MCMix: Anonymous Messaging via Secure Multiparty Computation

MCMix: Anonymous Messaging via Secure Multiparty Computation

Nikolaos Alexopoulos
Technische Universität Darmstadt, Germany
alexopoulos@tk.tu-darmstadt.de

Aggelos Kiayias
University of Edinburgh, UK
akiayias@inf.ed.ac.uk

Riivo Talviste
Cybernetica AS, Estonia
riivo@cyber.ee

Thomas Zacharias
University of Edinburgh, UK
tzachari@inf.ed.ac.uk

Abstract

We present MCMix, an anonymous messaging system that completely hides communication metadata and can scale in the order of hundreds of thousands of users. Our approach is to isolate two suitable functionalities, called dialing and conversation, that when used in succession, realize anonymous messaging. With this as a starting point, we apply secure multiparty computation ("MC" or MPC) and proceed to realize them. We then present an implementation using Sharemind, a prevalent MPC system. Our implementation is competitive in terms of latency with previous messaging systems that only offer weaker privacy guarantees. Our solution can be instantiated in a variety of different ways with different MPC implementations, overall illustrating how MPC is a viable and competitive alternative to mix-nets and DC-nets for anonymous communication.

1 Introduction

In an era in which privacy in communications is becoming increasingly important, it is often the case that two parties want to communicate anonymously, that is to exchange messages while hiding the very fact that they are in conversation. A major problem in this setting is hiding the communication metadata: while existing cryptographic techniques (e.g., secure point-to-point channels implemented with TLS) are sufficiently well developed to hide the communication content, they are not intended for hiding the metadata of the communication such as its length, its directionality, and the identities of the communicating end points. Metadata are particularly important, arguably some times as important to protect as the communication content. The importance of metadata is reflected in General Michael Hayden's quote "We kill people based on metadata"[1] and in the persistence of security agencies with programs like PRISM (by the NSA) and TEMPORA (by the GCHQ) in collecting metadata for storage and mining.

Anonymous communication has been pioneered in the work of Chaum, with mix-nets [16] and DC-nets [14] providing the first solutions to the problem of sender-anonymous communication. In particular, a mix-net enables the delivery of a set of messages from n senders to a recipient so that the recipient is incapable of mapping outgoing messages to their respective senders. A DC-net on the other hand, allows n parties to implement an anonymous broadcast channel so that any one of them can use it to broadcast a message to the set of parties without any participant being able to distinguish the source. While initially posed as theoretical constructs, these works have evolved to actual systems that have been implemented and tested, for instance in the case of Mixminion [25], that applies the mix-net concept to e-mail, in the case of Vuvuzela [49] that applies the mix-nets concept to messaging and in the case of Dissent [51] that implements DC-nets in a client-server model.

It is important to emphasize that the adversarial setting we wish to protect against is a model where the adversary has a *global view* of the network, akin say to what a global eavesdropper would have if they were passively observing the Internet backbone, rather than a localized view that a specific server or sub-network may have. Furthermore, the adversary may manipulate messages as they are transmitted and received from users as well as block users adaptively. Note that in a more "localized" adversary setting one may apply concepts like Onion routing [48], e.g., as implemented in the Tor system [27], or Freenet [20] to obtain a reasonable level of anonymity with very low latency. Unfortunately such systems are susceptible to traffic analysis, see e.g., [34], and, in principal, they cannot withstand a global adversary.

[1] Complete quote: "We kill people based on metadata. But that's not what we do with this metadata." General M. Hayden. The Johns Hopkins Foreign Affairs Symposium. 1/4/2014.

Given the complexity of the anonymous communication problem in general, we focus our application objective to the important special case of *anonymous messaging*, i.e., bidirectional communication with *both* sender and receiver anonymity that requires moderately low latency and has relatively small payloads (akin to SMS text messaging). The question we ask is whether it is possible to achieve it with *simulation-based security*[2] while scaling to *hundreds of thousands* of users. In particular, we consider two types of entities in our problem specification, clients and servers, and we ask how is it possible that the servers assist the clients that are online to communicate privately without leaking *any* type of metadata to a global adversary, apart from the fact that they are using the system. Furthermore, we seek a decentralized solution, specifically one where no single entity in the system can break the privacy of the clients even if it is compromised. We allow the adversary to completely control the network as well as a subset of the servers and adaptively drop clients' messages or manipulate them as it wishes.

Our Contributions. We present MCMix, the first anonymous messaging service that offers simulation-based security, under a well specified set of assumptions, and can scale to hundreds of thousands of users. In our solution, we adopt a different strategy compared to previous approaches to anonymous communication. Specifically, we provide a way to cast the problem of anonymous messaging natively in the setting of secure multiparty computation (MPC). MPC, since its initial inception [31], is known to be able to distribute and compute securely any function, nevertheless, it is typically considered to be not particularly efficient for a large number of parties and thus inconsistent with problems like anonymous messaging. However, the commodity-based approach for MPC [7] (client-server model), and more recent implementation efforts such as Fairplay [10], VIFF [23], Sharemind [11], PICCO [53], ObliVM [40], Araki et al. [5] and [30] increasingly suggest otherwise.

We first propose two ideal functionalities that correspond to the dialing operation and the conversation operation. The MCMix system proceeds in rounds, where in each round an invocation of either the dialing or the conversation ideal functionality is performed. The dialing functionality enables clients to either choose to dial another client or check whether anyone is trying to dial them (in practice in most dialing rounds the overwhelming majority of clients will be in dial-checking mode). If a matching pair is determined by the ideal functionality, then the caller will be notified that the other client has accepted their call and the callee will be notified about the caller. Moreover, the ideal functionality will deliver to both clients a random tag that can be thought of the equivalent of a "dead drop" or "rendezvous" point. Subsequently, the clients can access the conversation functionality using the established random tag. When two clients use the same random tag in the conversation functionality, their messages are swapped and thus they can send messages to each other (even concurrently).

The two ideal functionalities provide a useful abstraction of the anonymous messaging problem. We proceed now to describe how they can be implemented by an MPC system. It is easy to see that a straightforward implementation of the functionality programs results in a circuit of size $\Theta(n^2)$, where n is the number of online users accessing the functionalities. Such a solution would be clearly not scalable. We provide more efficient implementations that achieve $O(n \log n)$ complexity in both cases with very efficient constants using state of the art oblivious sorting algorithms [33, 13].

Given our high level functionality realizations, we proceed to an explicit implementation in the Sharemind system [11] using its SecreC programming language [12]. We provide benchmarks for the Dialing and Conversation solutions. The Sharemind platform provides a 3-server implementation of information theoretically secure MPC. Our results showcase that our system can handle hundreds of thousands of users in a reasonable latency (little over a minute), that is consistent with messaging.

In order to provide theoretical evidence of further improving performance and scaling to even larger anonymity sets, we provide a parallelized version of the conversation functionality. Parallelization is a non-trivial problem in our setting since we would like to maintain anonymity across the whole user set; thus, a simplistic approach that breaks users into chunks solving dialing and conversation independently will isolate them to smaller "communication islands"; if two users have to be on the same island in order to communicate, this will lead to privacy loss that is non-simulatable and we would like to avoid. Our parallelized solution manages to make the interaction between islands, in a way that maintains strong privacy guarantees, at the cost of a correctness error that can become arbitrarily small. In this way, by utilizing a large number of servers, we provide evidence that the system can scale up to anonymity sets of up to half a million of users. To sum up, our contributions can be expressed by the following points:

- A model for simulation-based anonymous messaging.
- A realization of this model with a set of programs that are provably secure and expressed in a way so that they can be implemented in any MPC platform.

[2]We use this term to refer to a level of metadata hiding that ensures, in a simulation based sense, that *no information* is leaked to an adversary. This is distinguished from weaker levels of privacy, such as e.g., a differential privacy setting where some controlled but non-trivial amount of information is leaked to the adversary.

- An implementation of our programs in Sharemind that can accomodate anonymity sets of hundreds of thousands of users.
- A novel parallelization technique that allows our system to scale, in theory, even beyond the order of hundreds of thousands of users.

Organization. After shortly presenting some preliminary topics in section 2, we formalize the concept of anonymous messaging via an ideal MPC functionality and introduce the Dialing and Conversation programs in an abstract form that together solve the sender and receiver anonymous messaging problem (cf. Section 3). In Section 4, we present the general architecture of MCMix and in Sections 5 and 6, we propose a way to realize the Dialing and Conversation programs, using MPC. Then, in Section 7, we give more details regarding how the MCMix system implements anonymous messaging in a provably secure and privacy-preserving way. In Section 8, we present the results of benchmarking our prototype and in Section 9, we account for the client-side load of our system. In Section 10, we provide an overview of noticeable anonymous communication systems and when applicable, we compare their performance and security level to MCMix. Finally, in Section 11, we introduce a novel way to parallelize our conversation protocol in order to achieve even better scalability.

2 Background

Secure Multiparty Computation. Secure Multiparty Computation (MPC), is an area of cryptography concerned with methods and protocols that enable a set of users $\mathcal{U} = u_1, \ldots, u_n$ with private data $d_1, \ldots, d_n$ from a domain set D, to compute the result of a public function $f(d_1, \ldots, d_n)$ in a range set Y, without revealing their private inputs. For clarity, we also assume that f accepts $\perp$ as input, which denotes abstain behavior.

Sharemind. Sharemind [11] is an MPC framework that offers a higher level representation of the circuit being computed in the form of a program written in a C-like language, namely the SecreC language [12]. It uses three-server protocols that offer security in the presence of an honest server majority. That is, we assume that no two servers will collude in order to break the systems privacy. Our implementation is designed over the Sharemind system, but the general approach that we introduce for anonymous messaging can also be deployed over other MPC protocols. The security of Sharemind has been analyzed several settings including semi-honest and active attacks (e.g., [11, 43]).

Oblivious Sorting. Sorting is used as a vital part of many algorithms. In the context of MPC, sorting an array of values without revealing their final position, is called oblivious sorting. The first approach to sorting obliviously is using a data-independent algorithm and performing each compare and exchange execution obliviously. This approach uses sorting networks to perform oblivious sorting. Sorting networks are circuits that solve the sorting problem on any set with an order relation. What sets sorting networks apart from general comparison sorts is that their sequence of comparisons is set in advance, regardless of the outcome of previous comparisons. Various algorithms exist to construct simple and efficient networks of depth $\mathcal{O}(\log^2 n)$ and size $\mathcal{O}(n \log^2 n)$. The three more used ones are Batcher's odd-even mergesort and bitonic sort [6] and Shellsort [46]. All three of these networks are simple in principle and efficient. Sorting networks that achieve the theoretically optimal $\mathcal{O}(\log n)$ and $\mathcal{O}(n \log n)$ complexity in depth and total number of comparisons, such as the AKS-network [1] exist, but the constants involved are so large that make them impractical for use. Note that even for 1 billion values, i.e., $n = 10^9$, it holds that $\log n < 30$ so, in practice, the extra log factor is preferable to the large constants. A major drawback of all sorting network approaches is that sorting a matrix by one of its columns would require oblivious exchange operations of complete matrix rows, which would be very expensive.

In recent years, techniques have been proposed from Hamada et. al [33] to use well known data-dependent algorithms, such as quicksort, in an oblivious manner to achieve very efficient implementations, especially when considering a small number of MPC servers, which is very often the case. This approach uses the "shuffling before sorting" idea, which means that if a vector has already been randomly permuted, information leaked about the outcome of comparisons does not leak information about the initial and final position of any element of the vector. More specifically, the variant of quicksort proposed in [33], needs on average $\mathcal{O}(\log n)$ rounds and a total of $\mathcal{O}(n \log n)$ oblivious comparisons. Complete privacy is guaranteed when the input vector contains no equal sorting keys, and in the case of equal keys, their number leaks. Furthermore, performance of the algorithm is data-dependent and generally depends on the number of equal elements, with the optimal case being that no equal pairs exist. Practical results have shown [13] that this quicksort variant is the most efficient oblivious sorting algorithm available, when the input keys are constructed in a way that makes them unique.

In our algorithms, we utilize the Quicksort algorithm together with a secret-shared index vector as described in [13]. This way, each sortable element becomes a unique value-index pair, providing us the optimal Quicksort performance and complete privacy. In addition, it has the added benefit of making the sorting algorithm stable.

Identity-Based Key Agreement Protocols. Like in [39], we make use of identity-based cryptography [45] to circumvent the need for a Public Key Infrastructure (PKI), here, for the computation of the dead drops[3]. In identity-based cryptography, a Key Generation Center (KGC) using a master secret key, generates the users' secret keys, while the users' public keys are a deterministic function of their identity. In an *identity-based key agreement (ID-KA) protocol* (e.g. [32, 44, 47, 18, 52, 29, 50]), any pair of users can execute a GenerateKey algorithm to agree on a shared key value, on input their obtained secret keys and the other user's identity.

