Dark Crystal - Report

The Social and Technical Applications of Threshold Based Secret-Sharing in an Internet Freedom Context

Magma Collective

Contents

Acknowledgements

This work was supported by the Open Technology Fund and the National Democratic Institute. We are extremely grateful to both these organisations for making this possible. In particular we would like to thank Madeleine Nicoloff and Fiona Krakenbürger.

1 Introduction

This report is about threshold-based consensus mechanisms for groups, and their application in the context of internet freedom. It is about ways to use trust relationships to make collective decisions or actions in software.

By 'threshold' we mean a critically-sized subset of a group, often referred to as '*m* of *n*', where *n* is the size of the group, and *m* is the minimum number of members who must consent before a particular action is taken on behalf of the group. By 'mechanism' we mean some way of ensuring this rule is upheld. It could simply be social convention, or it could be a legal mechanism, but in the context of the internet we are talking about cryptographic techniques.

Traditionally, cryptographic applications have been centered around individual actors rather than groups, and have focussed on eliminating trust, rather than reinforcing it, with sentiments like "don't trust, verify". This has been made necessary as a result of the social landscape of the online world, where the absence of direct human contact makes it is very easy to deceive or impersonate others. This requirement to be individualist and untrusting is alienating for many people, particularly for those from cultures who highly value group membership and activities. But cryptographic applications can also be designed for groups or social networks, where trust-relationships play a complementary role to cryptographic techniques such as signing and encryption.

This report is based on our work developing Dark Crystal, a set of techniques and guidelines for secure management of cryptographic keys using trust in small groups. We aim to help application developers build tools where users rely on other users in a network of trust, as opposed to relying on a third-party service provider. Our primary project is the Dark Crystal Key Backup protocol, which allows 'social recovery' of sensitive data using threshold consensus. To assess the utility of this protocol, we have built two features for the mobile messaging app Briar, which is aimed at high-risk users with security concerns.

We discuss our protocol and the Briar features in the context of internet freedom in sections 2 and 3, and go on to look at further use-cases for the techniques we use in section 4. In particular, we focus on group governance mechanisms in collaborative software. We cover some cryptographic protocols for group collaboration and asses to what extent they allow the group to be managed in a secure and democratic way, and discuss some existing applications for group collaboration and governance.

We also propose a design for a new protocol for 'Key Re-issuance' based on threshold signatures which would complement our existing 'Key Backup' protocol.

2 The Dark Crystal Key Backup Protocol

Terminology

- *Secret Owner* - a person who has a sensitive piece of data they want to backup.
- *Shard* - a single share of the backup data, generated by a secret-sharing algorithm.
- *Custodian* - someone who holds a shard on behalf of the secret owner.

The Key Backup protocol [12] is designed for safeguarding data which we don't want to loose, but which we don't want others to be able to access. There is always a trade-off between these two needs. The more elaborate our methods of securing data are, the more easily we loose access ourselves. Our approach is to require the consent of multiple trusted parties in order to recover access. This is achieved using Shamir's Secret Sharing algorithm [24] together with some other cryptographic techniques to improve security and check integrity.

The goal is to be able to back up and recover some secret data. The secret owner chooses a set of custodians, who each receive a shard of the secret data. If the data is lost, the custodians can return their shards to the secret owner and the secret recovered.

