RTMessenger: A differentially-private, secure file exchange service

Alex Jones

June 21, 2016

Abstract This paper describes an extension to the (ϵ, δ)-differentially private message-service, Vuvuzela, which improves the scalability and functionality of anonymous user-use data exchange. Through a rephrasing of the trust model and a focus on server network topology, RTMessenger secures user communication patterns againt powerful adversaries who can control a large portion of the server network and all of the network traffic.

1 Introduction

A goal of some cryptographers in the past few years has been the development of secure and private messaging services [1],[2],[3]. Development of these services would guarantee differential privacy and security, essentially protecting user $metadata$ with mathematical precision. RTMessenger aims to generalize and improve one proposed system, Vuvuzela, introduced in [1]. RTMessenger operates under the assumption that the adversary is powerful enough to observe and manipulate all network traffic and compile statistics . Under these assumptions, strong preventative measures built into the protocol keep prying eyes away from users' data and communication patterns. Like Vuvuzela, RTMessenger is intended for use by millions of users which intend to share thousands of messages.

2 Overview

This section will sketch the protocol in rough strokes. Then next develops how privacy is achieved, followed by psuedo-code for two algorithms and a proof of privacy.

The backbone of this service is a global pool of N $servers$, S, which act as internet-connected intermediaries between $clients$, which represent users. Clients instruct servers to exchange data via anonymous $mailbox$ exchanges, similar to the dead drops of Vuvuzela.

The exchange service operates in synchronized intervals, called $rounds$, intended to be the order of one second. Sychronization prevents the adversary from using time as an informative variable. It also enables servers to bundle data in random order, effectively unlinking the input and output from servers during each round. The details of the synchronized message exchange algorithm for servers and clients can be found in section 4.

Data exchanged through the service is packaged into $file\ exchange\ requests$. Servers must operate according to the time-dependent instructions carried in each request – a disobeying server invalidates a request at any stage of the file exchange process. Because servers do not affect how these requests are constructed, each acts as an agnostic facilitator of secure file exchange. Details of how these requests are constructed can be found in section 4.1.

Clients designate an $entry\ server$ as the entry point into the server network, which we represented by a graph $G = (V(S), E)$. For each request, clients sample κ servers to form a $request\ path$ in G, $\mathcal{P} = (S_1, S_2, ..., S_\kappa)$. Requests give instructions to be forwarded and then returned along $\mathcal{P}$, moving one hop every round. The $destination\ server$, S_κ, reads the mailbox location (a pseudo-random 128-bit ID M), and exchanges the contents of this request with any other request tagged with M. All request content is encrypted to preserve the privacy of user data.

This construction gives each server $S_j \in \mathcal{P}$ the opportunity to sanitize requests , making the server location and mailbox ID of the exchanges anonymous and differentially privacy-preserving for ν-sampled trust model over sampling procedure $\mathcal{S}$, see section 3, def 1 for details on the trust model.

RTMessenger improves scalability by thinking about the server network as a congifurable pool, rather than a path, and relies on pairs of trustworthy servers to sanitize the request traces through the network. To give context, the topology of the server network and entry server in Vuvuzela is a fixed chain; failure of any server blocks all messages sent between users. Also, total compromise of N servers breaks differential privacy of the service. The topology of the network in the proposed protocol can be decided independently for each request, and can depend on the computational powers of the network. Failure of a single server only affects requests which pass through that server, increasing robustness to partial server failure. Moreover, compromise of N servers does not always lead to a total break in privacy, only for requests which travel entirely in these servers.

This paper shows how considering G as a clique, tree, or hierarchy allow configuration of the service for different network capabilities. In particular, we show that RTMessenger reduces to a similar protocol to Vuvuzela under a path topology with iterative sampling. We expect exploration of new topologies and sampling procedures to yield fruitful extensions of this protocol. See section 4.2 for details.

3 Security Goals and Trust Model

As the primary goal of RTMessenger, the adversary should not be able to learn if users Alice and Bob communicate over any set of k file exchange requests. This must hold even if the adversary tampers with the network in any way. We say a server is $trusted$ if it runs bug-free implementations of the file-exchange algorithm and does not give the adversary any information that is not visible over the network.

RTMessenger defines a more general notion of trust than [1], framing trust in terms of the likelihood of maliciousness when sampling server indices, called a ν-sampled trust model over sampling procedure $\mathcal{S}$. Intuitively, selections of servers using a procedure $\mathcal{S}$ has a probability ν that a server is trusted. The following definition formalizes the power of the adversary respect to a sampling procedure and security parameter ν.

Let S be a set of parties possibly controlled by the adversary, and let $\mathcal{S}$ be a sampling strategy over an indexing of S, which has access to uniform random

number generator.

Definition 1 *A mechanism M operates with a ν-sampled trust model over $\mathcal{S}$ if, given a set of samples $(S_1, ..., S_{n-1})$ generated by $\mathcal{S}$, we have for the next sample $S_n \neq S_{n-1}$*

$$Pr(S_n \text{ is trusted }) \geq \nu.$$

This definition is convenient for our protocol, but it is not unnatural. For deterministic $\mathcal{S}$, users can be asked the following in simple language: what percent do you believe each of the following operators will operate faithfully? The minimum user response over samples drawn from $\mathcal{S}$ gives appropriate choice for ν. For arbitrary stochastic $\mathcal{S}$, accurate natural language representations are more difficult, but possible for simple $\mathcal{S}$ like uniform sampling.