In our setting, we will apply ID-KA for the computation of the dead drops, where now the users compute their secret keys by combining partial secret keys issued by the MPC servers. Therefore, we adjust ID-KA to a multiple KGC setting where each MPC server plays the role of a KGC. In general, we can manage distributed key generation in a fault tolerant manner, using threshold secret-sharing techniques. However, since our threat model considers a passive (semi-honest adversary), we consider an m-out-of-m instantiation, keeping protocol description simple. In particular, we can naturally extend a single KGC ID-KA protocol to a setting with m KGCs denoted by $\mathsf{KGC}_1, \ldots, \mathsf{KGC}_m$. In the full version of our paper, we present at length two multiple KGC ID-KA constructions based on ID-KA protocols that use *cryptographic pairings*.

In the first construction, we build upon the SOK ID-KA protocol introduced in [44] and proven secure in [42]. The key agreement in SOK is non-interactive and the shared key between two fixed users is fixed and can be computed only by knowing the other user's identity.

In the second construction, we build upon the ID-KA protocol introduced in [47] as modified in [18] that achieves security *and* forward secrecy as proven in [17]. In this construction, the users must additionally exchange some additional random values in every new session that is necessary for forward secure key agreement.

Both constructions match the original single ID-KA protocols, when $m = 1$. Therefore, it is straightforward that the first construction (resp. the second construction) preserves security (resp. security and forward secrecy) against any polynomially bounded semi-honest adversary that corrupts all-but-one of the m KGCs.

In the current version of MCMix, we do not focus on forward security. Hence, our system's description (cf. Dialing protocol in Section 5) is based on the simpler first ID-KA construction, where knowledge of the users' usernames is enough for shared key computation. Nonetheless, in Section 7 (cf. Remark 5), we briefly discuss on how the second construction could be adopted to a forward-secure version of our system, leaving detailed description for future work.

[3] If users' public keys have been distributed in a PKI setting, then we can turn to the easier solution of classic Diffie-Hellman key exchange for dead drop computation.

3 Ideal Anonymous Messaging

We formalize the concept of anonymous messaging in line with standard MPC security modeling. In particular, we capture the notion of an *ideal MPC functionality* $\mathcal{F}$ that in presence of an ideal adversary $\mathcal{S}$ receives inputs from a number of n users and computes the desired result w.r.t. some program f. An MPC protocol is said to be secure w.r.t. a class of programs, if its execution running in the presence of a real-world adversary results in input/output transcripts that are indistinguishable from the ideal setting that $\mathcal{F}$ specifies for program f.

Subsequently, inspired by Tor, Vuvuzela and other related systems, we make use of the "rendezvous points" idea. Specifically, we instantiate $\mathcal{F}$ w.r.t. two distinct "abstract" programs $\mathtt{DLN}_{\mathsf{abs}}$ and $\mathtt{CNV}_{\mathsf{abs}}$ that reflect the Dialing and Conversation functionalities respectively; the two programs are abstract in the sense that, in this section, they will be described at a high level algorithmic way that we will make concrete in the coming sections. The use of a random rendezvous point in the establishment of a communication channel between two users averts any denial of service attacks targeting specific users by other users at the conversation phase.

Notation. We write $x \xleftarrow{\$} X$ to denote that x is sampled uniformly at random from set X. For a positive integer n, the set $\{1, \ldots, n\}$ is denoted by $[n]$. The j-th component of n-length tuple a is denoted by $a[j]$, i.e. $a := (a[1], \ldots, a[n])$. We use $\overset{c}{\approx}$ to express indistinguishability between transcripts, seen as random variables. By $\mathsf{negl}(\cdot)$ we denote that a function is negligible, i.e. asymptotically smaller than the inverse of any polynomial. We use λ as the security parameter.

Let $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ be a vector of users' inputs. We denote by $\mathrm{EXEC}^{\mathcal{F},f}_{\mathcal{S},\mathbf{x}}(\lambda)$ the transcript of input/outputs in an ideal MPC execution of $\mathcal{F}$ interacting with the ideal adversary $\mathcal{S}$, and by $\mathrm{EXEC}^{\mathbb{P},f}_{\mathcal{A},\mathbf{x}}(\lambda)$ the transcript of inputs/outputs in a real-world execution of MPC protocol $\mathbb{P}$ w.r.t. f under the presence of adversary $\mathcal{A}$. By PPT, we mean that $\mathcal{A}$ runs in probabilistic polynomial time.

Entities and threat model. We consider a client-server MPC setting. Namely, the entities involved in an MPC protocol $\mathbb{P}$ are (i) a number of n users $u_1, \ldots, u_n$ that provide their inputs $\langle x_1, \ldots, x_n \rangle$ and (ii) a number of m servers $\mathsf{Ser}_1, \ldots, \mathsf{Ser}_m$ that collectively compute an evaluation on the users' inputs w.r.t. a program f. The users engaged in a specific MPC execution round form an ac-

tive set $\mathcal{U}_{\mathsf{act}}$. We consider an *ad-hoc* setting [8] of secure computation, where the program f is known in advance, but *not* the active user set $\mathcal{U}_{\mathsf{act}}$.

An adversary against $\mathbb{P}$ is allowed to have a *global view* of the protocol network. In addition, it may corrupt up to a fixed subset of θ servers and has limited computational resources preventing it from breaking the security of the underlying cryptographic primitives.

In standard MPC cryptographic modeling, the security of $\mathbb{P}$ is argued w.r.t. the functionality $\mathcal{F}$ that specifies an "ideal" evaluation of f, where the privacy leakage is the minimum possible for the honest users. Thus, indistinguishability between the ideal and the real world setting implies that an adversary against $\mathbb{P}$ obtains essentially no more information than this minimum leakage. In our description, $\mathcal{F}$ merely leaks whether an honest user is online or not. This information is impossible to hide against a network adversary and hence it is a minimum level of leakage. On the other hand, information that can be typically inferred by traffic analysis, is totally protected by $\mathcal{F}$. This level of anonymity, sometimes referred to as unobservability, requires the participation of all online parties and the generation of "dummy traffic" independently of whether or not they wish to send a message in a particular round. As a result, any protocol $\mathbb{P}$ that securely realizes $\mathcal{F}$ where f represents a dialing or conversation program, should incorporate such a methodology. As we demonstrate, using MPC to realize $\mathbb{P}$ is a natural way to determine the appropriate level and form of "dummy traffic" needed to realize this level of anonymity.

An ideal MPC functionality for a family of programs. In a messaging system, dialing and conversation among users are operations where conflicts are likely to appear, e.g. two users may dial the same person, or conversation may be accidentally established on colluding communication channels (three equal rendezvous points are computed). One can think several other examples of operations where conflicts are possible, such as election tally where exactly one out of multiple ballots per voter must be counted, or deciding on the valid sequence of transactions on a blockchain ledger when forking occurs. Any program implementing this type of an operation must be able to resolve these conflicts. The way that conflict resolution is achieved, may depend on parameters like computation efficiency, communication complexity or user priority, yet in any case, a set of programs that implement the same operation are in some sense equivalent and may be clustered under the same family. A plausible requirement is that the choice of the family member that will be utilized should not affect the security standards of the operation implementation.

Consequently, in an MPC setting that supports the realization of any program in the family, it is desirable that security is preserved w.r.t. to the entire family, so that one can choose the family member that suits their custom requirements. To express this formally, we introduce a relaxation of the usual MPC functionality. Namely, the relaxed ideal MPC functionality $\mathcal{F}$ is for a family of programs $\{f_z\}_z$ in the presence of an ideal adversary $\mathcal{S}$ that chooses the index z (this is the relaxation), where z can be parsed as the "code" that determines the family member f_z. The program f_z accepts as input a vector $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ of (i) valid messages from some domain D or (ii) $\perp$, if the user is inactive, i.e. not in $\mathcal{U}_{\mathsf{act}}$. In our description, computation takes place even when a subset of users abstain from the specific execution by not providing inputs. To formalize the abstain behavior of user u_i, for every $i \in [n]$ we define an '$\mathsf{abstain}_i(\cdot)$' predicate over $D \cup \{\perp\}$ as follows:

$$\mathsf{abstain}_i(x_i) := \begin{cases} 1, & \text{if } x_i = \perp \\ 0, & \text{if } x_i \in D \end{cases} \tag{1}$$

The ideal MPC functionality $\mathcal{F}$ is presented in Fig. 1. Note that the relaxation suggests that the users will receive output from a program f_z for z that will be the ideal adversary's choosing.

Ideal MPC functionality $\mathcal{F}$ for programs $\{f_z\}_z$

– Upon receiving 'start' from $\mathcal{S}$, it sets the status to 'input' and initializes two lists L_{input} and L_{corr} as empty.

– Upon receiving $(\mathsf{corrupt}, u_i)$ from $\mathcal{S}$, it adds u_i to L_{corr}.

– Upon receiving $(\mathsf{send\_input}, x_i)$ from u_i, if $u_i \in L_{\mathsf{corr}}$, then it sends $(\mathsf{send\_input}, u_i, x_i)$ to $\mathcal{S}$. If $u_i \notin L_{\mathsf{corr}}$, then it sends (i) $\big(\mathsf{send\_input}, u_i, \mathsf{abstain}_i(x_i)\big)$ to $\mathcal{S}$, where $\mathsf{abstain}_i(\cdot)$ is defined in Eq. (1).

– Upon receiving $(\mathsf{receive\_input}, u_i, \tilde{x}_i)$ from $\mathcal{S}$, if (i) the status is 'input' and (ii) $(u_i, \cdot) \notin L_{\mathsf{input}}$, then if $u_i \notin L_{\mathsf{corr}}$, it adds (u_i, x_i) to L_{input}, else it adds $(u_i, \tilde{x}_i)$ to L_{input}.

– Upon receiving $(\mathsf{compute}, z)$ from $\mathcal{S}$, if L_{input} contains records for all users in $\mathcal{U}_{\mathsf{act}}$, it executes the following steps: first, then it computes the value vector

$$\mathbf{y} = \langle y_1, \ldots, y_n \rangle \leftarrow f_z(x_1, \ldots, x_n) \, .$$

Then, it sends y_i to u_i for $i, \ldots, n$, (hence, $\mathcal{S}$ obtains $\{y_i\}_{u_i \in L_{\mathsf{corr}}}$).

Figure 1: The ideal MPC functionality $\mathcal{F}$ for family of programs $\{f_z : \big(D \cup \{\perp\}\big)^n \longrightarrow Y\}_z$, interacting with the ideal adversary $\mathcal{S}$.

The security of a real-world MPC protocol $\mathbb{P}$ is defined w.r.t. a class of programs $\mathbf{F}$ as well as a family selected from $\mathbf{F}$ as follows:

Definition 1. *Let $\mathbb{P}$ be an MPC protocol with n users and m servers and let $\mathbf{F}$ be a class of programs. We say that*

$\mathbb{P}$ *is a* (θ, m)-secure MPC protocol *w.r.t.* $\{f_z\}_z \subseteq \mathbf{F}$*, if for every active user set* $\mathcal{U}_{\mathsf{act}} \in \mathcal{U}$*, every program* f_z*, every input vector* $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ *and every PPT adversary* $\mathcal{A}$ *corrupting up to* θ *out of m servers, there is an ideal adversary* $\mathcal{S}$ *s.t.*

$$\text{EXEC}^{\mathcal{F}}_{\mathcal{S},\mathbf{x}}(\lambda) \overset{c}{\approx} \text{EXEC}^{\mathbb{P},f_z}_{\mathcal{A},\mathbf{x}}(\lambda) \,.$$

The family of programs $\mathtt{DLN}_{\mathsf{abs}}$ ***and*** $\mathtt{CNV}_{\mathsf{abs}}$**.** An anonymous messaging scheme comprises the following two functionalities: (i) the *Dialing functionality*, which consists of the computation of a rendezvous point for a given pair of users who want to communicate, and (ii) the *Conversation functionality*, which represents the actual exchange of messages. For the families $\mathtt{DLN}_{\mathsf{abs}}$ and $\mathtt{CNV}_{\mathsf{abs}}$, the parameter z, enables the adversary to choose (i) *how to handle collisions between multiple dialers* in the case of $\mathtt{DLN}_{\mathsf{abs}}$, and (ii) *how to handle the presence of three or more equal dead drops* in the case $\mathtt{CNV}_{\mathsf{abs}}$ (which happens only in the case of malicious users). We note that this minimum level of adversarial manipulation does not affect the security features of the anonymity system, yet it allows for substantial performance gains in terms of the implementation.