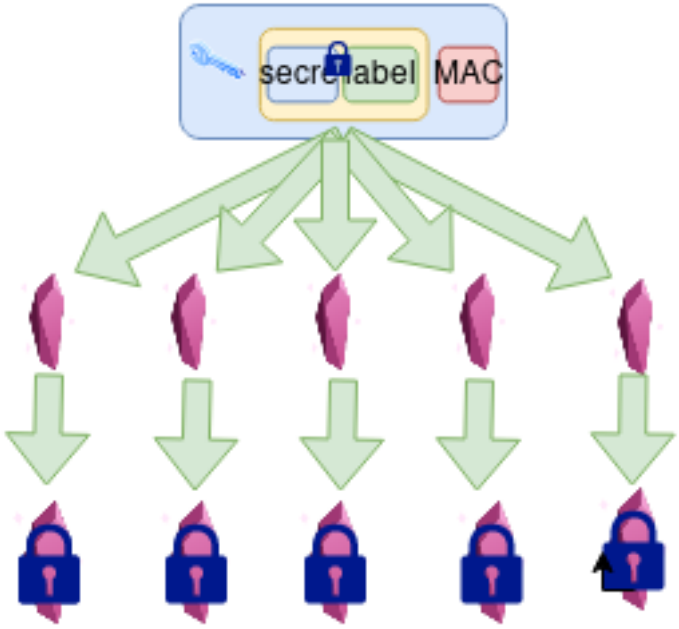

Figure 1: Diagram from the protocol specification showing the backup process. The encrypted secret together with it's key is distributed into shards. The shards are encrypted for each custodian.

The protocol attempts to address a fundamental usability issue with cryptographic applications: the '*Key Custody Problem*'.

Anyone who has experienced the frustration of forgetting the password for their GPG key will probably agree that having a whole system propped up by our ability to look after a single piece of secret data is rather precarious. The 'Global Encryption Trends Study 2018' [18], indicates that key management issues pose a major barrier to the adoption of encryption tools. Modern encryption techniques are strong, but the fear of loosing access to critical data mean they are often not adopted by those who need them.

Let's break down the Key Custody Problem. Most cryptographic systems rely on two assumptions:

1. You can access your secret key.
2. No-one other than you can access your secret key.

When these assumptions no longer hold, the system no longer works as intended, regardless of how secure the algorithms used are.

Assumption 1 is broken when a secret key is lost (which could mean inaccessible or permanently destroyed), and assumption 2 is broken when a secret key is compromised.

The Key Backup protocol addresses assumption 1, key loss, but not assumption 2. We can further break down the key loss scenario into three categories:

- ***'Swim'*** loss without compromise. For example, the device with the secret key falls deep into the sea and we assume it to be permanently inaccessible to anybody.
- ***'Theft'*** loss with compromise - both assumptions 1 and 2 are broken. For example, the device with the secret key was stolen, or we lost it in a busy public place and assume that someone else has it.
- ***'Inheritance'*** a situation where we actually want our key to be compromised following loss. Generally, this means we died or have lost the capacity to do the things our secret key allows us to do, and would like the key to be recovered by heirs.

The key backup protocol addresses 'swim' and 'inheritance', but it only partially helps us with 'theft'. The protocol does not offer any way to resolve the problems associated with a key being compromised.

For the theft scenario, its important to consider what our secret key is used for. If an encryption key is both lost and compromised, it is still important to be able to recover the key, since we would re-gain access to any data encrypted with that key. But if a signing secret key is both lost and compromised, we gain very little in recovering it. We can continue to use it to sign messages, but nobody can be sure whether it was really us who signed. We will discuss a possible solution to this problem later.

2.1 Is local storage safer?

The idea of the key backup protocol is to offer a distributed backup of a key held locally, so that we are able to recover that local copy in case we loose it. This recovery mechanism makes it more practical to

use the ‘local first’ model, where data is held on an individual’s device, giving the user ultimate control over who can access it. Local-first application design is often touted as solving the problems associated with centralised data storage, such as data extraction or abuse, and a lack of control for users.

However, the report ‘The Limits to Digital Consent’ [23] finds that the local-first model is ‘not inherently safer for people or communities’ because ‘the risks of data accumulation are placed on the individual’. When data is held in a centralised system, the organisation hosting it may have better resources to protect that data than the individual themselves. For example, they may have legal expertise, better access to legal representation, or employ better digital security techniques than the user.

So the risks associated with local-first software are not only about data loss, but about being a target for coercion into giving up that data. This might include legal action, physical threats, or exploiting security vulnerabilities. Therefore, storing the data on a centralised server with a well-chosen organisation is often safer than storing it on your own device.

Indeed, several applications for secure evidence gathering for high-risk users offer this model. The user uploads their evidence to a server run in a different jurisdiction by an organisation with good access to legal resources. The local copy can then be deleted for the safety of the user.

