Cwtch: Privacy Preserving Infrastructure for Asynchronous, Decentralized, Multi-Party and Metadata Resistant Applications

Sarah Jamie Lewis
sarah@openprivacy.ca

June 28, 2018

Abstract

Communications metadata is known to be exploited by various adversaries to undermine the security of systems, to track victims, and to conduct large scale social network analysis to feed mass surveillance. Metadata resistant tools are in their infancy and research into the construction and user experience of such tools is lacking.

In this paper we present Cwtch, an extension of the metadata resistant protocol Ricochet to support asynchronous, multi-peer communications through the use of discardable, untrusted, anonymous infrastructure.

Additionally, we present a threat model for evaluating metadata resistant systems, and discuss the current limitations and future work required to fully realize usable metadata resistance tools.

1 Introduction

During a debate at Johns Hopkins University in 2014, the former director of the NSA Michael Hayden infamously stated, *"We kill people based on metadata"* [13]. Even with such a proclamation, metadata resistance, or the security discipline of hiding not only the *content*, but also the *context* of a communication from an adversary, has not received much attention.

In recent years, public awareness of the need and benefits of end-to-end encrypted solutions has increased with applications like Signal[1], Whatsapp[2] and Wire[3] now providing users with secure communications. However, these tools require various levels of metadata exposure to function, and much of this metadata can be used to gain details about how and why a person is using a tool to communicate. [19].

One tool that does seek to reduce metadata is Ricochet [1], first released in 2014. Ricochet uses Tor onion services to provide secure end-to-end encrypted communication, and to protect the metadata of communications. There are no centralized servers that assist in routing Ricochet conversations. No one other than the parties involved in a conversation can know that such a conversation is taking place.

Ricochet isn't without limitations; there is no multi-device support, nor is there a mechanism for supporting group communication or for a user to send messages while a contact is offline. This makes adoption of Ricochet a difficult proposition; with even those in environments that would be served best by metadata resistance unaware that it exists [9] [18].

Inspired by these reasons we present Cwtch, a protocol for providing asynchronous, metadata-resistant, multi-party messaging based on Ricochet through the use of untrusted, discardable relay servers - in doing so we maintain the metadata-resistance of ricochet's decentralized design, while adding capabilities that support all the features we have mentioned.

Further, we demonstrate how Cwtch can be used

[1]https://signalapp.org
[2]https://whatsapp.com
[3]https://wire.org

as privacy preserving infrastructure to build new decentralized, metadata resistant applications without reinventing lower protocol level functionalities.

Our work makes the following contributions:

- We define a threat model for metadata resistance, including metadata resistance for group communications.
- We introduce Cwtch[4], the first free and open-source software providing metadata resistant multi-party communications, that uses untrusted, discardable infrastructure to facilitate offline sending. Cwtch is designed as an infrastructure layer on which other applications can be built.
- We discuss the various open problems that limit the mass adoption of metadata resistant tools, and propose a number of future research directions.

We define the metadata resistance problem in Section 2. Section 3 describes Cwtch, our system implementing metadata resistant group communications. Section 4 contains an analysis of the metadata resistance properties of Cwtch, and a scalability evaluation is provided in Section 5. In Section 6 we discuss building metadata resistant applications using Cwtch as an infrastructure layer, explore the current limitations of Cwtch and enumerate a number of open problems that must be solved in order to realize the mass adoption of metadata resistant tools. Related work is discussed in Section 7, and Section 8 concludes.

2 Problem Setting

This section describes the metadata resistance model addressed by Cwtch. It first describes the problem and enumerates the properties of an ideal solution. We describe how an extension of Ricochet can be used to partially solve this problem, and then walk through a toy solution.

[4]Cwtch is a Welsh word that roughly translates to "a hug that creates a safe-space".

2.1 Metadata Resistance

It is important to identify and understand that metadata is ubiquitous in communication protocols. However, information that is used to facilitate communication between peers and servers is also highly relevant to adversaries wishing to exploit such information. [17]

For our purposes, we will assume that the content of a communication is encrypted in such a way that an adversary is practically unable to break, and as such will limit our scope to the context of a communication (i.e. the metadata).

We seek to protect the following communication contexts, from all adversaries (including supporting infrastructure):

- **Who** is involved in a communication? It may be possible to identify people or simply device or network identifiers. E.g., "this communication involves Alice, a journalist, and Bob a government employee.".
- **Where** are the participants of the conversation? E.g., "during this communication Alice was in France and Bob was in Canada." - Using communications metadata to track the location of users is well studied. [16]
- **When** did a conversation take place? The timing and length of communication can reveal a large amount about the nature of a call, e.g., "Bob a government employee, talked to Alice on the phone for an hour yesterday evening. This is the first time they have communicated."
- **How** was the conversation mediated? Whether a conversation took place over an encrypted or unencrypted email can provide useful intelligence. E.g., "Alice sent an encrypted email to Bob yesterday, whereas they usually only send plaintext emails to each other."
- **What** is the conversation about? Even if the content of the communication is encrypted it is sometimes possible to derive a probable context of a conversation without knowing exactly what is said, e.g. "a person called a pizza store at

dinner time" or "someone called a known suicide hotline number at 3am."

Beyond individual conversations, we also seek to defend against context correlation attacks, whereby multiple conversations are analyzed to derive higher level information:

- **Relationships**: Discovering social relationships between a pair of entities by analyzing the frequency and length of their communications over a period of time. E.g. Carol and Eve call each other every single day for multiple hours at a time.
- **Cliques**: Discovering social relationships between a group of entities that all interact with each other. E.g. Alice, Bob and Eve all communicate with each other.
- **Loosely Connected Cliques and Bridge Individuals**: Discovering groups that communicate to each other through intermediaries by analyzing communication chains (e.g. everytime Alice talks to Bob she talks to Carol almost immediately after; Bob and Carol never communicate.)
- **Pattern of Life**: Discovering which communications are cyclical and predictable. E.g. Alice calls Eve every Monday evening for around an hour.