For comparison, Vuvuzela guarantees privacy under a 1-sampled trust model over $\mathcal{S}$, where $\mathcal{S}$ is the iteration $S_1, ..., S_N$. This reresents absolute faith in a single server. We explore other choices of $\mathcal{S}$ in section 4.2.

Security Parameters As ν is increased, our overall skepticism in κ samples of S increases. As κ is increased, the more parties we expect we need to sample to find one we can trust. In our context, these statements translate probability an adversary can get lucky and deanonymize a file exchange. We will show in section 6 how the values of κ, ν, ϵ, and δ give a tradeoff between differential privacy and the amount of noise which is added into the system.

For a server pool S of size N, we guarantee (ϵ, δ)-differential privacy over k file exchanges for ν as long as κ can be chosen large enough (see 6 for bounds on κ in terms of ν) without being larger than the entire network size. To setup, we define differential privacy for a single round, directly from [1].

Definition 2 *[1] A randomized algorithm M is (ϵ, δ)-differentially private for adjacent inputs x and y if, for all sets of outcomes S, $Pr[M(x) \in S] \leq e^{\epsilon} \cdot Pr[M(y) \in S] + \delta$.*

In our context, x and y are adjacent inputs if they differ in the actions of one user. For example, if x is the real action that Alice shares a file with Bob this round, then y would be a "cover story" for the same round, in which Alice shared a file with Charlie, or shared files with no user. By using this definition, all cover stories appear almost equally plausible, allowing Alice to defend herself against accusations. We condition this statement on the fact that Alice's cover story spans at most k requests.

Limitations Not all information can be hidden from the adversary, like client connection status, entry server, or security parameter ranged used. We also require that clients trust one another when they engage in conversation (dishonest users can simply give file-exchange records to the adversary and break privacy).

4 Achieving Privacy for File Exchanges

A user Alice generates her user key pair (sk_A, pk_A), which need not be unique in the system. To exchange files, users Alice and Bob gain knowledge of the public keys pk_A, pk_B and issue requests until the other user responds.

Assume that secret keys are not known to the adversary when Alice and Bob create them; servers and clients have access to a collision-resistant hash function H which produces C-bit output (ideally large, such as $C = 128$).

4.1 File Exchange Overview

Alice and Bob each decide on a request path length κ, κ',resp., and compute a shared secret key $sk_{A,B}$ using a key agreement protocol like Diffie-Hellman [4]. To exchange a file segment, Alice and Bob compute a pseudo-random $destination\ server\ S_\kappa$ and pseudo-random ID M, specific to the request, from padded hashes of their shared secret key. Then, they anonymously request a file exchange on the chosen server, tagged with that ID, and send the request out to their resp. entry servers. Since IDs and destination servers are chosen pseudo-randomly for each request, these values do not reveal information about which users created them.

Each server has a list of clients to which it is responsible to forward/return requests to/from the server network. A server decrypts requests with an ephemeral shared secret key computed using DH on the server public key and an ephemeral public key given in the request, then forwards to a server designated in the decrypted data. Requests with matching request ID during one round are content-swapped and returned. Servers encrypt and return this request along the reverse request path, returning the request to the client. This ensures that each computational step is cryptographically indistinguishable.

As the service is synchronized, servers start a new round once request parsing has finished, or a maximal timeout has been reached (to reduce the global effect of laggards), whichever comes first. All requests forwarded during one round are randomly permuted before sending, so incoming and outgoing messages cannot be linked by their content alone. Servers collaborate in pairs to add $cover\ traffic$ – fake requests statisticaly indistinguishable from user-issued requests—to hide the remaining visible variables in the protocol, described in detail later.

Request Construction To make requests cryptographically indistinguishable while including routing directions, public ephemeral keys and server IDs for the next server in the request are encrypted in a layered request onion. Requests are encrypted using ephemeral keys, in turn, so that exactly one server $S_j \in \mathcal{P}$ should be able to compute the jth decrypted layer of the request as it is forwarded, and the $(\kappa - j)$th encryption of the request as it is returned. As a result, the adversary cannot tamper with requests given to an honest server to forward without garbling all subsequent requests; nor can the adversary spoof requests on the return path without garbling request content. In short, each trusted server can enforce its portion of the intended request scheduling, regardless of other server behavior.

4.2 Server Network Topology and Sampling Procedures

Different system configurations can be achieved by generalizing the topology of the server network. To this end, servers and clients find common data to generate common terminal paths in G, accomplished by a global weighting of servers. This topology enables requests path to vary between each request, allowing clients to exchange data in a distributed manner. We must be consider how the adversary can exert his control over the network given this distributed agreement process to

ensure that privacy is maintained under various topologies of connected G. We explain how to do so for a tree, clique, and k-partite hierarchy.