We formally express the above functionalities by instantiating the generic MPC functionality $\mathcal{F}$ w.r.t. the *Dialing program family* $\mathtt{DLN}_{\mathsf{abs}}$ and the *Conversation program family* $\mathtt{CNV}_{\mathsf{abs}}$ (i.e. we set f as $\mathtt{DLN}_{\mathsf{abs}}$ and $\mathtt{CNV}_{\mathsf{abs}}$). We note that for both the dialing and conversation program families, the verification that the parameter z has the proper structure can be suitably restricted so that it is tested efficiently by the program. For brevity, we omit further details.

The Dialing program family $\mathtt{DLN}_{\mathsf{abs}}$. In the Dialing functionality, a rendezvous point for users u_i and u_j is set when two requests of the form (DIAL, u_i, u_j) and $(\text{DIALCHECK}, u_j)$ have been produced. Thus, the Dialing program family $\mathtt{DLN}_{\mathsf{abs}}$ receives inputs that are vectors of $(\text{DIAL}, \cdot, \cdot)$ or $(\text{DIALCHECK}, \cdot)$ requests, as well as $\perp$ to denote user inactivity. That is, $\mathcal{U}_{\mathsf{act}}$ is the set of users that do not provide a $\perp$ input. The program $\mathtt{DLN}_{\mathsf{abs}}$ is parameterized by z, that specifies a deterministic program $R^z_{\mathtt{DLN}}(\cdot,\cdot)$ over pairs of inputs to resolve the case where more than one dial requests address the same user/dial checker. The Dialing program family $\mathtt{DLN}_{\mathsf{abs}}$ is presented formally in Figure 2.

By the definition of $\mathtt{DLN}_{\mathsf{abs}}$, two active users u_i, u_j that have submitted matching dialing and dial check requests are going to be provided the same random integer $t_i = t_j \in \{t_{i,j}, t_{j,i}\}$, which establishes a rendezvous point. We will refer to these non-$\perp$ values in $t_1, \ldots, t_n$ as *dead drops*. In addition, $\mathtt{DLN}_{\mathsf{abs}}$ returns to each dialchecker u_i a bit c_i which is 1 iff u_i has succesfully established a rendezvous with some dialer. Such information is reasonable to be provided to a dialchecker, as t_i might be a random value that is not an actual dead-drop. Hence, the bit c_i communicates to the dialchecker that she has an incoming call (if nobody calls the dialchecker, then a random dead drop value is returned that nobody else shares with her). On the other hand, a dialer should not be able to infer information about the dial traffic and availability concerning some dialchecker, therefore $\mathtt{DLN}_{\mathsf{abs}}$ does not provide this success check to the dialers.

The Conversation program family $\mathtt{CNV}_{\mathsf{abs}}$. Given the establishment of the dead drops, as set by $\mathtt{DLN}_{\mathsf{abs}}$, the Conversation program family $\mathtt{CNV}_{\mathsf{abs}}$ realizes the operation of message exchange, where messages lie in some space $\mathcal{M}$. The program family $\mathtt{CNV}_{\mathsf{abs}}$ is presented in Figure 3.

Program family $\mathtt{DLN}_{\mathsf{abs}}$ ***parameterized by*** z

– **Domain:** $(D_{\mathtt{DLN}_{\mathsf{abs}}} \cup \{\perp\})^n$, where

$$D_{\mathtt{DLN}_{\mathsf{abs}}} := \Big\{ \big\{(\text{DIAL}, u_i, u_j)\big\}, (\text{DIALCHECK}, u_i) \Big\}_{u_i \neq u_j \in \mathcal{U}}$$

Namely, let $\mathcal{U}_{\mathsf{act}} := \{u_i \in \mathcal{U} \mid x_i \neq \perp\}$; a valid input x_i for user $u_i \in \mathcal{U}_{\mathsf{act}}$ consists of either (i) a (DIAL, u_i, u_j) request for some user u_j that u_i wants to dial, or (ii) a $(\text{DIALCHECK}, u_i)$ request.
For a vector of inputs $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$, if $x_i = (\text{DIALCHECK}, u_i)$ then $M_i(\mathbf{x}) = \{j \mid x_j = (\text{DIAL}, u_j, u_i)\}$, else is $\emptyset$. Parse z as a deterministic program $R^z_{\mathtt{DLN}}$, such that for any $\mathbf{x}$ if $M_i(\mathbf{x}) \neq \emptyset$, then $R^z_{\mathtt{DLN}}(i, \mathbf{x}) \in M_i(\mathbf{x})$, else it is equal to $\perp$.

– **Range:** $Y_{\mathtt{DLN}_{\mathsf{abs}}} := \langle \{y_i \mid y_i \in [a,b]\} \rangle_{u_i \in \mathcal{U}_{\mathsf{act}}}$, where $[a,b]$ is a predetermined integer interval.

– **Function:** On input a vector $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ where each non-$\perp$ value x_i is either a (DIAL, u_i, u_j) request, or a $(\text{DIALCHECK}, u_i)$ request, $\mathtt{DLN}_{\mathsf{abs}}$ computes a vector $\mathbf{y} = \langle y_i \rangle_{u_i \in \mathcal{U}_{\mathsf{act}}}$, as follows:

• Let $\mathcal{I}_{\mathsf{act}} := \{i \mid u_i \in \mathcal{U}_{\mathsf{act}}\}$ be the set of indices that refer to active users. For $i, j \in \mathcal{I}_{\mathsf{act}}$, $\mathtt{DLN}_{\mathsf{abs}}$ samples distinct random integers $t_{i,j}$ from range $[a,b]$.

• For every $i \in \mathcal{I}_{\mathsf{act}}$:

∘ If $x_i = (\text{DIAL}, u_i, u_j)$, then if there is a $j \in \mathcal{I}_{\mathsf{act}}$ such that $x_j = (\text{DIALCHECK}, u_j)$ and $i = R^z_{\mathtt{DLN}}(j, \mathbf{x})$, then it sets $t_i = t_{i,j}$. Otherwise (i.e., there is no such j), it sets $t_i = t_{i,i}$. In both cases, it sets $y_i = t_i$.

∘ If $x_i = (\text{DIALCHECK}, u_j)$, then if there is a $j \in \mathcal{I}_{\mathsf{act}}$ such that $j = R^z_{\mathtt{DLN}}(i, \mathbf{x}) \neq \perp$, then it sets $t_i = t_{i,j}$ and a bit $c_i = 1$. Otherwise (i.e., there is no such j), it sets $t_i = t_{i,i}$ and a bit $c_i = 0$. In both cases, it sets $y_i = (t_i, c_i)$.

• It returns the value vector $\mathbf{y} := \langle y_i \rangle_{u_i \in \mathcal{U}_{\mathsf{act}}}$.

Figure 2: The Dialing program family $\mathtt{DLN}_{\mathsf{abs}} : (D_{\mathtt{DLN}_{\mathsf{abs}}} \cup \{\perp\})^n \longrightarrow Y_{\mathtt{DLN}_{\mathsf{abs}}}$ with parameter z, where non-$\perp$ range values are integers sampled from range $[a,b]$.

By the definition of $\mathtt{CNV}_{\mathsf{abs}}$, if every dead drop is not shared among three or more users, then two users u_i, u_j are going to exchange their messages m_i, m_j only if they provide the same dead drop $t_i = t_j$. Recall that if the dead drops are computed as outputs of the Dialing program family $\mathtt{DLN}_{\mathsf{abs}}$ w.r.t. the same active set $\mathcal{U}_{\mathsf{act}}$, then no more than two users share the same dead drop, which implies the correctness of $\mathtt{CNV}_{\mathsf{abs}}$. In the other cases, either (i) there is no matching dead drop or (ii) more than 2 matching dead drops exist. In case (ii), the parameter z specifies a deterministic program $R^z_{\mathtt{CNV}}$ among inputs which in turn determines the pair of matching dead drops. In any case, when a message exchange fails for some user, then $\mathtt{CNV}_{\mathsf{abs}}$ returns back this message to the user for resubmission in an upcoming round.

Program family $\mathtt{CNV}_{\mathsf{abs}}$ ***parameterized by*** z

– **Domain:** $(D_{\mathtt{CNV}_{\mathsf{abs}}} \cup \{\perp\})^n$, where

$$D_{\mathtt{CNV}_{\mathsf{abs}}} := \left\{(\mathrm{CONV}, t_i, m_i)\right\}_{u_i \in \mathcal{U}}^{t_i \in [a,b], m_i \in \mathcal{M}}$$

Namely, let $\mathcal{U}_{\mathsf{act}} := \{u_i \in \mathcal{U} \mid x_i \neq \perp\}$; a valid input for user u_i consists of a $(\mathrm{CONV}, t_i, m_i)$ request for rendezvous point tagged by t_i for sending message m_i.
For a vector of inputs $\mathbf{x}$, define $N_i(\mathbf{x}) = \{j \mid x_j = (\mathrm{CONV}, t_i, m_j)\}$. Parse z as a deterministic program $R^z_{\mathtt{CNV}}$, such that for any $\mathbf{x}$ if $N_i(\mathbf{x}) \neq \emptyset$ then $R^z_{\mathtt{CNV}}(i, \mathbf{x}) \in N_i(\mathbf{x})$, else it is equal to $\perp$.

– **Range:** $\langle \{m_i \mid m_i \in \mathcal{U}_{\mathsf{act}}\} \rangle_{u_i \in \mathcal{U}_{\mathsf{act}}}$.

– **Function:** On input a vector $\langle x_1, \ldots, x_n \rangle$ where each non-$\perp$ value x_i is a $(\mathrm{CONV}, t_i, m_i)$ request, $\mathtt{CNV}_{\mathsf{abs}}$ returns a value $\mathbf{y} = \langle y_i \rangle_{u_i \in \mathcal{U}_{\mathsf{act}}}$, as follows:

- Let $\mathcal{I}_{\mathsf{act}} := \{i \mid u_i \in \mathcal{U}_{\mathsf{act}}\}$ be the set of indices that refer to active users. For every $i \in \mathcal{I}_{\mathsf{act}}$: if $j = R^z_{\mathtt{CNV}}(i, \mathbf{x}) \neq \perp$, then it sets $y_i = m_j$. Otherwise, it sets $y_i = m_i$.
- It returns the value vector $\mathbf{y} := \langle y_1, \ldots, y_n \rangle$.

Figure 3: The Conversation program family $\mathtt{CNV}_{\mathsf{abs}} : (D_{\mathtt{CNV}_{\mathsf{abs}}} \cup \{\perp\})^n \longrightarrow Y_{\mathtt{CNV}_{\mathsf{abs}}}$ with parameter z, where non-$\perp$ dead drop values are integers sampled from a predetermined interval $[a,b]$ and messages are taken from space $\mathcal{M}$.

Anonymous Messaging Systems. An anonymous messaging system is a pair of protocols that realize any two members of the families $\mathtt{DLN}_{\mathsf{abs}}$ and $\mathtt{CNV}_{\mathsf{abs}}$ under the security guarantee provided in Definition 1. Given such realization, anonymous communication can be achieved as a continuous sequence of interleaved invocations of dialing and conversation. In principle, dialing can be more infrequent compared to conversation, e.g., perform only a single dialing every certain number of conversation "rounds." We note that the value of our relaxation of MPC security is on the fact that we can realize any member of the respective families.

Sharemind as a secure MPC platform. As already discussed, Sharemind will be the building platform for the implementation of our anonymous messaging scheme. As shown in [11], Sharemind is information theoretically secure against a passive (honest-but-curious) adversary that corrupts 1-out-of-3 MPC servers. Subsequent work [43] provides interesting directions regarding the active security of Sharemind, even specifically for novel oblivious sorting algorithms [38]. However, in our implementation, we consider the case of passive security.

In more detail, let $\mathbb{S}$ be the class of programs that can be written in Sharemind's supporting language SecreC. In our analysis, we claim that Sharemind operates as a $(1,3)$-*secure MPC platform for any program family member of the class* $\mathbb{S}$ *against passive adversaries*, as in Definition 1. Using the above claim, we provide two SecreC programs and prove that they realize two members of the families $\mathtt{DLN}_{\mathsf{abs}}$ and $\mathtt{CNV}_{\mathsf{abs}}$, (cf. Sections 5 and 6) hence obtaining an anonymous messaging system.