So in some cases, whats needed is not re-designing our software architecture for the needs of high risk users, but forming better organisations to provide services in the traditional client-server model. That said, internet traffic to such organisations can be blocked or monitored. So as always, this is a solution that fits some cases but not others.

Our key backup protocol doesn’t need to be used for a backup copy, it can also be used to distribute the data whilst the original is intentionally deleted, just as users of the secure evidence gathering app delete their copy after uploading it to a trusted server. The ‘remote wipe’ feature described below is an example of this. Furthermore, the ‘trusted contacts’ for the protocol do not need to be individuals that the user knows personally. They can also be organisations with good access to resources as we have described here.

2.2 Relevance to high risk users

The Key Backup protocol addresses the needs of high risk users in two ways. Firstly, by providing a way of recovering cryptographic keys which are lost accidentally, it makes applications which use client-side encryption easier to use, and more likely to be adopted. Secondly, this recovery mechanism makes it possible to intentionally delete sensitive data in a high-risk situation, such as when crossing a checkpoint or border, or at a demonstration. Our two features for Briar are intended to give practical examples of how the protocol can address these needs.

Our Threat Model report [13] discusses the security of the approach for various possible adversaries.

2.3 Technical overview of the protocol

As described in the protocol specification document [12], there are several enhancements to the basic Shamir's Secret Sharing algorithm to improve security.

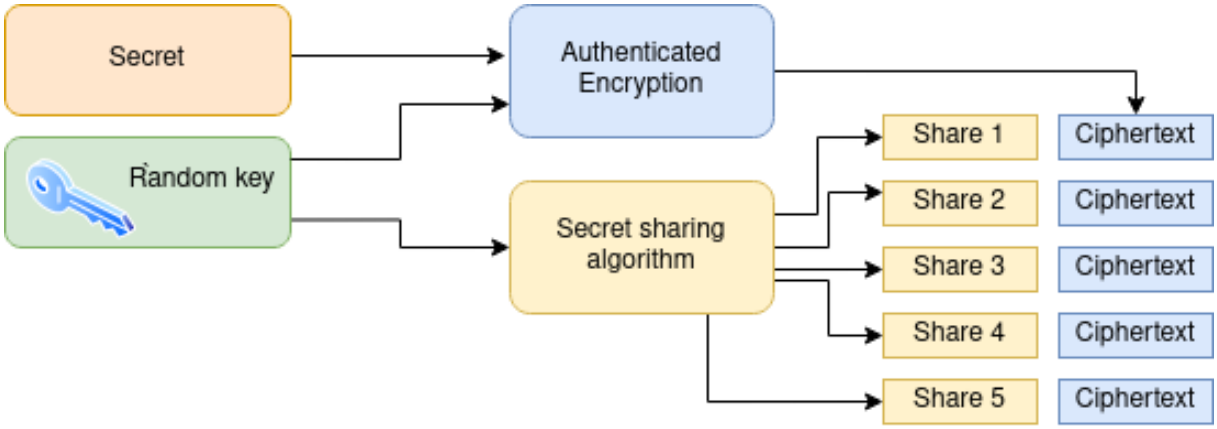

Figure 2: The input to the secret sharing algorithm is not the secret, but the key used to encrypt the secret

Perhaps the most important one is adding authenticated encryption of the secret, and using the encryption key as input to the secret sharing. This guarantees a uniformly random secret, and enables us to programmatically determine whether or not recovery was successful.

Other features aim to reduce the amount of metadata given to custodians to the absolute minimum necessary. The share-index is obfuscated, meaning custodians cannot know how many other custodians there are, and optionally the secret is padded with zero-bytes to obfuscate its length.

The secret owner adds a signature to each share. This is to protect against shards being modified, maliciously or accidentally, and achieves the same goal as schemes known as 'Verifiable Secret Sharing'.

The protocol version number and a time-stamp are added to the shard message, and each shard is encrypted using the public key of it's destined custodian. The shards are then sent to each custodian, and they are stored on the custodian devices. The security of this step is dependent on the transport protocol used to send the shards.