2.1.1 Centralized Servers

The vast majority of communication protocols rely on centralized servers to handle routing, and otherwise facilitate communications. Even if participants were to trust such a server unilaterally, that does not solve the metadata analysis problem.

An adversary who is able to observe traffic between each participant and the centralized server will likely be able to perform statistical correlation and reconstruct the routing table of the server – even if communication between the server and the participants is encrypted [14] [6] [20].

Such an adversary would likely be able to derive much of the metadata we have described above.

The Tor Network [8] provides some degree of protection from this kind of metadata analysis: a (non-global) adversary cannot simply observe communication between participants because traffic is obfuscated by onion routing. Other anonymization networks such as i2p[5] provide similar properties.

However, even if we were to ensure all communication took place over an anonymizing network, the centralized server still holds a wealth of information regarding relationships between participants. In order to provide metadata-resistance we must eliminate the need for the centralized server to be aware of this information.

2.2 Goals

To motivate our work we define the properties that any solution to the metadata resistance problem should have. For this we will rely on definitions provided by a previous systemization of secure messaging properties by Unger et al. [21].

Any modern, secure messaging solution must provide *Confidentiality*, *Integrity* and *Authentication*.

As mentioned in our introduction and problem statement we require any solution be *Asynchronous* and provide *Multi-Device Support*.

For this paper we will separate the concepts of *Asynchronous Key-Exchange* and *Asynchronous Conversation*, focusing on achieving *Asynchronous Conversation*. We will discuss *Asynchronous Key-Exchange* further in Section 6.

We define that any metadata resistant system must be *Anonymity Preserving*, providing *Sender Anonymity*, *Recipient Anonymity*, *Participation Anonymity* and *Unlinkability* between protocol messages belonging to the same conversation.

In an adversarial environment requiring metadata protection we must provide mechanisms for both *Forward Secrecy* and *Backward Secrecy*, such that compromise of key material does not enable decryption of previous or succeeding data.

For group application the properties of *Participant Consistency*, *Speaker Consistency* and *Destination Validation* are necessary to ensure a cryptographi-

[5]https://i2p.org

cally secure and *Global Transcript*, and as such be *Causality Preserving.*

Some properties of secure group communications are ostensibly desired, but also difficult to implement in a way that does not compromise usability in a decentralized, metadata resistant environment. *Computational Equality* and *Trust Equality* require a substantial amount of pairwise communication in a group context, the overhead for which is costly in an environment where there may be high latency and low (or no) availability between some participants.

Additionally, while the properties of *Contractible Membership* and *Expandable Membership* potentially reduce the overhead of group management in such an environment, we will not consider them in our solution, because they are generally only achievable with the same communication symmetry necessary to achieve computational and trust equality, and the aforementioned overhead of such algorithms is not desirable

Finally we will state usability criteria for any solution. Taking into account the networking environment and the likelihood of dropped or delayed packets, we require any solution to be *Out-of-Order Resilient* and *Dropped Message Resilient.*

2.3 Ricochet: An Overview

Ricochet is a secure messaging protocol which, through its use of the Tor hidden service protocol, provides online 2-party instant messaging with *sender anonymity*, *recipient anonymity*, *participation anonymity* and partial *unlinkability* (to network adversaries with limited scope) [1].

Ricochet is *Anonymity Preserving* and provides a number of other properties including *Confidentiality*, *Integrity*, *Authentication*, *Speaker Consistency*, and *Causality Preservation* for instant messaging between two parties.

In this paper we will build upon the Ricochet protocol to define and implement a metadata-resistant group chat protocol. However, to start, it is important to understand the properties that Ricochet *cannot* provide us at all, as well as properties which Ricochet does provide for two-party exchanges but which we cannot extend to multi-party protocols.

Ricochet is not *Asynchronous*; it requires both parties to be online at the same time in order to exchange messages.

Further, properties like *Forward Secrecy*, *Participation Anonymity*, and *Authentication* are derived from the hidden service connection between two servers, and thus cannot be trivially extended to provide equivalent end to end protection in a group setting, and must instead be reimplemented at a higher level.

2.4 Toy Solution: Online Metadata Resistant Group Chat

A naive implementation of metadata-resistant group chat which meets the majority of our goals is a scheme we will call *Online Group Chat*:

- Setup: Each client involved in the group chat establishes a Ricochet channel with every other client.

- Messaging: When a client wishes to send a message they must first encrypt the message to every participating client, then send these messages to every client along with a signature.

- Message Receipt / Attestation: Once a client has received a message, they must decrypt it and compare the contents of the message with the signature If the signature is verified, they must then check with all the other clients to ensure that they all received the same message as well.

- Teardown: The group chat ends when the clients destroy their Ricochet channels. Offline clients are unable to participate in the chat from then on.

Online Group Chat requires every client involved in the group chat to maintain a connection to every other client, which requires $\frac{n*(n-1)}{2}$ communication channels. While this scheme does provide many of our desired properties, it cannot provide *Asynchronous* communication, and lacks the usability properties that we desire, such as support for dropped packets, unavailable group members and multi-device support.

Further, requiring any one party to stay online is undesirable; we cannot guarantee that any two peers are available to communicate with each other in a given period of time. Because of this, it is necessary to introduce long-lived supporting infrastructure (outside the peers themselves).

2.5 Introducing a Relay Server

The originator of the group chat, Alice, generates and sends a group-key K_g and a group-chat-server identifier S to participants Bob and Carol, with whom she has previous initiated pairwise Ricochet connections with.

Alice, Bob, and Carol all create Ricochet connections to S using ephemeral Ricochet identifiers – S gains no information regarding who is connecting to it.

When any of Alice, Bob and Carol wish to send a message, they sign it using long term signing keys, and encrypt it using K_g and send the resulting ciphertext c_g to S where it is relayed to all other connected peers.

Even if S is only used by a single group then S is able to approximate the number of ephemeral connections it serves, and thus can derive the number of participants in a group. However, because of the ephemeral anonymous connections, S gains no information as to who is speaking or what is being said, and as such is unable to gain any useful metadata.