Global Server Weighting The servers and clients come to consensus on a weight for each server based on the amount of requests that can be processed during one round, denoted $\lambda_i \in \mathbb{Z}$ for server S_i. Reindex via $i \mapsto i'$ for index set I', by sorting in non-increasing order of $\lambda_{i'}$. This allows us to compute a weight function, convex for $i' \in I'$, $R(i') := R_{i'} = \frac{\lambda_{i'}}{\sum_{i' \in [N]} \lambda_{i'}}$. From this, a natural extension is to describe a CDF $F(x) = \sum_{1 \leq j \leq x} R_j$, which is unique for sets $\{\lambda_i\}_i$ of size N (easy to achieve by adding small random noise to each weight).

Client and servers download a compact approximation of F, which can be provided with a signature by each server to prevent spoofing. For sub-ϵ' resolution, provide a $\lceil \frac{1}{\epsilon'} \rceil$-bit relative weight value for each S_i in order $1, ..., N$, and use the interval between the i and $i-1$th value. Sorting S_i by these weights gives our reindexed I', giving a globally-known approximation of $R_{i'}$.

This allows all clients to efficiently pseudo-randomly sample servers through the standard inverse transform sampling method. Importantly, this construction allows all servers and clients to perform the same sampling, so that cryptographically indistinguishable samples can be generated by any honest party. Because our trust model only restricts samplings to omit repeats, servers do not need to know the previous servers along the request path to generate such a path.

Clique Topology Sampling If our server network is capable of quickly transmitting data blocks between all pairs of servers in one round, then a clique topology is an appropriate choice for our network. Requests bounce randomly through this network, with the next node chosen in proportion to their relative ranking weight in R (without the current server included).

In the following sampling procedure $\mathcal{S}_\kappa$ for a given ranking R and security level κ, we refer to servers by their index in I'. To ensure that client request meet at the destination server, clients share S_κ by computing $F^{-1}(H(z)/2^C)$, where C is the bit length of our hash and z is known secretly and varies for each request. Clients sample the rest of $\mathcal{P}$ using $F^{-1}(u), u \xleftarrow{\$} [0,1]$ until κ have been drawn, rejecting repeated samples. The request path $\mathcal{P} = (S_1, ..., S_\kappa)$.

Crucially, this procedure allows client pairs to agree on server destinations pseudo-randomly, but allows servers to create indistinguishable cover traffic. Restricting the sampling to omit repeats reduces the likelihood of some poor samplings, with many repeated indices.

The load of request swaps is distributed according to the distribution described by F, balancing the overall load of the file exchanges appropriately. This effectively multiplexes the cryptographic operations across the servers in proportion to capability, maximizing the efficiency of the network as a whole while limiting the ability of any set of compromised servers to deanonymize users.

Tree Topology Sampling If we have centralized server cluster, a tree topology is appropriate. We fix a globally known tree T so all requests paths propogate up and down the tree with respect to the root. This topology places a large computational burden on the root of T.

Let the value l represents the the longest request path possible and the highest level of security offered. To choose a tree, put $l = \lceil log_k(N) \rceil$ and T can be a

k-ary tree. We build T of depth l with N nodes from the ranking R by putting placing i into the i'th place in an array and creating the k-ary tree from the array representation of T. Assume $\kappa < l$ and denote the m nodes at depth κ as $v_0, ..., v_{m-1}$.

Bottom-up $\mathcal{S}$: Let the m nodes at depth κ be $v_0, ..., v_{m-1}$. Sample uniformly $u \xleftarrow{\$} \{0, ..., m-1\}$ and put $S_1 = v_u$, and recursively define $S_{i+1} = Parent(S_i)$ in T, $i \leq \kappa$.

Computing request paths in this method is simple and efficient for clients and servers. We note that the Vuvuzela protocol uses a tree topology using a 1-ary tree, or path, with $l = \kappa = N$ for all messages.

Hierarchical K-partite Network If our network is heterogeneous in terms of processing power, we can organize our network into a ranked K-partite heirarchical graph, using K bins to group elements of S. The groups are indexed by $1, ..., K$, with $\lfloor N/K \rfloor$ servers in each group (barring possibly one group). In this setting, the maximum value of κ is K.

Our sampling procedure $\mathcal{S}$ samples $S_{K-\kappa+j}$th server of $\mathcal{P}$ from the $(K-\kappa+j)$th bin uniformly at randomly, with client agreement on S_K for $j \in (1, ..., \kappa)$. All requests travel from one bin group to a neighboring group during each round, so each server must maintain at most $2\lceil N \rceil K$ i/o connections with other servers. This network is simple to organize, makes our network less prone to malicious variation of rankings R, and leads to a balance between destination server load and noise request requirements. See section 7.

4.3 Four Ingredients for Differential Privacy

Using the temporary mailbox design, RTMessenger achieves privacy through a combination of fixed-bandwidth/fixed-schedule protocols, mixnets, request path disguising, and cover traffic.

Fixing bandwith and schedules The adversary should be able to listen to all of Alice's network traffic without compromising her privacy. Her client then should not reveal information based on the frequency or size of the requests she issues.

All request paths of length κ have the same bit-size s_κ by fixing content and pad size, so request size reveals only κ. Constant frequency of sending request is maintained by always sending ρ requests every round, filling unused request slots with fake requests. Constant frequency of receiving returned requests is maintained by setting request persitence in the network, defined below, as a deterministic function of κ.

