers. We define following privacy properties:

Sender Anonymity: When a chat message is received, no non-global entities except for the sender can determine which entity produced the message.

Recipient Anonymity: No non-global entities except the receiver of a chat message know which entity received it.

Participation Anonymity: No non-global entities except the conversation participants can discover which set of network nodes are engaged in a conversation.

Unlinkability: No non-global entities except the conversation participants can discover that two protocol messages belong to the same conversation.

Global Adversary Resistant: Global adversaries cannot break the anonymity of the protocol.

B. Usability Properties

Contact Discovery: The system provides a mechanism for discovering contact information.

No Message Delays: No long message delays are incurred.

No Message Drops: Dropped messages are retransmitted.

Easy Initialization: The user does not need to perform any significant tasks before starting to communicate.

No Fees Required: The scheme does not require monetary fees to be used.

C. Adoption Properties

Topology Independent: No network topology is imposed on the conversation security or trust establishment schemes.

No Additional Service: The architecture does not depend on availability of any infrastructure beyond the chat participants.

Spam/Flood Resistant: The availability of the system is resistant to denial-of-service attacks and bulk messaging.

Low Storage Consumption: The system does not require a large amount of storage capacity for any entity.

Low Bandwidth: The system does not require a large amount of bandwidth usage for any entity.

Low Computation: The system does not require a large amount of processing power for any entity.

Asynchronous: Messages sent to recipients who are offline will be delivered when the recipient reconnects, even if the sender has since disconnected.

Scalable: The amount of resources required to maintain system availability scales linearly with the number of users.

D. Evaluation

1) Store-and-Forward (baseline): To evaluate the effectiveness and costs of different transport privacy architectures in Table III, we compare the solutions to a baseline. For the baseline protocol, we assume a simple store-and-forward messaging protocol. This method is employed by email and text messaging, causing minor message delays and storage requirements for intermediate servers. Since email headers contain sender and recipient information, a simple store-and-forward mechanism does not provide any privacy properties.

2) Peer-to-Peer Solutions: Instead of relying on centralized servers for message storage and forwarding, peer-to-peer based schemes try to establish a direct message exchange between the participants. Since end users frequently change their IP addresses, these systems often use Distributed Hash Tables (DHTs) to map usernames to IP addresses without a central authority. Examples of popular DHT systems are Chord, Kademlia (used by BitTorrent), and GNUnet [108]–[110]. In addition to acting as an IP address lookup table, it is possible to store exchanged messages directly in a DHT. Various query privacy extensions have been proposed to prevent other users from learning what data is being requested. They can be used in advanced DHT overlays allowing anonymous queries and message exchange [111]–[113].

Global network adversaries are still able to see the traffic flow between participants during message exchange. Thus, clients have two options to protect the data flow: fake message transmissions, or use anonymization techniques. End-user clients might use services such as onion routing, which is evaluated in the next section, to hide their identities.

From the usability and adoption perspective, peer-to-peer networks require a synchronous environment. DHTs can be used for *contact discovery* with *easy initialization*, but they introduce *message delays* and *message drops*.

In practice, various end-user applications use the BitTorrent or GNUnet networks for their secure messaging service. For instance, Tox, Bleep, and other messengers use BitTorrent for message exchange. The GNUnet Name Service (GNS) offers privacy-preserving name queries for contact discovery [53].

3) Onion Routing: Onion routing is a method for communicating through multiple proxy servers that complicates end-to-end message tracing [114]. In onion routing, senders send messages wrapped in multiple layers of encryption through preselected paths of proxy servers. These servers unwrap layers of encryption until the original message is exposed, at which point it is relayed to the final destination. Each node in the path only knows the immediate predecessor and successor in the path. The routing process adds some latency to messages, but otherwise retains the baseline usability features. An onion routing protocol, such as the widely used Tor protocol [115], provides *sender anonymity*, *participant anonymity*, and *unlinkability* against network attackers with limited scope.

Global network adversaries are still able to break the anonymity properties of simple onion routing designs by performing statistical analysis incorporating features such as content size, transmission directions, counts, and timing, among others. The success of such an adversary can be limited by individually eliminating these features. Protection can be added, for example, by introducing random delays to transmissions. The longer the allowed delays, the less statistical power is available to the adversary. Of course, this imposes potentially long *message delays* and *additional storage requirements* for relays, making it unusable for synchronous instant messaging. Mixminion is an implementation using this technique [116].

Unfortunately, random delays do not completely defeat global adversaries. The only way to do so is to make transmission indistinguishable from no transmission (e.g., by saturating the bandwidth of all connections). However, in practice, this is likely infeasible. Additionally, concrete implementations such as Tor often provide weaker anonymity guarantees than idealized onion routing schemes. Several prominent attacks against Tor have been based on implementation defects, limited resources, weaknesses introduced by performance tradeoffs, and predictability of the content being transmitted [117]–[120]. Adoption of onion routing is limited by the requirement to establish a large network of nodes to provide a sufficient anonymity set and cover traffic.