Alternative MPC platforms. For the purpose of the proposed anonymous messaging, Sharemind can be viewed as a black box providing MPC functionality. Hence, it is also possible to swap Sharemind for another MPC implementation providing different deployment or security properties. For example, recently, Furukawa et al. proposed a highly-optimised protocol for computation with an honest majority and security for malicious adversaries [30], that was further improved by Araki et al. [4]. Similarly, it is possible to support more than three computation parties. SPDZ [24] is a practical MPC implementation that provides statistical security against an active adversary that corrupts up to $m-1$ parties. Its online computation and communication complexities are both $O(m|C| + m^3)$, where $|C|$ stands for the computable arithmetic circuit size. In our setting, the lower bound for this circuit size is the number of users, n. Both actively secure MPC implementations mentioned here work in a preprocessing (i.e. offline/online) model.

4 System Architecture

Our work is presented in a manner that makes it easy to implement using any of the aforementioned MPC protocols in Section 2 and with any number of servers. However, for the sake of presentation, we assume three MPC servers, denoted by $\mathsf{Ser}_1, \mathsf{Ser}_2, \mathsf{Ser}_3$. As a general idea, the protocol works in rounds, where in each round users break their input into shares and forward the shares to the servers, with each server receiving one share. Then, the servers interactively compute the desired output shares,

which are in turn returned to the respective users. In our description, for simplicity we choose additive secret sharing, but other sharing schemes would not affect the functionality of our architecture.

Besides the MPC servers, the complete architecture of our system comprises an *entry* and an *output* server used to handle user requests. The entry and output servers may be located on the same or on different physical machines and are only trusted to relay messages.

Registration phase. At the beginning, the MPC servers $\mathsf{Ser}_1, \mathsf{Ser}_2, \mathsf{Ser}_3$ run the Setup phase of the secure multiple KGC ID-KA protocol (cf. Section 2) playing the role of three KGCs: $\mathsf{KGC}_1, \mathsf{KGC}_2, \mathsf{KGC}_3$ generating their partial master secret keys $\mathsf{msk}_1, \mathsf{msk}_2, \mathsf{msk}_3$.

Before starting to use the system, each user u_i registers with a unique username UN_i of 64 bits. Then, each MPC server Ser_ℓ, $\ell \in \{1,2,3\}$ generates u_i's partial secret key $\mathsf{sk}_{i,\ell}$ and sends it u_i. Upon receiving $\mathsf{sk}_{i,1}, \mathsf{sk}_{i,2}, \mathsf{sk}_{i,3}$, u_i combines the partial keys to obtain her ID-KA secret key sk_i as output of the secret key derivation algorithm. In addition, by performing standard key exchange operation, u_i obtains a symmetric key $k_{i,\ell}$ for communication with each of $\mathsf{Ser}_\ell, \ell \in \{1,2,3\}$. From this point on, any authentication and communication between u_i and the servers is performed using symmetric key cryptography. In the client-side, u_i can compute u_j's ID-KA public key pk_j as a function of her username UN_j and agree on the ID-KA key $K_{i,\ell}$. In the rest of this paper, we set the length of the usernames $\mathsf{UN}_1, \ldots, \mathsf{UN}_n \in \mathcal{UN}$, to be 64 bits.

Main phase. The main phase of the protocol for each round r, consists of the following steps:

1. Encoding: Each user u_i generates a request a_i, as input to the MPC that is to be executed.

2. Secret sharing: Each user u_i creates three shares of the request using additive secret sharing, so that $a_i = a_{i,\mathsf{Ser}1} + a_{i,\mathsf{Ser}2} + a_{i,\mathsf{Ser}3}$ holds. Note that the subscripts denote the MPC server that will process the share. Then each of the three shares intended for one of the MPC servers is encrypted with the respective symmetric key $k_{i,\ell}$ using authenticated encryption. The result is a triple of the form $a'_i = (a'_{i,\mathsf{Ser}1}, a'_{i,\mathsf{Ser}2}, a'_{i,\mathsf{Ser}3})$, where $a'_{i,\mathsf{Ser}_\ell} := \mathsf{Enk}_{k_{i,\ell}}(a_{i,\mathsf{Ser}_j})$, $\ell = \{1,2,3\}$. Then each user sends the encrypted shares along with her username UN_i, as a package to the entry server.

3. MPC input preparation: Before the start of round r, the entry server groups the packages received already and sends each share along with its associated username to the respective MPC servers. It is important to note that the use of an entry server is only to synchronize the MPC servers and to provide the shares in the same order to each of them. For notation simplicity and without loss of generality, we assume that the entry server arranges u_i as the user that submitted the i-th input. Then, each MPC server Ser_ℓ receives a sequence of the form $a'_{\mathsf{Ser}_\ell} = \langle a'_{1,\mathsf{Ser}_\ell}, \cdots, a'_{n,\mathsf{Ser}_\ell} \rangle$. We denote as n the number of users that provided an input in round r. In addition to a'_{Ser_ℓ}, the MPC servers also receive a sequence of the users' usernames in corresponding order, that is a sequence of the form $\mathbf{UN} = \langle \mathsf{UN}_1, \cdots, \mathsf{UN}_n \rangle$, where UN_i is the registered username of the user that provided input i.

4. Order check: Each MPC server computes a hash of the usernames in the order they appear in its input sequence, as $H(\mathsf{UN}_1 || \cdots || \mathsf{UN}_n)$, and exchanges it with the other MPC servers. In case the three hashes do not match, it is implied that the order of the usernames provided to the three servers was different. Thus, a denial of service attack has taken place by either the entry server or one of the MPC servers (considering they reported a false hash). This step is optional when considering only privacy implications of a malicious entry server.

5. Decryption and authentication: At this point, authentication is performed implicitly by each server via decrypting the received share with the symmetric key corresponding to the username that came with the share. Thus shares $a_{\mathsf{Ser}_\ell} = \langle a_{\mathsf{Ser}_\ell,1}, \cdots, a_{\mathsf{Ser}_\ell,n} \rangle$, with $a_{\mathsf{Ser}_\ell,i} := \mathsf{Dec}_{k_{i,\ell}}(a'_{\mathsf{Ser}_\ell,i})$ are ready for the MPC.

6. MPC algorithm: The MPC servers execute the MPC protocol.

7. Encryption and return: Each MPC server encrypts each output share with the respective symmetric key and forwards shares of the form $b'_{\mathsf{Ser}_\ell} = \langle b'_{1,\mathsf{Ser}_\ell}, \cdots, b'_{n,\mathsf{Ser}_\ell} \rangle$ to the output server. The output server collects the shares corresponding to the same user and returns a package of the form $(b'_{i,\mathsf{Ser}_1}, b'_{i,\mathsf{Ser}_2}, b'_{i,\mathsf{Ser}_3})$ to each user u_i.

8. Decryption and reconstruction: Each user decrypts the received shares with the respective symmetric key and adds them, resulting in $b_i = b_{i,\mathsf{Ser}_1} + b_{i,\mathsf{Ser}_2} + b_{i,\mathsf{Ser}_3}$, where $b_{i,\mathsf{Ser}_\ell} = \mathsf{Dec}_{k_{i,\ell}}(b'_{i,\mathsf{Ser}_\ell})$. The value b_i is the final output of the MPC protocol for each user u_i for round r.

Remark 1. The entry and output servers are used for practical reasons. The main function they perform is grouping the received packages of shares and forwarding them to/from the servers. As they have no information about the symmetric keys exchanged between users and servers at the registration phase, they schedule the traffic consisting of encrypted shared data. Hence, if entry and output servers are malicious, they can do no more than an adversary controlling the network.

5 The Dialing Protocol

The dialing protocol enables a user u_i to notify another user u_j that she wants to start a conversation, much like how the telephone protocol works. The protocol runs in

rounds to deter possible timing attacks, where in each round, every online active user will either send a DIAL request or a DIALCHECK request. All requests are mutually indiscriminate. For clarity, we first provide a description of the Dialing protocol steps. Then, we proceed with the efficient program $\mathtt{DLN}_{\mathsf{sort}}$ implementing it.

Protocol description. The protocol runs in seven steps, where steps 2-6 are executed by the MPC servers. Steps 1 and 7 are executed locally by each user.

1. Encoding: The inputs $x_1,\ldots,x_n$ are of the form of (DIAL,u_i,u_j) requests, $(\text{DIALCHECK},u_i)$ requests, or $\perp$, representing the action each user takes for this dialing round. For simplicity, assume that the users are enumerated as $u_1,\ldots,u_n$ consistently with the input sequence $x_1,\ldots,x_n$, i.e. u_i is the user that submitted the i-th input. As a result, the active users that submitted non-$\perp$ values, are enumerated as $u_1,\ldots,u_{\mathsf{act}}$, where act is the size of the active set $\mathcal{U}_{\mathsf{act}}$. The inputs of the active users are encoded as triples of the form $a_i := (a_i[1],a_i[2],a_i[3])$ where the third component is an *input wire ID* wid_i. The wire IDs are initially set to zero, but in the following Step 2, each wid_i will be set unique for each u_i.

In particular, if u_i wants to dial u_j, then the (DIAL,u_i,u_j) request is encoded as $(\mathsf{UN}_i,\mathsf{UN}_j,0)$ where UN_i and UN_j are the usernames of the dialer and the dialee respectively. If u_i is a dial checker, then the $(\text{DIALCHECK},u_i)$ request is encoded as $(C,\mathsf{UN}_j,0)$, where (i) C is a special value designated to denote a dial check and is different from any possible username value, and (ii) UN_j is the checker's own username.

2. Assigning wire ID values: As a first step, the MPC protocol assigns unique wire IDs for each user. This is done by setting the third component $a_i[3]$ of the encoded triple a_i to i. Therefore, for each u_i, we have that $\mathsf{wid}_i := i$. These wire IDs are needed internally for the MPC calculation and express the order in which the inputs were received so that the respective outputs will be delivered in the same order.

3. Checking input validity: The protocol then checks if any of the first two members of each triple, denoted by $a_i[1]$ and $a_i[2]$, is equal to the submitter's username. This check ensures that inputs are encoded in a way that does not compromise the security of the system. The threat here is that a user u_i might try to impersonate a user u_j by encoding a DIALCHECK input as $a_i = (C,\mathsf{UN}_j,\mathsf{wid}_i)$. That attack would allow user u_i to receive a dial request that was intended for user u_j. A similar problem arises when considering a user u_i encoding a DIAL input as $a_i = (\mathsf{UN}_l,\mathsf{UN}_j,\mathsf{wid}_i)$. In this case, user u_j will think the dial originated from user u_l. To avert such impersonation attacks, it is enough for the MPC protocol to check that either the first or the second member of an input tuple is equal to the username of the user that submitted that input. This, along with the fact that the input is sent from the user to each MPC server using authenticated encryption (cf. step 2 of the architecture in section 4) guarantees that no impersonation attack can take place.

In more detail, if the input is a DIALCHECK request, then this check ensures that the second member of the tuple is the user's own username. In the case of a DIAL request, the check ensures that a user can only impersonate another user when she dials herself, that is a request of the form $a_i = (\mathsf{UN}_j,\mathsf{UN}_i,\mathsf{wid}_i)$ is created by user u_i. In this case, this request does not affect the protocol. If the check fails for the encoded input a_i, then the input is set to $a_i = (0,0,\mathsf{wid}_i)$ and does not affect the protocol.

4. Sorting by usernames: The encoded input triples are first sorted according to their second components using the oblivious Quicksort algorithm of [33], implemented according to [13]. Observe that every non-zero second component is either (i) the username UN_j of dialee u_j in a dial request from some user u_i, or (ii) the username UN_j from dial checker u_j. Thus, when a triple $(C,\mathsf{UN}_j,\mathsf{wid}_j)$ is adjacent to some triple $(\mathsf{UN}_i,\mathsf{UN}_j,\mathsf{wid}_i)$ with a non-zero second component, this determines a dial pair between u_i, u_j. We note that two special conflict cases may appear:

I. $(C,\mathsf{UN}_j,\mathsf{wid}_j)$ is adjacent to two dial triples as $\ldots,(\mathsf{UN}_i,\mathsf{UN}_j,\mathsf{wid}_i),(C,\mathsf{UN}_j,\mathsf{wid}_j),(\mathsf{UN}_{i'},\mathsf{UN}_j,\mathsf{wid}_{i'}),\ldots$

II. Two or more adjacent dial triples correspond to $(C,\mathsf{UN}_j,\mathsf{wid}_j)$. The sorting would then appear as $\ldots,(\mathsf{UN}_{i'},\mathsf{UN}_j,\mathsf{wid}_{i'}),(\mathsf{UN}_i,\mathsf{UN}_j,\mathsf{wid}_i),(C,\mathsf{UN}_j,\mathsf{wid}_j),\ldots$

5. Connecting neighbors: Next, requests are processed individually by looking at both their neighbors' triples to determine if there is a dial for any given dial check request. Of course, requests at the first and last place of the sorted vector need only look at one neighbor. Thus, we can claim that any dial check request will have a suitable dial request as its neighbor or not at all.