Further, if each message includes a signature of a previously seen message identifier, then S has no ability to modify the transcript (by e.g. not distributing a message to the rest of the group) without being detected and because each connection to S is ephemeral (and is regularly torn-down and rebuilt) S gains no information useful to target modifications, and thus can be assumed to be completely untrusted.

3 The Cwtch Protocol

We will now introduce the Cwtch Protocol, a decentralized, metadata resistant, group communication protocol based and expanded upon on the concepts we have described above.

We begin by defining the actors within our system.

- **Cwtch Peer**: A single actor within the network who can initiate connections with other Cwtch Peers or with Cwtch Servers.
- **Cwtch Server**: Independently operated and untrusted support infrastructure which can be used by one of more groups to facilitate communication between Cwtch Peers.

Metadata-resistant in Cwtch is achieved through the use of untrusted Servers that relay messages to all parties. This broadcasting mechanism means that a Cwtch Server is not aware of which peers a given message was intended for.

We assume all supporting infrastructure is untrusted, even in cases where it may be set up by one of the chat participants. We define the properties a Cwtch Server must satisfy as follows:

- A Cwtch Server may be used by multiple groups or just one.
- A Cwtch Server, without collaboration of a group member, should never learn the identity of participants within a group.
- A Cwtch Server should never learn the content of any communication.
- A Cwtch Server should never be able to distinguish messages as belonging to a particular group.

Further, all participants within a cwtch session must be able to detect and/or successfully mitigate when a cwtch server is acting dishonestly. Dishonest behavior is defined as:

- Failing to relay any message: this will be detected when a message ID appears in subsequent messages, but which is not known to some participants. Failure to deliver can be related to server reliability (see section 6) and is not necessarily malicious. Peers can request others resend older messages (see section 3.3.1).

- Modifying a relayed message: this will be detected as a failure to decrypt, and trigger a resend.

- Attaching duplicate messages to the timeline - duplicate messages will be ignored as they will share the same signature as a previously seen message and be discarded.

While iterating these behaviors, it should be noted that it is assumed that because a server is never aware of the peers and groups that it is being used by, it is only ever able to act in malicious ways in a random, untargeted fashion.

3.1 Protocol Specification

Every Cwtch Peer is initialized with an Ed25519 public/private key pair, vk and sk, as well as an RSA-1024 public/private key pair, $O\mathsf{pk}$ and $O\mathsf{sk}$, used to set up an onion service.

3.2 Connections

All communication in Cwtch, whether between Cwtch Peers or between a Peer and a Server, takes place over direct peer-to-peer Ricochet connections (over Tor onion services).

Each Cwtch peer establishes an onion service using $O\mathsf{pk}$ and $O\mathsf{sk}$. This onion service is set up to receive Ricochet connections.

When a peer connects to another it must first authenticate itself using the $im.ricochet.auth.hidden-service$ channel[6] - after which it is free to send messages on a *im.cwtch.peer* channel.

3.2.1 Identity Key Exchange

When two Peers wish the connect, at least one peer must obtain the Cwtch identifier ofthe other and initiate an identity key exchange.

This pair-wise key exchange, shown in Figure 4 can be done when a pair of clients are both online by establishing a Ricochet connection between them and then simply transmitting their name and vk to each other.

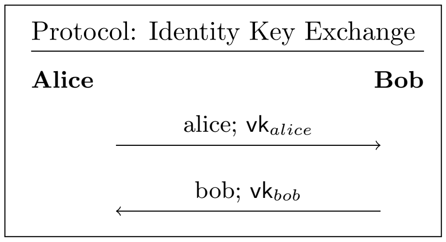

Figure 1: An overview of Identity Exchange

3.2.2 Group Setup

To create a new group the initiator chooses a group server S_g, and generates a symmetric group key k_g. The group identity I_g is randomly generated and then concatenated with S_g and signed with sk, to produce a ticket, T_g tied to the identity of the group. All group parameters are then sent to each participating member of the group.

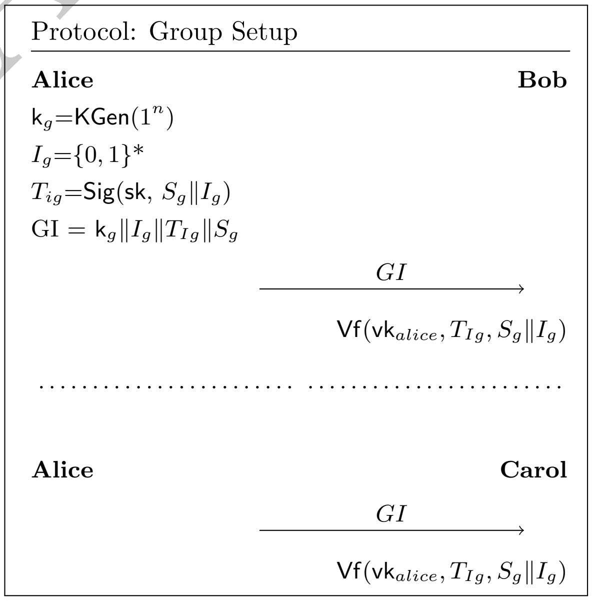
Protocol: Group Setup
Alice
Bob
k_g=$\mathsf{KGen}(1^n)$
I_g=$\{0,1\}^*$
T_{ig}=$\mathsf{Sig}(\mathsf{sk}, S_g \| I_g)$
GI = $\mathsf{k}_g \| I_g \| T_{Ig} \| S_g$
GI
$\mathsf{Vf}(\mathsf{vk}_{alice}, T_{Ig}, S_g \| I_g)$
Alice
Carol
GI
$\mathsf{Vf}(\mathsf{vk}_{alice}, T_{Ig}, S_g \| I_g)$

Figure 2: An overview of Group Setup

Once an invite is received, the recipient verifies $S_g \| I_g$ against T_g to ensure that the ticket has been

[6]https://github.com/ricochet-im/ricochet/blob/master/doc/protocol.md#authentication

signed by the inviter. This prevents an attack wherein someone who is not the originator of a group attempts to invite an unsuspecting party to a group - by doing this they would gain the ability to segment the conversation and thus break the integrity of the *Global Transcript*. The conversation ticket is sent on every future message to the group so participants can ensure that every member of the chat is listening on the same server under the same group ID.