In the default mode, onion routing systems do not attempt to provide *recipient anonymity*. However, Tor can be extended to achieve this property using an extension called *hidden services*. To create a Tor hidden service, the recipient uses traditional Tor circuits to upload a set of *introduction points* and a public key to a public database. The sender later uses a circuit to acquire this information from the database. The sender chooses a *rendezvous point* and sends it along with a nonce to the recipient through an introduction point. The recipient and sender both connect to the rendezvous point and use the nonce to relay transmissions. Without the aforementioned defenses, this scheme is also vulnerable to global adversaries.

To provide *asynchronous* communication support, store-and-forward servers can be incorporated into the onion routing model. Each user is associated with a Tor hidden service that remains online. To send a message, the sender constructs a circuit to the recipient's server and transmits the message. Users periodically poll their own servers to determine if any messages are queued. Ricochet is an example of this approach.

Pond uses this design for its transmission architecture [121] but adds random delays between connections, all of which transmit the same amount of data, to weaken statistical analysis by network adversaries. While some protection against global network adversaries is provided by the onion routing model, this protection is strictly weaker than Tor because connections are made directly from senders to recipient mail servers. This design requires *storage commitments* by servers and also introduces very *high latency*.

Without additional protections, this scheme is also highly vulnerable to denial-of-service attacks because connection delays and fixed transmission sizes artificially limit bandwidth to very low levels. Pond addresses this by requiring users to maintain group lists secured by zero-knowledge-group-proof schemes (ZKGP). This way, recipients can upload contact lists without revealing their contacts. Simultaneously, senders can authenticate by providing zero-knowledge proofs that they are in this list. The BBS signature scheme [122] is currently used by Pond to achieve this. Additional work is underway to provide a similar mechanism in more efficient manner by using one-time delivery tokens [121].

The ZKGP schemes used by Pond are related to secret handshake protocols. Secret handshakes enable authentication between parties that share some attributes, while keeping identities hidden from others [123].

4) DC-nets: Dining Cryptographer networks (DC-nets) are anonymity systems that are often compared to onion routing schemes. Since they are primarily used as a general-purpose transport privacy mechanism, many varieties have been proposed [124]–[131]. In our brief overview, we focus on recently introduced schemes that explicitly list secure messaging as an intended use case.

DC-nets are group protocols that execute in rounds. At the start of each round, each participant either submits a secret message or no message. At the end of the round, all participants receive the xor of all secret messages submitted, without knowing which message was submitted by which participants. In this way, DC-nets provide *sender anonymity* while also achieving *global adversary resilience* — no statistical analysis can reveal the sender of a message. *Recipient anonymity* can be achieved by using the protocol to publish an ephemeral public key. Messages encrypted with this key are then sent and, since the owner of the matching private key is unknown, the participant able to decrypt the messages cannot be determined. Since messages are sent in rounds, DC-nets add message latency and do not support *asynchronous* communication; dropped messages prevent the protocol from advancing. Messages are easily *linked* by observing which network nodes participate in a round. Additionally, DC-nets have limited *scalability* due to requiring pairwise communication.

The basic DC-net design has a problem with collisions: if two parties submit a message in the same round, the result will be corrupted. A malicious participant can exploit this to perform an anonymous denial-of-service attack by submitting garbled messages each round. Worse still, an active network attacker can also perform this attack by perturbing transmitted bits. There are several approaches to mitigate this problem. Anonycaster [127] adds pseudorandomly determined "silent rounds" where all members know that no message should be contributed. Receipt of a message during a silent round indicates a denial-of-service attack by an active network attacker. However, malicious participants can still launch attacks by sending garbled messages only during non-silent rounds.

Dissent [128]–[130] and Verdict [131] take a different approach by constructing a DC-net system through the use of a verifiable shuffle and bulk transfer protocol. Shuffle-based DC-nets can include a blame protocol to pinpoint the entity that caused a round to fail. Dissent appoints one participant as a leader to manage round timing, the blame protocol, and exclusion of disconnected members from rounds, thereby restoring support for *asynchronicity*. Verdict uses an alternative approach where the DC-net protocol is executed by a set of central servers that clients connect to, providing greater *scalability* and maintaining security as long as any one server is honest.

While DC-nets are primarily a transport privacy mechanism, they are distinguished from other schemes by their use of rounds and the fact that every network node is also a participant in the conversation. When using DC-nets to transmit higher-level conversation security protocols, it is important for designers to consider how these properties affect the overall security of the scheme (e.g., the use of synchronous rounds creates a *global transcript*, and the details of the DC-net key exchanges may cause a loss of *participation repudiation*).

5) Broadcast Systems: There is a simple approach to providing recipient anonymity against all attackers, including global adversaries: distributing messages to everyone. This approach provides *recipient anonymity*, *participation anonymity*, and *unlinkability* against all network attackers. It also provides a natural way to *discover contacts* because requests for contact data can be sent to the correct entity without knowledge of any addressing information. However, there are some serious downsides that hinder adoption: broadcasting a message to everyone in the network requires high bandwidth, there is no support for *asynchronicity*, and it has extreme *scalability* issues. Additionally, it is easy to attack the availability of the network through flooding. Bitmessage [132], a broadcast-based transport system, either requires a proof of work or monetary fees to send messages in order to limit spam, adding *computation requirements* and *message delays* as represented by the *blockchains* row in Table III. It is also possible to alleviate *scalability* problems by clustering users into smaller broadcast groups, at the cost of reduced anonymity set sizes.