Definition 3 *A request with path $\mathcal{P} = (S_1, ..., S_\kappa)$ in G has persistence 2κ, if intended to be forwarded to the next server in the path before the next round, then returned along the reversed path in the same manner, returning to the user after 2κ rounds.*

A correct implementation of the protocol at all servers in $\mathcal{P}$ should enable request forwarding between servers $\kappa - 1$ times in $\kappa - 1$ rounds, followed by returning along the $\kappa-1$ servers along the reversal of the path during $\kappa-1$ rounds. Fixing persistence in terms of κ makes request schedule independent of request content if all servers act correctly.

Because the adversary may tamper with network traffic and compromised servers may disobey the protocol, we must ensure that the presence of at least one server along the request path can ensure constant frequency. As an example of an attack, consider what could happen if the adversary blocks all network traffic except that from Alice and Bob, who issue requests of size s_κ during round r. In round $r + \kappa$, the adversary can delete all requests which have no mailbox "partner". In round $r + 2\kappa$, the adversary can determine if Alice and Bob are not talking to one another if no request is returned to them.

This and similar frequency-attacks can be thwarted if every server calculates the expected round that a request will return along the reversal of the request path. During that round, a server will return some indistinguishable request for each request it is expected to return. Thus, the adversary cannot learn schedule-related information, as long as one server in $\mathcal{P}$ is trusted. Persistence is then constant to the adversary which cannot control all servers along $\mathcal{P}$, making the protocol fixed-schedule and fixed-bandwidth with respect to client actions.

Server Mixnet To unlink the sender of a reqest and the mailbox, the set of all requests decrypted and scheduled to be forwarded to server S_j before round r is shuffled according to a random permutation $\pi(j, r)$. When a trusted server performs this operation, then the adversary cannot track how incoming requests are linked to outgoing requests, although the adversary can count the sizes of incoming and outgoing messages. This will be hidden by adding cover traffic, covered next. When a request is returned to a server, the index of the request is provided, so the server can use π^{-1} to determine where to return the request and what ephemeral key to use during encrytion.

4.4 Cover Traffic and Hiding Mailbox Access Counts

Before continuing, consider the following strawman setup which naive applies the cover traffic strategy of Vuvuzela. Clients build request onions to travel through servers according to samplings of $\mathcal{S}$, meeting the partner request at a common destination server with a random mailbox ID. Content is exchanged and returned to the client. If cover traffic and request mixing is applied by each server indepedently, are we releasing privacy-breaching information to the adversary? The adversary can potential build statistics of mailbox IDs and destination servers, but that is not informative in itself since requests cannot be linked to users and cover traffic was added to be cryptographically indistinguishable. However, the adversary can trace request paths from each mailbox access back through the network, to the last trusted server on the request path. Because the adversary has control of the network, all communication except that of two users Alice and Bob can be blocked. If one trusted server sourced both requests ending at the same mailbox, then with a large pool of servers, there is a small chance that this request was issued by clients Alice and Bob, since Bob's request did not likely pass through this server during the same round as Alice's.

We can prevent tracing attacks by relying on every pair of servers which is the same number of hops from some destination server to collaborate to generate cover traffic. Each such pair of servers agrees in secret on mailbox locations for each request generated, distributed according to the final sampled value of $\mathcal{S}$ so that pairs of fake mailbox accesses can be generated by any pair of trusted servers

without adversary knowledge. Thus, tracing request from mailboxes in reverse to two trusted servers results in $dead\ ends$ of traced requests. For example, in the tree topology all server pairs for each layer collaborate to send requests to the same mailbox on the root. In the clique topology, all servers must collaborate with all other servers. In the hierarchy topology, all server pairs for each bin perform this operation. This restriction further highlights the tradeoffs presented by selecting a given topology: increasing the possible number of request path "choke points" reduces the overall fraction of the load for destination servers.

To hide mailbox access counts at destinations, we leverage the noise adding argument of [1]. RTMessenger servers adds noise requests—randomly generated and indistinguishable from real user requests—to prevent statistical correlation attacks. Since an adversary knows request sizes from network traffic, we only need to add cover traffic of size s_k, $1 \leq k < \kappa$ for each pair of servers which received some request size s_{k+1} in the previous round. As an added benefit, infrequently visited pairs of servers will likely create less cover traffic than the worst case scenario during light network loads, reducing the overall load of all servers under normal scenarios. We conjecture that this saving will appear most in network topologies where the server distribution after sampling has a heavy-tailed distribution.

We note this strategy can also be applied to [1] without loss of privacy, since the adversary knows if clients are logged on and sending requests by their network traffic. The adversary can immediately invalidate all forwarded requests when no client requests reach the entry server, making cover traffic unneccessary.

The amount of cover traffic needed for privacy is independent of the number of users operating clients, and depends only on overall privacy and user privacy requirements. See section 6 for details.

4.5 File-exchange requests and user settings

In most systems, a global security level is recommended for all communications for simplicity. As added functionality, a user should be able to specify a different κ for each active file-exchange, based on the sensitivity of that request, understanding that the minimum chosen value of a request pair will dictate the level of privacy. If we are not careful, the distribution of κ over time can leak information about user's communication pattern. There are two approaches we can take to solve this problem.