At this point, assuming peers accept the invitation to join the group, peers then initiate a connection with S_g.

It is worth noting that peers who are invited to a group may no know each other. When an invitation is accepted a peers identifier is revealed to all group members.

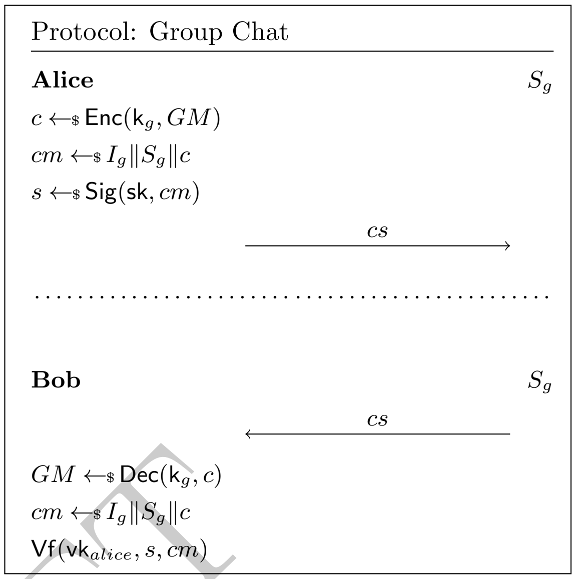
Protocol: Group Chat
Alice
S_g
$c \leftarrow_\$ \mathsf{Enc}(\mathsf{k}_g, GM)$
$cm \leftarrow_\$ I_g \| S_g \| c$
$s \leftarrow_\$ \mathsf{Sig}(\mathsf{sk}, cm)$
cs
Bob
S_g
cs
$GM \leftarrow_\$ \mathsf{Dec}(\mathsf{k}_g, c)$
$cm \leftarrow_\$ I_g \| S_g \| c$
$\mathsf{Vf}(\mathsf{vk}_{alice}, s, cm)$

Figure 3: An overview of Group Chat

3.3 Group Chat

During Group Chat, when a Peer wishes to send a message, they construct a GroupMessage GM, containing the message M (padded and limited to 1024 bytes), the current timestamp T, the most recently seen message signature s_m from the group, an incrementing sequence number for this peers messages, and the signed group identifier T_{ig}. This group message is encrypted with k_g to produce a ciphertext c.

The original GM is concatenated with c and the result is signed with sk to produce a signature s that can be verified by the other group members.

To receive messages each Peer sets up a listen channel on the Server, and for each new message the Server receives the Peer will receive a copy. It is important to note that the Peer will not just receive messages for their groups, but also ciphertexts from other groups. Because they do not have an associated key, these message will fail to decrypt and are simply discarded.

To verify a message is from a given peer, the recipient must construct $GM \| c$ and verify s using the peer's vk. Groups can be constructed in such a way that some group members may not have performed key exchanges with other group members, and as such they will be unable to verify that a given message came from the stated Cwtch peer. This behavior could trigger a peer to attempt to initiate an identity exchange with an onion, but in many usecases this may not be necessary or desirable; that a peer held a valid group encryption key may be sufficient. A Cwtch peer will attempt identity key exchanges with all unknown group participants, but will not trust or peer with these participants until user intervention.

3.3.1 Resending

Each peer maintains a group message sequence number which increments every time they send messages to a given group. When a peer receives a message from a group, they check to see if the author has previously sent any messages to the group, and if they have they compare the sequence number of the last received message from that peer, to the sequence number in the message they just received. If the sequence numbers are not sequential the Peer can send a message to the group requesting that someone resends the messages.

How this resend is implemented is ultimately an application level detail (for some applications, a lost GroupMessage might be seen as inevitable and not critical).

Ricochet Channel im.cwtch.server.listen

1. **Setup:**
   (a) Peer sends OpenChannel message to Server
   (b) Server sends ChannelResult to Peer
2. **On Group Message:**
   (a) When any Peer sends a GroupMessage to the Server, the Server forwards the GroupMessage to every other connected Peer

Cwtch servers store messages for a specific length of time and then discard them. Peers who are not online when messages are sent can setup a fetch channel on the server when they are next online and the server will send all messages currently stored.

3.4 Group Expansion and Contraction

Adding and removing members from groups are desirable functions; however, carrying them out in a metadata resistant environment presents a number of challenges. In particular, while we make the assumption that participants are online during group formation. For reasons given above, this assumption is less defensible during group modification (which we presume can happen at any time).

Further, adding state (such as a cryptographic ratchet) to our system presents problems due to the adversarial and availability model of servers and participants. We must assume that participants never receive certain messages, servers may clear their cache at-will, and longer periods of disconnection would result in certain members being unable to participate in the conversation without online intervention.

Forward Secrecy and *Backward Secrecy* in the Cwtch protocol relies on *opportunistic group renegotiation* with the group leader periodically sending new group invites to online parties. Conversations happening on the set of peers will migrate from one group to the other, sending messages to both groups until the transition is complete. See n 4 for more information on mitigating potential attacks on this approach.

A new member joining is a specific case of a group expansion being explicitly triggered, but afterwards acts no differently.

Backward Secrecy is also achieved from this mechanism: once a member leaves a group, the group leader simply ceases to send them new group invites, and once the group has rotated completely to a new key then they will no longer be able to participate in the conversation.

This mechanism means that it is impossible for the initiator of a group to leave the group. Any transfer of ownership of a group is logically the destruction of one group and the creation of another by a different peer.

We assume the most likely avenue of compromise of a group key is from the physical compromise of a participants computer, detecting and recovering from such a compromise is beyond the scope of this paper, however, if any group key is compromised at any point in time, adversaries will only be able to compromise group messages during that particular conversation. As all group key exchanges happen out of band over peer to peer channels, once the group has switched to a new key (and any compromised participant has regenerated their private keys and initiated new identity key exchanges), the conversation is no longer compromised.