6) PIR: Private Information Retrieval (PIR) protocols allow a user to query a database on a server without enabling the server to determine what information was retrieved. These systems, such as the Pynchon Gate [133], can be used to store databases of message inboxes, as well as databases of contact information. *Recipient anonymity* is provided because, while the server knows the network node that is connecting to it, the server cannot associate incoming connections with protocol messages that they retrieve. For the same reason, the protocols offer *participation anonymity* and *unlinkability*. By default, there is no mechanism for providing *sender anonymity*. These systems are naturally *asynchronous*, but they result in *high latency* because inboxes must be polled. The servers also incur a *high storage cost* and are vulnerable to flooding attacks.

PIR schemes can also be used to privately retrieve presence information, which can be useful for augmenting synchronous protocols lacking this capability. For example, DP5 [134] uses PIR to privately provide presence data for a secure messaging protocol; DP5 does not facilitate message transmission itself.

PIR implementations can be divided into computational schemes, which rely on computational limitations of the server, information-theoretic schemes, which rely on non-collusion of servers, and hybrid schemes that combine properties of both. PIR implementations differ in their bandwidth, computation, and initialization costs, as well as their scalability. PIR is not widely adopted in practice because one or more of these costs is usually prohibitively high.

E. Discussion

If messages are secured end-to-end, leaving only identifiers for anonymous inboxes in the unencrypted header, then metadata is easily hidden from service operators. Assuming that each message is sent using new channels, an adversary is not able to link single messages to conversations. However, such schemes introduce adoption and usability issues; they are prone to spam, flooding, and denial-of-service attacks, or require expensive operations such as zero-knowledge authentication posing barriers to adoption. Worse still, hiding metadata from a global adversary in these schemes necessitates serious usability problems such as long delays.

In contrast, decentralized schemes either exhibit synchronicity issues or have serious scalability problems. Most decentralized projects, especially BitTorrent-based approaches, lack detailed documentation that is required for complete evaluation. Some tools claiming to hide metadata only do so in the absence of global network adversaries, which recent surveillance revelations suggest may exist.

Broadcast-based schemes can achieve the best privacy properties, but exhibit serious usability issues, such as lost or delayed messages, in addition to apparently intractable scalability issues. Even if anonymous transmission schemes are adopted, they require a large user base to provide a high degree of anonymity, potentially discouraging early adopters. Finally, care must be taken when selecting a conversation security scheme to avoid leaking cryptographic material or identifiers that might lead to deanonymization.

VII. Concluding Remarks

The vast majority of the world's electronic communication still runs over legacy protocols like SMTP, SMS/GSM, and centralized messengers, none of which were designed with end-to-end security in mind. We encourage the research community to view the high-profile NSA revelations in the United States as a golden opportunity to encourage the adoption of secure systems in their place. As the old adage goes: "never let a crisis go to waste."

Unfortunately, while we have seen considerable progress in practical tools over the past two years, there is little evidence suggesting that academic research on secure messaging has dramatically increased. This is unfortunate for two reasons: First, many interesting problems of practical importance remain unresolved. In particular, apparent practical deployment constraints, including limitations for asynchronous communication, multiple independent devices, and zero user effort, are not fully appreciated in most published research papers. Second, many theoretically *solved* problems are not considered in practice, whether because developers are unaware of their existence, or because they cannot immediately translate the cryptographic publications into working systems.

Our effort to systematize existing knowledge on secure messaging suggests three major problems must be resolved: *trust establishment*, *conversation security* and *transport privacy*. The schemes can largely be chosen independently, yielding a vast design space for secure messaging systems. Yet we also caution against a proliferation of a-la-carte systems for specific niches. The main purpose of communication networks is to interact with others and there is considerable value in having a small number of popular protocols that connect a large number of users. Currently, everyone uses email and hence many people fall back to this method despite its insecurity.

We also note that, disappointingly, most of the exciting progress being made right now is by protocols that are either completely proprietary (e.g., Apple iMessage) or are open-source but lack a rigorously specified protocol to facilitate interoperable implementations (e.g., TextSecure). An open standard for secure messaging, combining the most promising features identified by our survey, would be of immense value.

Inevitably, trade-offs have to be made. We conclude that secure approaches in trust establishment perform poorly in usability and adoption, while more usable approaches lack strong security guarantees. We consider the most promising approach for trust establishment to be a combination of central key directories, transparency logs to ensure global consistency of the key directory's entries, and a variety of options for security-conscious users to verify keys out of band to put pressure on the key directory to remain honest.