In more detail, for every user u_i, the protocol produces a pair $b := (b_i[1],b_i[2])$, where $b_i[2]$ is wid_i and $b_i[1]$ is either (i) the username UN_j of some user u_j that dialed u_i, or (ii) 0, if no dial request has been made for u_i, or u_i has made a dial request.

6. Sorting by wire IDs: As a final sorting step, the protocol needs to sort the processed requests according to their wire IDs in order for the correct requests to be forwarded to each user. The latter sort, performed on $\langle b_1,\ldots,b_{\mathsf{act}}\rangle$ according to the wire IDs can again be implemented by the Quicksort algorithm of [33].

The result of the last sorting is a vector $\langle\hat{b}_1,\ldots,\hat{b}_{\mathsf{act}}\rangle$ where $\hat{b}_i$ is a pair $(\hat{b}_i[1],\hat{b}_i[2])$ that corresponds to u_i and $\hat{b}_1$ is essentially either (i) a username UN_j or (ii) a zero value, in both cases indexed by $\hat{b}_2 := \mathsf{wid}_i$.

The Dialing Program $\mathtt{DLN}_{\mathsf{sort}}$

Input: a sequence $\langle x_1,\ldots,x_n\rangle$ where x_i is either a (DIAL,u_i,u_j) request, a $(\text{DIALCHECK},u_i)$ request, or $\perp$. All $\perp$ inputs are stacked last.

Output: a sequence $\langle y_i\rangle_{i:x_i\neq\perp}$, where y_i either is a κ-bit integer t_i, if $x_i=(\text{DIAL},u_i,u_j)$, or a pair of a κ-bit integer t_i and a bit c_i, if $x_i=(\text{DIALCHECK},u_i)$.

---

1. For each $i\leftarrow 1,\ldots,n$
if $x_i=\perp$ **then**
  Set $\mathsf{act}:=i-1$;
  Break loop ;
else if $x_i=(\text{DIAL},u_i,u_j)$ **then**
  Set $a_i:=(a_i[1],a_i[2],a_i[3])\leftarrow(\mathsf{UN}_i,\mathsf{UN}_j,0)$;
else if $x_i=(\text{DIALCHECK},u_i)$ **then**
  Set $a_i:=(a_i[1],a_i[2],a_i[3])\leftarrow(C,\mathsf{UN}_i,0)$;
end if
2. For each $i\leftarrow 1,\ldots,\mathsf{act}$
Set wid_i as $a_i[3]\leftarrow i$;
3. For each $i\leftarrow 1,\ldots,\mathsf{act}$
if $a_i[1]\neq\mathsf{UN}_i$ AND $a_i[2]\neq\mathsf{UN}_i$ **then**
  Set $a_i[1]=a_i[2]=0$;
end if
4. $\langle a_i\rangle_{i:x_i\neq\perp}$ according to second coordinate using Quicksort;
5. For each $i\leftarrow 1,\ldots,\mathsf{act}$
if $a_i[1]=C$ AND $a_i[2]=a_{i-1}[2]$ **then**
  Set $b_i:=(b_i[1],b_i[2])\leftarrow(a_{i-1}[1],a_i[3])$;
else if $a_i[1]=C$ AND $a_i[2]=a_{i+1}[2]$ **then**
  Set $b_i:=(b_i[1],b_i[2])\leftarrow(a_{i+1}[1],a_i[3])$;
else
  Set $b_i:=(b_i[1],b_i[2])\leftarrow(0,a_i[3])$;
end if
6. Sort tuples $\langle b_i\rangle_{i:x_i\neq\perp}$ according to second coordinate using Quicksort;
7. For each $i\leftarrow 1,\ldots,\mathsf{act}$
if $a_i[1]=\mathsf{UN}_i$ **then**
  Set $t_i\leftarrow H\big(\mathsf{GenerateKey}(a_i[1],a_i[2]),r\big)$;
  Set $y_i\leftarrow t_i$;
else if $a_i[1]=C$ AND $b_i[1]\in\mathcal{UN}$ **then**
  Set $t_i\leftarrow H\big(\mathsf{GenerateKey}(a_i[1],b_i[1]),r\big)$;
  Set $y_i\leftarrow(t_i,1)$;
else if $a_i[1]=C$ AND $b_i[1]=0$ **then**
  Pick $\rho_i\xleftarrow{\$}\{0,1\}^{64}$;
  Set $t_i\leftarrow H\big(\mathsf{GenerateKey}(\mathsf{sk}_i,\rho_i),r\big)$;
  Set $y_i\leftarrow(t_i,0)$;
end if
return $\mathbf{y}:=\langle y_i\rangle_{i:x_i\neq\perp}$.

Figure 4: The Dialing program $\mathtt{DLN}_{\mathsf{sort}}$ realizing the Dialing program $\mathtt{DLN}_{\mathsf{abs}}$ for dialing round r, and users $u_1,\ldots,u_n$ with usernames $\mathsf{UN}_1,\ldots,\mathsf{UN}_n\in\{0,1\}^{64}$. The value C denotes a dial check request.

7. Computing the dead drops: After the Quicksort algorithm is completed, the active users $u_1,\ldots,u_{\mathsf{act}}$ are delivered the values $\hat{b}_1[1],\ldots,\hat{b}_1[\mathsf{act}]$ respectively. Having received $\hat{b}_i[1]$, dialer u_i that knows UN_j, and dial checker u_j that obtained UN_i, can calculate their shared dead drop value for dialing round r as follows:

$$t_i:=H\big(K_{i,j},r\big)\ ,\quad \text{if } \hat{b}_i[1]=0$$
$$t_j:=H\big(K_{j,i},r\big)\ ,\quad \text{if } \hat{b}_i[1]=\mathsf{UN}_j$$

Above, H is a standard cryptographic hash function, r is the round number. The values $K_{i,j},K_{j,i}$ are the ID-KA keys that u_i and u_j compute by running the key agreement algorithm $\mathsf{GenerateKey}$ on input $(\mathsf{sk}_i,\mathsf{UN}_j)$ and $(\mathsf{sk}_j,\mathsf{UN}_i)$ respectively (cf. Section 2), where $\mathsf{sk}_i,\mathsf{sk}_j$ are the secret keys of u_i and u_j. Recall that the operations for ID-KA key generation are over a finite multiplicative group of prime order q. We stress that the dead drop value is at least 64 bits long to make accidental collisions unlikely, although our system can tolerate them. By the correctness of the ID-KA protocol, it holds that $K_{i,j}=K_{j,i}$, hence we have that $t_i=t_j$.

On the other hand, if user u_i dial checked but $\hat{b}_i[1]=0$ (no one dialed u_i), then for uniformity reasons, she computes a random dead drop as above by inserting a random value ρ_i in place of UN_j, i.e. she sets $t_i:=H\big(\mathsf{GenerateKey}(\mathsf{sk}_i,\rho_i),r\big)$.

Note that if u_i has dialchecked, then either (i) she established a rendezvous point with u_j, if $\hat{b}_1=\mathsf{UN}_j$, or (ii) no one dialed her, if $\hat{b}_1=0$. Thus, she can set a "success" bit c_i to 1 or 0 respectively, indicating her successful engagement in the dialing round r. Besides, if u_i is a dialer that dialed u_j, then she always computes the value $t_i:=H\big(\mathsf{GenerateKey}(\mathsf{sk}_i,\mathsf{UN}_j),r\big)$, regardless of the success of her dialing request. Hence, she can not infer a success bit.

The Dialing program $\mathtt{DLN}_{\mathsf{sort}}$**.** The program $\mathtt{DLN}_{\mathsf{sort}}$ implementing the Dialing protocol is presented in Fig. 4.

Following Section 3, we show that $\mathtt{DLN}_{\mathsf{sort}}$ realizes the member of the Dialing program family $\mathtt{DLN}_{\mathsf{abs}}$ that corresponds to our sorting process. Namely, in Step 4 of $\mathtt{DLN}_{\mathsf{sort}}$ (Sorting by usernames), the inputs are arranged according to an ordering of their second coordinate. Thus, we set the index z that parameterizes the family $\mathtt{DLN}_{\mathsf{abs}}$ to be the string z_{qs2} as follows: z_{qs2} is parsed as the deterministic program $R_{\mathtt{DLN}}^{z_{\mathsf{qs2}}}$ that takes as takes as input an index i and array of triples $\mathbf{x}$ in encoded form, and outputs the index j so that when the array is sorted according to Quicksort ordering on the second coordinate, x_i is the left neighbor of the encoded x_j. Formally, we state the following theorem and provide the proof in the full version of the paper.

Theorem 1. *Let n be the number of users, $\kappa \geq 64$ be the dead drop string length and q be the prime order of the underlying ID-KA group. Let H be the cryptographic hash function modeled as a random oracle. Then, the Dialing program* $\mathtt{DLN}_{\mathsf{sort}}$ *described in Fig. 4 implements the member of the Dialing program family* $\mathtt{DLN}_{\mathsf{abs}}$ *described in Fig. 2 for parameter* z_{qs2} *with correctness error* $\frac{n^4}{q} + \frac{n}{2^\kappa}$.

Remark 2. The correctness error $\frac{n^4}{q} + \frac{n}{2^\kappa}$ is typically a negligible value in our setting. To provide intuition, consider the case with a number of $n = 100000 < 2^{17}$ users, dead drop size $\kappa = 64$ bits and group size $q \geq 2^{128}$. The error for this case is less than $\frac{2^{17 \cdot 4}}{2^{128}} + \frac{2^{17}}{2^{64}} \approx 2^{-47}$.

6 The Conversation Protocol

The Conversation protocol facilitates the actual exchange of messages associated with the same t dead drop value, which represents a rendezvous point computed in the final step of a Dialing protocol execution. It is expected that no more than two messages will have the same t value due to its large bit-size, although our system can handle collisions as we will see later. As in the previous section, we first provide a description of the Conversation protocol and then the corresponding program labeled $\mathtt{CNV}_{\mathsf{sort}}$ that implements it. At this point, we have to highlight our assumption that a valid message m_i at the input has its least significant bit (LSB) equal to 0. This flag which could also be a discrete fourth member of our tuple, is useful at (i) conflict resolution when more than two dead drops are identical and (ii) the parallelization of our protocol discussed in section 11 and in the full version of the paper.

Protocol description. The protocol is executed via the following steps, where steps 2-6 are executed by the MPC servers. Step 1 is executed locally by each user.

1. **Encoding:** The inputs are of the form of (CONV, t_i, m_i) requests, or $\perp$. Again, we assume that the users are enumerated as $u_1, \ldots, u_n$ consistently with the order they submitted their input sequence $x_1, \ldots, x_n$, hence all $\perp$ values are stacked last. Active users' inputs are encoded as triples of the form $a_i := (a_i[1], a_i[2], a_i[3])$ where the third component is an input wire ID wid_i that will be uniquely assigned in the following step. In particular, if u_i wants to engage in conversation, then the (CONV, t_i, m_i) request is encoded as $(t_i, m_i, 0)$. In case u_i is not engaging in conversation the request will use a random dead drop value and a random message.

2. **Assigning wire ID values:** As a first step, the MPC protocol assigns unique wire IDs for each user. This is done by setting the third component $a_i[3]$ of the encoded triple a_i to i. Thus, for each u_i, we have that $\mathsf{wid}_i := i$.

3. **Sorting by dead drops:** The encoded input triples are first sorted according to their first components using the oblivious Quicksort algorithm of [33]. As a result, the inputs of any two users that share the same dead drop value will become adjacent.

4. **Exchanging adjacent messages:** By construction, two inputs with the same dead drop value indicate a pair of users u_i and u_j that wish to communicate. Thus, the protocol generates a vector $\langle b_1, \ldots, b_n \rangle$, where each b_i is a pair $(b_i[1], b_i[2])$, of which the second component is wid_i and the first component is either (i) the message of some adjacent encoded input, or (ii) the original message m_i, if message exchange did not take place for u_i because there was no matching dead drop or due to conflict (three or more equal dead drops). As already mentioned, the LSB of two exchanged messages is set to 1. In the special conflict case where three or more values share the same dead drop t, an arrangement would be as follows:

$$\ldots, (t', m_k, k), (t, m_j, j), (t, m_i, i), (t, m_{i'}, i'), \ldots$$

In this case, the messages of u_i and u_j will be exchanged and $u_{i'}$ will obtain back his message at the end of the protocol, notifying him to resubmit.

5. **Sorting by wire IDs:** As in the Dialing protocol (Step 5), the Conversation protocol performs a Quicksort on the processed requests according to their mutually distinct wire IDs in order for the correct requests to be forwarded to each user. The result is a vector $\langle \hat{b}_1, \ldots, \hat{b}_n \rangle$ where $\hat{b}_i$ is a pair $(\hat{b}_i[1], \hat{b}_i[2])$ that corresponds to u_i and is either (i) a message m_j from some user u_j or (ii) the original message m_i, in both cases indexed by wid_i.

