Interoperability in End-to-End Encrypted Messaging

Julia Len
Cornell Tech

Esha Ghosh
Microsoft Research

Paul Grubbs
University of Michigan

Paul Rösler
FAU Erlangen-Nürnberg

Abstract

The Digital Markets Act (DMA) is a nascent European Union regulation adopted in May 2022. One of its most controversial provisions is a requirement that so-called "gatekeepers" offering end-to-end encrypted messaging apps, such as WhatsApp, implement "interoperability" with other messaging apps: in essence, encrypted messaging across service providers. This requirement represents a fundamental shift in the design assumptions of existing encrypted messaging systems, most of which are designed to be centralized. Technologists have not really begun thinking about the myriad security, privacy, and functionality questions raised by the interoperability requirement; given that the DMA's interoperability mandate may take effect as soon as mid-2024, it is critical for researchers to begin understanding the challenges and offering solutions.

In this paper, we take an initial step in this direction. We break down the DMA's effects on the design of encrypted messaging systems into three main areas: *identity*, or how to resolve identities across service providers; *protocols*, or how to establish a secure connection between clients on different platforms; and *abuse prevention*, or how service providers can detect and take action against users engaging in abuse or spam. For each area, we identify key security and privacy requirements, summarize existing proposals, and examine whether proposals meet our security and privacy requirements. Finally, we propose our own design for an interoperable encrypted messaging system, and point out open problems.

1 Introduction

End-to-end encrypted (E2EE) messaging has, in the last decade, become perhaps the most widely-used privacy technology in the world. Well over two billion people use some form of E2EE messaging to communicate privately. Most of these use a centralized E2EE messaging app run by a single service provider, such as WhatsApp or Signal. These service providers manage user identities, forward traffic between users, and detect and prevent harmful content on the platform, like abuse and spam. Generally, users of these apps can only message other users of the same app—e.g., a WhatsApp user cannot message someone on Signal.

In May of last year, the European Union adopted a regulation called the Digital Markets Act (DMA). Its focus is to reduce what regulators view as unfair and anti-competitive behavior by large tech companies. The DMA places a particular emphasis on reducing the extent to which large "gatekeeper" companies can "lock in" users to their ecosystems, and exploit network effects to make switching costs high and deter users moving to other services. Among its provisions is a rule that gatekeepers who operate E2EE messaging services must make them interoperable with E2EE messaging from non-gatekeeper providers. So, for example, if Signal, a non-gatekeeper, requests to interoperate with WhatsApp, Signal users would be able to message WhatsApp users without making a WhatsApp account, or indeed an account on any Meta service.

Almost as soon as the interoperability mandate was announced, a loud, polarized, and often acrimonious debate [39, 47] began online between supporters of the mandate and its opponents. What both sides generally agree on is that this interoperability mandate represents a fundamental change to the basic centralized architecture of today's E2EE systems.

But how will these interoperable E2EE messaging systems look, exactly? What new security and privacy risks could be introduced by interoperability across service providers? How do the details of the DMA's requirements affect the design space? These and other crucial questions lack clear answers today; since the mandate may take effect as soon as mid-2024, it is critical to answer them as soon as possible.

Our contributions. In this work, we begin the formal study of interoperable end-to-end encrypted messaging. We begin with a detailed analysis of Article 7 of the DMA, which describes the interoperability mandate. Our analysis points both to places where requirements are very clear—for example, that end-to-end encryption must be preserved in interoperable chats—and places where requirements are ambiguous.

With this analysis, we then identify the components of existing centralized E2EE messaging systems that will be affected the most. We find that the DMA's mandate will affect three components in particular: (1) identity systems, (2) E2EE protocols, and (3) abuse prevention. For each component, we explain why the DMA will affect it, give its API in an interoperable setting, explain the unique security and privacy challenges that arise because of interoperability, and briefly survey some early proposals for implementing interoperability. Finally, we propose our own design for interoperable versions of these three components. We also highlight open problems we believe need further research.

General findings. Our study of interoperability uncovers many interesting challenges that are unique to specific components, such as identity or abuse prevention; these are described in the next few paragraphs. Our study also uncovers other, more general, facts about interoperable E2EE.

The first is that there are two basic architectures for interoperable E2EE: client-to-server, where a client of one provider directly communicates with another provider, and server-to-server, where the client's own provider helps them communicate with other providers. We explain the security and privacy tradeoffs of these two designs in Section 3; ultimately, we believe server-to-server is the right design.

Our second finding is that strong privacy for communication metadata is easier to achieve in interoperable E2EE than in non-interoperable settings. Intuitively, this is because each client's server can act as an anonymizing proxy, hiding their identity from the other provider.

Identity. To set up a session in E2EE messaging, the sender and receiver must agree on each others' long-term identity keys to perform mutual authentication. This process is what we refer to as the identity component of the E2EE messaging system, which we examine in further detail in Section 4. A key finding of ours is that a new service will need to be incorporated into interoperable identity systems: service discovery, which will provide a means for users to learn which apps they can use to message other users. (Service discovery is irrelevant in centralized E2EE messaging, as there is only one app.) We note that although this component is not explicitly required by the DMA, we deem this functionality important for the end-to-end interoperable chat pipeline. We then explore the second major step of identity, which is how a user retrieves the recipient's key material.

Protocols. Once the sender and receiver learn each others' identity keys, they can initiate a secure session and begin exchanging data; we refer to this as the "protocols" component. In Section 5 we discuss how to interoperate protocols. The challenge here is readily apparent: if two users are on different apps that do not "speak" the same protocol, they cannot hope to establish a secure session. We describe three possible solutions to this problem: (1) clients either know the protocol of every app their app interoperates with, and they choose the appropriate one when making an interoperable session, (2) clients are protocol-agnostic, and servers translate between protocols by terminating connections across providers, or (3) all interoperable apps agree on a common protocol for interoperability, such as MLS [11]. A key observation we make is that the text of the DMA seems to specifically preclude option (2), which has been suggested several times [30, 68]: it requires end-to-end encryption to be preserved for interoperable chats. In Appendix B, we also point to a number of interesting subtleties in the case of interoperable group chats.

Abuse prevention. If the sender's message is unwanted by the receiver, either because it is annoying, a scam, or even threatening, the receiver may want to prevent the sender from sending them another message. If the sender sends unwanted messages to enough people, the platform may detect them as a spammer and prevent them from sending more messages. Collectively, we refer to this set of features of an E2EE messaging system as "abuse prevention". In Section 6 we explore interoperable abuse prevention. Our analysis of the DMA's text uncovers an important fact about abuse prevention, namely that the DMA gives gatekeepers the explicit right to protect their users from harmful content coming from other platforms. Thus, interoperable abuse prevention is not only advantageous, it is likely to be required in practice, given the magnitude of the spam problem on existing E2EE messaging [1]. We also identify an interesting challenge with interoperable abuse prevention, namely how to enable functionality like cross-platform reporting and blocklisting without revealing metadata across platforms.

Our interoperable messaging design. We use the API we define for identity, protocols, and abuse prevention to propose a design for two-party interoperable E2EE messaging in Section 7. At a high level, our design introduces a third-party server to handle service name discovery, to which clients can query to learn target recipients' preferred apps when initiating new communication sessions. Upon finding the service provider of the recipient, a message sender can retrieve the recipient's key material by encrypting the recipient's identifier to the recipient provider, thereby hiding the recipient's id from the sender's service provider. Furthermore, our design makes use of a sender-anonymous E2EE protocol to hide the sender identity from the recipient provider during the established secure session. Hiding the recipient identity from the sender's provider is even easier in our protocol: the sender can simply encrypt the recipient's identity to the recipient's provider. We also identify an interesting benefit of the interoperable setting over centralized E2EE messaging: sender-anonymity that is resistant to network traffic analysis comes naturally to interoperability because providers are limited to seeing only partial traffic. Finally, our design introduces mechanisms to address spam filtering, user reporting, and blocklisting.

We highlight that we neither describe an existing protocol

for interoperability (ours is novel) nor do we propose a protocol meant for standardization. Instead, the goal of this work is to describe how the existing components of widely used E2EE messengers can and should be extended to fulfill the requirements of the DMA. Our interoperable messaging design therefore incorporates elements of messaging that are necessary (e.g., key distribution, metadata hiding, blocklisting) and those that are optional but useful (e.g., service discovery, verifiable user reporting).

Related work and concepts. There has been some initial interest in interoperability from the IETF, who have convened a new working group, More Instant Messaging Interoperability (MIMI) [45], to standardize protocols for interoperability. This group's focus is on specific protocols for a general notion of interoperable E2EE messaging, whereas in this work we try to build a holistic view of the problem space for interoperability as specifically defined by the DMA. There have also been analyses that provide a high level view of open problems and challenges with the DMA as it relates to interoperable E2EE messaging, such as that by Brown [21].

One concept related to interoperability is third-party clients, where users interact with a service provider via a client other than the "official" one distributed by the provider itself. Third-party client usage varies widely across E2EE apps—for example, Telegram officially sanctions third-party clients, whereas WhatsApp actively tries to stop the distribution of third-party clients [60, 66]. Third-party clients are distinct from interoperable E2EE because they do not allow sending messages across providers.

Another concept related to interoperable E2EE is decentralized messaging, in which messages are routed between clients without the use of a centralized service provider. Examples include Briar and Bridgefy [2, 19]. Decentralized E2EE apps sometimes use a distributed ledger or blockchain to manage metadata and identities. Interoperable E2EE is incomparable to decentralized E2EE because centralized service providers still exist in interoperable E2EE, and messages are still routed between users by centralized service providers.

Finally, one messaging architecture that could be used for interoperability is federated messaging, such as Matrix [43]. In federated messaging, users register with and are based at a "home" server, but can interact with users on other servers via some common protocol. Servers connected together via this protocol need not be controlled by a single party. Notably, this is the approach that has thus far been recommended by MIMI.

However, we note that federation is not the only solution for interoperability and is not required by the DMA. Furthermore, techniques and design patterns from federated messaging might not fit neatly into the model for interoperable E2EE messaging as defined by the DMA. First, in federated settings typically all servers are equal participants in the federation; in interoperable E2EE, gatekeeper servers have different roles in the protocol than non-gatekeepers. Also, federated E2EE apps are designed from the ground up to be federated, and have only a single protocol that everyone uses; in contrast, interoperability as defined by the DMA means connecting systems that were not designed to be connected, and that have two distinct ways of being used: within-platform and across platforms.

Identity management presents another case where it is unclear whether federation is a practical solution for DMA interoperability. In a federated system like Matrix, for instance, different servers must adhere to a common convention for identity—for example, everyone is identified by their username at their home server URL. In contrast, interoperating E2EE identity means using different kinds of identifiers, such as usernames and phone numbers, for which there is no obvious common semantics.

2 Digital Markets Act (DMA) Background

The DMA was adopted by the EU parliament in May 2022, after having been introduced roughly two years earlier. The law is focused on "fair markets in the digital sector"; specifically, ensuring that digital markets remain competitive even in the presence of large players that control a majority of the market. These large players have a special designation—"gatekeepers"—under the DMA. The DMA defines certain conditions to test whether a company is a gatekeeper; the details are not relevant for this work.

The DMA can be understood as a series of requirements gatekeepers must abide by if they want to conduct business in the EU. These requirements govern the way gatekeepers offer services to their customers. They also prevent gatekeepers from compelling users to interact with gatekeeper services in a certain way—for example, forcing users to make an account with one product to use another for the same gatekeeper. For this work, the most important part of the DMA is Article 7, which describes the interoperability requirement. We quote the full text of Article 7 in Appendix C.

Interoperability requirements for gatekeepers. Article 7 requires that gatekeepers who provide "number-independent interpersonal communication services" [48] (NIICS) interoperate their NIICSs with third parties upon request and free of charge. The legal definition of an NIICS relies upon EU precedent [48]; the reader can safely understand an NIICS as a means for two or more people to communicate directly that (a) is not operated by a traditional phone company—hence "number-independent" and (b) enables an interactive and interpersonal exchange of information. So, for example, WhatsApp is an NIICS provided by the gatekeeper Meta; in contrast, the Facebook social networking platform is not an NIICS. The most important paragraph is paragraph 3. We quote it here verbatim:

"The level of security, including the end-to-end encryption,

where applicable, that the gatekeeper provides to its own end users shall be preserved across the interoperable services."

Paraphrasing, this paragraph says that if the gatekeeper operates an end-to-end encrypted NIICS for its own end users, and the third party requesting interoperation also provides end-to-end encryption, then cross-platform messages must be end-to-end encrypted. (That the regulation goes out of its way to mention end-to-end encryption is important to note; we will return to this point later.) Notably, the paragraph also refers to the "level of security," which could refer to specific properties of the E2EE protocol, such as forward secrecy.