Our observations on the conversation security layer suggest that asynchronous environments and limited multi-device support are not fully resolved. For two-party conversation security, per-message ratcheting with resilience for out-of-order messages combined with deniable key exchange protocols, as implemented in Axolotl, can be employed today at the cost of additional implementation complexity with no significant impact on user experience. The situation is less clear for secure group conversations; while no approach is a clear answer, the TextSecure group protocol provides pragmatic security considerations while remaining practical. It may be possible to achieve other desirable properties, such as participant consistency and anonymity preservation, by incorporating techniques from the other systems. It remains unclear exactly what consistency properties are required to match users' expectations and usability research is sorely needed to guide future protocol design. Finally, transport privacy remains a challenging problem. No suggested approaches managed to provide strong transport privacy properties against global adversaries while also remaining practical.

We consider this systematization to be a useful assessment of published research and deployment experience. We have uncovered many open challenges and interesting problems to be solved by the research community. The active development of secure messaging tools offers a huge potential to provide real-world benefits to millions; we hope this paper can serve as an inspiration and a basis for this important goal.

Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful comments, and Trevor Perrin and Henry Corrigan-Gibbs for their discussion and excellent feedback. Joseph Bonneau is supported by a Secure Usability Fellowship from the Open Technology Fund and Simply Secure. We gratefully acknowledge the support of NSERC and the Ontario Research Fund.

REFERENCES

[1] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, "SoK: Secure Messaging," in *Symposium on Security and Privacy*. IEEE, 2015.

[2] A. Whitten and J. D. Tygar, "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0," in *Security Symposium*. USENIX, 1999.

[3] S. L. Garfinkel and R. C. Miller, "Johnny 2: A User Test of Key Continuity Management with S/MIME and Outlook Express," in *Symposium on Usable Privacy and Security*. ACM, 2005, pp. 13–24.

[4] S. L. Garfinkel, D. Margrave, J. I. Schiller, E. Nordlander, and R. C. Miller, "How to Make Secure Email Easier To Use," in *SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2005, pp. 701–710.

[5] K. Renaud, M. Volkamer, and A. Renkema-Padmos, "Why Doesn't Jane Protect Her Privacy?" in *Privacy Enhancing Technologies*. Springer, 2014, pp. 244–262.

[6] M. Madden. (2014, Nov) Public Perceptions of Privacy and Security in the Post-Snowden Era. [Online]. Available: http://www.pewinternet.org/2014/11/12/public-privacy-perceptions/

[7] GoldBug Project. GoldBug - Secure Instant Messenger. [Online]. Available: http://goldbug.sourceforge.net/

[8] Telegram. Telegram Messenger. [Online]. Available: https://telegram.org/

[9] Wickr. Wickr – Top Secret Messenger. [Online]. Available: https://wickr.com/

[10] Confide. Confide - Your Off-the-Record Messenger. [Online]. Available: https://getconfide.com/

[11] Electronic Frontier Foundation. Secure Messaging Scorecard. [Online]. Available: https://www.eff.org/secure-messaging-scorecard

[12] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky, "Deniable Encryption," in *Advances in Cryptology–CRYPTO*. Springer, 1997, pp. 90–104.

[13] Y. Dodis, J. Katz, A. Smith, and S. Walfish, "Composability and On-Line Deniability of Authentication," in *Theory of Cryptography*. Springer, 2009, pp. 146–162.

[14] H. Liu, E. Y. Vasserman, and N. Hopper, "Improved Group Off-the-Record Messaging," in *Workshop on Privacy in the Electronic Society*. ACM, 2013, pp. 249–254.

[15] R. Anderson, "Two Remarks on Public Key Cryptology," 1997, available from https://www.cl.cam.ac.uk/users/rja14.

[16] R. Shirey, "Internet Security Glossary," RFC 2828 (Informational), Internet Engineering Task Force, 2000, obsoleted by RFC 4949. [Online]. Available: http://tools.ietf.org/rfc/rfc2828.txt

[17] O. W. Systems. Advanced cryptographic ratcheting. [Online]. Available: https://whispersystems.org/blog/advanced-ratcheting/

[18] S. E. Schechter, R. Dhamija, A. Ozment, and I. Fischer, "The Emperor's New Security Indicators: An evaluation of website authentication and the effect of role playing on usability studies," in *Symposium on Security and Privacy*. IEEE, 2007, pp. 51–65.

[19] R. Stedman, K. Yoshida, and I. Goldberg, "A User Study of Off-the-Record Messaging," in *Symposium on Usable Privacy and Security*. ACM, 2008, pp. 95–104.

[20] S. Clark, T. Goodspeed, P. Metzger, Z. Wasserman, K. Xu, and M. Blaze, "Why (Special Agent) Johnny (Still) Can't Encrypt: A Security Analysis of the APCO Project 25 Two-way Radio System," in *Security Symposium*. USENIX, 2011.

[21] S. Fahl, M. Harbach, T. Muders, M. Smith, and U. Sander, "Helping Johnny 2.0 to Encrypt His Facebook Conversations," in *Symposium on Usable Privacy and Security*. ACM, 2012.

[22] S. Ruoti, N. Kim, B. Burgon, T. van der Horst, and K. Seamons, "Confused Johnny: When Automatic Encryption Leads to Confusion and Mistakes," in *Symposium on Usable Privacy and Security*. ACM, 2013.