2.3.1 Consent for being a custodian

The issue of gaining consent for taking custody of a shard, is complex and has no perfect solution. The key backup protocol does not recommend a particular way of implementing consent, but offers three possible models for developers to choose from based on the nature of their application.

With the ‘weak consent’ model, the shard is sent right away, but on receiving it the custodian is asked for their consent. If they refuse then the shard is deleted locally and a message sent to the secret-owner to inform them. This has the advantage that in the ‘happy path’ (when everything goes well), the backup is made very quickly, and implementation on the side of the secret-owner is fairly simple. The disadvantage is that if a custodian rejects, the secret owner needs to send a new share set to the others, and things get messy if some custodians try to return shards from the original set. Worse still, the secret owner can never be sure the share was really deleted, which presents a security issue. Furthermore, until the custodian reads and responds to the request notification, the shard exists in the application’s ‘inbox’ on their device such that they have in effect unwillingly accepted (at least temporary) custody.

With ‘strong consent’, the secret owner sends no shards until they have received an acceptance message from all custodians. This is better for both security and for offering genuine ‘opt-in’ consent to the custodians. However, until all these acceptance messages are receive there is no backup in place whatsoever, and in practice we have found it often takes a long time to get responses for everyone, particularly with apps where not everyone is always connected or signed in. It also presents a usability issue to the secret owner, as they will not immediately see confirmation that the backup is set up.

The other model is simply ‘no consent’. Depending on the nature of the secret and the usability and security trade-offs of the other models in some cases we would recommend no explicit consent mechanism. Of course users are still able to ask potential custodians themselves before making a backup.

The different consent models are described further in the Threat Model report [13].

3 Use-cases implemented for this project

As part of this project we built two features for Briar. This was a case-project to asses the utility of the protocol in the context of high risk users. The process and outcomes are discussed in detail in our project case report [11].

3.1 Introduction to Briar

Briar is a mobile messaging app with a focus on security [7]. The interface is similar to many popular messaging apps, but it works in a very different way in order to meet the needs of high risk users.

Most mobile messaging applications consist of two pieces of software, an app which runs on the user's device and a program which runs on a server hosted by the company who provides the app. This server-side program coordinates the sending of messages between users and usually stores messages so that messages are always delivered as soon as possible regardless of the user's connectivity. Usually client-side encryption is used, meaning the server operators are unable to access the message content. However, they can access message metadata such as who sent a message to who, when it was sent, how big the message was. They may also revoke the service, willingly or unwillingly, or have traffic to their servers blocked or monitored. For high risk users these are serious issues.

Figure 3: Screenshot of Briar

Briar consists of only a client side application. There is no Briar server running somewhere. There is no way for the developers of Briar to know how many people are using the application or who they are. Messages are sent either over the anonymity network, Tor, or directly over the local network or bluetooth connection if possible.