Article 7 contains some language that specifies how interoperability requests between providers work, and a paragraph that requires users on both providers to be able to opt out of using interoperable messaging. The regulation makes two more statements that are important for us. The first, in paragraph 8, relates to what information the gatekeeper can learn about end users to be able to implement interoperability. Specifically, it says that the gatekeeper can collect personal information of end users, and exchange it with the third-party provider, only if it is "strictly necessary" to implement interoperability. This suggests that data and metadata about users and communications should be protected wherever possible.

The second statement, in paragraph 9, says the gatekeeper cannot be prevented from protecting its own platform from third-party providers who may endanger the "integrity, security, and privacy of its services." (The statement says that measures taken must, as above, be "strictly necessary.") We interpret this paragraph as allowing gatekeepers to implement abuse prevention (e.g. spam filtering) for interoperable traffic.

Requirements for interoperable functionality. Our paper focuses on one particular kind of NIICS functionality: basic chat messaging and media sharing between two parties. We note briefly that Article 7 will also require interoperability for other NIICS functionality in the future. Two years after the initial implementation of the DMA, interoperating group chats will be required; two years after that, interoperating voice and video calls will be required as well. E2EE must be preserved in all cases. To keep the scope of this paper manageable we focus on the first set of requirements and leave it to future work to extend our framework to the case of group, voice, and video chats.

Important questions. Some things in Article 7 are very clear: for example, E2EE must be preserved, transfer of personal data across providers must be minimal, and gatekeepers must be able to protect their own users from harm coming from other providers. The scope of interoperability requirements is also fairly clear: paragraph 2 spells out which NIICS features must be interoperated. A few important points are left ambiguous by the text. We summarize here the ones that affect the design of interoperable systems.

First, the phrase "strictly necessary" occurs in several places in Article 7. What this actually means is presumably left up to implementors. However, deciding what really is "strictly necessary" has important implications for the security and privacy of interoperable systems, since that phrase governs both the amount and type of user data that can be exchanged between service providers (paragraph 8) and what measures gatekeepers can take to protect their own users from harms from third-party providers (paragraph 9). Indeed, one of the goals of this paper is to give an initial answer of what is "strictly necessary," to inform the discussion and debate between gatekeepers and regulators.

A related question is the meaning of "level of security" in paragraph 3. It is unclear what kinds of features pertain to the "level" of security (e.g., do disappearing messages count?); also unclear is how fine-grained the measurement is (e.g., does forward-secrecy count as more secure?).

Other forms of interoperability. While this section, and the remainder of this paper, focuses on interoperability as required by the DMA, we want to briefly discuss other examples of interoperability in the E2EE ecosystem. Recently, a few client-facing apps have been created with the express purpose of interoperating messaging. These apps include Beeper, Texts, and Mio [12, 46, 59]. All claim to support interoperating widely-used apps like WhatsApp and iMessage, but little information about their implementation is publicly available. All seem to require the user to have accounts on all services, so are unlikely to satisfy the DMA.

3 Overview

The DMA's Article 7, as described above, does not explain how any of its requirements could, or should, be implemented. If we want to understand the possible security and privacy implications of interoperability, understanding the things that must change in an interoperable setting, and what the design space of those changes is, is the first step.

Because end-to-end encryption must be preserved, two components of the E2EE messaging system are implicated immediately: first, the *identity* component must be involved, so that users can acquire other users' identity keying material for cross-provider chats. We discuss interoperable identity in Section 4. Second, the *protocol* itself must change so that a secure session can be established across service providers, even if the providers use different protocols for intra-provider sessions. Section 5 discusses interoperable protocols. A third system component that must be changed is *abuse prevention*: since paragraph 9 gives gatekeepers the right to protect their users from abuse from third-party providers, there must be a way of (e.g.) detecting spam messages sent across providers, and giving users the ability to block users on other providers. Section 6 covers interoperable abuse prevention.

In each of these three sections, we explain what pieces of the component would change and give an API that specifies the logical input/output behavior of an interoperable solution

for the given component. We also highlight security and privacy properties needed for each solution in an interoperable setting. Finally, we give a brief overview of solutions that have been proposed (if any), and study how (or if) they achieve the security and privacy properties we identify.

Notation. We assume communication between a clearly-defined sender and recipient through their service providers, with the following notation. We refer to the user U_{sid} that sends the message as the sender and denote sid as the sender's identity on its own provider, which is a unique identifier string (e.g., an email or phone number) used by the provider. Likewise, the recipient U_{rid} is the user that receives the message and rid denotes the recipient's identity on its provider. We refer to the identifier for some user U in which it does not matter whether they are the sender or the recipient as uid.

We refer to the sender's provider as SP, the recipient's provider as RP, and a generic provider as P. We assume SP and RP function both as identity providers and message delivery servers. This means they also maintain key directories that map usernames to their public key bundles.

Threat model and assumptions. We provide an independent threat model for each component. This approach is a standard one in E2EE messaging, where entities can have different capabilities. For instance, when considering the protocol, we might assume a provider can be malicious because its goal is to learn user conversations; in contrast, we assume that for abuse prevention the provider behaves honestly, since it has little reason to deviate from its own platform policies. We make the following assumptions throughout our paper. We assume each provider has an identity certificate signed by some trusted certificate authority, and that each provider's certificate is publicly available. We also assume that each user we describe has an account with one or more providers.

A meta-question: communication patterns. Before we go into more detail about the design of the three different components, we want to highlight and discuss an overarching design question about interoperable messaging: how the sender interacts with the recipient's service provider and vice versa. There are two basic ways this interaction could happen, each with tradeoffs: either U_{sid}'s client interacts with RP (perhaps even as an RP client would)—we call this "client-to-server"—or U_{sid} interacts with RP *through* the sender provider SP—we call this "server-to-server". We highlight the difference between these two architectures in Figure 1.

We base our component designs on the server-to-server architecture, which we argue is the most realistic to deploy. For example, this design minimizes changes to clients, which continue to communicate only with their provider as before. This also results in easier authentication techniques for interoperable traffic—the trust relation between providers means that clients do not need to handle authenticating their messages when sending them to another provider. Furthermore, server-to-server interactions seem to make implementing abuse prevention measures like spam detection easier, since SP can use the metadata of interoperable traffic to find and ban spammers on its platform. Finally, as we expand on in Section 5, server-to-server has the counterintuitive property that hiding communication metadata from the providers comes essentially for free. Specifically, by encrypting the receiver's identity with RP's public key, the sender can hide the receiver from SP (and vice versa); since the providers transfer messages, network-level metadata like IP addresses of users is not revealed either. This is strictly better than what existing centralized E2EE apps can do: while techniques like Signal's Sealed Sender can hide application-layer metadata, they cannot also hide network metadata.

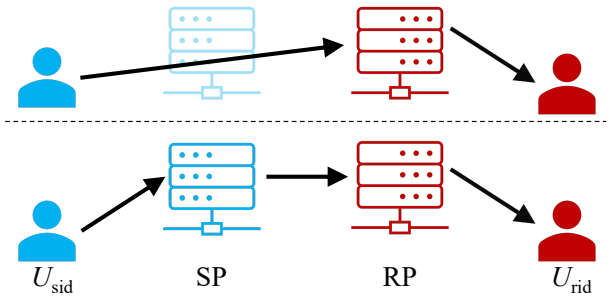

Figure 1: (Top) The client-to-server architecture for interoperable message communication. (Bottom) The server-to-server architecture. U_{sid}, SP, RP, and U_{rid} are defined as in Section 3.

4 Identity Discovery and Interoperability

Consider a very basic E2EE chat scenario: user Alice, who uses app AliceChat, wishes to chat with user Bob, who is on BobChat. To enable a cross-provider encrypted channel of communication, two important steps need to happen: (1) Alice needs to discover which app Bob is using, and (2) Alice's client app will need to obtain Bob's keying material from BobChat. We identify these steps as the two fundamental problems for identity interoperability:
(1) *Service Name Discovery* (SND): the way the sender discovers the preferred app name of the recipient without interacting with them directly.
(2) *Retrieving Keying Material* (RKM): the way the sender retrieves the keying material of the recipient from the recipient's preferred app.

While Alice could obtain the name of Bob's provider (BobChat, in our example) through some out-of-band mechanism such as talking to Bob directly over a different channel (which is sufficient based on the DMA requirements), it could be that Alice does not have the means to do so. The focus of SND, therefore, is in formalizing this step as an automated mechanism executed by a client and understanding its desir-

able security and privacy features.

4.1 Service Name Discovery (SND)

The most natural solution that comes to mind, when thinking of SND, is a directory service that maintains a mapping between user identifiers and the list of app names (corresponding to the apps they are registered on). We could imagine a server maintaining this directory, responding to lookup queries and updating the directory on appropriate (i.e., authorized) update requests. We build SND on this intuitive notion. SND is a primitive that is executed between the querying party (sid) and a server, which we denote as the Service Directory Server (SDS). SDS is a logical server, distinct from SP and RP (but could be implemented in a distributed way) that maintains a directory mapping user identifiers to their preferred messaging apps, along with their preference number pref.

User identifier collisions. Before discussing the API for SND, let us highlight a subtle issue that arises in maintaining this directory. In the current E2EE centralized ecosystem, each service provider uses its preferred user identifier (e.g., phone number [65], email address [41], or random string [49]). While the identifier is unique for a given service provider, there is no global namespace where each identifier is unique. In SND, we strive to develop an API (and, consequently, a protocol) that allows for any identifier chosen by the corresponding provider without enforcing a global namespace. We believe this will lead to the least amount of friction in adoption as compared to developing a new globally unique namespace such as that for email identifiers, which reveal the name of the service provider by default.

This leads us to handling the issue of user identifier collision at the API layer, where there can be such collisions across different apps. For instance, a user can have accounts on multiple apps with the same identifier, or different users can have the same identifier across multiple apps. We model this in our API by letting SDS maintain a multi-map for the directory and report the list of apps in response to a given query. We note that this is common on social media platforms, where a search returns all profiles that match a queried username.

API. We now describe the SND API. We assume that the connection between the client and the server happens over an encrypted and authenticated channel (i.e., over TLS).

- $\mathsf{SND.UpdateEntry}(\mathsf{st_{SDS}}, \mathsf{uid}, \mathsf{pref}, \mathsf{app}, b, \mathsf{aux}) \rightarrow 1/0$: The SDS receives its input from U_{uid}, runs this algorithm locally, and responds with a bit indicating success or failure. In more detail, the SDS receives a request from U_{uid} to add or remove app to their list of apps stored in the directory along with some authorization token (from their corresponding provider) in aux. If the authorization is valid, the SDS does the following. If $b = 1$, then the SDS adds app with preference pref. If $b = 0$, then the SDS removes app from their existing list of apps. Once the request has been successfully processed, the SDS responds with a success bit (failure otherwise).
- $\mathsf{SND.Lookup}^{\mathsf{Auth}(\mathsf{st_P})}(\mathsf{st_{uid}}, \mathsf{query}\ ;\ \mathsf{st_{SDS}}) \rightarrow (\mathsf{resp}\ ;\ \perp)$: We model lookup as an interactive algorithm where the client U_{uid} requests query to the SDS. The parties also have access to U_{uid}'s provider P to authenticate U_{uid} as a registered user of P. This is to ensure that no arbitrary party can run Lookup with the SDS and extract the entire directory. At the end of the protocol execution, U_{uid} receives response resp while SDS has no output.

Basic Threat Model. The **servers** P and SDS are considered semi-honest. They are assumed to not collude and to faithfully execute the protocol, but are not trusted for privacy. The **clients** are considered malicious—they can try to learn the entire SDS directory via enumeration, or try to forge updates.

Security Goals. First we enumerate the basic security and privacy requirements of a SND and then discuss some possible extensions.

- *Unforgeability of authorization mechanism*. We require that no unauthorized user should be able to change the SND database. In other words, no PPT adversary should be able to forge a service provider's authorization mechanism except with negligible probability.
- *Privacy of* uid *from* SDS. We require that identifier uid of the user U_{uid} making the query should not be revealed to the SDS. This, in turn, implies that the transcript of communication of the SDS with $\mathsf{Auth}(\mathsf{st_P})$ and U_{uid} should contain no information about uid.
- *Privacy of* query *from* P. We also require that query is hidden from P. Since P already knows uid, i.e., identity of the querying party, if P learns for which identifier U_{uid} is querying, then P can learn sensitive information about with whom U_{uid} is initiating communication. So, the privacy property requires that the social network of U_{uid} is hidden from the P.