3.5 Signatures

In order to preserve metadata resistance, the system requires that a participant have access to the group and access to the peer's identity in order to verify a message came from a given peer.

However, this presents a potential attack vector. If a malicious actor is a member of a particular group, they could extract messages from one group and relay them to another group that they control (with the same GroupID).

Top prevent this attack, signatures in Cwtch have the following structure $\mathsf{Sig}(\mathsf{sk}, I_g \| S_g \| c)$ - This struc-

Ricochet Channel im.cwtch.server.send

1. **Setup**:
   (a) Peer sends *OpenChannel* message to Server
   (b) Server generates a Challenge C
   (c) Server sends a successful *ChannelResult* and C to Peer
2. **Spam Guard**: When the Peer wishes to send a GroupMessage M
   (a) Peer generate a random nonce N
   (b) Peer calculates $P = sha256(N \| M \| C)$
   (c) If $P[0:2]$!= $\{0,0\}$; goto (a) Otherwise the Peer proceeds to the next step.
3. **Send**:
   (a) Peer sends M and P to Server
   (b) Server calculates $P = sha256(N \| M \| C)$
   (c) If $P[0:2] == \{0,0\}$; The Server accepts the messages as having provided adequate proof of work, otherwise the server discards the message as spam
   (d) Server closes the channel.

ture binds a given encrypted group message to a group ID and a specific server and ciphertext. As such even if an adversary were to extract the message from a given group and relay it to another group the members of that group would not be able to verify that the message unless they shared the same server and group key (this would mean that they would receive all messages from that group anyway, mitigating the attack, and making the compromise obvious.

3.6 Spam Resistance

One major potential pitfall with this kind of design is spam. While somewhat counteracted by the decentralized nature of the protocol (anyone can setup and use a cwtch server), we must consider how to prevent an individual cwtch server from being overwhelmed by bogus messages.

Proof-of-Work places a cap on the number of messages that are accepted by the server that is proportional to the computational power of an adversary.

This certainly doesn't prevent a moderately funded adversary from overwhelming a given Cwtch server, but combined with the ability of groups to select and move to an arbitrary Cwtch server, it makes targeted attacks on the communication of particular groups difficult.

3.7 Multi-device Support

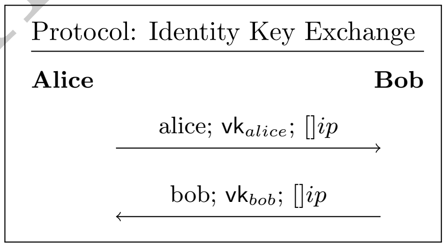
Protocol: Identity Key Exchange
Alice
Bob
alice; vk_{alice}; $[]ip$
bob; vk_{bob}; $[]ip$

Figure 4: Modified Identity Exchange to support Multiple Devices

We can achieve support for sharing an identity between multiple end-user devices in a number of different ways.

Naively, we could simply have a primary device share identity and group keys with another device over a dedicated ricochet channel. However this increases the attack surface needed to compromise a Cwtch peer.

Instead we introduce a third parameter in our identity key exchange. This parameter contains a list of cross-signed identity verifications $[]iv$.

To obtain these verifications a peer with an active profile connects to a peer with a newly created profile over a dedicated ricochet channel. Once authenticated, each peers signs a concatenation of their own Cwtch address, and the address of the connected peer with vk and sends the resulting signature to the connected peer.

This signature can then be exchanged with contacts along with a list of Cwtch addresses as proof that any of those Cwtch address belong to the same identity.

4 Threat Model / Metdata Resistance Analysis

Now that we have outlined the Cwtch Protocol, we will present an analysis of the metadata that a malicious entity could attempt to extract from the system.

4.0.1 A Note on Adversaries

Adversary models of protecting metadata in decentralized systems has been previously systematized [11]. We will discuss Cwtch within the context of those adversaries.

4.1 Adversary: Relay Servers

Cwtch Servers are responsible for supporting asynchronous, multi-party communication through relaying messages. While they have access to metadata about the receipt times of GroupMessages and the access pattern of ephemeral identifiers, they should not be able to derive further information from these.

4.1.1 Peer / Server Metadata

Peers connect to servers over Ricochet using an ephemeral identifier. There is no connection between a Peer's long-term identity and its interaction with a server or any set of servers.

Every peer receives every message sent to a particular server. This is equivalent to the naive PIR design, in which every participating peer receives all the information in the system, and all decryption is conducted by the peer. In this manner, the server gains no information about which sent messages are intended for which recipients.

Such a design can be see as inefficient, but the potential inefficiencies must be weighed against the understanding that Cwtch has no central servers. Each group can choose any Cwtch server to act as a relay for the messages for that group, or they can set up their own. The load for Cwtch messaging is not centered on a single server or set of servers.

4.1.2 Attack Vector: Timing Sidechannels

There is a potential for information leak through the timing differences of various peers as they listen to messages from the server.

A Peer must attempt decryption of all messages under all group keys. If peers find a successful decryption they can stop trying other group keys. Depending on implementation, this has the potential to leak into the listen channel in the form of a noticeable difference in time between processing a decryptable message vs. a non-decryptable one.

While such a leak would be difficult to capitalize on, based on the other metadata resistance properties we discuss in this section, it is still undesirable.

Our prototype implementation separates receiving messages on the listen channel from attempted decryption through message passing; thus the listen channel timing runs independently from any decryption attempts.

4.1.3 Attack Vector: Malicious Servers Estimating Group Cliques

One possible attack vector present in Cwtch is the ability for a malicious server to deny certain connections with the aim of isolating a group on a given server. This may lead to the server being able to determine or estimate certain properties about the group (size, message frequency, pattern of life).

Such analysis would be difficult and opportunistic, as mentioned peers connect using ephemeral identifiers which they rotate periodically, and multiple devices belonging to the same peer will exhibit behaviour similar to multiple users. Groups are free to rotate servers at any time. Any malicious server attempting to perform such an attack would either have to rely on data outside of the system (targeted surveillance) and/or make a number of assumptions about how their server is being used (e.g. assuming two ephemeral peer identifiers are actually the same peer).