Asymmetric, Fixed Security Parameters We can permit assymetric request path lengths for each half of file exchange with little alteration to the sampling procedure, so that each user can independently select her security parameter for all file-exchanges. Conversation partners need not know both security values for this to work. Moreover, since requests contain information only about the current request size, there is no information for the adversary to link mailbox accesses with the round the request was created. The adversary can learn a client's security settings, which may not be palatable to some users.

Symmetric, Variable Security Parameters Alternatively, we can allow different privacy levels for each conversation, with less simple alterations to the protocol.

Seemingly contradictory, we do not want to the adversary to learn communication patterns, but a client must advertise a variability in request size. We argue

that as long as all users agree to the same sampling procedure to choose request security parameter κ, and issue $some$ request of size s_κ (i.e. a fake request if no pending request has that security parameter) every time a value is sampled, then the adversary learns nothing about conversation patterns by observing network traffic from a client.

As a simple example, assume the globally acceptable range of κ is $[\kappa_{min}, \kappa_{max}] = K$, and the set of some client's pending requests is $Q = \{(\kappa, q)\}$. Sample uniformly $k \xleftarrow{\$} K$, and place some (k, q) in Q in the outgoing request queue for this round, or place a fake request of size s_k in Q if no such request exists. Repeat ρ times each round. This will necessarily increase the latency of all requests. Similar procedures can be created to reduce latency by using the shared secret of file exchange partners, but care must be taken not to reveal any more information than is publically available or risk breaking privacy.

5 File Exchange Protocol Details

The following illustrate how to implement file exchange using a RTMessenger client, with security parameter κ chosen according to section 6, which assumes (ϵ, δ) are global parameters.

Every server advertises a public key pk_{S_i}. We refer to S_i in this section by the weight-ranked index, rather than the global index. Set the maximum number of rounds to r_{max}.

5.1 File Exchange Algorithm: Client

Alice's identifies active file-exchange parter Bob according to a public key-group pair (pk_B, g). The group g is unique amongst all of Alice's file exchange groups, created locally by Alice. To initialize , the client queries a trusted server(s) for the current round r, and anonymously registers with entry server S_0 by sharing the public portion of the ephemeral key pair (sk_0, pk_0). Let the shared key $shk_0 = DH(sk_{serv}, pk_0) = DH(sk_0, pk_{serv})$.

When Alice wants to send a file with security parameter κ to Bob who uses arbitrary security parameter κ', she splits the file into into equal size chunks, one being F, and adds header data, including date of submission and file index, to each chunk. All necessary information to issue a file request is contained in $(\kappa, (pk_B, g), c = Enc(DH(sk_A, pk_B), F))$, which is added to a list L in arbitrary order after user submission. Alice's client performs steps the algorithm in order and then repeats. If she wants to idle, she stops transmitting, but can continue to queue files.

1. **Queue Files**: The client waits for S_0 to announce the round r has begun. Sample κ from the global security parameter distribution. Choose the request $q = (\kappa, (pk_B, g), c)$ with smallest index in L and transfer to Q. If no such request exists, create a random key pk_{rand} and put $q_0 = (x, (pk_{rand}, 0), 0)$ in Q. Repeat until $|Q| = \rho$.

2. **Address Requests**: Remove all $\{q_i\}$ from Q destined for Bob. For q_i, compute the destination server $S_\kappa = F^{-1}(H(i|r + \kappa|sk_{A,B})/2^C)$, and mailbox ID $M = H(i|3r_{max} - (r + \kappa)|sk_{A,B})$. This establishes the same pseduorandom server and mailbox location for the ith request of Alice to colocate

with Bob's ith request sent during round $r + \kappa - \kappa'$ to meet during round $r + \kappa$. Repeat this process for each user represented in Q, storing M and S_κ for each request.

3. **Compute Remaining Request Paths and Encrypt**: Remove the first element of Q: an addressed request q, with security parameter x. Sample $\mathcal{S}$ using a random seed to get non-repeating $(S_1, ..., S_{x-1})$. Create an ephemeral key pair (sk_j, pk_j) and shared key $shk_j = DH(sk_j, pk_{S_j})$ for $j \in [x]$. For the base case of the layer-encrypted reqest, put $e_{x+1} = (M|c)$. Recursively re-encrypt and pad, so $e_i = (S_{i+1}|pk_{i+1}|Enc(shk_{i+1}, e_{i+1}))$, for $x > i \geq 0$, omiting S_{x+1}. Add $Enc(shk_0, e_0)$ to the end of Q. Repeat $\rho - 1$ times and upload Q to S_0.

4. **Recieved and Decrypt Returned Requests**: S_0 returns requests in the form (j, r', q'), the j-th indexed request uploaded during a previous round r'. If this request was fake, discard it. Otherwise, use shared keys $shk_j, j \in \{0, ..., x\}$ associated with the initial request to decrypt q' in order $0, ..., x$. Finally, use $shk_{A,B}$ to decrypt the request content c and add to the local record of file-exchange with (pk_B, g).

5.2 File Exchange Algorithm: Server

The server network is in charge of passing the client's requests along to their destination server, swapping message content when pairs of requests are located in the same mailbox, and returning swapped request. Servers also add cover traffic - fake requests - to ensure privacy. A server's $forward\ queue$, $Q_f(r)$, contains all request to forward during round r. Similarly define the $return\ queue$ $Q_b(r)$.