6. **Forwarding messages:** At the end, the protocol discards the wire IDs and creates the output vector $\mathbf{y} = \langle y_1, \ldots, y_n \rangle := \langle \hat{b}_1[1], \ldots, \hat{b}_n[1] \rangle$. Thus, each y_i is either (i) a message m_j from some user u_j or (ii) the self-generated message m_i. Finally, the users $u_1, \ldots, u_n$ are delivered the values $y_1, \ldots, y_n$.

Remark 3. In reality, the dead drop value t_i of some user u_i is not exactly the value she received from a dialing protocol execution. For conversation round r it is computed as $t_i := H(t_{(dialing)i}, r)$, where $t_{(dialing)i}$ is the dead drop for u_i, generated by the dialing protocol and acts as the seed for the creation of an ephemeral dead drop for each conversation round.

Remark 4. Due to the size of dead drops values, the probability that a collision on randomly generated dead drop values will occur can be made very small. Even in the case of a collision, the client of the user that was affected would just resend that message in the next round, as it would know that a collision occurred because it received a message it could not decrypt.

The Conversation program $\mathtt{CNV}_{\mathsf{sort}}$ **.** The program

$\mathtt{CNV}_{\mathsf{sort}}$ implementing the Conversation protocol is presented in Fig. 5.

The Conversation Program $\mathtt{CNV}_{\mathsf{sort}}$

Input: a sequence $\langle x_1,\ldots,x_n\rangle$ where x_i is either a (CONV,t_i,m_i) request, or $\perp$. All $\perp$ inputs are stacked last.

Output: a sequence of messages $\langle y_i\rangle_{x_i\neq\perp}$.

---

1. For each $i\leftarrow 1,\ldots,n$
if $x_i=\perp$ **then**
  Set $\mathsf{act}:=i-1$;
  Break loop ;
end if
if $x_i=(\text{CONV},t_i,m_i)$ **then**
  Set $a:=(a_i[1],a_i[2],a_i[3])\leftarrow(t_i,m_i,0)$;
end if

2. For each $i\leftarrow 1,\ldots,\mathsf{act}$
Set wid_i as $a_i[3]\leftarrow i$;

3. Sort tuples $\langle a_i\rangle_{x_i\neq\perp}$ according to first coordinate $a_i[1]$ using Quicksort;

4. For each $i\leftarrow 1,\ldots,\mathsf{act}-1$
if $a_i[1]=a_{i+1}[1]$ AND $LSB(a_i[2])=LSB(a_{i+1}[2])=0$ **then**
  Set the LSB of $a_i[2]$ and $a_{i+1}[2]$ to 1 ;
  Set $b_i\leftarrow(a_{i+1}[2],a_i[3])$;
  Set $b_{i+1}\leftarrow(a_i[2],a_{i+1}[3])$;
end if

5. Sort tuples $\langle b_i\rangle_{i:x_i\neq\perp}$ according to second coordinate (which is the wire id) using Quicksort;

6. For each $i\leftarrow 1,\ldots,\mathsf{act}$
Set $y_i\leftarrow b_i[1]$;

return $\mathbf{y}:=\langle y_i\rangle_{i:x_i\neq\perp}$.

Figure 5: The Conversation program $\mathtt{CNV}_{\mathsf{sort}}$ realizing the Conversation program $\mathtt{CNV}_{\mathsf{abs}}$ for conversation round r, dead drop size $\kappa\geq 64$ and users $u_1,\ldots,u_n$ with messages taken from space $\mathcal{M}$.

Following Section 3, we show that $\mathtt{CNV}_{\mathsf{sort}}$ realizes the member of the Conversation program family $\mathtt{CNV}_{\mathsf{abs}}$ that corresponds to our sorting process. Namely, in Step 3 of $\mathtt{CNV}_{\mathsf{sort}}$ (Sorting by dead drops), the inputs are arranged according to an ordering of their first coordinate. Thus, we set the index z that parameterizes the family $\mathtt{CNV}_{\mathsf{abs}}$ to be the string z_{qs1} as follows: z_{qs1} is parsed as the deterministic program $R^{z_{\mathsf{qs1}}}_{\mathtt{CNV}}$ that takes as input an index i and array of triples $\mathbf{x}$ in encoded form, and outputs the index j so that when the array is sorted according to Quicksort ordering on the first coordinate, the encoded triple of u_i (or resp. u_j) has no neighbors on the left of the sorted array and the encoded triple of u_j (or resp. u_i) is the right neighbor of the encoded triple of u_i (or resp. u_j). Formally, we state the following theorem and provide the proof in the full version of the paper.

Theorem 2. *Let n be the number of users and $\kappa\geq 64$ be the dead drop string length. The Conversation program $\mathtt{CNV}_{\mathsf{sort}}$ described in Fig. 5 implements the member of the Conversation program family $\mathtt{CNV}_{\mathsf{abs}}$ described in Fig. 3 for parameter z_{qs1}.*

7 The MCMix Anonymous Messaging System

Having presented the general architecture of our system in Section 4 and the Dialing and Conversation protocols and programs in Sections 5 and 6 respectively, we now show how these programs are implemented in our architecture. Our system consists of two MPC instances of the general architecture in Section 4, executing one after the other or independently in parallel. One implements the Dialing protocol and the other the Conversation protocol. Below, we specify the operations of general architecture for each of our two protocols. We note with the prime symbol, e.g. **1'.**, the specification of the respective step, e.g. **1.**, of the general architecture.

Dialing. The execution of the Dialing protocol for round r follows the steps of section 4 with the following particularities:

1'. Encoding: The input of user u_i is encoded as $a_i=(\mathsf{UN}_i,\mathsf{UN}_j,0)$, in the case of a dial to user u_j, or as $a_i=(C,\mathsf{UN}_i,0)$ in the case of a dial request, as specified by Step 1 of the Dialing program $\mathtt{DLN}_{\mathsf{sort}}$ in Fig. 4.

6'. MPC algorithm: The MPC server secure computation consists of Steps 2-6 of $\mathtt{DLN}_{\mathsf{sort}}$.

8'. Decryption and reconstruction: The reconstructed value b_i received by user u_i is the output b_i of Step 6 of $\mathtt{DLN}_{\mathsf{sort}}$.

9'. Dead drop calculation: As an extra step, the dead drop value t_i is calculated by each user by performing Step 7 of $\mathtt{DLN}_{\mathsf{sort}}$.

Conversation. The execution of the conversation protocol for round r follows the steps of Section 4 with the following particularities:

1'. Encoding: Input is encoded as $a_i=(t_i,m_i,0)$, with t_i being a dead drop calculated by the final step of a previous dialing round in the case of a real conversation request (also taking into account Remark 3), or a random value in the case the user does not want to send a message (but still wants to protect her privacy), according to the Conversation program $\mathtt{CNV}_{\mathsf{sort}}$ in Fig. 5.

6'. MPC algorithm: The MPC server secure computation consists of Steps 2-6 of $\mathtt{CNV}_{\mathsf{sort}}$.

8'. Decryption and reconstruction: The reconstructed value b_i received by the user that provided input i is the output y_i of Step 6 of $\mathtt{CNV}_{\mathsf{sort}}$ and is the message intended for this user.

Security of MCMix. We prove our security theorem for the general θ-out-of-m case, as in Definition 1, using the parameters z_{qs2} and z_{qs1} defined in Sections 5 and 6 respectively. We provide the proof in the full version.

Theorem 3. *Let κ be the dead drop size, n be the number of users, m be the number of servers and q the size of the underlying Diffie-Hellman group, where n,m are polynomial in λ, $\kappa = \Theta(\lambda)$ and $q = \Omega(2^{\lambda})$. Let $\mathbb{P}$ be a (θ,m)-secure MPC protocol with n users w.r.t. (i) the Server Computation Steps 2-6 of the Dialing program $\mathtt{DLN}_{\mathsf{sort}}$ described in Fig. 2 and (ii) the Server Computation Steps 2-6 of the Conversation program $\mathtt{CNV}_{\mathsf{sort}}$ described in Fig. 3. Then,* MCMix implemented over $\mathbb{P}$ is an anonymous messaging system *by securely realizing the program families $\mathtt{DLN}_{\mathsf{abs}}$ and $\mathtt{CNV}_{\mathsf{abs}}$ for parameters z_{qs2} and z_{qs1} respectively.*

Remark 5 (**On forward security of MCMix**). MCMix in its current form does not offer forward security. Nevertheless it is possible to provide forward security as follows. First, clients could refresh their exchanged keys with the servers in regular time intervals, e.g., once a day. Alternatively to avoid interaction, forward secure encryption can be used, e.g., see [9]. With respect to the dead drop calculation we can obtain forward security by applying our second ID-KA construction with forward secrecy (cf. Section 2 and the full version). The additional communication cost to the Dialing protocol would be one extra random group element per user as now the active inputs $x_1,\ldots,x_n$ for dialing need to be used for the first round of the exchange; they are of the form of $(\mathrm{DIAL},u_i,u_j,r_i)$ and $(\mathrm{DIALCHECK},u_i,r_j)$, where r_i,r_j are random elements from the ID-KA cyclic group. Sorting would still be executed on the users' usernames and the wire IDs as before thus incurring no additional overhead. We omit further details.

8 Implementation and Benchmarking

We implemented a prototype of our system using the Sharemind platform and performed extensive evaluation. ***Experiment setting.*** Benchmarks were run on a cluster of three machines with point-to-point 1 Gbps network connections using various profiles for network latency aiming to simulate WAN behavior. Each machine has a 12-core 3 GHz Hyper-Threading CPU and 48 GB of RAM. However, even though the hardware supports it, Sharemind MPC protocols are not optimized to use multiple CPU cores or network layer in a parallel manner. The servers running Sharemind employ only 2 cores, one for executing the computations and another for pseudo-random number generation. To simulate real-world environment, we use the `tc` tool to manipulate operating system's network traffic control settings. This tool is used to both cap the available network bandwidth, as well as introduce communication latency by adding round-trip delay (ping).

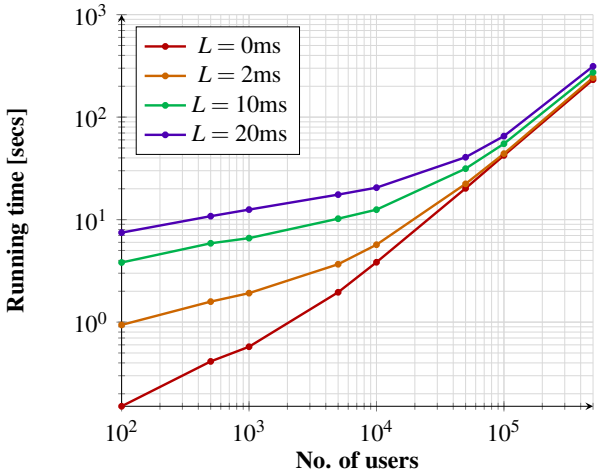

Figure 6: Running time in secs of the Dialing protocol implementation for a number of $n = 100$, 500, 1K, 5K, 10K, 50K, 100K, 500K users and latency $L = 0$, 2, 10, 20 ms. The benchmarks were run with message size 8 Bytes and 1 Gbps network bandwidth.

Dialing protocol. We benchmarked our dialing protocol for various numbers of users and various latency values. The results are presented in Fig. 6. As we can see, the dialing protocol has a runtime for each round of around one minute for 100,000 users and around 300 seconds for 500,000 users, considering the worst case of 20 ms of latency. The latter value might still be considered acceptable for some settings, as dialing rounds need not be executed very often. Another interesting observation is that the effect of latency diminishes as the number of users increases, due to the fact that the number of communication rounds of our algorithm scales logarithmically to the number of inputs. This in turn happens because Quicksort needs $\mathcal{O}(\log(n))$ steps to sort n inputs when executed in parallel. The vectorized nature of our implementation succeeds in taking advantage of the parallelizable nature of the algorithm. The time a user needs to encode her request and send it, as well as the time required by each MPC server to decrypt the requests it received have no effect on the per round runtime of our system. This is be-

cause these operations are performed in a pipelined fashion. This means that the encoding, encryption and decryption of the requests for round $r+1$ takes place while the MPC servers perform the computations for round r. In the dialing protocol this is acceptable as a user's intent on whether to dial or perform a dial check might not depend on the output of the previous dialing round.