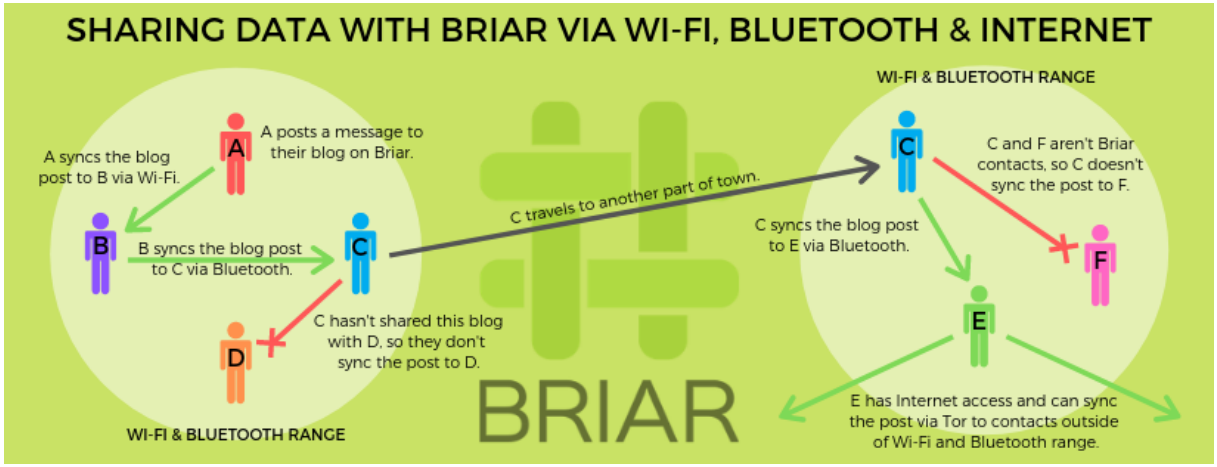

Figure 4: Diagram from Briar's website showing how connections can be made

Furthermore Briar authenticates users using cryptography alone. Unlike some other messaging apps, a Briar account is never linked to a phone number. This is beneficial to high risk users as phone numbers can often be linked to other personal information.

These features give a great security benefit, but do introduce some usability issues. All responsibility for the Briar account is left to the user. There is no way to recover the account if the password is forgotten or the device is lost.

3.2 User testing

As part of the case project we conducted two rounds of user testing sessions. The first of which was organised by ourselves and the second by the National Democratic Institute and was hosted by their partner project, Southeast Asia Freedom of Expression Network (SAFEnet) with participants of one of their security training events. This gave us the opportunity to get feedback from people who work on political campaigns or journalism who might have a genuine need for these features. For the session with SAFEnet, the app was translated to Indonesian and the session was conducted with live interpretation. This also gave us an insight into internationalisation issues around the concepts and terminology we use in the features.

3.3 Social backup

3.3.1 Explanation

The social backup feature allows users to backup their account by distributing an encrypted backup amongst a small group of trusted contacts. The user selects a set of their Briar contacts, and then

chooses a threshold value - the minimum amount of contacts needed to recover the account. Shards of their account backup are then distributed to these contacts using Dark Crystal's Key Backup protocol.