It is possible to make this attack even more difficult by having peers create a new connection to the server whenever they wish to send a message. This would reduce the server's ability to link messages to a given group to zero, but comes at the expense of increased latency in addition to the cost of setting up new Ricochet connections.

4.1.4 Attack Vector: Cwtch Servers Overwhelming Peers

A Cwtch Server can construct their own arbitrary message and overwhelm a peer/multiple peers. As with other attacks outlined here, such a denial of service attack could not be targeted and as such would be opportunistic. Peers gain knowledge about how busy a given server is and so could detect and decide to move servers if message latency/processing was greatly impacted by load.

4.2 Adversary: Harvesters

Peers and Servers are able to observe all traffic in a given Cwtch system. Only peers belonging to certain groups are able to decrypt messages for those groups.

Harvesters are adversaries that can gather and store such information, and may also attempt connections to every Cwtch peer they identify to collect and store Cwtch Peer Identity mappings.

The usefulness of a widely collected mapping of Cwtch Peer Identities is limited. Basic identity information in Cwtch (name, onion address) is assumed to be public. Assuming the Harevester was left as untrusted, they would gain no more information from which to build a more detailed social map.

4.2.1 Attack Vector: Decryption after Key Compromise

The main risk of such harvester collection is future key compromises that provide unencrypted access to message communications and allow context to be derived after the fact.

The main defence against this is shorter group contexts, as discussed in Section 3.4.

4.2.2 Attack Vector: Peers Estimating Group Cliques

Peers gain some information regarding the utilization of the server. They are able to make note of the number of messages (which they are unable to decrypt) and the time these messages are received. Peers gain no information about who is sending these messages and, unlike servers, are not able to link any two given messages together (unless they are part of the group).

4.3 Friends

Cwtch Peers build direct connections and trusted relationships with other Cwtch Peers. They do gain access to *who* and by being active participants in communication they de-facto gain access to the other kinds of metadata discussed. Friends, however, should only ever gain information about groups that they are invited too, and identities presented to them.

Friends can also attempt to get friends to joining a group. However, a Peer must explicitly accept a GroupInvite before joining a server.

We will not consider the threat implications of friends exploiting trust relationships while in a group or to gain access to a group (e.g. leaking plaintext GroupMessages, social engineering).

4.4 Sniffers

We categorize a *Sniffer* as any adversary able to observe or otherwise monitor connections to and from Cwtch Peers and Servers. The capabilities of this adversary are proportional to the protection of the underlying anonymity network.

While we assume that most practical adversaries are unable to derive any useful information from the underlying anonymity network, our prototype uses Tor onion service connections.

Tor is currently the largest anonymization network with the location hiding properties necessary for the security of Cwtch. As such, it is the obvious choice to protect against the majority of network sniffing adversaries.

However, Tor is not designed to be resistant to a global passive adversary and any such adversary should be assumed to be able to deduce the real identities of those communicating over Cwtch through correlating peer channels and the resulting server channels. In such a scenario the mixing properties provided by Cwtch Servers would not survive long-term analysis.

Should a design of a GPA-resistant anonymization network gain wider adoption it would be prudent to adopt that as the base communication layer in Cwtch's implementation.

5 Usage Patterns and Scalability

Cwtch is designed to be decentralized, with individual groups being self-contained and independent of any specific infrastructure.

It is possible to imagine two specific usage patterns for Cwtch, which have a significant impact on the scalability, latency and overall performance of a Cwtch system.

At a high level, these usage patterns are:

- The number of Cwtch servers is proportional to the number of groups. At the extreme we can imagine each group sets up their own Cwtch server.
- A large number of groups use a small number of Cwtch servers. At the extreme we can imagine a single Cwtch server handling every single group.

The distinction between these two usage patterns is important to convey. In the former, Cwtch servers deal with a low number of peers and messages at any particular point in time. The latter usage pattern

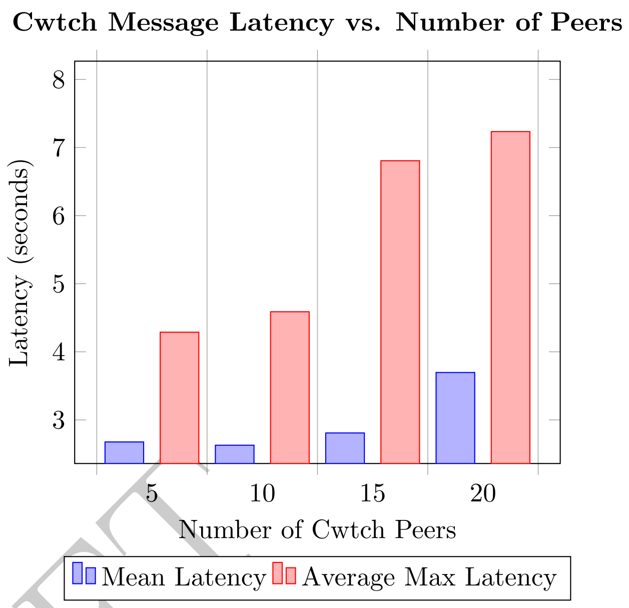

Figure 5: Bar graph showing the increase in mean latency and max latency as the number of peers using a single server increases.

subjects any single Cwtch server to a large number of peers and messages, thus making the specification and design of the server the main factor in determining latency and storage requirements.

We ran multiple experiments to understand the expected latency of Cwtch given the above two usage patterns, which is shown in Figure 5.

For a server handling small numbers of peers and a single group, we saw mean latencies of between *2.6s* and *2.73* from the time a message was sent by a peer, to the time it was received by the other peers. Max latencies were between *3.9s* and *4.6s*. Much of this time (30-50%) is attributable to the spamguard proof-of-work construction defined in section 3.6, and not to onion routing or server performance.