[23] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, "The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes," in *Symposium on Security and Privacy*. IEEE, 2012. [Online]. Available: http://www.jbonneau.com/doc/BHOS12-IEEESP-quest_to_replace_passwords.pdf

[24] J. Clark and P. C. van Oorschot, "SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements," in *Symposium on Security and Privacy*. IEEE, 2013, pp. 511–525.

[25] J. Nielsen, "Finding Usability Problems Through Heuristic Evaluation," in *SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1992, pp. 373–380.

[26] ——, "Usability inspection methods," in *Conference Companion on Human Factors in Computing Systems*. ACM, 1994, pp. 413–414.

[27] B. E. John and M. M. Mashyna, "Evaluating a Multimedia Authoring Tool with Cognitive Walkthrough and Think-Aloud User Studies," DTIC, Tech. Rep., 1995.

[28] S. Sheng, L. Broderick, C. A. Koranda, and J. J. Hyland, "Why Johnny Still Can't Encrypt: Evaluating the Usability of Email Encryption Software," in *Symposium On Usable Privacy and Security*. ACM, 2006.

[29] D. Wendlandt, D. G. Andersen, and A. Perrig, "Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing," in *Annual Technical Conference*. USENIX, 2008, pp. 321–334.

[30] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor, "Crying Wolf: An Empirical Study of SSL Warning Effectiveness," in *Security Symposium*. USENIX, 2009, pp. 399–416.

[31] P. Zimmermann, A. Johnston, and J. Callas, "ZRTP: Media Path Key Agreement for Unicast Secure RTP," RFC 6189 (Informational), Internet Engineering Task Force, 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6189.txt

[32] E. A. Blossom, "The VP1 Protocol for Voice Privacy Devices Version 1.2," Communication Security Corporation, 1999.

[33] P. Gupta and V. Shmatikov, "Security Analysis of Voice-over-IP Protocols," in *Computer Security Foundations Symposium*. IEEE, 2007, pp. 49–63.

[34] M. Petraschek, T. Hoeher, O. Jung, H. Hlavacs, and W. N. Gansterer, "Security and Usability Aspects of Man-in-the-Middle Attacks on ZRTP," *Journal of Universal Computer Science*, vol. 14, no. 5, pp. 673–692, 2008.

[35] M. Shirvanian and N. Saxena, "Wiretapping via Mimicry: Short Voice Imitation Man-in-the-Middle Attacks on Crypto Phones," in *Conference on Computer and Communications Security*. ACM, 2014, pp. 868–879.

[36] O. W. Systems. Open WhisperSystems. [Online]. Available: https://whispersystems.org/

[37] M. Jakobsson and M. Yung, "Proving Without Knowing: On Oblivious, Agnostic and Blindfolded Provers," in *Advances in Cryptology–CRYPTO*. Springer, 1996, pp. 186–200.

[38] C. Alexander and I. Goldberg, "Improved User Authentication in Off-The-Record Messaging," in *Workshop on Privacy in the Electronic Society*. ACM, 2007, pp. 41–47.

[39] F. Boudot, B. Schoenmakers, and J. Traoré, "A fair and efficient solution to the socialist millionaires' problem," *Discrete Applied Mathematics*, vol. 111, no. 1, pp. 23–36, 2001.

[40] M. Farb, Y.-H. Lin, T. H.-J. Kim, J. McCune, and A. Perrig, "SafeSlinger: Easy-to-Use and Secure Public-Key Exchange," in *International Conference on Mobile Computing & Networking*. ACM, 2013, pp. 417–428.

[41] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software," in *Conference on Computer and Communications Security*. ACM, 2012, pp. 38–49.

[42] M. Marlinspike, "More tricks for defeating SSL in practice," in *Black Hat USA*, 2009.

[43] VASCO. (2011, Sep) DigiNotar reports security incident. [Online]. Available: https://www.vasco.com/company/about_vasco/press_room/news_archive/2011/news_diginotar_reports_security_incident.aspx

[44] A. Langley. (2013) Enhancing digital certificate security. [Online]. Available: http://googleonlinesecurity.blogspot.de/2013/01/enhancing-digital-certificate-security.html

[45] B. Laurie, A. Langley, and E. Kasper, "Certificate Transparency," RFC 6962 (Experimental), Internet Engineering Task Force, 2013. [Online]. Available: http://tools.ietf.org/rfc/rfc6962.txt

[46] Google. End-To-End. [Online]. Available: https://github.com/google/end-to-end

[47] M. D. Ryan, "Enhanced Certificate Transparency and End-to-end Encrypted Mail," in *Network and Distributed System Security Symposium*. Internet Society, 2014.

[48] M. S. Melara, A. Blankstein, J. Bonneau, M. J. Freedman, and E. W. Felten, "CONIKS: A Privacy-Preserving Consistent Key Service for Secure End-to-End Communication," Cryptology ePrint Archive Report 2014/1004, 2014. [Online]. Available: https://eprint.iacr.org/2014/1004

[49] A. Ulrich, R. Holz, P. Hauck, and G. Carle, "Investigating the OpenPGP Web of Trust," in *Computer Security–ESORICS*. Springer, 2011, pp. 489–507.

[50] R. L. Rivest and B. Lampson, "SDSI – A Simple Distributed Security Infrastructure," 1996, manuscript.