Conversation protocol. For the conversation protocol we made extensive benchmarks considering the number of users, the latency of the network, as well as the message size. In Fig. 7, we can see that the running time of the conversation protocol with a very small message size of 8 Bytes (B) is similar to the running time of the dialing protocol. That is, the system can serve 100,000 users with in around one minute for maximum latency of 20 ms. Again, we see that latency is a minor performance factor for a large number of users. This fact enables us to claim that our system will have similar running times even with greater latency values.

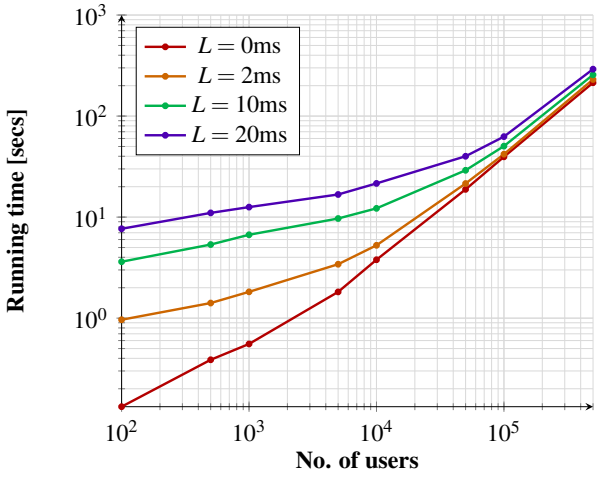

Figure 7: Running time in secs of the Conversation protocol implementation for a number of $n = 100$, 500, 1K, 5K, 10K, 50K, 100K, 500K users and latency $L = 0$, 2, 10, 20 ms. The benchmarks were run with message size 8 Bytes and 1 Gbps network bandwidth.

In Fig. 8, we consider how the message size affects performance. We have benchmarked various message sizes ranging from 8 B to 1 KB messages. No artificial latency has been injected for these experiments. We see that message size affects performance in a significant way as opposed to latency, but the system can still support anonymity sets of tens of thousands of users even with 1KB messages and certainly SMS long messages for hundreds of thousands.

Finally, in Fig. 9, we provide the peak network bandwidth consumption during the Dialing and Conversation protocols. We note that the total bandwidth is shown, i.e. bytes sent and received and to both other computing nodes. We observe that in both protocols the bandwidth consumption remains at a low level of less than 100Mbps for the Dialing protocol for (usernames of 64bits) as well as the Conversation protocol for messages of up to SMS size. For bigger message sizes and 100,000 users, we get that the total consumption is roughly 150Mbps and 300Mbps for messages of 256B and 1KB respectively, which can be realistic for a large scale setting.

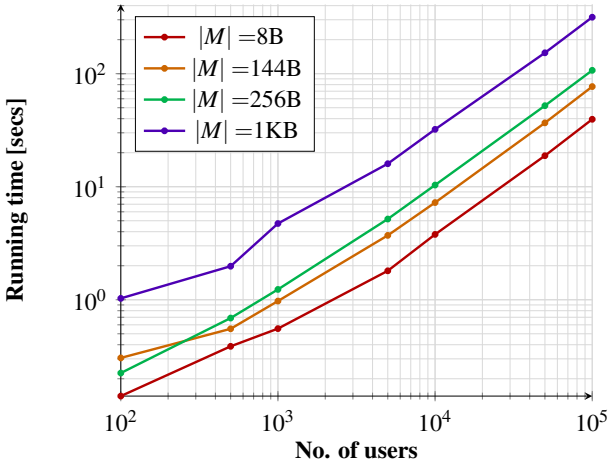

Figure 8: Running time in secs of the Conversation protocol implementation for a number of $n = 100$, 500, 1K, 5K, 10K, 50K, 100K users and message size $|M| = 8$, 144, 256, 1K Bytes. The benchmarks were run with no latency and 1 Gbps network bandwidth.

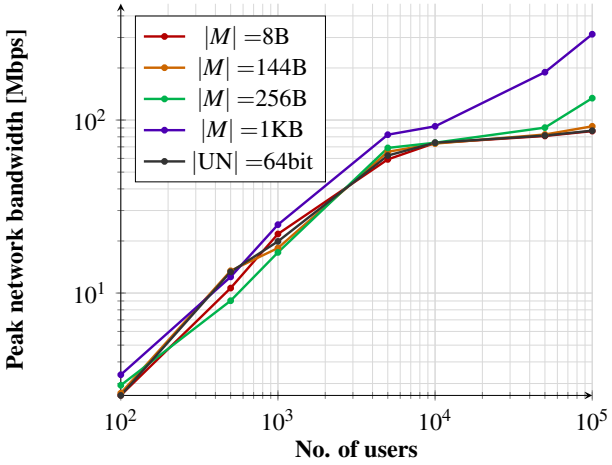

Figure 9: The peak network bandwidth consumption in Mbps during the Dialing protocol for usernames (UNs) of 64bits and the Conversation protocol for message size $|M| = 8$B, 144B, 256B, 1KB, given a number of $n = 100$, 500, 1K, 5K, 10K, 50K, 100K users. The benchmarks were run with no latency and 1 Gbps network bandwidth.

9 Client Load and Adoption Incentives

Anonymous communication systems critically rely on having adequately large anonymity sets to be effective. In other words: "Anonymity Loves Company" [26], and the usability aspects of anonymous communication systems should be an important design consideration. MCMix strives to offer strong adoption incentives by offering strong security, while minimizing the computation and communication load on the client side.

Computation load: For the Dialing protocol, each client performs an ID-KA operation (cf. Section 2) to compute the dead drop value, plus a few symmetric operations to encrypt/decrypt the shares. The Key Exchange operation consists of a few hashes and a single bilinear symmetric pairing computation. In [2], symmetric pairing time is estimated at 14.9 ms running on a commodity device, or around three times the time needed for a modular exponentation in the corresponding cyclic group. For the Conversation protocol, the load is low, consisting only of symmetric encryption/decryption operations.

Communication load: In Table 1, we depict the total monthly bandwidth costs of the clients in an example setting with (i) SMS message size of 140 B, (ii) fixed block size for AES of 128 bits, (iii) standard 20/20 B TCP/IP headers, (iv) SHA-256 HMACs and (v) dialing and conversation rounds assumed to be executed every one minute (simultaneously). For a detailed discussion on the communication load of our system, we refer to the full version.

$\|M\|$ (B)	bandwidth per month (MB)
8	47
144	78
256	106
1K	296

Table 1: Communication costs of clients (Dialing and Conversation combined) w.r.t. message size.

The theoretical analysis of the computational and communication overhead of our system shows, that it is lightweight on the client side and the bandwidth needs of a device to be constantly connected are in the range of tens of MB per month, which we consider easily manageable. While we expect MCMix to be practical for mobile users, further experiments may be needed to compute actual battery consumption and bandwidth usage in a real-world setting.

10 Related Work and Comparison

This section attempts to place our work in relation to the state of the art in the expanding field of anonymity-preserving communication systems.

First, regarding Onion-routing based approaches, like POND [37] which uses the Tor network [27], we emphasize that they do not fit the model of a global adversary who can easily defeat them, see e.g., [34]. Systems that attempt to defeat global adversaries operate in rounds and expect each online user to send encrypted messages in each round. Furthermore, our interpretation of anonymous messaging is one of unobservable bilateral communication. Therefore, unilateral shuffling mechanisms based on mixnets or recent MPC constructions [41] do not satisfy our application scenario.

Our work is most closely related to the Vuvuzela system [49] that uses mixnets in addition to dummy messages, to add noise and achieve a differentially private (cf. [28]) solution to anonymous messaging. By definition, differential privacy protects users as individuals and also allows for some (albeit small) leakage to an observer and thus it is weaker than the simulation-based privacy that we achieve. For example, when all users talk to each other compared to when no user is talking to anyone is completely distinguishable in Vuvuzela, but indistinguishable for MCMix that does not leak any metadata at all. Furthermore, Vuvuzela puts a burden on the client side that requires to finish the dial protocol by downloading a substantial amount of user data (or losing substantially in terms privacy); note that using Bloom filters as described in [39] can help in making this a one time cost. Another drawback of this system is that it cannot scale down in a tight way, due to the burden imposed by the added noise that needs to be always added to maintain acceptable privacy guarantees. On the up side, the system has good architecture and is extremely scalable to millions of users under the assumption of a single honest server, whereas (non-parallelized) MCMix can scale to 100,000 users with similar latency and assuming an honest server majority. However, our parallelized MPC approach can reach that level of performance and in any case, we anticipate that further advances in secure MPC protocols can improve performance substantially even in the non-parallelized version.

Riffle [36], uses hybrid mixnets and private information retrieval (PIR, [19]) techniques to implement anonymous messaging. It offers good privacy guarantees, but unlike MCMix and Vuvuzela, it can not handle network churn. During the setup phase of the protocol, client keys are verifiably shuffled by a mixnet. During each communication phase, the same permutations as the ones established in the setup phase are applied to the clients' authenticated messages by the mix servers. As a result of this setup, a single client momentarily leaving or entering the system would require to re-run the expensive setup phase of the protocol.

cMix [15] introduces a mixnet design that can shuf-

fle messages faster than previous work by avoiding public key operations in the real-time phase. cMix provides sender anonymity, yet it may leak the number of messages received by each user, exhibiting a similar security performance as Vuvuzela's dialing protocol.

Dissent [22, 51] is based on DC-nets and achieves anonymity sets up to a few thousand users, in an anonymous broadcasting scenario. Riposte [21] uses PIR techniques to implement a distributed database that users can anonymously write and read from, assuming no two servers collude (in the efficient scheme). Specifically, the authors implement the write stage on the database as a "reverse" PIR, where a client spreads suitable information for writing in the database. Subsequently, when used for messaging, users can read using PIR from the position in the database that the sender wrote the message (which can be a random position calculated from key information available to the users). Riposte can scale to millions of users but it requires many hours to perform a complete operation; a significant bottleneck is the write-operation that requires $O(\sqrt{L})$ client communication for an L-long database which is proportional to the number of users. In contrast, in our system, client bandwidth is minimal, i.e. a single message per server is sent by each user. Additionally, the application scenario is more related to that of Dissent, rather than ours, i.e. anonymous broadcasting, instead of private point to point message exchange, as the authors specify that their approach is suitable "for latency-tolerant workloads with many more readers than writers". Finally, our technical approach is very different compared to Riposte, as Riposte uses MPC techniques only to detect and exclude malformed client requests, while MCMix offers a native MPC solution for the complete messaging functionality.

BAR [35] uses a "broadcast to all" approach to achieve perfect privacy. A central untrusted server receives all messages in each round and then broadcasts them to all participants. This approach induces a very large communication overhead and therefore anonymity sets are limited to hundreds of users. Pung [3] is a system that like BAR operates on fully untrusted setting, while it uses state-of-the-art PIR techniques and smart database organization to scale to a much larger number of users. However, Pung can only implement the equivalent of our conversation functionality and not the dialing functionality, and exhibits substantial client load.

11 Parallelizing the conversation protocol

As discussed in previous sections, our protocols are provably secure assuming a secure MPC framework and are also scalable enough to support hundreds of thousands of users. While these anonymity sets can accommodate a lot of use cases, we recognize the need for anonymity systems to offer as large an anonymity set as possible. Therefore, we propose a technique that leads to an even more scalable system, by describing a parallel realization of the Conversation protocol, as this is the latency-critical component of our system. Note that the Dialing protocol can be executed independently of the Conversation protocol and in much longer time intervals, e.g. every five minutes. Therefore, the implementation on a single MPC instance can cover very large anonymity sets, e.g. 500,000 users as seen in Fig. 6.

In the following paragraph, we provide the general idea behind our parallelization technique and refer the reader to the full version for a detailed description of the parallelized Conversation protocol.

General Idea. Our main challenge is to come up with a protocol that can run in different MPC instances (islands) in parallel with minimal communication between those instances, while achieving strong privacy. Additionally, the anonymity set should be the whole user population. The problem of anonymous communication, where two users may submit their messages to different islands and still expect to communicate with perfect correctness, while leaking no information at all, is hard to be parallelized. In our approach, we choose to maintain the strongest possible privacy standards. As a result, in our parallelized version of MCMix, we relax our quality of service (qos) guarantees. That is, in each round, an adjustable small number of requests that would have been served when using the algorithm of Fig. 5, will fail to do so, and affected users will have to resend their messages. The probability of this phenomenon can be made arbitrarily small in the expense of performance, which is shown in the full version.

As evident by the algorithmic representation of our two protocols, the integral part of their function is matching equal values in pairs and performing a swap action on these pairs. Our parallelizable technique for performing this action benefits from the fact that the values in question (dead drops) output by a hash function (modeled as a random oracle) are uniformly distributed.