Server processing and connection management also accounts for a significant portion fo the latency. We believe that while some latency increase is to be expected as the number of peer connections rises, improvements in the Cwtch server code and optimistic setup of send channels can likely reduce *perceived* latency by the peer.

In testing we observed that as we increase the number of peers and groups using a single server we see both mean latencies and max latencies increase. We tested our prototype server with 20 online peers spread across 4 groups, we observed max latencies of up to *7.6s* while average latencies increased to around *7.2s*. We expect that as peer connections grow, mean and max latency (as well as server resource usage) grows linearly. Further testing in larger environments is needed to experimentally validate that belief.

It is important to note that the latency figures described above are derived from extreme testing conditions, with all peers sending a continual stream of messages to the server, and the server streaming these messages to all the peers. In the real world, conversations are spread over time, with no more than a few peers participating simultaneously,and thus we expect far smaller per-peer load on the server. Future testing is planned to validate this experimentally.

6 Discussion and Future Work

6.1 Metadata Resistant Applications

As stated in our introduction, one of our goals is not to produce a single, monolithic catch-all application/protocol, but to provide a framework for building privacy-preserving tools.

The base Cwtch protocol provides metadata resistant communication for private groups. However, we can build protocols on top of Cwtch to provide other kinds of services.

These services are enabled by structured data being passed as part of a GroupMessage. For example, a simple group text chat application might contain the structure shown in Figure 6.

```
{
"type": "im.cwtch.text"
"fields" : {
"text": "An example message..."
}
}
```

Figure 6: An example of what a standard Cwtch text message might look like.

Furthermore, we can provide an implementation of a public message or notice board that is also metadata resistant. The only difference between a public board and private group at the protocol level in Cwtch is how invites and keys are managed. Open groups would require an additional component to manage registration requests and/or automate invites for a given group.

In an open group the secrecy of the group key is not a concern, as information posted to the group is deemed public. This does open up the group to potential censorship – if a server becomes aware that a given open group is using it then the server can discover the key and use that to identify and block messages from that group. Other censorship resistant publishing systems have suggested using threshold cryptography to split data between a number of servers (and thus requiring pieces to be obtained from a subset of servers before decryption can occur) [22].

Outside of such strategies, given a diverse set of Cwtch servers, even if certain Cwtch servers did not want to host a given open group (enough to actively censor it), others hopefully would.

It is worth nothing that servers cannot modify any data in an open group, even with possession of the group key. Attempting to modify a group message would mean that members would be unable to verify the signature.

We can define a structure for such notice board messages; for example, a Craiglist-esque[7] personals section might look like the one shown in Figure 7.

[7]https://craigslist.com

```
{
"type": "im.cwtch.listing",
"fields" : {
"title":"24 W4WM Just moved to...",
"message":"Hi Everyone,....",
"tags": "platonic, w4w, w4m"
}
}
```

Figure 7: An example of how a notice board application based on Cwtch might be implemented.

Clients can interpret structured messages in a variety of ways depending on the type of the message sent, and these can be tailored to the users' preferences and threat models.

This does open up a potential issue with conflicting message types being sent over the same channel. It is expected that clients will focus on a particular type of Cwtch application, and ignore message types that do not comply, e.g. a dating app based on Cwtch might support groups of both listings and group chats, but have specific application handling to determine how these are displayed.

6.2 Limitations and Open Problems

In this paper, we have not addressed a variety of adoption and usability concerns, which are essential to resolve before widespread use of metadata resistant tools becomes a practical possibility.

- **Offline Key Exchange**: Cwtch requires that any two peers are online at the same time before a key exchange/group setup is possible. One potential way to overcome this is through encoding an additional public key and a Cwtch server address into a Cwtch peer identifier. This would allow peers to send encrypted messages to an offline Cwtch peer via a known server, with the same guarantees as a Cwtch group message.

  This approach is not without issues, as by encoding metadata into the Cwtch identifier we allow adversaries to mount partially targeted attacks (in particular denial-of-service attacks against the Cwtch server with the aim of disrupting new connections). However, the benefit of first contact without an online key exchange is likely worth the potential DoS risk in many threat models.

- **User Experience of Failures**: Environments that offer metadata resistance are plagued with issues that impact usability, e.g. higher latencies than seen with centralized, metadata-driven systems, or dropped connections resulting from unstable anonymization networks. Additional research is needed to understand how users experience these kinds of failures, and how apps should handle and/or communicate them to users.

- **Increasing Anonymity Sets vs. Decentralization**: Heavily utilized Cwtch servers increase message latency and the resources a client requires to process messages. While Cwtch servers are designed to be cheap and easy to set up, and Cwtch peers are encouraged to move around, there is a clear balance to be found between increasing the anonymity set of a given Cwtch server (to prevent targeted disruptions) and the decentralization of Cwtch groups.

- **Discovering Cwtch Servers**: Much of the strength of Cwtch rests on the assumption that peers and groups can change groups at any time, and that servers are untrusted and discardable. However, in this paper we have not introduced any mechanism for finding new servers to use to host groups. We believe that such an advertising mechanism could be built over Cwtch itself.

- **Usability of Cwtch Identifiers**: Tor onion service addresses are not user friendly, and v3 onion service addresses compound the issue by making identifiers much longer. Ideally users would be able to provide friends and associates with a more friendly handle, or their friends and associates would be able to discover them in a consistent, reliable way.

- **Delivery Reliability vs. Server Resource/Peer Availability**: In Cwtch, servers have full control over the number of messages they store and for how long. This has an unfortunate impact on the reliability of group messages: if groups choose an unreliable server, they might find their messages have been dropped. While we provide a mechanism for detecting dropped/missing messages, we do not currently provide a way to recover from such failures. There are many possible strategies from asking peers to resend messages to moving to a different server, each one with benefits and drawbackss. A full evaluation of these approaches should be conducted to derive a practical solution.

7 Related Work

In recent years secure, private messaging (including group messaging) has risen in prominence and is now commonly deployed[10] through apps such as Signal. However, systems that aim to protect not only the contents of the communication but also any metadata are still an active area of research.