Privacy Discussion. Given that SND employs a centralized oracle service, it is important to highlight some of the privacy implications with this component. In particular, mitigation that prevent attackers from scraping the database stored by the SDS would need to be deployed, such as rate limiting. Note, however, that the authentication mechanism we require during Lookup naturally provides some rate-limiting: it prevents arbitrary parties from querying the SDS without authorization from a valid service provider. In addition, because the service reveals all apps used by a particular user during a valid query, users would need to be careful in what apps they add to the SDS by refraining from adding any sensitive apps on which they might not want to be publicly discoverable.

4.2 Retrieving Key Material (RKM)

RKM is the primitive used for retrieving the keying material of the user U_{rid}. Recall that after a user learns their target recipient's app, which can happen either through SND or some out-of-band mechanism, this is the next step in forming the handshake for establishing a secure channel. As discussed in Section 3, we model this using the server-to-server architecture.

- $\mathsf{RKM.Init}(\mathsf{st_{sid}}, \mathsf{rid}, \mathsf{SP}, \mathsf{RP}) \to (\mathsf{st_{sid}}, q)$: This algorithm is run by U_{sid} to prepare the query packet for retrieving U_{rid}'s keying material from RP. U_{sid} prepares the query packet q which it forwards to its own provider, i.e., SP. It also updates its state in case it needs to open a session for this query (which it will close upon receiving the corresponding response back from SP).
- $\mathsf{RKM.Lookup}(\mathsf{st_{SP}}, q \ ; \ \mathsf{st_{RP}}) \to (\mathsf{resp} \ ; \ \perp)$: We model RKM.Lookup as an interactive algorithm between providers SP and RP. SP gets an input q and (possibly processes and) forwards it to RP. RP parses the query, prepares response resp, and sends it back to SP.
- $\mathsf{RKM.Reconstruct}(\mathsf{st_{sid}}, \mathsf{resp}) \to (\mathsf{st_{sid}}, \mathsf{km})$: U_{sid}, which runs this algorithm locally, parses the response from SP to extract key material km for U_{rid} (which can be $\perp$ if the extraction fails). U_{sid} updates its state to reflect any change in possibly ongoing sessions (i.e., close the appropriate session).

Basic Threat Model. The servers SP and RP are considered semi-honest. They are assumed to not collude and to faithfully execute the protocol, but they are not trusted for privacy. The clients are considered malicious. This means they may not follow the protocol and can try to send ill-formed query packets.

Security Goals. Even though clients can be potentially malicious in our threat model, there is not much they can achieve beyond sending ill-formed queries. This can be cheaply mitigated by having the providers check for well-formedness of queries before processing them; we therefore do not make it an explicit security goal.

- *Privacy of* sid *from* RP. We require that no single provider can learn both sid and rid since that would reveal the social graphs of U_{sid} and U_{rid} to the providers. This means that since RP already knows rid, we require that RP does not learn sid even when U_{sid} initiates an RKM.Lookup for rid.
- *Privacy of* rid *from* SP. Likewise, SP already knows sid. We therefore require that SP does not learn rid, even though it contacts RP to retrieve U_{rid}'s key material.

Comparing to centralized E2EE. It is worth noting here that in the current ecosystem of centralized providers, the provider naturally learns the social graphs of its users. However, the decentralization enabled by interoperability points to an interesting benefit of this setting: we can aim for the strong privacy property of hiding the social graph from providers.

4.3 Current Proposals

In this section, we describe existing proposals for identity discovery and interoperability.

Introducing a global namespace. Rescorla [54] suggests two possible ways to introduce a global namespace: (1) introducing a hierarchical namespace or (2) introducing an unqualified namespace where any user identifier can be attached to any service (requiring a PKI where some roots of trust will have to attest to user identities and their public keys in a uniform manner.). In proposal (1) the uid will be fully qualified, so SND will not be needed any more. But proposal (2) will require SND to find out which service a certain identity is using. Both these proposals will need to implement RKM.

Creating uids with each provider. Rescorla [54] also proposes an alternative where each provider can continue using their identity system, but the users would need to create an identifier with each provider, which already violates the spirit of interoperability. If each user has an account on each of the E2EE providers, SND may not be required any more. However, it is unclear which uid a certain user should use for discovering/retrieving key material for another user on a different provider. RKM will still be needed if such a proposal is implemented.

Phone numbers as globally unique uids. A second line of proposals [53, 55] suggest using phone numbers as universal identifiers. Rescorla [53] proposes using a centralized directory service for identity mapping where each phone number can be mapped to the list of apps it uses. This is very close to our SND proposal where the centralized directory can be directly mapped to our SDS, but violates our security and privacy goals (SDS should not learn the social graphs of the users, since an unauthorized user can scrape this directory to find out which app(s) a certain user is using). Rosenberg et al. [55] proposed their protocol SPIN as an alternative to replace the centralized server by having each client do its own phone number-to-app mapping via SMS. This is a complex protocol that will require OS vendors to implement new APIs. But most notably, with SPIN, a user will not be able to discover other users who are not online at the same time, which will be a significant problem for asynchronous E2EE chats. Viewed through the lens of SND and RKM, in SPIN the SDS is implemented in a distributed way and the clients run SND.Lookup and RKM.Lookup directly with the (distributed) SDS and the RP. Finally, there has been some discussion about using Decentralized Identifiers (DIDs) as a globally unique identifier for discoverability [25].

5 Protocol-Layer Interoperability

Communication between users will be done by an (interoperable) messaging protocol. To understand requirements for such a protocol, we specify the parties' APIs for sending and

delivering messages, list established security goals and define extended properties that we derive from our interpretation of the DMA, and discuss available tools.

5.1 API

We specify the API for the protocol of sending and delivering messages, which we refer to as PRO, as generically as possible—focusing on a server-to-server architecture (see Section 3). This is illustrated with the following protocol flow:

- $\mathsf{PRO.Send}(\mathsf{st}_{\mathsf{sid}}, m, \mathsf{rid}, \mathsf{RP}, aux) \to (\mathsf{st}_{\mathsf{sid}}, c)$: Executed locally by user U_{sid} to compile ciphertext c to user U_{rid}, whose intended Recipient Provider is RP, taking possibly auxiliary information such as authorization credentials.
- $\mathsf{PRO.Forward}(\mathsf{st}_{\mathsf{SP}}, c, \mathsf{RP}, aux) \to (\mathsf{st}_{\mathsf{SP}}, 1/0)$: SP receives a request to deliver the ciphertext c and either forwards it to Recipient Provider RP or rejects c (q.v. §6).
- $\mathsf{PRO.Deliver}(\mathsf{st}_{\mathsf{RP}}, c, aux) \to (\mathsf{st}_{\mathsf{RP}}, 1/0)$: Recipient Provider RP obtains the (processed) ciphertext c from SP to internally process and deliver or reject it (q.v. §6).
- $\mathsf{PRO.Receive}(\mathsf{st}_{\mathsf{rid}}, \mathsf{SP}, c) \to (\mathsf{st}_{\mathsf{rid}}, m)$: Recipient U_{rid} receives ciphertext c from its provider RP to recover plaintext message m. It also takes as input (an identifier of) the Sender Provider SP to help with authenticating the sender.

5.2 Security and Privacy Properties

We divide the security properties into *basic concepts* and *more sophisticated extensions*. The basic concepts are achieved by the majority of messaging apps for two-party communication. Furthermore, the DMA clearly states that these basic concepts must be carried over from ordinary messaging to the interoperable setting. The extended properties are also achieved by many popular messaging apps, and achieving them in an interoperable setting is in some cases not difficult. It is not clear if such properties are covered by paragraph 3 of the DMA.

Basic Threat Model. All **users** are considered malicious except for those who a user expects to have a secure communication with. All involved **servers** are considered malicious.

Security Goals. The first goal is **Authorized Messaging**: only registered users can send messages to other providers' users; second, we want **Confidentiality**: messages can only be read by their senders and recipients; third, **Authenticity**: messages as well as their metadata (e.g., sender identity) cannot be altered in transmission. Because of the DMA's requirement to limit data collection and exchange across providers to what is strictly necessary (see paragraph 8), we also include two metadata hiding goals. First, we want **Sender Privacy**: providers learn as little information about senders as possible (e.g., RP learns sid infrequently or not at all); second, we want **Recipient Privacy**: providers learn as little information about recipients as possible (same as above for SP learning rid).

With our protocol proposal in Section 7, we demonstrate that interoperable messaging naturally supports metadata hiding, in particular sender privacy and recipient privacy.

Extended Threat Model: Device Corruption. Beyond these basic security goals with respect to the above weak adversary model, we consider temporary corruption of user devices a realistic threat: all **user devices** may be corrupted temporarily. That means, for a limited time, all stored secrets can be exposed and all random coins can be sampled adversarially. This is a standard assumption in messaging literature [6,10,14,15,17,22,23,28,52,56,57]. Extra considerations exist for the setting of groups. Due to space limitations, our discussion of the group setting appears only in Appendix B.

Extended Security Goals. An extension of our basic security goals is to also include **Forward Secrecy** (FS) and **Post-Compromise Security** (PCS). This means we require confidentiality, authenticity, and metadata hiding even if a device is corrupted in the future (FS) and even after a temporary past corruption (PCS). Many messengers that implement variants of Signal's Double Ratchet [50] (e.g., WhatsApp [65], Messenger [41], Matrix [43], Wire [67]) as well as the upcoming MLS standard [11] achieve FS and PCS.

5.3 Possible Solutions

Standard protocols for secure two-party communication can be used for interoperable messaging, but there are important obstacles and open problems that complicate their use. We structure our discussion with the three major components of a protocol: (1) **session initialization** to establish (2) a **secure channel** with which users communicate; this channel also uses (3) **metadata hiding** to preserve privacy.

Session Initialization. Messengers generally use a One-Pass Key Exchange (OPKE) protocol to establish a shared secret between parties. The most common OPKE, called X3DH [40], is based on Signal's protocol stack [36,40,50]. It derives the initial secret via a composition of Diffie-Hellman exchanges between long-term and ephemeral key pairs. For this, users refresh their ephemeral key pairs with the server periodically. A similar KEM-based session initialization is implemented in MLS [11].

Secure Channel. One may consider Signal's Double Ratchet [50] as the de-facto standard secure channel protocol since its variants are implemented in Signal as well as in WhatsApp [65], Facebook Messenger [41], Skype [44], Matrix [43], Wire [67], and many other messengers. Two issues exist for making this the channel for interoperability: incompatible implementations exist (e.g., Signal uses AES-CBC but Wire uses ChaCha20) and some messengers use totally different secure channels (e.g., iMessage and Telegram both use custom protocols). Agreeing on a unified protocol and harmonizing these variations may require a difficult standard-

ization effort. An alternative is using a near-standard like MLS [5, 7, 8, 11, 16, 24, 37, 64].

Given the short timeline of the DMA, it seems likely that gatekeepers will fix a protocol that other apps must "speak". Yet, a global interoperability standard might be an option in the long run, too. Either way, the gatekeepers are required to publish a corresponding protocol documentation in a so-called reference offer that the non-gatekeepers may implement in their app. These implementations serve as so-called *client bridges*. A variant of the client bridge approach would be that the gatekeepers publish a library that implements their protocol along with their reference offer. We note that this solution comes with drawbacks: if a gatekeeper's implementation contains vulnerabilities, then these will be propagated to other interoperating apps, which will have limited power to resolve these themselves.

Metadata Hiding. A simple way to realize metadata protection is via anonymous encryption: essentially, encrypting ciphertexts and recipient information in a nested way. This allows SP to only learn the *sender identity* and destination RP, but not the actual recipient identity; this is also true in reverse for RP. Key-private public-key encryption [13] is a basic tool to achieve this, but more evolved designs of efficient anonymous wrappers are deployed in practice—an example is Signal's Sealed Sender protocol [36]. However, a recent study [63] showed a battery-draining attack by malicious senders against Sealed Sender; fixes are possible with some performance overhead. Achieving FS and PCS with anonymous wrappers can be challenging, but first proposals exist in the literature [28].

Other Issues. An important subtlety with using different protocols for within- and across-provider messaging is the possibility of cross-protocol (aka. protocol confusion) attacks [9, 20, 49]. Care must be taken to enforce proper labelling and domain separation in implementations. Finally, as we discuss in Appendix B, group messaging poses many more open problems that we leave for future work—note that paragraph 2.b of the DMA requires interoperable group messaging only 2 years later than two-party chats.

6 Abuse Prevention

As discussed above, the DMA gives gatekeepers the explicit right to protect their users from potential abuse coming from third-party providers (Article 7, paragraph 9). This raises questions about what reasonable measures gatekeepers may demand for abuse prevention while still maintaining the DMA's strict user privacy requirements (Article 7, paragraph 8). In this section, we consider how three common abuse prevention mechanisms—server-side spam filtering, user reporting, and blocklisting—could work in an interoperable setting.