[51] C. M. Ellison, "Establishing Identity Without Certification Authorities," in *Security Symposium*. USENIX, 1996, pp. 67–76.

[52] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen, "SPKI Certificate Theory," RFC 2693 (Experimental), Internet Engineering Task Force, 1999. [Online]. Available: http://tools.ietf.org/rfc/rfc2693.txt

[53] M. Wachs, M. Schanzenbach, and C. Grothoff, "A Censorship-Resistant, Privacy-Enhancing and Fully Decentralized Name System," in *Cryptology and Network Security*. Springer, 2014, pp. 127–142.

[54] ——, "On the Feasibility of a Censorship Resistant Decentralized Name System," in *Foundations and Practice of Security*. Springer, 2014, pp. 19–30.

[55] A. Boldyreva, V. Goyal, and V. Kumar, "Identity-based Encryption with Efficient Revocation," in *Conference on Computer and Communications Security*. ACM, 2008, pp. 417–426.

[56] B. Libert and D. Vergnaud, "Adaptive-ID Secure Revocable Identity-Based Encryption," in *Topics in Cryptology–CT-RSA*. Springer, 2009, pp. 1–15.

[57] C. Wang, Y. Li, X. Xia, and K. Zheng, "An Efficient and Provable Secure Revocable Identity-Based Encryption Scheme," *PLOS ONE*, 2014.

[58] A. Thukral and X. Zou, "Secure Group Instant Messaging Using Cryptographic Primitives," in *Networking and Mobile Computing*. Springer, 2005, pp. 1002–1011.

[59] Z. Bin, F. Meng, X. Hou-ren, and H. Dian-you, "Design and Implementation of Secure Instant Messaging System Based on MSN," in *International Symposium on Computer Science and Computational Technology*, vol. 1. IEEE, 2008, pp. 38–41.

[60] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008, self-published.

[61] N. Project. Namecoin. [Online]. Available: https://namecoin.info/

[62] W. Diffie, P. C. van Oorschot, and M. J. Wiener, "Authentication and Authenticated Key Exchanges," *Designs, Codes and Cryptography*, vol. 2, no. 2, pp. 107–125, 1992.

[63] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core," RFC 6120 (Proposed Standard), Internet Engineering Task Force, 2011. [Online]. Available: http://tools.ietf.org/rfc/rfc6120.txt

[64] S. Schrittwieser, P. Frühwirt, P. Kieseberg, M. Leithner, M. Mulazzani, M. Huber, and E. R. Weippl, "Guess Who's Texting You? Evaluating the Security of Smartphone Messaging Applications," in *Network and Distributed System Security Symposium*. Internet Society, 2012.

[65] Microsoft. (2014) Does Skype use encryption? [Online]. Available: https://support.skype.com/en/faq/FA31/does-skype-use-encryption

[66] Google. (2014) Google Hangouts - Video Conferencing & Meeting for Business. [Online]. Available: https://www.google.com/work/apps/business/products/hangouts/

[67] Facebook. (2014) Facebook Help Center. [Online]. Available: https://www.facebook.com/help/

[68] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, "OpenPGP Message Format," RFC 4880 (Proposed Standard), Internet Engineering Task Force, 1999, updated by RFC 5581. [Online]. Available: http://tools.ietf.org/rfc/rfc4880.txt

[69] B. Ramsdell and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification," RFC 5751 (Proposed Standard), Internet Engineering Task Force, 2010. [Online]. Available: http://tools.ietf.org/rfc/rfc5751.txt

[70] D. Fomin and Y. Leboulanger. Gajim, a Jabber/XMPP client. [Online]. Available: https://gajim.org/

[71] D. Davis, "Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML," in *Annual Technical Conference, General Track*. USENIX, 2001, pp. 65–78.

[72] C.-J. Wang, W.-L. Lin, and H.-T. Lin, "Design of An Instant Messaging System Using Identity Based Cryptosystems," in *International Conference on Emerging Intelligent Data and Web Technologies*. IEEE, 2013, pp. 277–281.

[73] I. Brown, A. Back, and B. Laurie, "Forward Secrecy Extensions for OpenPGP," Draft, Internet Engineering Task Force, 2002. [Online]. Available: https://tools.ietf.org/id/draft-brown-pgp-pfs-03.txt

[74] R. Canetti, S. Halevi, and J. Katz, "A Forward-Secure Public-Key Encryption Scheme," in *Advances in Cryptology–EUROCRYPT*. Springer, 2003, pp. 255–271.

[75] M. Green and I. Miers, "Forward Secure Asynchronous Messaging from Puncturable Encryption," in *Symposium on Security and Privacy*. IEEE, 2015.

[76] M. Mannan and P. C. van Oorschot, "A Protocol for Secure Public Instant Messaging," in *Financial Cryptography and Data Security*. Springer, 2006, pp. 20–35.

[77] C.-H. Yang and T.-Y. Kuo, "The Design and Implementation of a Secure Instant Messaging Key Exchange Protocol," 2007, available from http://crypto.nknu.edu.tw/psnl/publications/2007Technology_SIMPP.pdf.