A server $serv$ operates the protocol by running the algorithm below, beginning at round r.

1. **Parse Requests from Servers**: If $serv$ has an entry in $Q_f(r)$, it will include a public key pk and ciphertext e. Compute the shared key $shk = DH(pk, sk_{serv})$ and symmetrically decrypt $e \mapsto m$ for all requests in $Q_f(r)$. For each mailbox-bound request $m = (M|c)$, swap c with another request if it matches the value M and put c in $Q_b(r + 1)$ with the return address of the matched request. For each other $m = (S_{next}, pk_{next}, e_{next})$, put (pk_{next}, e_{next}) in $Q_f(r + 1)$, bound for server with index S_{next}.

2. **Announce Traffic Size Distribution:** Each server shares a list of input request sizes to all other servers, signed with their public key.

3. **Add Cover Traffic Singles**: If a request of size s_{y+1} was received, then generate n_1 from Laplace(μ, β) capped below at 0. Add $\lceil n_1 \rceil$ single accesses to mailboxes in the same fashion as fake requests generated by clients with path length y.

4. **Add Cover Traffic Pairs**: If both this and another server S_c recieved requests of size s_{y+1} which may meet at a common destination under $\mathcal{S}$, collaborate to form fake requests. Compute the shared secret $shk_c = DH(pk_c, sk_{serv})$. Sample n_2 from Laplace(μ, β) capped below at 0. Add $\lceil n_2/2 \rceil$ file exchange requests in same fashion as steps 2 and 3 of the client algorithm with shared key shk_c, security parameter y, new ephemeral keys, and random content.

Use the counter i which counts total number of cover traffic requests which have been generated with S_c to meet at round $r + y$. Add all requests to $Q_f(r + 1)$.

5. **Suffle Requests:** Compute a random permutation $\pi(j, r + 1)$, mixing all requests to forward to S_j contained in $Q_f(r + 1)$. Upload $Q_f(r + 1)$ to S_j.

6. **Encrypt Results and Return**: A server receives returned requests tagged with the index and round it left $serv$. If a reqest was expected to be returned to $serv$ in round r but is missing in $Q_b(r)$, fill the content with a special message (of the expected size, padded with a nonce) indicating some subsequent server disobeyed the protocol, and add to $Q_b(r + 1)$. Each request in $Q_b(r + 1)$ is encrypted using the associated shk and returned to the previous server or client, tagged with the index in the bundle and round it left the other party.

5.3 File Exchange Commentary

These algorithms mirror Algorithms 1 and 2 in [1], with modifications to include variable request paths and sever addressing client-side, and paired-cover traffic on server-side. Intuitively, since the the next server a request should be forwarded to can be read only by a party with the shared secret key in the decrypted request onion, then one honest server can hide the total request path from the adversary, allowing fake requests to serve as equally plausible cover stories. The amount of noise added in step 2 of the server algorithm is determined by values described in the next section, and the security values of each client can be respected since noise volume depends only on (ϵ, δ). We will see in the next section how this algorithm is sufficient for private file-exchange.

We included terminology to enable group file-exchange and multiple file-exchange partners during a single round. Users share files in a group pair-wise between users, which prevents group size from becoming a visible variable on the destination server. We considered using group mailboxes and all-pairs content sharing, with noise to hide accesses for each mailbox group size. We omitted this for clarity.

Retransmission of unmatched requests is handled by resending requests client-side, explained in section 7.

6 Analysis

We want to show that given our trust model parameters ν, we can provide enough noise to provide (ϵ, δ) differential privacy. We start by analyzing one request, and then consider k requests. Most of the analysis follows directly from section 6 of [1].

Assume our encryption scheme is cryptographically secure, and H is collision-resistant. Without loss of generality, we ignore all requests created by the adversary, since they do not affect the intended actions of clients, and an adversary gains no more information by introducing them since honest servers processes all requests indepedently of one another (except for mailbox pairs at the destination, but adversaries cannot match mailbox IDs on an honest server with non-negligible probability).

6.1 Observable Variables

To start, consider the observable variables the adversary can learn from one request. Clients send ρ requests of fixed-size for a given request path length κ, whose schedule in the server network is deterministic. Each request is encrypted in layers padded with the next server index in $\mathcal{P}$ using new ephemeral keys; the returned request is similarly encrypted using shared secret keys. Thus, the adversary cannot learn which requests correspond to which mailboxes on destination servers before they reach the destination and only if all layers of the onion have been stripped properly.

If the destination server is honest, then we do not need cover traffic to hide mailbox access counts: an adversary cannot decrypt the last layer of encryption and all requests are returned according to the same schedule regardless of the mailbox configuration. However, we still require cover traffic to fuzz the distribution of network traffic between servers. Because pairs of cover traffic requests choose destination servers according to $\mathcal{S}$, then the overall distribution of requests cannot be distinguished from samples of $\mathcal{S}$. We argue that because the destination is resampled for each message, mailbox access distribution compiled over all compromised servers is the only informative variable revealed by this choice. The privacy preservation of this variable will be covered in the next section.