Spam filtering. Spam will form a dominant concern for gatekeeper apps, with WhatsApp already citing it as one of the main challenges to adopting interoperability [47]. Providers like WhatsApp [1] combat spam by deploying server-side spam classifiers to block spam from reaching users. Spam classifiers use as features the age of the sender's account, the rate of messages being sent, and other reputational features of the account. Once an account is classified as a spammer, the platform can block it. Making a new account is possible but can be expensive—e.g., on WhatsApp, banned users would need to get new phone numbers.

These methods work primarily because existing centralized messaging apps have all the features needed to make accurate spam classifications. Even still, messaging apps struggle to remove spam from their platforms. Adding interoperability to this setting could make this much worse. A notable example of an interoperable system with a severe spam problem is email—spam comprises upwards of 50 percent of sent emails [38]. E2EE spam detection is harder than for emails, since *only* metadata can be used. Available metadata may vary for interoperable chats: for instance, a provider's automatic spam detection might be based on reputational features such as account registration and phone numbers, but this would not work on a message coming from a different provider that does not use phone numbers to identify users. Even if a spammer was blocked on a gatekeeper platform, the spammer could potentially create accounts on other platforms with lower barriers to entry and continue their abuse. Gatekeepers could potentially demand information about users on other providers, such as their identities or account ages, as a necessary condition for protecting their users in accordance with the DMA, but it is possible that this could violate the DMA's Article 7, paragraph 8, which specifies that only "strictly necessary" cross-provider information flow is allowed. Therefore, "strictly necessary" metadata leakage will be determined by technical solutions, and we argue that there exist practical solutions that obviate the need for such leakage.

One option is client-side spam classifiers, used (e.g.) by Google Messages [31]. This is better for privacy but creates storage and compute overheads for clients and has poor UX.

We thus claim that server-side spam filtering will continue to be the main method for spam prevention but can be deployed in a privacy-preserving way such that it still satisfies the requirements of the DMA. Gatekeepers could include an API function for spam filtering and require other providers to run the filter on all messages before sending them to the gatekeeper. This approach presents a major benefit of our server-to-server architecture: the sender provider will know the account information needed to classify spammers and filter messages. Furthermore, this solution shows that any demands by the gatekeeper to learn potentially sensitive identifying information about users on other apps is not in fact "strictly necessary."

User reporting. User reporting, in which users can report

messages they believe constitute spam or harassment to the provider, is another important tool in abuse prevention. In particular, Pfefferkorn found from a survey of companies and organizations on trust and safety techniques that user reporting was the most popular approach [51]. When users report spam, their client often includes information about the sender's identity, such as their phone number, and the message in question, which is the approach taken by iMessage [35] and Google Messages [32].

For user reporting in interoperable messaging, the DMA could enable a gatekeeper to require other providers to reveal the identities of all senders so that the gatekeeper could directly block any reported senders. However, as we argued above, RP should not be able to learn an arbitrary sender identity. Thus, we need to hide an arbitrary sender's identity from RP while still enabling RP to ban malicious users.

Blocklisting. Blocklisting is a common feature of many E2EE apps and presents an interesting challenge when considering interoperable abuse prevention. In particular, we explore how users can block other users across providers. Just as for user reporting, the goal for blocklisting should be that the recipient does not receive messages from specified senders using other providers without the need for RP to learn arbitrary sender identities.

Furthermore, a novel goal introduced in the interoperability setting is how to blocklist the same identity across several apps. For example, imagine Alice has an account on WhatsApp while Bob has accounts on both WhatsApp and Telegram. If Alice blocks Bob on WhatsApp, then Bob could continue harassing Alice on WhatsApp from his Telegram account. Therefore, an extended challenge here is how to detect and prevent such cross-platform Sybil attacks by blocklisting an identity registered across various platforms.

6.1 API

Here we present our API ABP for abuse prevention. It handles spam filtering, user reporting, and blocklisting.

- $\mathsf{ABP.SpFilter}(\mathsf{st_{SP}}, \mathsf{sid}, c, aux) \rightarrow (\mathsf{st_{SP}}, 1/0)$: Sender Provider SP runs the spam filtering algorithm to determine which ciphertexts it sees are spam. It takes as input its state, the sender identifier sid, the ciphertext c to be delivered, and auxiliary information *aux* about the sender's identity for spam classification. It outputs the updated state and a spam/not spam (0/1) bit.
- $\mathsf{ABP.GenReport}(\mathsf{st_{rid}}, \mathsf{sid}, aux) \rightarrow \rho$: Recipient U_{rid} reports a message sent by the sender identified by sid by generating report ρ to send to the recipient provider RP. The algorithm additionally takes as input any auxiliary information to help with the report, such as the messages being reported or an explanation of why the user is being reported.
- $\mathsf{ABP.ReportRcv}(\mathsf{st_{RP}}, \rho) \rightarrow \mathsf{st_{RP}}$: Recipient Provider RP processes user reports by taking as input its state and a report ρ and outputting its updated state.
- $\mathsf{ABP.Block}(\mathsf{st_{rid}}, \mathsf{sid}\ ;\ \mathsf{st_{RP}}) \rightarrow (\perp\ ;\ \mathsf{st_{RP}})$: Recipient U_{rid} blocks U_{sid} by running this interactive algorithm with Recipient Provider RP. U_{rid} takes as input its state and the identifier sid of the user to block and has no output. RP takes as input its state and outputs its updated state.

6.2 Security and Privacy Properties

We now describe the threat model we consider and expand on the needed security and privacy properties.

Threat model. All users are considered malicious and capable of abusive behavior, including spam and harassment. Servers are considered honest-but-curious for abuse prevention – they seek to learn information, including identities, about users but execute all appropriate abuse prevention mechanisms honestly. (As discussed in Section 3, servers have little motivation not to follow their own abuse prevention policies.)

Minimizing metadata leakage. A property we need is preventing providers from collecting sensitive data about users on other providers, (e.g. RP learning the sender identity). This causes a tension with abuse prevention, for which many of the classic mechanisms rely on providers already knowing such data about their users. Gatekeepers may argue this information is "strictly necessary" (q.v.§2) for abuse prevention; we argue in Section 7 that this is not the case and should be disallowed. Our proposed abuse prevention mechanisms hide arbitrary sender identities from RP and vice versa.

Verifiable user reporting with deniability. For user reporting, it is important that the provider can verify a sender actually sent a reported message. This is formalized by [33] as receiver binding. They also formalize *sender binding*, which guarantees that an abusive message sender cannot send an unreportable message. Both properties are needed for interoperable reporting as well. However, a potential issue with user reporting described in [61] is that the reporting protocol may render sent messages non-repudiable by users after device compromise. Thus, user reporting also needs deniability, i.e. only the provider can verify reports. This is a goal of Messenger [41].

6.3 Current Proposals

Proposals for interoperable abuse prevention have thus far been relatively sparse. Element has proposed their "scalable, crowdsourced" moderation tooling approach could work for interoperable communication mandated by the DMA [29]. Their approach is tailored for a decentralized network, not interoperable E2EE. For instance, their suggestion of sharing Access Control Lists across providers breaks our privacy property; it also requires global identifiers for users. Notably, their proposal does not address interoperable spam filtering

or blocklisting. Scheffler and Mayer's SoK on content moderation in E2EE [58] shows that prior literature exists on detecting spam for E2EE email, but most rely either on client-side filtering or more expensive cryptographic primitives like homomorphic encryption. They also show that prior literature exists for (centralized) E2EE messaging, but for which nearly all rely on client-side detection. To our knowledge, there has been no prior work or proposals focused on interoperable blocklisting.

An approach that we believe does work is *asymmetric message franking* [61], which allows report verification in a deniable way. In Section 7 we describe how to use asymmetric message franking for interoperable abuse prevention.

7 Design Proposal

In this section, we provide an overview of our proposed design for interoperable E2EE messaging using our APIs introduced in Sections 4, 5 and 6. As described in Section 3, our design uses the server-to-server architecture, with the client-to-server model used only for identity discovery. We present an overview of the design in Figure 2.

Preliminaries. We describe the building blocks we use in our protocol in detail in Appendix A and give a brief overview here.

Initial Key Exchange Protocol. Scheme $\mathsf{KE} = (\mathsf{Gen}, \mathsf{EGen}, \mathsf{Send}, \mathsf{Recv})$ is a quadruple of algorithms which enables the initial key exchange between two parties at the start of a session in a secure messaging protocol. Gen and EGen are used to generate static and ephemeral key pairs for the client, respectively. KE.Send and KE.Recv are run by the sender and the recipient, respectively, when initiating a new session to derive a shared key. We assume a KE scheme provides authenticity and confidentiality of the established key with forward secrecy.

Secure Messaging Protocol. Scheme $\mathsf{SM} = (\mathsf{InitS}, \mathsf{InitR}, \mathsf{Send}, \mathsf{Recv})$ is a quadruple of algorithms which forms the core secure E2EE messaging protocol between the sender and recipient. SM.InitS and SM.InitR are run by the sender and the recipient, respectively, to initialize their states, on input a symmetric key k. SM.Send is run by the sender on its state and plaintext message m to produce ciphertext c, which is received and decrypted by the receiver through SM.Recv. We assume an SM scheme satisfies authenticity and confidentiality of transmitted messages with forward secrecy and post-compromise security.

Sender-Anonymous Wrapper. Scheme $\mathsf{AW} = (\mathsf{Gen}, \mathsf{Send}, \mathsf{Recv})$ is a triple of algorithms which enables sender-anonymous encryption on top of a non-sender-anonymous SM scheme. Gen generates a pair $(\mathsf{sk}, \mathsf{pk})$. AW.Send takes the sender secret key, the recipient public key, and the message as input and outputs a ciphertext that hides the sender's identity. AW.Recv takes recipient secret key and a ciphertext as input and outputs the corresponding plaintext m as well as the sender identifier sid. AW should achieve cryptographic sender anonymity (in addition to authenticity and confidentiality of the payload) so that the service provider cannot learn the identity of the sender from c.

Asymmetric Message Franking. An Asymmetric Message Franking (AMF) scheme [61] is a cryptographic primitive that enables secure metadata-private moderation. An AMF scheme is run between three parties: the message sender, the message recipient, and a third-party moderator (also known as the judge) to which message recipients can report abusive messages. The message franking algorithm Frank is run by the message sender to generate a franking tag which can be verified by the recipient using Verify. The judge authentication algorithm is run by the third-party moderator when a message has been reported; it takes as input the sender public key, the recipient public key, the judge's secret key, the message, and the message franking tag and outputs a bit that determines whether the message franking tag is valid. We require that AMFs should achieve (1) sender binding, which prevents a sender from forming a signature that can be verified by the receiver but not the moderator; (2) receiver binding, which prevents a receiver from creating a franking tag on a message that was not sent by the sender; and (3) deniability, for when keys or messages are posted publicly after a compromise.

Key Material. Our design relies on clients and providers to maintain various key material. As described in Section 3, we assume some PKI which verifies the public keys of providers and which can be accessed by clients. As such, in our design we assume that both U_{sid} and U_{rid} know the Sender Provider's public key $\mathsf{pk}_{\mathsf{SP}}$ and the Recipient Provider's public key $\mathsf{pk}_{\mathsf{RP}}$.

We further assume that each client has a static public key that serves as their identity key registered with their provider. In particular, we express U_{sid}'s public key as $\mathsf{pk}_{\mathsf{sid}}$ and U_{rid}'s public key as $\mathsf{pk}_{\mathsf{rid}}$. In actuality, our design makes use of multiple cryptographic primitives that each has its own keying material, so a static key pk is actually a tuple of static keys $(\mathsf{pk}_{\mathsf{KE}}, \mathsf{pk}_{\mathsf{AW}}, \mathsf{pk}_{\mathsf{AMF}})$. To simplify notation, we overload terminology and simply refer to a client's identity public key or identity secret key where it is obvious what protocol, and hence which key, is used.