[78] C.-H. Yang, T.-Y. Kuo, T. Ahn, and C.-P. Lee, "Design and Implementation of a Secure Instant Messaging Service based on Elliptic-Curve Cryptography," *Journal of Computers*, vol. 18, no. 4, pp. 31–38, 2008.

[79] C.-P. Lee and C.-H. Yang, "Design and Implement of a Secure Instant Messaging Service with IC Card," 2009, available from http://crypto.nknu.edu.tw/psnl/publications/2009CPU_SIMICCard.pdf.

[80] O. W. Systems. Simplifying OTR deniability. [Online]. Available: https://whispersystems.org/blog/simplifying-otr-deniability

[81] N. Borisov, I. Goldberg, and E. Brewer, "Off-the-Record Communication, or, Why Not To Use PGP," in *Workshop on Privacy in the Electronic Society*. ACM, 2004, pp. 77–84.

[82] V. Moscaritolo, G. Belvin, and P. Zimmermann, *Silent Circle Instant Messaging Protocol Protocol Specification*, 2012.

[83] T. Perrin. (2013) Axolotl Ratchet. [Online]. Available: https://github.com/trevp/axolotl/wiki

[84] A. J. Menezes, M. Qu, and S. A. Vanstone, "Some New Key Agreement Protocols Providing Implicit Authentication," in *Selected Areas in Cryptography*, 1995, pp. 22–32.

[85] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone, "An Efficient Protocol for Authenticated Key Agreement," *Designs, Codes and Cryptography*, vol. 28, no. 2, pp. 119–134, 2003.

[86] N. S. Agency. SKIPJACK and KEA Algorithm Specifications. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/documents/skipjack/skipjack.pdf

[87] B. LaMacchia, K. Lauter, and A. Mityagin, "Stronger Security of Authenticated Key Exchange," in *Provable Security*. Springer, 2007, pp. 1–16.

[88] K. Lauter and A. Mityagin, "Security Analysis of KEA Authenticated Key Exchange Protocol," in *Public Key Cryptography – PKC 2006*. Springer, 2006, pp. 378–394.

[89] R. Ankney, D. Johnson, and M. Matyas, "The Unified Model," Contribution to X9F1, 1995.

[90] H. Krawczyk, "HMQV: A High-Performance Secure Diffie-Hellman Protocol," in *Advances in Cryptology–CRYPTO*. Springer, 2005, pp. 546–566.

[91] C. Kudla and K. G. Paterson, "Modular Security Proofs for Key Agreement Protocols," in *Advances in Cryptology–ASIACRYPT*. Springer, 2005, pp. 549–565.

[92] T. Frosch, C. Mainka, C. Bader, F. Bergsma, J. Schwenk, and T. Holz, "How Secure is TextSecure?" Cryptology ePrint Archive Report 2014/904, 2014. [Online]. Available: https://eprint.iacr.org/2014/904

[93] O. W. Systems. Open Whisper Systems partners with WhatsApp to provide end-to-end encryption. [Online]. Available: https://whispersystems.org/blog/whatsapp/

[94] SILC Project. SILC – Secure Internet Live Conferencing. [Online]. Available: http://silcnet.org/

[95] H. Kikuchi, M. Tada, and S. Nakanishi, "Secure Instant Messaging Protocol Preserving Confidentiality against Administrator," in *International Conference on Advanced Information Networking and Applications*. IEEE, 2004, pp. 27–30.

[96] J. A. Cooley, R. I. Khazan, B. W. Fuller, and G. E. Pickard, "GROK: A Practical System for Securing Group Communications," in *International Symposium on Network Computing and Applications*. IEEE, 2010, pp. 100–107.

[97] M. D. Van Gundy and H. Chen, "OldBlue: Causal Broadcast In A Mutually Suspicious Environment (Working Draft)," 2012, available from http://matt.singlethink.net/projects/mpotr/oldblue-draft.pdf.