Assume that Alice and Bob attempt to exchange files and send one request along paths whose node set is S, T, resp.. As the worst case, we must consider when the destination $S_\kappa = T_\kappa$ is compromised. Because κ servers (including the entry server) must decrypt each request onion before it reaches S_κ, we assume that some server pair T_j, S_j for $0 \leq j < \kappa$ is mutually trusted. According to the protocol, S_j and T_j mix all incoming and outgoing messages and agree on request destinations so the adversary cannot trace a request path from a client to the destination (without arbitrarily guessing). Further, since all pairs of requests (real and fake) intended for the same mailbox leave from T_j, S_j, the adversary cannot trace which request path is linked to which client. The unmatched pairs of cover traffic requests generated by T_j, w.l.o.g, cannot be linked to T_j, so the different in cover traffic amount sampled by T_j, S_j appears as ordinary single mailbox accesses, given the collision-resistant hash function. Two honest parties (servers or clients) chose the mailbox ID indistinguishably and request content is encrypted (or random for fake requests), so no information is revealed by these variables.

According to the protocol, S_j, T_j furnish fake requests in shared pairs or singly, which are mixed randomly into real requests to be forwarded next round. The distribution of request paths, mailbox IDs, ephemeral keys, and request content is indistinguishable from client-created request paths, so no information is revealed through any one request's content about it's validity. The adversary does know, however, which mailboxes are accessed as well as the request content on compromised servers.

Honestly generated mailbox IDs collide with probability 2^{-C}, so we can assume with very high probability that mailboxes are accessed once — if a file exchange is not reciprocated – or twice — if both parties issued a file exchange. Thus, the variables known to the adversary are the network address of a client and the distribution of single and double mailbox access counts on compromised servers.

6.2 One round of requests

Choosing κ Let the user select ν and assign κ from $f(x) = \lceil \frac{lnx}{ln(1-\nu^2)} \rceil$. This gives the minimal κ value such that all pairs on two request paths sampled from $\mathcal{S}$, equidistant from the destination server, fail to be honest with probability less than x, derived from $(1-\nu^2)^\kappa \leq x$. So our path length for a given ν increases in proportion to the magnitude of $ln(x)$. For scope, given a ν of 0.50, $f(10^{-4}) = 33$; with $\nu = 0.95$, $f(10^{-4}) = 4$.

Since the likelihood of a server pair being trusted is less than an individual server being trusted, then choosing a $f(x) = \kappa$-length $\mathcal{P}$ (plus entry server) ensures than both the collaborative and single cover traffic generation steps fail along coterminal requests paths, $before$ the destination server, with probability less than x. Thus, when choosing security value δ for the system, we must subtract some $x < \delta$ to account for this failure, then use the remaining $\delta - x = \eta$ instead of δ to decide desired noise volume. Luckily, the amount of noise depends only logarithmically on δ, so we can afford to use a large fraction of the desired δ in x to keep path lengths short.

Required Noise Values We now must consider how much noise must be added by each server to fuzz mailbox access distribution. Let m_1 and m_2 be the number of mailboxes that are accessed once and twice, resp., on all destination servers. Because we want to prove privacy for arbitrary $\mathcal{S}$, we do not assume any particular set of destination servers are trusted, so we assume they all can be compromised. View figure 6 from [1], which displays the changes in m_1 and m_2 for all possible combinations of Alice's real actions and her cover story. In each case, the absolute value of m_1 changes by no more than 2, and m_2 by no more than 1.

We use Theorem 1 in [1] to show that the noise added to m_1 and m_2 gives differential privacy.

Theorem 1 *[1] Consider the algorithm M that adds noise $\lceil max(0, Laplace(\mu, b)) \rceil$ to m_1 and $\lceil max(0, Laplace(\frac{\mu}{2}, \frac{b}{2})) \rceil$ to m_2. Then M is (ϵ, δ)-differentially private with respect to changes of up to 2 in m_1 and 1 in m_2, for $\epsilon = \frac{4}{b}$ and $\delta = exp(\frac{2-\mu}{b})$*

From this, we can compute the inputs to the Laplacian, μ and β, in terms ϵ and δ.

$$b = 4/\epsilon;\ \mu = 2 - \frac{4ln\delta}{\epsilon}.$$

Finally, by choosing using $\eta < \delta$ instead of δ in Theorem 1, and choosing $\kappa = f(\delta - \eta)$ we achieve at least (ϵ, δ)-differential privacy (specifically, with parameters $(\epsilon, \eta + f(\delta - \eta))$) for M, for arbitrary parameters (ϵ, δ), assuming $\kappa \leq N$.

6.3 Multiple Rounds of Conversation

We also want to maintain privacy over k rounds of the algorithm, where the adversary can operate any strategy over these rounds to try and break privacy. As a loose rule, privacy does not degrade more than k times the privacy parameters for one round. Optimal privacy parameters under this process, called adaptive composition, are presented in [6] in theorem 3.3. We use the general, simpler, version presented in [5].