Identity. When user U_{sid} on service provider SP wants to begin communication with U_{rid}, the first steps are to execute SND to find U_{rid}'s provider and then RKM to retrieve their public key material from RP. For SND, we assume there is one SDS (which could be implemented as a single third-party server or could be implemented in a distributed manner). The server maps user identifiers to their associated lists of apps. Recall that there could be identifier collisions on different apps (Alice can have user accounts on multiple apps with the same $\mathsf{uid} = \mathsf{alice}$, or there can be multiple accounts with the same $\mathsf{uid} = \mathsf{alice}$ on different apps belonging to different persons). In our protocol, we let the SDS return all the apps

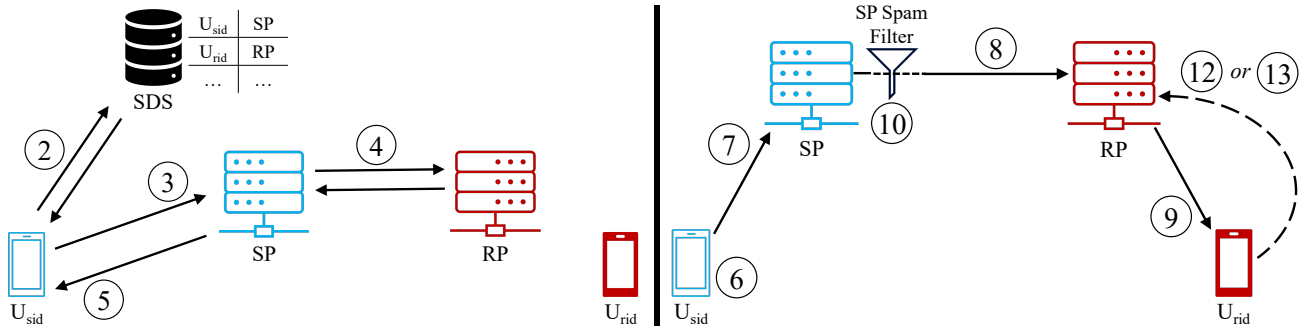

Figure 2: An overview of the flow for our proposed design, with the numbers referring to the specific API functions as defined in Section 7. **(Left)** The flow for SND and RKM. **(Right)** The flow for PRO and ABP, with dashed lines indicating those for ABP.

corresponding to a given query uid.

① $\mathsf{SND.UpdateEntry}(\mathsf{st_{SDS}}, \mathsf{uid}, \mathsf{pref}, \mathsf{app}, b, \mathsf{aux}) \to 1/0$: The SDS receives a request from U_{uid} to add or remove app to their list of apps stored on the server. Before processing this request, the SDS first retrieves the signature σ_{P} that provider P signed over $(\mathsf{uid}, \mathsf{app})$ and verifies it by computing $\mathsf{SIG.Verify}(\mathsf{pk_{RP}}, \sigma_{\mathsf{P}}, (\mathsf{uid}, \mathsf{app}))$. If $b = 1$, then the SDS adds app with preference pref, meaning it adds it as index pref into their list of apps. Else if $b = 0$, then the SDS removes app from their existing list of apps. Once the request has been processed, the SDS responds with a success or failure bit.

② $\mathsf{SND.Lookup}^{\mathsf{Auth}(\mathsf{st_P})}(\mathsf{st_{uid}}, \mathsf{query}\ ;\ \mathsf{st_{SDS}}) \to (\mathsf{resp}; \perp)$: The client U_{uid} looks up the user with identifier rid by initiating a lookup request with the SDS. The SDS first authenticates the client as a valid user of an approved service provider by beginning a challenge-response protocol with U_{uid}. For this, the SDS chooses a random nonce n and sends it to the client. The client then calls protocol Auth to send a request to its provider P to sign over n by computing $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sk_P}, n)$. The client finally completes the challenge-response protocol by sending back P, n, σ and the identifier rid to look up. The SDS verifies the signature by computing $\mathsf{SIG.Verify}(\mathsf{pk_P}, \sigma, n)$ and looks up rid to retrieve the associated list of apps $\overrightarrow{\mathsf{app}}$, which it returns to the client.

③ $\mathsf{RKM.Init}(\mathsf{st_{sid}}, \mathsf{rid}, \mathsf{SP}, \mathsf{RP}) \to (\mathsf{st_{sid}}, q)$: Since RP's response back to client U_{sid} should be hidden from SP, the client first runs $(\mathsf{sk}_e, \mathsf{pk}_e) \leftarrow\!\!\$\ \mathsf{PKE.KeyGen}()$ to generate an ephemeral key pair, which RP will use to encrypt its response. U_{sid} then computes ciphertext $c \leftarrow\!\!\$\ \mathsf{PKE.Enc}(\mathsf{pk_{RP}}, \mathsf{rid} \| \mathsf{pk}_e)$ and forms $q \leftarrow (\mathsf{RP}, c)$, which is sent to SP.

④ $\mathsf{RKM.Lookup}(\mathsf{st_{SP}}, q\ ;\ \mathsf{st_{RP}}) \to (\mathsf{resp}\ ;\ \perp)$: SP receives query $(\mathsf{RP}, c) \leftarrow q$ from U_{sid} and forwards c to RP. RP decrypts by computing $\mathsf{rid} \| \mathsf{pk}_e \leftarrow \mathsf{PKE.Dec}(\mathsf{sk_{RP}}, c)$ and then looks up the associated initial key material for its user with identifier rid. Key material $\mathsf{km_{rid}}$ is composed of $(\mathsf{pk_{rid}}, \mathsf{epk_{rid}}, \mathsf{anon})$: the static identity public key and the ephemeral public key of U_{rid}, respectively, and the flag anon which indicates whether U_{rid} accepts sender-anonymous packets from anybody. RP then encrypts this by computing $c' \leftarrow\!\!\$\ \mathsf{PKE.Enc}(\mathsf{pk}_e, \mathsf{km})$ and forms $\mathsf{resp} \leftarrow c'$, which it sends back to SP who then forwards this back to U_{sid}.

⑤ $\mathsf{RKM.Reconstruct}(\mathsf{st_{sid}}, \mathsf{resp}) \to (\mathsf{st_{sid}}, \mathsf{km})$: Client U_{sid} receives resp from its provider SP and decrypts it by retrieving sk_e from its state and computing $\mathsf{km_{rid}} \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_e, \mathsf{resp})$. It then stores $\mathsf{km_{rid}}$ in its state.

Messaging Protocol. Here we propose an interoperable messaging protocol for two-party communication. As discussed in Appendix B, such a solution offers basic tools that suffice already for a simple, yet inefficient, extension to interoperable group messaging: intuitively, group messages are sent via the pairwise two-party channels between all group members—causing a linear communication overhead—, and the membership management is conducted on the user-to-user layer. We assume the sender has already executed the SND and RKM steps to retrieve the recipient's key material.

⑥ $\mathsf{PRO.Send}(\mathsf{st_{sid}}, m, \mathsf{rid}, \mathsf{RP}) \to (\mathsf{st_{sid}}, c)$: This protocol is executed locally by sender U_{sid}. The final ciphertext is sent to SP.

(a) If the calling user U_{sid} does not have an ongoing session with recipient U_{rid} in their local state $\mathsf{st_{sid}}$ yet, the client establishes a new session by extracting U_{rid}'s key material $(\mathsf{pk_{rid}}, \mathsf{epk_{rid}}, \mathsf{anon})$ from $\mathsf{st_{sid}}$. The sending client runs an initial, one-pass key exchange protocol (e.g., X3DH [40]) to establish key material for U_{sid}'s part of the session state with U_{rid} by retrieving U_{sid}'s key exchange state $\mathsf{st_{KE,sid}}$ from $\mathsf{st_{sid}}$ and then computing $(k, c_{\mathsf{KE}}) \leftarrow\!\!\$\ \mathsf{KE.Send}(\mathsf{st_{KE,sid}}, \mathsf{pk_{rid}}, \mathsf{epk_{rid}})$. It then creates new session state by running $\mathsf{st_{SM,sid,rid}} \leftarrow \mathsf{SM.InitS}(k)$, which U_{sid} stores in its state. U_{sid} also retrieves SP's signature σ_{pk} over $(\mathsf{sid}, \mathsf{pk_{sid}})$ so that U_{rid} can later verify that U_{sid} actually owns $\mathsf{pk_{sid}}$.

(b) The client encrypts message m with the interoperable pairwise messaging protocol (e.g., Double Ratchet), using U_{sid}'s part of the session state with U_{rid}. If U_{sid} has not yet shared its access key π_{sid} to enable U_{rid} to form sender-anonymous packets back to U_{sid}, it appends this to m as $m \leftarrow m \| \pi_{\mathsf{sid}}$. It then computes $(\mathsf{st}_{\mathsf{SM,sid,rid}}, c_{\mathsf{SM}}) \leftarrow\!\!\$\, \mathsf{SM.Send}(\mathsf{st}_{\mathsf{SM,sid,rid}}, m)$. Finally, it computes the message franking tag by running $\sigma_{\mathsf{fr}} \leftarrow\!\!\$\, \mathsf{AMF.Frank}(\mathsf{sk}_{\mathsf{sid}}, \mathsf{pk}_{\mathsf{rid}}, \mathsf{pk}_{\mathsf{RP}}, m)$.

(c) If $\mathsf{anon} = \mathsf{true}$ or U_{sid} already received something from U_{rid} in this session, U_{sid} can hide their own identity from the recipient provider as follows:

i. The client wraps ciphertext c_{SM} and message franking tag σ_{fr} in a sender-anonymous packet (e.g., via Sealed Sender [36]) to U_{rid}. It does so by retrieving its anonymous wrapper secret key $\mathsf{ask}_{\mathsf{sid}}$ from $\mathsf{st}_{\mathsf{sid}}$ and computing $c_{\mathsf{AW}} \leftarrow\!\!\$\, \mathsf{AW.Send}(\mathsf{ask}_{\mathsf{sid}}, \mathsf{pk}_{\mathsf{rid}}, c_{\mathsf{SM}} \| \sigma_{\mathsf{fr}})$ if a session already exists between U_{sid} and U_{rid}. Else if U_{sid} is initiating a session with U_{rid}, then U_{sid} computes $c \leftarrow c_{\mathsf{KE}} \| c_{\mathsf{SM}} \| \sigma_{\mathsf{fr}} \| (\mathsf{sid}, \mathsf{pk}_{\mathsf{sid}}) \| \sigma_{\mathsf{pk}}$ and $c_{\mathsf{AW}} \leftarrow\!\!\$\, \mathsf{AW.Send}(\mathsf{ask}_{\mathsf{sid}}, \mathsf{pk}_{\mathsf{rid}}, c)$.

ii. The client then attaches U_{rid}'s access key π_{rid} that shows to RP that U_{rid} accepts sender-anonymous packets from U_{sid}, which results in $c' = c_{\mathsf{AW}} \| \pi_{\mathsf{rid}}$. Just as for Signal's Sealed Sender [36], this proof is a symmetric secret that U_{rid} distributes to all its session partners (including U_{sid}) as well as RP. If $\mathsf{anon} = \mathsf{true}$, then $\pi \leftarrow \perp$.

(d) Else if $\mathsf{anon} = \mathsf{false}$ and this is the first time U_{sid} is contacting U_{rid}, U_{sid} sets $c' = c_{\mathsf{KE}} \| c_{\mathsf{SM}} \| \sigma_{\mathsf{fr}} \| (\mathsf{sid}, \mathsf{pk}_{\mathsf{sid}}) \| \sigma_{\mathsf{pk}}$.

(e) To hide the recipient's identity from SP, encrypt $c' \| \mathsf{rid}$ to RP by computing $c'' \leftarrow\!\!\$\, \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{RP}}, c' \| \mathsf{rid})$.

(f) The client then returns its updated state and c'', which it sends to SP.

(7) $\mathsf{PRO.Forward}(\mathsf{st}_{\mathsf{SP}}, c, \mathsf{sid}, \mathsf{RP}, aux) \rightarrow (\mathsf{st}_{\mathsf{SP}}, 1/0)$: In this protocol, SP forwards the ciphertext c it received from U_{sid} to RP. SP verifies that U_{sid} is permitted to send a ciphertext to recipient provider RP (e.g. U_{sid} could have been banned from RP because of spam or harassment) and performs other abuse prevention mechanisms like spam filtering. If these fail, then SP returns 0 and drops the message. Otherwise, SP returns 1 and proceeds with delivery of c to RP.

(8) $\mathsf{PRO.Deliver}(\mathsf{st}_{\mathsf{RP}}, c) \rightarrow (\mathsf{st}_{\mathsf{RP}}, 1/0)$: RP receives ciphertext c from SP. It processes c as follows:

(a) RP first decrypts c to recover the recipient identifier: $c' \| \mathsf{rid} \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_{\mathsf{RP}}, c)$.

(b) If $c' = c_{\mathsf{KE}} \| c_{\mathsf{SM}} \| \sigma_{\mathsf{fr}} \| (\mathsf{sid}, \mathsf{pk}_{\mathsf{sid}}) \| \sigma_{\mathsf{pk}}$, RP retrieves the blocklist from its state. If rid has blocked sid, then RP drops the ciphertext and returns 0. Otherwise, it forwards c' and SP to U_{rid} and returns 1.