In our approach, requests are split obliviously between MPC islands based on the fact that equal dead drop values are likely to be located at roughly the same indexes of different arrays after sorting, considering these values are uniformly distributed. In summary, and in the simple case of 2 islands, the procedure is as follows. As a first step, requests in each island are sorted according to their dead drop values. Then, one island collects the lower half of both islands' sorted requests, and the second island the upper half. A swap operation, identical to the one of the initial conversation protocol, is performed as a next step, followed by a sort according to the wire IDs of the requests. Assuming the first island assigns strictly smaller wire ID values to the incoming requests,

exactly the bottom /upper half of the requests held by each island belongs to the first/second one. These halves are sent to their respective islands. Finally, each island merges the array of requests it received, with the one it kept, according to their wire IDs. The final order of requests corresponds to the order in which they were initially received, and the requests with the same dead drop that found themselves on the same island during the swap phase, represent successful instances of the conversation protocol.

Performance of the parallelized Conversation protocol. Considering the fact that we did not have access to a great number of physical machines, in order to run the parallelized Conversation protocol with a variety of island numbers, we ran the parallel algorithm on a single island for different user numbers and then extrapolated to give predictions for a real multi-island implementation. Except from the running time of the MPC that we measured, we also added the communication time calculated by assuming commodity 100 Mbps connections between the islands. In the parallelised setting, in both inter-island communication rounds, each party sends and receives in total $n/m \cdot (m-1)/m$ elements to/from other parties, where n is the number of messages and m is the number of islands. In our benchmarks, we have not added any overhead for symmetric encryption between the islands, as even a commodity laptop can keep up with encrypting and decrypting data at a rate of 100 Mbps. Thus, we expect that the results presented in Fig. 10 realistically highlight the scalability of the system. From the results

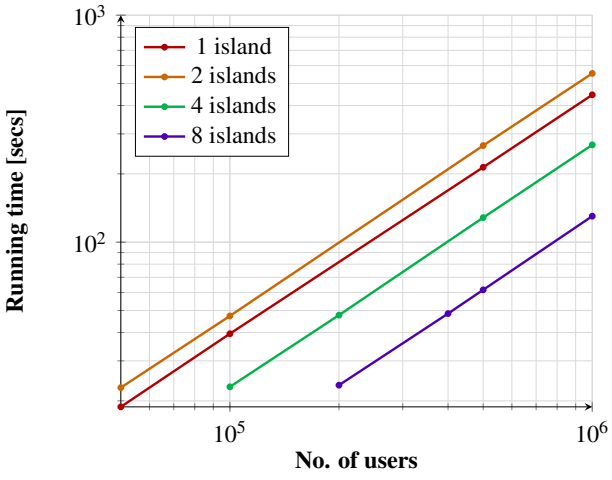

Figure 10: Running time in secs of the Conversation protocol implemented in 1,2,4,8 island setting. The benchmarks were run with no latency, 1 Gbps network bandwidth (intra-island) and 64 bit message size. Bandwidth between the islands was modeled at 100 Mbps.

of Fig. 10, we can see deploying our system over 2 islands does not provide any performance gain. This is due to a constant overhead, roughly of a factor of 2, that follows from the description of the parallelized algorithm (cf. full version for details). However, when using 4 or more islands, our parallelization technique gets very rewarding. In the case of 8 islands, the system can support an anonymity set of 500,000 users with a latency of 60 seconds. We expect this trend to continue for even more than 8 islands, thus enabling even larger anonymity sets.

Acknowledgements

Alexopoulos, Kiayias and Zacharias were supported by the Horizon 2020 PANORAMIX project (Grant Agreement No. 653497). Alexopoulos was also supported by the DFG as part of project S1 within the CRC 1119 CROSSING. Talviste was supported by the Estonian Research Council (Grant No. IUT27-1). The authors would like to thank Tim Grube and Chris Campbell for their comments on a previous version of this paper.

References

[1] Ajtai, M., Komlós, J., and Szemerédi, E. An O(n log n) sorting network. In *ACM STOC* (1983), pp. 1–9.

[2] Akinyele, J. A., Garman, C., and Hohenberger, S. Automating fast and secure translations from type-I to type-III pairing schemes. In *ACM CCS* (2015), pp. 1370–1381.

[3] Angel, S., and Setty, S. Unobservable communication over fully untrusted infrastructure. In *OSDI* (2016), pp. 551–569.

[4] Araki, T., Barak, A., Furukawa, J., Lichter, T., Lindell, Y., Nof, A., Ohara, K., Watzman, A., and Weinstein, O. Optimized Honest-Majority MPC for Malicious Adversaries – Breaking the 1 Billion-Gate Per Second Barrier. In *IEEE Symposium on Security and Privacy* (2017), pp. 843–862.

[5] Araki, T., Furukawa, J., Lindell, Y., Nof, A., and Ohara, K. High-throughput semi-honest secure three-party computation with an honest majority. In *ACM CCS* (2016), pp. 805–817.

[6] Batcher, K. E. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference* (1968), ACM, pp. 307–314.

[7] Beaver, D. Commodity-based cryptography. In *ACM STOC* (1997), pp. 446–455.

[8] Beimel, A., Gabizon, A., Ishai, Y., and Kushilevitz, E. Distribution design. In *ITCS* (2016), pp. 81–92.

[9] Bellare, M., and Yee, B. S. Forward-security in private-key cryptography. In *CT-RSA* (2003), pp. 1–18.

[10] Ben-David, A., Nisan, N., and Pinkas, B. Fairplaymp: a system for secure multi-party computation. In *ACM CCS* (2008), pp. 257–266.

[11] Bogdanov, D. *Sharemind: programmable secure computations with practical applications*. PhD thesis, University of Tartu, 2013.

[12] Bogdanov, D., Laud, P., and Randmets, J. Domain-polymorphic programming of privacy-preserving applications. In *PLAS* (2014), pp. 53–65.

[13] BOGDANOV, D., LAUR, S., AND TALVISTE, R. A Practical Analysis of Oblivious Sorting Algorithms for Secure Multi-party Computation. In *Proceedings of the 19th Nordic Conference on Secure IT Systems, NordSec 2014*, vol. 8788 of *LNCS*. Springer, 2014, pp. 59–74.

[14] CHAUM, D. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology 1*, 1 (1988), 65–75.

[15] CHAUM, D., JAVANI, F., KATE, A., KRASNOVA, A., DE RUITER, J., AND SHERMAN, A. T. cMix: Anonymization by high-performance scalable mixing. *IACR Cryptology ePrint Archive* (2016).

[16] CHAUM, D. L. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM 24*, 2 (1981), 84–90.

[17] CHEN, L., CHENG, Z., AND SMART, N. P. Identity-based key agreement protocols from pairings. *Int. J. Inf. Sec. 6*, 4 (2007), 213–241.

[18] CHEN, L., AND KUDLA, C. Identity based authenticated key agreement protocols from pairings. In *CSFW-16* (2003), pp. 219–233.

[19] CHOR, B., KUSHILEVITZ, E., GOLDREICH, O., AND SUDAN, M. Private information retrieval. *Journal of the ACM (JACM) 45*, 6 (1998), 965–981.

[20] CLARKE, I., SANDBERG, O., WILEY, B., AND HONG, T. W. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies* (2001), Springer, pp. 46–66.

[21] CORRIGAN-GIBBS, H., BONEH, D., AND MAZIÈRES, D. Riposte: An anonymous messaging system handling millions of users. In *IEEE Symposium on Security and Privacy* (2015), pp. 321–338.

[22] CORRIGAN-GIBBS, H., AND FORD, B. Dissent: accountable anonymous group messaging. In *ACM CCS* (2010), pp. 340–350.

[23] DAMGÅRD, I., GEISLER, M., KRØIGAARD, M., AND NIELSEN, J. B. Asynchronous multiparty computation: Theory and implementation. In *PKC* (2009), pp. 160–179.

[24] DAMGÅRD, I., PASTRO, V., SMART, N., AND ZAKARIAS, S. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO* (2012), pp. 643–662.

[25] DANEZIS, G., DINGLEDINE, R., AND MATHEWSON, N. Mixminion: Design of a type III anonymous remailer protocol. In *IEEE Symposium on Security and Privacy* (2003), pp. 2–15.

[26] DINGLEDINE, R., AND MATHEWSON, N. Anonymity loves company: Usability and the network effect. In *WEIS* (2006).

[27] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. Tech. rep., DTIC Document, 2004.

[28] DWORK, C. Differential privacy. In *Automata, languages and programming*. Springer, 2006, pp. 1–12.

[29] FIORE, D., AND GENNARO, R. Identity-based key exchange protocols without pairings. *Trans. Computational Science 10* (2010), 42–77.

[30] FURUKAWA, J., LINDELL, Y., NOF, A., AND WEINSTEIN, O. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *EUROCRYPT* (2017), pp. 225–255.

[31] GOLDREICH, O., MICALI, S., AND WIGDERSON, A. How to play any mental game. In *ACM STOC* (1987), pp. 218–229.

[32] GÜNTHER, C. G. An identity-based key-exchange protocol. In *EUROCRYPT* (1989), pp. 29–37.

[33] HAMADA, K., KIKUCHI, R., IKARASHI, D., CHIDA, K., AND TAKAHASHI, K. Practically efficient multi-party sorting protocols from comparison sort algorithms. In *Information Security and Cryptology–ICISC 2012*. Springer, 2012, pp. 202–216.

[34] JOHNSON, A., WACEK, C., JANSEN, R., SHERR, M., AND SYVERSON, P. Users get routed: Traffic correlation on tor by realistic adversaries. In *ACM CCS* (2013), pp. 337–348.

[35] KOTZANIKOLAOU, P., CHATZISOFRONIOU, G., AND BURMESTER, M. Broadcast anonymous routing (BAR): scalable real-time anonymous communication. *Int. J. Inf. Sec. 16*, 3 (2017), 313–326.

[36] KWON, A., LAZAR, D., DEVADAS, S., AND FORD, B. Riffle: An efficient communication system with strong anonymity. *PoPETS 2016*, 2 (2015), 115–134.

[37] LANGLEY, A. Pond (v0.1.1). `https://github.com/agl/pond`, 2015.

[38] LAUD, P., AND PETTAI, M. Secure multiparty sorting protocols with covert privacy. In *NordSec* (2016), pp. 216–231.

[39] LAZAR, D., AND ZELDOVICH, N. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *OSDI* (2016), pp. 571–586.

[40] LIU, C., WANG, X. S., NAYAK, K., HUANG, Y., AND SHI, E. Oblivm: A programming framework for secure computation. In *IEEE Symposium on Security and Privacy* (2015), pp. 359–376.

[41] MOVAHEDI, M., SAIA, J., AND ZAMANI, M. Shuffle to baffle: Towards scalable protocols for secure multi-party shuffling. In *ICDCS* (2015), pp. 800–801.

[42] PATERSON, K. G., AND SRINIVASAN, S. On the relations between non-interactive key distribution, identity-based encryption and trapdoor discrete log groups. *Des. Codes Cryptography 52*, 2 (2009), 219–241.

[43] PETTAI, M., AND LAUD, P. Automatic proofs of privacy of secure multi-party computation protocols against active adversaries. In *CSF* (2015), pp. 75–89.

[44] SAKAI, R., KASAHARA, M., AND OGHISHI, K. Cryptosystems based on pairing. SCIS, Okinawa, Japan, 2000.

[45] SHAMIR, A. Identity-based cryptosystems and signature schemes. In *CRYPTO* (1984), pp. 47–53.

[46] SHELL, D. L. A high-speed sorting procedure. *Communications of the ACM 2*, 7 (1959), 30–32.

[47] SMART, N. P. An identity based authenticated key agreement protocol based on the weil pairing. *IACR Cryptology ePrint Archive* (2001).

[48] SYVERSON, P. F., GOLDSCHLAG, D. M., AND REED, M. G. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy* (1997), pp. 44–54.

[49] VAN DEN HOOFF, J., LAZAR, D., ZAHARIA, M., AND ZELDOVICH, N. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *SOSP* (2015), pp. 137–152.

[50] WANG, Y. Efficient identity-based and authenticated key agreement protocol. *Trans. Computational Science 17* (2013), 172–197.

[51] WOLINSKY, D. I., CORRIGAN-GIBBS, H., FORD, B., AND JOHNSON, A. Dissent in numbers: Making strong anonymity scale. In *OSDI* (2012), pp. 179–182.

[52] YUAN, Q., AND LI, S. A new efficient id-based authenticated key agreement protocol. *IACR Cryptology ePrint Archive* (2005).

[53] ZHANG, Y., STEELE, A., AND BLANTON, M. Picco: a general-purpose compiler for private distributed computation. In *ACM CCS* (2013), pp. 813–826.