[98] J. Reardon, A. Kligman, B. Agala, and I. Goldberg, "KleeQ: Asynchronous Key Management for Dynamic Ad-Hoc Networks," University of Waterloo, Tech. Rep., 2007.
[99] J. Bian, R. Seker, and U. Topaloglu, "Off-the-Record Instant Messaging for Group Conversation," in *International Conference on Information Reuse and Integration*. IEEE, 2007, pp. 79–84.
[100] O. W. Systems. Private Group Messaging. [Online]. Available: https://whispersystems.org/blog/private-groups/
[101] A. Fiat and M. Naor, "Broadcast Encryption," in *Advances in Cryptology–CRYPTO'93*. Springer, 1994, pp. 480–491.
[102] I. Goldberg, B. Ustaoğlu, M. D. Van Gundy, and H. Chen, "Multi-party Off-the-Record Messaging," in *Conference on Computer and Communications Security*. ACM, 2009, pp. 358–368.
[103] M. Van Gundy, "Improved Deniable Signature Key Exchange for mpOTR," 2013, available from http://matt.singlethink.net/projects/mpotr/improved-dske.pdf.
[104] R. Kainda, I. Flechais, and A. W. Roscoe, "Usability and Security of Out-Of-Band Channels in Secure Device Pairing Protocols," in *Symposium on Usable Privacy and Security*. ACM, 2009.
[105] B. Warner. (2014) Pairing Problems. [Online]. Available: https://blog.mozilla.org/warner/2014/04/02/pairing-problems/
[106] (2015) (n+1)sec. [Online]. Available: learn.equalit.ie/wiki/Np1sec
[107] M. Abdalla, C. Chevalier, M. Manulis, and D. Pointcheval, "Flexible Group Key Exchange with On-Demand Computation of Subgroup Keys," in *Progress in Cryptology–AFRICACRYPT*. Springer, 2010, pp. 351–368.
[108] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," *SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.
[109] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.
[110] J. A. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The BitTorrent P2P File-Sharing System: Measurements and Analysis," in *Peer-to-Peer Systems IV*. Springer, 2005, pp. 205–216.
[111] A. Kapadia and N. Triandopoulos, "Halo: High-Assurance Locate for Distributed Hash Tables," in *Network and Distributed System Security Symposium*. Internet Society, 2008.
[112] Q. Wang and N. Borisov, "Octopus: A Secure and Anonymous DHT Lookup," in *International Conference on Distributed Computing Systems*. IEEE, 2012, pp. 325–334.
[113] M. Backes, I. Goldberg, A. Kate, and T. Toft, "Adding Query Privacy to Robust DHTs," in *Symposium on Information, Computer and Communications Security*. ACM, 2012, pp. 30–31.
[114] M. G. Reed, P. F. Syverson, and D. M. Goldschlag, "Anonymous Connections and Onion Routing," *Selected Areas in Communications*, vol. 16, no. 4, pp. 482–494, 1998.
[115] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," DTIC, Tech. Rep., 2004.
[116] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a Type III Anonymous Remailer Protocol," in *Symposium on Security and Privacy*. IEEE, 2003, pp. 2–15.
[117] S. J. Murdoch and G. Danezis, "Low-Cost Traffic Analysis of Tor," in *Symposium on Security and Privacy*. IEEE, 2005, pp. 183–195.
[118] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-Resource Routing Attacks Against Tor," in *Workshop on Privacy in the Electronic Society*. ACM, 2007, pp. 11–20.
[119] N. S. Evans, R. Dingledine, and C. Grothoff, "A Practical Congestion Attack on Tor Using Long Paths," in *Security Symposium*. USENIX, 2009, pp. 33–50.
[120] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website Fingerprinting in Onion Routing Based Anonymization Networks," in *Workshop on Privacy in the Electronic Society*. ACM, 2011, pp. 103–114.
[121] A. Langley. Pond. [Online]. Available: https://pond.imperialviolet.org/
[122] D. Boneh, X. Boyen, and H. Shacham, "Short Group Signatures," in *Advances in Cryptology–CRYPTO*. Springer, 2004, pp. 41–55.
[123] D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, and H.-C. Wong, "Secret Handshakes from Pairing-Based Key Agreements," in *Symposium on Security and Privacy*. IEEE, 2003, pp. 180–196.
[124] M. Waidner and B. Pfitzmann, "The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability," in *Advances in Cryptology–EUROCRYPT*. Springer, 1989.
[125] S. Goel, M. Robson, M. Polte, and E. Sirer, "Herbivore: A Scalable and Efficient Protocol for Anonymous Communication," Cornell University, Tech. Rep. TR2003-1890, 2003.
[126] P. Golle and A. Juels, "Dining Cryptographers Revisited," in *Advances in Cryptology–EUROCRYPT*. Springer, 2004, pp. 456–473.
[127] C. C. Head, "Anonycaster: Simple, Efficient Anonymous Group Communication," 2012, available from https://blogs.ubc.ca/computersecurity/files/2012/04/anonycaster.pdf.
[128] H. Corrigan-Gibbs and B. Ford, "Dissent: Accountable Anonymous Group Messaging," in *Conference on Computer and Communications Security*. ACM, 2010, pp. 340–350.
[129] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson, "Dissent in Numbers: Making Strong Anonymity Scale," in *Conference on Operating Systems Design and Implementation*. USENIX, 2012, pp. 179–182.
[130] E. Syta, H. Corrigan-Gibbs, S.-C. Weng, D. Wolinsky, B. Ford, and A. Johnson, "Security Analysis of Accountable Anonymity in Dissent," *Transactions on Information and System Security*, vol. 17, no. 1, p. 4, 2014.
[131] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford, "Proactively Accountable Anonymous Messaging in Verdict," arXiv e-prints, Tech. Rep. arXiv:1209.4819, 2012.
[132] B. Project. Bitmessage. [Online]. Available: https://bitmessage.org/
[133] L. Sassaman, B. Cohen, and N. Mathewson, "The Pynchon Gate: A Secure Method of Pseudonymous Mail Retrieval," in *Workshop on Privacy in the Electronic Society*. ACM, 2005, pp. 1–9.
[134] N. Borisov, G. Danezis, and I. Goldberg, "DP5: A Private Presence Service," CACR, Tech. Rep. 2014-10, 2014.