Multiple proposed metadata resistant communication architectures have relied on PIR. Onion-PIR [7] divides server responsibilities between a control server to handle user registrations, and multiple PIR servers which are assumed to be non-colluding. In contrast, Talek [3] presents a private pub-sub architecture based on PIR and introduces two novel techniques: Oblivious Logging and Private Notifications. Talek is designed to protect a large number of client communications from a small number of untrusted servers. Talek require clients to issue dummy requests when they do not need to read and write.

Riposte [4] describes a reverse-PIR scheme in which clients are able to write to a remote database without revealing which row they have written to. This scheme is then used as the basis of an anonymous messaging protocol.

Outside of PIR-based schemes,a proposal by Hopeman [12] uses the construct of a public bulletin board to provide unlinkable message exchange between two parties. When posting a party includes a ”tag”, a way of identifying the location their next post, because the messages are encrypted the messages cannot be linked to each other. A mixnet ensures that requests to the server are unlinkable.

Pond [15] presented a prototype for forward secure, asynchronous messaging using Tor onion services. Pond servers would receive and store messages for users. Pond users interact with their *home* server for receiving messages, but interact with the recipient’s server in order to send messages.

Dissent [5] presents an approach based on DC-Nets [2], which allows members of a well-defined group to communicate anonymously in a way that provides accountability and resistance to denial-of-service and Sybil attacks.

8 Conclusion

There are a many open problems that need to be solved before metadata resistant tools are a suitable option for most people.

In this paper we presented Cwtch, a protocol for building metadata resistant applications based on Ricochet. The protocol solves a number of open problems with contemporary metadata resistant designs, providing asynchronous, anonymous multi-party communication through the use of untrusted, discardable infrastructure.

We have released an open source prototype of Cwtch which can be found at `https://cwtch.im/`.

References

[1] John Brooks. *Ricochet: Anonymous instant messaging for real privacy.* `https://ricochet.im`. Accessed: 2018-03-10.

[2] David Chaum. “The dining cryptographers problem: Unconditional sender and recipient untraceability”. In: *Journal of cryptology* 1.1 (1988), pp. 65–75.

[3] Raymond Cheng et al. *Talek: a Private Publish-Subscribe Protocol.* Tech. rep. Technical Report. University of Washington, 2016.

[4] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. "Riposte: An anonymous messaging system handling millions of users". In: *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE. 2015, pp. 321–338.

[5] Henry Corrigan-Gibbs and Bryan Ford. "Dissent: accountable anonymous group messaging". In: *Proceedings of the 17th ACM conference on Computer and communications security*. ACM. 2010, pp. 340–350.

[6] George Danezis. "Statistical disclosure attacks". In: *IFIP International Information Security Conference*. Springer. 2003, pp. 421–426.

[7] Daniel Demmler, Marco Holz, and Thomas Schneider. "OnionPIR: Effective Protection of Sensitive Metadata in Online Communication Networks". In: *International Conference on Applied Cryptography and Network Security*. Springer. 2017, pp. 599–619.

[8] Roger Dingledine, Nick Mathewson, and Paul Syverson. *Tor: The second-generation onion router*. Tech. rep. Naval Research Lab Washington DC, 2004.

[9] Ksenia Ermoshina, Harry Halpin, and Francesca Musiani. "Can Johnny Build a Protocol? Co-ordinating developer and user intentions for privacy-enhanced secure messaging protocols". In: *Proceedings of the 2nd European Workshop on Usable Security. Internet Society. Available at: https://pdfs.semanticscholar.org/41e4/f623d838ead1a782de46a94e44fb762cdd32.pdf*. 2017.

[10] Ksenia Ermoshina, Francesca Musiani, and Harry Halpin. "End-to-end encrypted messaging protocols: An overview". In: *International Conference on Internet Science*. Springer. 2016, pp. 244–254.

[11] Benjamin Greschbach, Gunnar Kreitz, and Sonja Buchegger. "The devil is in the metadataNew privacy challenges in Decentralised Online Social Networks". In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*. IEEE. 2012, pp. 333–339.

[12] Jaap-Henk Hoepman. "Privately (and unlinkably) exchanging messages using a public bulletin board". In: *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*. ACM. 2015, pp. 85–94.

[13] Johns Hopkins. *The Johns Hopkins Foreign Affairs Symposium Presents: The Price of Privacy: Re-Evaluating the NSA*. 2014. URL: https://www.youtube.com/watch?v=kV2HDM86XgI.

[14] Dogan Kedogan, Dakshi Agrawal, and Stefan Penz. "Limits of anonymity in open environments". In: *International Workshop on Information Hiding*. Springer. 2002, pp. 53–69.

[15] Adam Langley. *Pond*. https://github.com/agl/pond. Accessed: 2018-05-21.

[16] Stevens Le Blond et al. "I know where you are and what you are sharing: exploiting P2P communications to invade users' privacy". In: *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM. 2011, pp. 45–60.

[17] Jonathan Mayer, Patrick Mutchler, and John C Mitchell. "Evaluating the privacy properties of telephone metadata". In: *Proceedings of the National Academy of Sciences* 113.20 (2016), pp. 5536–5541.

[18] Karen Renaud, Melanie Volkamer, and Arne Renkema-Padmos. "Why doesnt Jane protect her privacy?" In: *International Symposium on Privacy Enhancing Technologies Symposium*. Springer. 2014, pp. 244–262.

[19] Christoph Rottermanner et al. "Privacy and data protection in smartphone messengers". In: *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services*. ACM. 2015, p. 83.

[20] Andrei Serjantov and Peter Sewell. “Passive attack analysis for connection-based anonymity systems”. In: *European Symposium on Research in Computer Security*. Springer. 2003, pp. 116–131.

[21] Nik Unger et al. “SoK: secure messaging”. In: *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE. 2015, pp. 232–249.

[22] Marc Waldman, Aviel D Rubin, and Lorrie Faith Cranor. “Publius: A Robust, Tamper-Evident Censorship-Resistant Web Publishing System”. In: *9th USENIX Security Symposium*. 2000, pp. 59–72.