(c) Else if $c' = c_{\mathsf{AW}} \| \pi$, RP verifies that π is a valid proof for rid. If $\pi = \perp$, then it verifies that U_{rid} has set $\mathsf{anon} = \mathsf{true}$. If the proof is invalid, it drops the message and returns 0. Otherwise, it forwards c_{AW} and SP to U_{rid} and returns 1.

(9) $\mathsf{PRO.Receive}(\mathsf{st}_{\mathsf{rid}}, \mathsf{SP}, c) \rightarrow (\mathsf{st}_{\mathsf{rid}}, m)$: U_{rid} receives a ciphertext from RP and processes it as follows:

(a) If $c = c_{\mathsf{AW}}$, U_{rid} gets $\mathsf{sk}_{\mathsf{rid}}$ from its state and runs $(\mathsf{sid}, c') \leftarrow \mathsf{AW.Recv}(\mathsf{sk}_{\mathsf{rid}}, c_{\mathsf{AW}})$. Then if $c' = c_{\mathsf{SM}} \| \sigma_{\mathsf{fr}}$, U_{rid} retrieves $\mathsf{st}_{\mathsf{SM,sid,rid}}$ from its state.

(b) If c or c' can be parsed as $c_{\mathsf{KE}} \| c_{\mathsf{SM}} \| \sigma_{\mathsf{fr}} \| (\mathsf{sid}, \mathsf{pk}_{\mathsf{sid}}) \| \sigma_{\mathsf{pk}}$, U_{rid} needs to establish a new session with U_{sid}. U_{rid} first verifies U_{sid}'s key by computing $\mathsf{SIG.Verify}(\mathsf{pk}_{\mathsf{SP}}, \sigma_{\mathsf{pk}}, (\mathsf{sid}, \mathsf{pk}_{\mathsf{sid}}))$. It then runs $(\mathsf{pk}'_{\mathsf{sid}}, k) \leftarrow \mathsf{KE.Recv}(\mathsf{st}_{\mathsf{KE,rid}}, c_{\mathsf{KE}})$ and verifies that $\mathsf{pk}'_{\mathsf{sid}} = \mathsf{pk}_{\mathsf{sid}}$, meaning the sender's public key used in the key exchange protocol is correct. It runs $\mathsf{st}_{\mathsf{SM,sid,rid}} \leftarrow \mathsf{SM.InitR}(k)$ to generate the new session state.

(c) Finally, U_{rid} computes $m \leftarrow \mathsf{SM.Recv}(\mathsf{st}_{\mathsf{SM,sid,rid}}, c_{\mathsf{SM}})$ to get back the plaintext message. If m can be parsed as $m \| \pi_{\mathsf{sid}} \leftarrow m$, it then stores the sender's access key π_{sid} in its state. U_{rid} next verifies $\mathsf{AMF.Verify}(\mathsf{pk}_{\mathsf{sid}}, \mathsf{sk}_{\mathsf{rid}}, \mathsf{pk}_{\mathsf{RP}}, m, \sigma_{\mathsf{fr}})$ and returns m and its updated state.

Abuse Prevention. Here we present our interoperable abuse prevention protocol that enables spam filtering, user reporting, and blocklisting.

(10) $\mathsf{ABP.SpFilter}(\mathsf{st}_{\mathsf{SP}}, \mathsf{sid}, c, aux) \rightarrow (\mathsf{st}_{\mathsf{SP}}, 1/0)$: This algorithm is run by SP when it receives ciphertext c to be sent. As part of its state, SP stores data about each user's account, such as the identifier used to create the account (e.g. phone number, email, etc.) and the age of the account. It also keeps track of the number of ciphertexts sent by each user over some time period, such as the last 60 seconds, which is used as the auxiliary data *aux*. The algorithm itself can be a spam classifier trained on data that enables the model to classify spam based on the mentioned features, such as that deployed for WhatsApp. If SP is a non-gatekeeper, this classifier could be provided as part of the gatekeeper's API. The algorithm outputs the updated state of SP and also outputs 1 if the message is deemed as benign or 0 if the message is deemed spam and should be dropped. In this latter case, SP drops the message and updates its state to note the sender of the message as a spammer. For instance, it could deactivate the account and notify the banned user.

(11) $\mathsf{ABP.GenReport}(\mathsf{st}_{\mathsf{rid}}, \mathsf{sid}, aux) \rightarrow \rho$: The auxiliary data *aux* consists of the message m to be reported, the associated message franking tag σ_{fr}, and an optional reason this violates the Terms of Service of the provider, which we call R. The recipient retrieves the sender's key $\mathsf{pk}_{\mathsf{sid}}$ from its state $\mathsf{st}_{\mathsf{rid}}$ and compiles the report as $\rho \leftarrow (\mathsf{sid}, \mathsf{pk}_{\mathsf{sid}}, \mathsf{rid}, m, \sigma_{\mathsf{fr}}, R)$.

(12) $\mathsf{ABP.ReportRcv}(\mathsf{st}_{\mathsf{RP}}, \rho) \rightarrow \mathsf{st}_{\mathsf{RP}}$: RP receives report $(\mathsf{sid}, \mathsf{pk}_{\mathsf{sid}}, \mathsf{rid}, m, \sigma_{\mathsf{fr}}, R) \leftarrow \rho$. It then retrieves the key $\mathsf{pk}_{\mathsf{rid}}$ of the recipient and its own secret key $\mathsf{sk}_{\mathsf{RP}}$ from its state $\mathsf{st}_{\mathsf{RP}}$ and verifies the message franking tag by computing $\mathsf{AMF.Judge}(\mathsf{pk}_{\mathsf{sid}}, \mathsf{pk}_{\mathsf{rid}}, \mathsf{sk}_{\mathsf{RP}}, m, \sigma_{\mathsf{fr}})$. If it returns 0, RP rejects the report and aborts. Otherwise, the provider reviews the message and explanation (if available) by the recipient. If the provider decides that the sender violated their Terms of Service and should be banned from their platform, then RP can send sid to SP and request them to block this user from sending messages to their platform. We note that it is possible for SP to refuse, but then RP could have grounds to discontinue interoperability.

(13) $\mathsf{ABP.Block}(\mathsf{st}_{\mathsf{rid}}, \mathsf{sid}\ ;\ \mathsf{st}_{\mathsf{RP}}) \rightarrow (\perp\ ;\ \mathsf{st}_{\mathsf{RP}})$: The client U_{rid} sends a block request for sender with identifier sid to the recipient provider RP. Once RP receives the request, it rotates the access key for U_{rid} and sends the new access key π'_{rid} back to U_{rid}. RP also updates the blocklist in its state to add sid as a blocked sender for U_{rid}. The client U_{rid} stores π'_{rid} in its state and sends messages to each of its non-blocked contacts so that they can get U_{rid}'s new access key. Note that if U_{rid} has set anon to true, meaning that they accept sender-anonymous messages from anybody, then they do not have the ability to blocklist users.

Security and Privacy Properties. We describe how our protocol meets our security and privacy goals.

Identity. The authorization mechanism used in SND is unforgeable due to the EUF-CMA unforgeability of the digital signature scheme. The privacy of uid from SDS is preserved by construction: the U_{uid} does not present it to the SDS when running a Lookup query and the challenge-response protocol transcript does not contain any information about uid either. The privacy of query from P is also preserved by construction: the challenge-response protocol does not need any information about the query. In RKM, SP does not learn any information about rid since it is encrypted to RP. The IND-CCA security of PKE ensures this privacy. By construction, SP does not send sid to RP, so sid remains hidden from RP.

Messaging Protocol. Depending on the properties of the employed protocols KE, SM, and AW, their overall composition achieves the security properties specified in Section 5.2. More concretely, by letting SP check whether U_{sid} is permitted to send something to the specified RP, a first authorization check is conducted. Due to the second authorization check, performed by RP, U_{rid} is guaranteed to only receive ciphertexts from accepted contacts. This enables our protocol to achieve *Authorized Messaging*. Furthermore, based on the key-indistinguishability of the initial key exchange as well as the FS and PCS of the messaging protocol, the transmitted payload remains confidential and authentic with FS and PCS.

Finally, as rid is encrypted to RP, SP never learns the recipient identity, which ensures *Recipient Privacy*. Similarly, U_{sid} hides their identity towards RP if the recipient accepts anonymous ciphertexts or as soon as the recipient accepted their shared, ongoing session by distributing a corresponding access key, which ensures *Sender Privacy*. However, as mentioned in Section 5.3, none of the deployed sender-anonymous wrapper protocols prevents the battery draining attack, provides FS and PCS, and is practically efficient simultaneously.

Abuse Prevention. Our spam filtering mechanism reduces metadata about sender identities leaked to RP since SP runs the filter on its own users. Furthermore, U_{sid}'s identity is only revealed to RP by U_{rid} during reporting or blocklisting. Note that outside of this, RP has no way of learning U_{sid}'s identity. We argue this is acceptable metadata leakage, since our model assumes malicious clients which could reveal a sender's identity to RP at any point anyways. The notions of verifiable reporting and deniability are achieved by the AMF scheme.

8 Future Directions

Identifying major open problems and challenges in the world of interoperable E2EE is an important contribution of this paper. Our attempt at defining a uniform API uncovers a fundamental tension between the communication pattern and security and privacy trade-offs. Our APIs mostly assume a server-to-server communication pattern (q.v.§3). A second important tension arises between privacy (metadata leakage minimization) and abuse/spam prevention. In our construction (Section 7) we strive to achieve a reasonable trade-off by choosing a mix of communication patterns and privacy-enhancing cryptographic building blocks. Achieving better trade-offs remains an interesting and important open question: for example, our protocol reveals both sender and receiver identity to RP in one case, but this may not be inherently needed. It would also be desirable to improve the privacy of reporting and blocklisting: for example, hiding the sender's identity from RP during reporting. Finally, by design our protocol does not allow cross-provider spam filtering: using secure computation, it may be possible to implement more accurate spam detection with privacy.

Another extension would be improving the privacy of SND. Tools like PIR and anonymous credentials may be useful here. Applying key transparency to protect against malicious providers giving incorrect keys during RKM is another open direction. Lastly, we have surfaced several nuances of extending our 1:1 chats to group chats in the Appendix. Extending our framework and analyses to encrypted voice and video chats also remains an open problem.

Acknowledgements

The authors thank Ian Brown, Richard Barnes, Alissa Cooper, Thomas Ristenpart, and Greg Zaverucha for helpful feedback. This work was supported by a generous gift from Meta.

References

[1] How WhatsApp reduced spam while launching end-to-end encryption. USENIX Enigma, 2017.

[2] Martin R Albrecht, Jorge Blasco, Rikke Bjerg Jensen, and Lenka Mareková. Mesh messaging in large-scale protests: Breaking Bridgefy. In *Topics in Cryptology–CT-RSA 2021*. Springer, 2021.

[3] Martin R. Albrecht, Sofía Celi, Benjamin Dowling, and Daniel Jones. Practically-exploitable cryptographic vulnerabilities in matrix. https://nebuchadnezzar-megolm.github.io/static/paper.pdf, 2022.

[4] Joël Alwen, Benedikt Auerbach, Miguel Cueto Noval, Karen Klein, Guillermo Pascual-Perez, and Krzysztof Pietrzak. DeCAF: Decentralizable continuous group key agreement with fast healing. Cryptology ePrint Archive, Report 2022/559, 2022. https://eprint.iacr.org/2022/559.

[5] Joël Alwen, Benedikt Auerbach, Miguel Cueto Noval, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, and Michael Walter. CoCoA: Concurrent continuous group key agreement. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 815–844. Springer, Heidelberg, May / June 2022.

[6] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 129–158. Springer, Heidelberg, May 2019.

[7] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 248–277. Springer, Heidelberg, August 2020.

[8] Joël Alwen, Sandro Coretti, Daniel Jost, and Marta Mularczyk. Continuous group key agreement with active security. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 261–290. Springer, Heidelberg, November 2020.

[9] Matilda Backendal, Miro Haller, and Kenneth G. Paterson. MEGA: malleable encryption goes awry. In *44rd IEEE Symposium on Security and Privacy, SP 2023*. IEEE, 2023.

[10] Fatih Balli, Paul Rösler, and Serge Vaudenay. Determining the core primitive for optimally secure ratcheting. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 621–650. Springer, Heidelberg, December 2020.

[11] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-17, Internet Engineering Task Force, December 2022. Work in Progress.

[12] Beeper app. https://www.beeper.com/.

[13] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 566–582. Springer, Heidelberg, December 2001.

[14] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 619–650. Springer, Heidelberg, August 2017.