Theorem 2 *For any $\epsilon > 0, \delta \in [0,1]$, and $\delta' \in (0,1]$, an (ϵ, δ)-differentially private mechanism M satisfies $(\epsilon', k\delta + \delta')$-differential privacy under k-fold adaptive composition, for*

$$\epsilon' = k\epsilon(exp(\epsilon) - 1) + \epsilon\sqrt{2klog(1/\delta')}$$

The important dependencies from the equations in Theorem 1 and 2 are how the amount of noise scales with k, ϵ, and δ. The mean μ is proportional to $1/\epsilon$ and depends on $log(\delta)$ scaling of b. To protect against adaptive composition, δ must additionally shrink in proportion to k, while ϵ must shrink in proportion to $\sqrt{k}$ to provide the stated guarantees over k rounds. Thus, per-round μ scales in proportion to $\sqrt{k}$.

A user may let her client idle when she is not currently exchanging files, which can extend her ability to provide a cover story to more than $\lfloor k/\rho \rfloor$ rounds. This holds as long as her cover story differs from her real actions in fewer than $\lfloor k/\rho \rfloor$ rounds. Since she can reveal her real actions for each round she was idling, she can stretch the privacy of the service by using a cover story for the remaining rounds.

Realistic Parameter Values Exact values of our parameters depend on the intended sensitivity and usage of the service. We recommend viewing section 6.4 of [1] for an in depth view of the latency when practical values of (ϵ, δ) for N servers in the path topology.

7 Additional Considerations

Achieving Linear Latency Scaling in Network Size The amount of noise added to disguise a request changes as it travels through the network, depending on the number of servers which can forward a request to the same destination. In this sense, the number of "choke points" which are the same number of hops away from some destination server must make noise pair-wise. As a result, the number of single noise requests is constant, but the number of pairs of servers which must collaborate to add noise will dominate the total noise in most cases. From construction, the hierarchical network gives a constant number of collaboration pairs for requests as they travel through the network. Evenly sized K groups implies $O(\frac{N}{K})$ collaborated fake request sets are added by each server along each step prior to the destination in a request path. In total, at most $O(\frac{N^2}{K^2})$ total collaborative requests will be generated by the servers in each bin for a given level of privacy. This amount of traffic will be added at most $K - 1$ times if a single requests is added to a server the first bin. However, the resulting request paths will be balanced uniformly across servers in each bin, reducing the total amount of requests that any one server must parse by a factor of $O(\frac{N}{K})$, enabling round lengths to scale according to $O(N)$ from one round's worth of requests. Thus, we have a delay of $O(N \cdot K)$ between request issue and reception, and K need not grow with network size, so we have a shot at linear latency scaling with network size.

Retransmission Clients must wait for requests to be returned before they can retransmit any unreciprocated requests. Overlap and repeated transmissions may occur, but including the tranmission round and file index allows clients to accurately reconstruct the other user's submission timeline eventually.

Unique User Global IDs Although users need not reveal their public key to any server, servers might request that users establish a user ID to track client status over time. To this end, entry servers can issue a random ID, valid for some fixed number of requests. This allows servers to mandate how often clients need to register while keeping privacy.

Denial-of-Service To reduce the potency of denial-of-service attacks, servers can implement requirements on top of the standard protocol. Note that clients cannot be spammed by other users, since the probability of guessing mailbox ID's is negligible. Because clients can send a limited number of messages per round, this worry is already minimal. Consider the ever-incentivizing monetary solution: a small fee, scaling with the number of request sent, can be levied against servers and clients, to discourage clients from setting up useless traffic-generating server or client bots. If the system is intended to be free use, then servers can request that clients generate a validation key for a large, randomly created data block for each request they issue. This acts as an anonymous proof-of-work, with little overhead for honest clients, but with detremental effects for bots and spammers.

Public Key Disclosure Clients need not release pk_A to servers. Consequently, the adversary will not be able to track Alice's physical movement patterns without using physical evidence, i.e. a network address linked with a particular location. Extra cautious users may operate their client through a set of proxies to further reduce the risk of location-based privacy attacks.

References

[1] *van den Hooff, J. Lazar, D. Zaharia, M. Zeldovich, N. Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis. SOSP2015*

[2] *Corrigan-Gibbs, H. Boneh, D. Maziéres, D. Riposte: An Anonymous Messaging System Handling Millions of Users. http://arxiv.org/abs/1503.06115*

[3] *Wolinksy,D.I. Corrigan-Gibbs, H. Ford, B., Johnson, A. Dissent in Numbers: Making strong anonymity scale. Proceedings of the 10^{th} Symposium on Operating Systems Design and Implementation (OSDI), Hollywood, CA, Oct. 2012.*

[4] *Diffie, W. Hellman, M. New Directions in Cryptography. IEEE Transactions on Information Theory, Vol. IT-22, No. 6, Nov 1976*

[5] *Dwork, C. Roth, A. The algorithmic foundations of differential privacy. Foundations and Trends in Theoretical Computer Science, 9(3-4):211-407, 2014.*

[6] *Kairouz, P. Oh, S. Viswanth, P. The Composition Theorem for Differential Privacy. Proceedings of the 32^{nd} Internation Conference on Machine Learning. 2015*

[7] *More Chernoff Bounds, Sampling, and the Chernoff + Union Bound method. https://www.cs.princeton.edu/courses/archive/fall09/cos521/Handouts/probabilityandcomputing.pdf*