[15] Alexander Bienstock, Yevgeniy Dodis, Sanjam Garg, Garrison Grogan, Mohammad Hajiabadi, and Paul Rösler. On the worst-case inefficiency of CGKA. In Eike Kiltz and Vinod Vaikuntanathan, editors, *Theory of Cryptography - 20th International Conference, TCC 2022, Chicago, IL, USA, November 7-10, 2022, Proceedings, Part II*, volume 13748 of *Lecture Notes in Computer Science*, pages 213–243. Springer, 2022.

[16] Alexander Bienstock, Yevgeniy Dodis, and Paul Rösler. On the price of concurrency in group ratcheting protocols. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 198–228. Springer, 2020.

[17] Alexander Bienstock, Jaiden Fairoze, Sanjam Garg, Pratyay Mukherjee, and Srinivasan Raghuraman. A more complete analysis of the Signal double ratchet algorithm. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 784–813. Springer, Heidelberg, August 2022.

[18] Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, and Douglas Stebila. Towards post-quantum security for Signal's X3DH handshake. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 404–430. Springer, Heidelberg, October 2020.

[19] Briar: secure messaging, anywhere. https://briarproject.org/how-it-works/.

[20] Marcus Brinkmann, Christian Dresen, Robert Merget, Damian Poddebniak, Jens Müller, Juraj Somorovsky, Jörg Schwenk, and Sebastian Schinzel. ALPACA: Application layer protocol confusion - analyzing and mitigating cracks in TLS authentication. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 4293–4310. USENIX Association, August 2021.

[21] Ian Brown. Private messaging interoperability in the EU Digital Markets Act. *OpenForum Academy*, 2022.

[22] Ran Canetti, Palak Jain, Marika Swanberg, and Mayank Varia. Universally composable end-to-end secure messaging. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 3–33. Springer, Heidelberg, August 2022.

[23] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 451–466. IEEE, 2017.

[24] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1802–1819. ACM Press, October 2018.

[25] The digital markets act explained in 15 questions. https://element.io/blog/the-digital-markets-act-explained-in-15-questions/.

[26] EU Digital Markets Act (DMA). https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32022R1925, 2022.

[27] Samuel Dobson and Steven D. Galbraith. Post-quantum signal key agreement with SIDH. Cryptology ePrint Archive, Report 2021/1187, 2021. https://eprint.iacr.org/2021/1187.

[28] Benjamin Dowling, Eduard Hauck, Doreen Riepel, and Paul Rösler. Strongly anonymous ratcheted key exchange. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part III*, volume 13793 of *Lecture Notes in Computer Science*, pages 119–150. Springer, 2022.

[29] Moderation needs a radical change. https://element.io/blog/moderation-needs-a-radical-change/.

[30] The digital markets act explained in 15 questions. https://element.io/blog/the-digital-markets-act-explained-in-15-questions/, 2022.

[31] Google messages: Your chats stay private with spam detection. https://support.google.com/messages/answer/9327903.

[32] Google messages report spam. https://support.google.com/messages/answer/9061432.

[33] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 66–97. Springer, Heidelberg, August 2017.

[34] Keitaro Hashimoto, Shuichi Katsumata, Kris Kwiatkowski, and Thomas Prest. An efficient and generic construction for signal's handshake (X3DH): Post-quantum, state leakage secure, and deniable. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 410–440. Springer, Heidelberg, May 2021.

[35] Block, filter, and report messages on iphone. https://support.apple.com/guide/iphone/block-filter-and-report-messages-iph203ab0be4/ios.

[36] jlund. Technology preview: Sealed sender for signal. https://signal.org/blog/sealed-sender/, 10 2018.

[37] Karen Klein, Guillermo Pascual-Perez, Michael Walter, Chethan Kamath, Margarita Capretto, Miguel Cueto, Ilia Markov, Michelle Yeo, Joël Alwen, and Krzysztof Pietrzak. Keep the dirt: Tainted TreeKEM, adaptively and actively secure continuous group key agreement. In *2021 IEEE Symposium on Security and Privacy*, pages 268–284. IEEE Computer Society Press, May 2021.

[38] Frederic Lardinois. Google says its machine learning tech now blocks 99.9% of gmail spam and phishing messages. *TechCrunch*.

[39] Natasha Lomas. Europe says yes to messaging interoperability as it agrees on major new regime for big tech. https://techcrunch.com/2022/03/24/dma-political-agreement/.

[40] Moxie Marlinspike and Trevor Perrin. The X3DH key agreement protocol. *Open Whisper Systems*, 2016.

[41] Facebook Messenger. Messenger secret conversations, technical whitepaper. https://about.fb.com/wp-content/uploads/2016/07/messenger-secret-

conversations-technical-whitepaper.pdf, 05 2017.

[42] Matrix Messenger. Megolm group ratchet. https://gitlab.matrix.org/matrix-org/olm/-/blob/master/docs/megolm.md, 11 2019.

[43] Matrix Messenger. Olm: A cryptographic ratchet. https://gitlab.matrix.org/matrix-org/olm/blob/master/docs/olm.md, 11 2019.

[44] Microsoft. Skype private conversation, technical white paper. https://az705183.vo.msecnd.net/onlinesupportmedia/onlinesupport/media/skype/documents/skype-private-conversation-white-paper.pdf, 06 2018.

[45] More instant messaging interoperability (mimi). https://datatracker.ietf.org/wg/mimi/about/.

[46] Mio app. https://www.m.io/.

[47] Casey Newton. Three ways the European Union might ruin WhatsApp. *The Verge*.

[48] European electronic communications code. https://eur-lex.europa.eu/eli/dir/2018/1972/oj, 2018.

[49] Kenneth G. Paterson, Matteo Scarlata, and Kien Tuong Truong. Three lessons from threema: Analysis of a secure messenger. *Attack Website*, 2022.

[50] Trevor Perrin and Moxie Marlinspike. The double ratchet algorithm. https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf, 11 2016.

[51] Riana Pfefferkorn. Content-oblivious trust and safety techniques: Results from a survey of online service providers. *Journal of Online Trust and Safety*, 1(2), 2022.

[52] Bertram Poettering and Paul Rösler. Towards bidirectional ratcheted key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2018.

[53] Eric Rescorla. Discovery mechanisms for messaging and calling interoperability. https://educatedguesswork.org/posts/messaging-discovery/.

[54] Eric Rescorla. End-to-end encryption and messaging interoperability. https://educatedguesswork.org/posts/messaging-e2e/#identity.

[55] J. Rosenberg, C. Jennings, A. Cooper, and J. Peterson. Simple protocol for inviting numbers (spin). https://www.ietf.org/archive/id/draft-rosenberg-dispatch-spin-00.html.

[56] Paul Rösler, Christian Mainka, and Jörg Schwenk. More is less: On the end-to-end security of group chats in signal, whatsapp, and threema. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 415–429. IEEE, 2018.

[57] Paul Rösler, Daniel Slamanig, and Christoph Striecks. Unique-path identity based encryption with applications to strongly secure messaging. In *Advances in Cryptology - EUROCRYPT 2023 - 42th Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2023 Proceedings*, Lecture Notes in Computer Science. Springer, 2023.

[58] Sarah Scheffler and Jonathan Mayer. SoK: Content moderation for end-to-end encryption. *arXiv preprint arXiv:2303.03979*, 2023.

[59] Texts app. https://texts.com/.

[60] Use a third-party whatsApp client and you could be banned for life. https://www.cultofmac.com/314343/use-a-third-party-whatsapp-client-and-you-could-be-banned-for-life/.

[61] Nirvan Tyagi, Paul Grubbs, Julia Len, Ian Miers, and Thomas Ristenpart. Asymmetric message franking: content moderation for metadata-private end-to-end encryption. In *Annual International Cryptology Conference*, pages 222–250. Springer, 2019.

[62] Nirvan Tyagi, Julia Len, Ian Miers, and Thomas Ristenpart. Orca: Blocklisting in sender-anonymous messaging. *Cryptology ePrint Archive*, 2021.

[63] Nirvan Tyagi, Julia Len, Ian Miers, and Thomas Ristenpart. Orca: Blocklisting in sender-anonymous messaging. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 2299–2316. USENIX Association, 2022.

[64] Matthew Weidner, Martin Kleppmann, Daniel Hugenroth, and Alastair R. Beresford. Key agreement for decentralized secure group messaging with strong security guarantees. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2024–2045. ACM Press, November 2021.

[65] WhatsApp. Whatsapp encryption overview, technical white paper. https://scontent-frt3-2.xx.fbcdn.net/v/t39.8562-6/309473131_

1302549333851760_6207638168445881915_n.pdf?_nc_cat=107&ccb=1-7&_nc_sid=ad8a9d&_nc_ohc=FWSdA0jgi5MAX8k8W57&_nc_ht=scontent-frt3-2.xx&oh=00_AfACgyhdpJlZ0uVsIDt_DH14-diu42Ik5QOeiA1lcyMMNQ&oe=63CEEFDD, 11 2021.

[66] About unofficial apps. https://faq.whatsapp.com/1217634902127718.

[67] Wire. Proteus github repository. https://github.com/wireapp/proteus, 10 2019.

[68] Interoperability without sacrificing privacy: Matrix and the dma. https://matrix.org/blog/2022/03/25/interoperability-without-sacrificing-privacy-matrix-and-the-dma.

A Preliminaries

We describe in more detail the cryptographic primitives that we use in our construction in Section 7.

Public Key Encryption. A public key encryption scheme is a triple of algorithms $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. Key generation is a probabilistic algorithm that generates a secret-public key pair: $(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\, \mathsf{PKE.KeyGen}()$. Encryption is a probabilistic algorithm that encrypts a message using a public key: $c \leftarrow\!\!\$\, \mathsf{PKE.Enc}(\mathsf{pk}, m)$. Finally, decryption is a deterministic algorithm that recovers the plaintext message using the corresponding secret key: $m \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, c)$. We assume any PKE scheme used meets the traditional notions of correctness and IND-CCA security.

Digital Signature Scheme. A digital signature scheme is a triple of algorithms $\mathsf{SIG} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. Key generation is a probabilistic algorithm that generates a secret signing key and public verification key: $(\mathsf{sk}, \mathsf{vk}) \leftarrow\!\!\$\, \mathsf{SIG.KeyGen}()$. Sign is a probabilistic algorithm that signs a message using the secret signing key to produce a signature: $\sigma \leftarrow\!\!\$\, \mathsf{SIG.Sign}(\mathsf{sk}, m)$. Verify is a deterministic algorithm that takes as input the public verification key, the message, and the signature and outputs a bit indicating either acceptance or rejection of the signature: $1/0 \leftarrow \mathsf{SIG.Verify}(\mathsf{vk}, \sigma, m)$. We assume any SIG scheme used meets the traditional notions of correctness and existential unforgeability under chosen message attack (EUF-CMA).

Initial Key Exchange Protocol. Scheme $\mathsf{KE} = (\mathsf{Gen}, \mathsf{EGen}, \mathsf{Send}, \mathsf{Recv})$ is a quadruple of algorithms which enables the initial key exchange between two parties at the start of a session in a secure messaging protocol:

- $(\mathsf{st}, \mathsf{pk}) \leftarrow\!\!\$\, \mathsf{KE.Gen}()$: Generates a static key pair, where the secret key is stored in a new client state.
- $(\mathsf{st}, \mathsf{epk}) \leftarrow\!\!\$\, \mathsf{KE.EGen}(\mathsf{st})$: Generates an ephemeral key pair, adding the secret component to the client state.
- $(k, c_{\mathsf{KE}}) \leftarrow\!\!\$\, \mathsf{KE.Send}(\mathsf{st}_s, \mathsf{pk}_r, \mathsf{epk}_r)$: Is run by a sender when initiating a new session with a recipient to derive a shared key. It takes as input the sender state, the recipient static public key, and the recipient ephemeral public key and outputs a shared symmetric key k and ciphertext c_{KE}.
- $(\mathsf{pk}_s, k) \leftarrow \mathsf{KE.Recv}(\mathsf{st}_r, c_{\mathsf{KE}})$: Is run by a recipient when receiving a session initiation request from a sender in order to derive a shared key. It takes as input the recipient state and the ciphertext output by Send and outputs the sender's public key and the shared symmetric key.

We assume a KE scheme that meets the security goals of authenticity and confidentiality of the established key with forward secrecy [18, 23, 27, 34].

Secure Messaging Protocol. Scheme $\mathsf{SM} = (\mathsf{InitS}, \mathsf{InitR}, \mathsf{Send}, \mathsf{Recv})$ is a quadruple of algorithms which forms the core secure E2EE messaging protocol between two parties,

the sender and recipient:

- $\mathsf{st}_s \leftarrow \mathsf{SM.InitS}(k)$ and $\mathsf{st}_r \leftarrow \mathsf{SM.InitR}(k)$: Is run by the sender resp. recipient to initialize its secret state on input a symmetric key k.
- $(\mathsf{st}_s, c) \leftarrow\!\!\$\, \mathsf{SM.Send}(\mathsf{st}_s, m)$: Is run by the sender on its state and plaintext message m to produce ciphertext c.
- $(\mathsf{st}_r, m) \leftarrow \mathsf{SM.Recv}(\mathsf{st}_r, c)$: Is run by the recipient on its state and a ciphertext c to obtain the corresponding plaintext message m.

We assume an SM scheme that meets the security goals of authenticity and confidentiality of transmitted messages with forward secrecy and post-compromise security [6, 17, 22, 23].

Sender-Anonymous Wrapper. Scheme $\mathsf{AW} = (\mathsf{Gen}, \mathsf{Send}, \mathsf{Recv})$ is a triple of algorithms which enables sender-anonymous encryption. This scheme forms a wrapper on top of the non-sender-anonymous E2EE scheme SM.

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\, \mathsf{AW.Gen}()$: Generates a key pair.
- $c \leftarrow\!\!\$\, \mathsf{AW.Send}(\mathsf{sk}_s, \mathsf{pk}_r, m)$: Takes as input the sender secret key, the recipient public key, and the message and outputs a ciphertext that hides the sender's identity.
- $(\mathsf{sid}, m) \leftarrow \mathsf{AW.Recv}(\mathsf{sk}_r, c)$: Takes as input the recipient secret key and a ciphertext and outputs the corresponding plaintext m as well as the sender identifier sid.

This scheme can be viewed as an abstraction of sender-anonymous protocols like Signal's Sealed Sender [36]. As described by Tyagi et al. [62], AW should achieve cryptographic sender anonymity (in addition to authenticity and confidentiality of the payload) so that the service provider cannot learn the identity of the sender from c.

Asymmetric Message Franking. An Asymmetric Message Franking (AMF) scheme [61] is a cryptographic primitive that enables secure metadata-private moderation. An AMF scheme is run between three parties: the message sender, the message recipient, and a third-party moderator (also known as the judge) to which message recipients can report abusive messages. It is comprised of algorithms $\mathsf{AMF} = (\mathsf{KeyGen}, \mathsf{Frank}, \mathsf{Verify}, \mathsf{Judge}, \mathsf{Forge}, \mathsf{RForge}, \mathsf{JForge})$. Key generation generates a secret-public key pair. The message franking algorithm is run by the message sender and takes as input the sender's secret key, the recipient's public key, the judge's public key, and the message to be sent and outputs the message franking tag: $\sigma_{\mathsf{fr}} \leftarrow\!\!\$\, \mathsf{AMF.Frank}(\mathsf{sk}_s, \mathsf{pk}_r, \mathsf{pk}_j, m)$. The verification algorithm is run by the message recipient and takes as input the sender public key, the recipient secret key, the judge's public key, the message, and the message franking tag and outputs a bit that determines whether the message franking tag is valid: $b \leftarrow \mathsf{AMF.Verify}(\mathsf{pk}_s, \mathsf{sk}_r, \mathsf{pk}_j, m, \sigma_{\mathsf{fr}})$. The judge authentication algorithm is run by the third-party moderator when a message has been reported; it takes as input the sender public key, the recipient public key, the judge's secret key, the message, and the message franking tag and outputs a bit that determines whether the message franking tag is valid: $b \leftarrow \mathsf{AMF.Judge}(\mathsf{pk}_s, \mathsf{pk}_r, \mathsf{sk}_j, m, \sigma_{\mathsf{fr}})$.

We assume any AMF scheme used meets the correctness, accountability, and deniability notions as defined in [61]. At a high level, we require that AMFs should achieve (1) sender binding, which prevents a sender from forming a signature that can be verified by the receiver but not the moderator; (2) receiver binding, which prevents a receiver from creating a franking tag on a message that was not sent by the sender; and (3) deniability for when keys or messages are posted publicly after a compromise.

B Group Messaging Overview

Generalizing the two-party case from Section 5 to interoperable group messaging adds challenges and opens several design choices. We present three relevant options for implementing interoperable group messaging. Each option comes with its own API requirements that we do not spell out formally.

B.1 Pairwise Channels

The first option is the simplest as it only relies on components that are necessary for interoperable *two-party* messaging already. Hence, it would directly extend a two-party solution to interoperable group messaging: every group message is sent as $n-1$ individual pairwise messages, where n is the number of group members. Messages between two users of *the same* provider are sent via their ordinary provider-internal messaging channel. We note that, thereby, a group conversation can be realized based on multiple independent pairwise channel protocols as we will elaborate in the next paragraph *Illustration of Federation vs. Interoperability*.

An existing real-world example for pairwise channel-based group messaging was an **earlier version of Signal's group chat** protocol (see Rösler et al. [56]). At a high level, it sends group messages as $n-1$ two-party messages, where each message is prepended with a secret, random group identifier. Whenever the set of group members changes, the group identifier is freshly sampled at random and distributed to all members. Thereby, this identifier serves as a simple proof of membership.

The main advantages of this approach are its simplicity as well as the fact that all peculiarities of groups, such as membership management, are handled on the user-to-user layer without the need for additional server functionalities. Beyond the most obvious disadvantage—the linear communication overhead—, handling concurrent, potentially colliding, membership changes poses a problem. E.g., consider a situation in which user A adds B to the group and removes C from the group, and, simultaneously, C removes A from the group. Solving this issue efficiently (i.e., without relying on expensive consensus mechanisms) remains an open problem;

Weidner et al. [64] propose a candidate solution that is not entirely applicable to the setting we consider here.

Illustration of Federation vs. Interoperability. Pairwise channels exemplify a difference between *federated* messaging and *interoperable* messaging. While in federated messaging, all users must execute *the same protocol*, pairwise channels for interoperable (group) messaging can be based on *multiple different, incompatible protocols*: consider a group of users A, B, and C who use providers SP_A, SP_B, and SP_C, respectively. When A sends a group message, she could send the message to B via a client bridge using SP_B's ordinary two-party messaging protocol, and to C via another client bridge using SP_C's ordinary two-party messaging protocol. A second group message sent from B might be sent back to A through A's client bridge via SP_B's ordinary two-party messaging protocol, and to C via B's client bridge using SP_C's ordinary two-party messaging protocol. That means users may speak different protocols depending on the their communication partners' *providers* and also depending on the *communication direction* with these partners.

We note that the DMA is not explicit about requiring more than two providers (one gatekeeper and one non-gatekeeper) to provide interoperability to their users. Thus, we want to mention that the above example can also be simplified by observing that the situation of using different pairwise channel protocols to realize a (group) conversation can already occur with two providers: two users A and B with different providers could use one protocol for messages sent from A to B and another protocol for messages sent from B to A. The decision which protocol is used for which communication direction is not only technical: it may depend on further interpretations of the DMA and/or negotiations between gatekeepers and other providers. However, there might be technical reasons (e.g., security or performance properties) to prefer one protocol over another.

B.2 Subset Groups

Just as pairwise channels, *subset groups* can rely on protocol components that are already necessary for intra-provider group messaging as well as interoperable two-party messaging. The core idea is to split the interoperable group into its natural intra-provider sub-groups. Based on this idea, the enhancement over pairwise chats is to leverage this sub-group structure by forwarding only the communication from one provider's sub-group to the other providers' sub-groups via interoperable channels: More concretely, for a group with users A, B, and C who use provider SP_1 and D, E, and F who use provider SP_2, messages sent by A are sent as 3 pairwise messages to D, E, and F, respectively, and as a single group message to B and C via SP_1's ordinary group messaging protocol. A refinement of this option could be: the message sent by A is converted into a single message to the group of D, E, and F at provider SP_2 (i.e., A must be permitted to send to this group from outside), and an ordinary provider-internal group message to B and C.

This approach solves the performance disadvantage of pairwise channels but not the membership consistency issue.

B.3 Standardized Protocol

A third option is to standardize a dedicated protocol that can process interoperable (group) messaging, which is, for example, the goal of the MIMI standardization initiative [45]. In contrast to the above to options, this third option may need further, potentially protocol-dependent interfaces at the involved servers.

Despite the fact that group messaging can be used to illustrate differences between federated and interoperable messaging, a federated messaging protocol can be used as a solution for interoperable messaging: The involved providers would need to agree on jointly deploying the same (federated) protocol instead of composing parts of their internal messaging protocols with connection-protocols for interoperability.

An existing example for federated messaging is **Matrix** [42]. However, recent attacks [3] suggest that further cryptographic analyses are necessary to establish trust in Matrix's protocols. Another example that is often discussed in the context of interoperable (group) messaging is the **MLS standardization initiative** [11]. Federated deployment is, however, not in the current scope of MLS. Moreover, the current protocol relies on a central server that maintains membership consistency and resolves concurrently executed protocol operations. Such a central server could be deployed by each of the involved providers; deciding which provider's server is responsible for which interoperable group might be a rather non-technical question.

Instead of using a centralized server to handle concurrently acting users, several recent articles propose messaging protocols that work in decentralized environments [4, 5, 16, 64]. While some of these protocols focus on maintaining low communication overhead [4, 5, 16], Weidner et al. [64] particularly focus on solving the membership consistency problem in a decentralized setting—yet, their protocol incurs linear communication overhead and relies on lossless, in-order delivery.

C Text of Article 7 of the DMA

Here we quote the full text of Article 7 of the DMA, which describes the interoperability mandate. We take the text from a PDF distributed by the European Parliament [26]. The text references Article 3 at points; this article lays out the process for designating a company as a gatekeeper for the purposes of the DMA.

1. Where a gatekeeper provides number-independent interpersonal communications services that are listed in

the designation decision pursuant to Article 3(9), it shall make the basic functionalities of its number-independent interpersonal communications services interoperable with the number-independent interpersonal communications services of another provider offering or intending to offer such services in the Union, by providing the necessary technical interfaces or similar solutions that facilitate interoperability, upon request, and free of charge.

2. The gatekeeper shall make at least the following basic functionalities referred to in paragraph 1 interoperable where the gatekeeper itself provides those functionalities to its own end users:

   (a) following the listing in the designation decision pursuant to Article 3(9):

      i. end-to-end text messaging between two individual end users;

      ii. sharing of images, voice messages, videos and other attached files in end-to-end communication between two individual end users.

   (b) within 2 years from the designation:

      i. end-to-end text messaging within groups of individual end users;

      ii. sharing of images, voice messages, videos and other attached files in end-to-end communication between a group chat and an individual end user;

   (c) within 4 years from the designation:

      i. end-to-end voice calls between two individual end users;

      ii. end-to-end video calls between two individual end users;

      iii. end-to-end voice calls between a group chat and an individual end user;

      iv. end-to-end video calls between a group chat and an individual end user.

3. The level of security, including the end-to-end encryption, where applicable, that the gatekeeper provides to its own end users shall be preserved across the interoperable services.

4. The gatekeeper shall publish a reference offer laying down the technical details and general terms and conditions of interoperability with its number-independent interpersonal communications services, including the necessary details on the level of security and end-to-end encryption. The gatekeeper shall publish that reference offer within the period laid down in Article 3(10) and update it where necessary.

5. Following the publication of the reference offer pursuant to paragraph 4, any provider of number-independent interpersonal communications services offering or intending to offer such services in the Union may request interoperability with the number-independent interpersonal communications services provided by the gatekeeper. Such a request may cover some or all of the basic functionalities listed in paragraph 2. The gatekeeper shall comply with any reasonable request for interoperability within 3 months after receiving that request by rendering the requested basic functionalities operational.

6. The Commission may, exceptionally, upon a reasoned request by the gatekeeper, extend the time limits for compliance under paragraph 2 or 5 where the gatekeeper demonstrates that this is necessary to ensure effective interoperability and to maintain the necessary level of security, including end-to-end encryption, where applicable.

7. The end users of the number-independent interpersonal communications services of the gatekeeper and of the requesting provider of number-independent interpersonal communications services shall remain free to decide whether to make use of the interoperable basic functionalities that may be provided by the gatekeeper pursuant to paragraph 1.

8. The gatekeeper shall collect and exchange with the provider of number-independent interpersonal communications services that makes a request for interoperability only the personal data of end users that is strictly necessary to provide effective interoperability. Any such collection and exchange of the personal data of end users shall fully comply with Regulation (EU) 2016/679 and Directive 2002/58/EC.

9. The gatekeeper shall not be prevented from taking measures to ensure that third-party providers of number-independent interpersonal communications services requesting interoperability do not endanger the integrity, security and privacy of its services, provided that such measures are strictly necessary and proportionate and are duly justified by the gatekeeper.