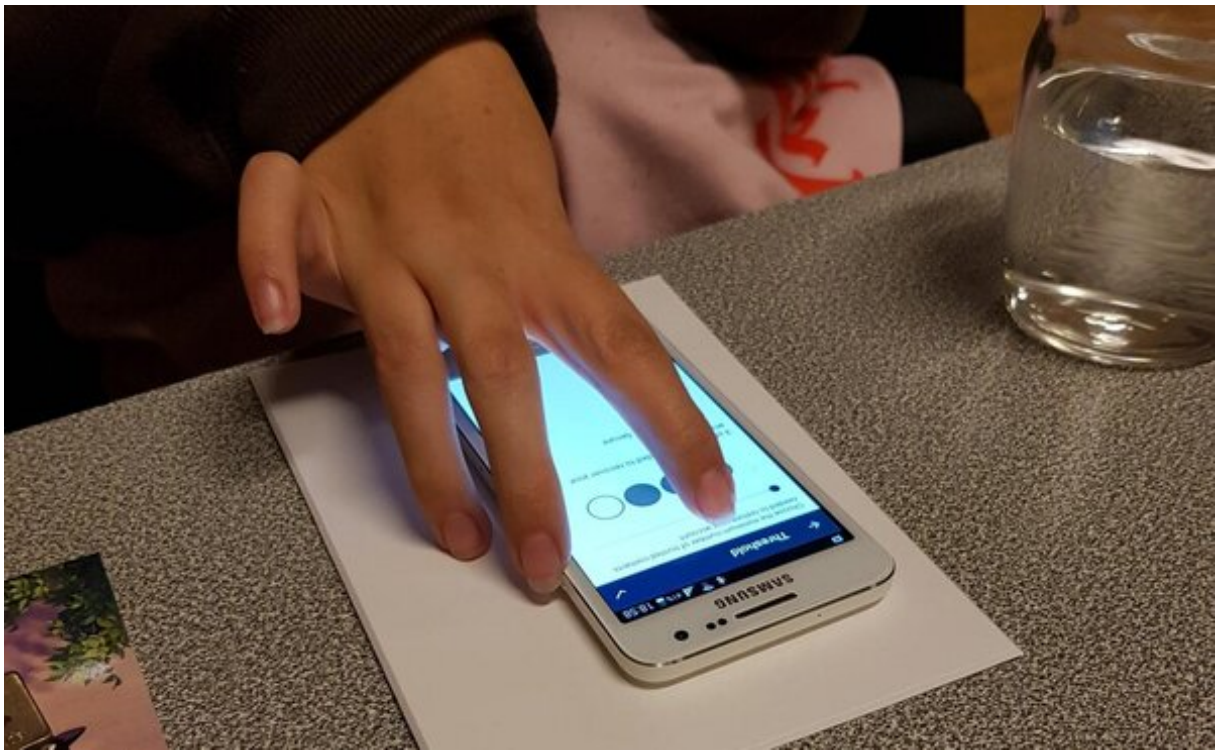

Figure 5: Selecting a threshold value

When the Briar app is opened for the first time, there is an additional screen where the user may choose whether to start a new account or recover their backup.

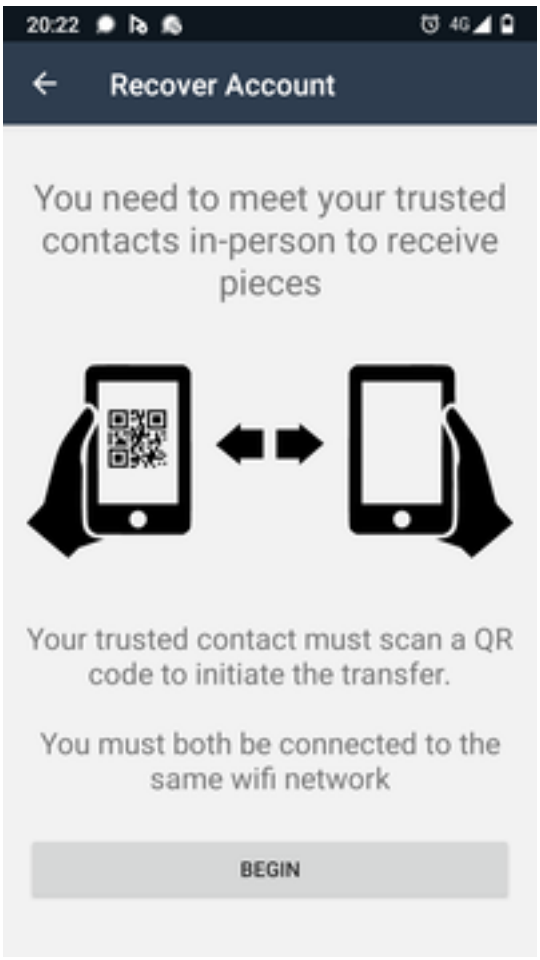

Figure 6: Explainer screen shown to the secret owner when receiving a shard

When recovering, the secret-owner must physically meet with their custodians to transfer the backup shards to their new device. Shards are transferred using a protocol similar to Briar's 'add contact nearby' feature, which involves scanning a QR code as well as transferring the encrypted data using a local Wifi connection.

On successfully retrieving the critical amount of shards, the user is prompted to choose a new password and the account is recovered.

3.3.2 How is the feature relevant to high-risk users?

Like many other security tools, Briar's security comes at a cost to usability. Forgetting the password or loosing the device means being permanently locked-out of your account. It is difficulties like this which make users less likely to adopt security tools. Social backup aims to improve usability.

Furthermore, the presence of a backup feature makes in possible to for users to intentionally delete their Briar data in a high-risk situation, with the possibility to recover it later, but without carrying any of the backup data. This can be done in situations such as at a demonstration or when crossing a border or checkpoint.

3.3.3 What data is backed up?

When we talk about backing up the 'Briar account', we mean backing up the keys which make the account 'yours', meaning your contacts will be able to continue to contact you.

But there are other kinds of data stored by the Briar application which peers might not want to loose, for example their contacts list, their messages, and their membership of particular groups, forums and blogs. It should be noted that the actual content of forums or blogs themselves does not need to be backed up, since when existing content can be retrieved from other peers when re-joining.

In deciding what to backup, there was a trade-off between information peers are most concerned about loosing, and what information they are most concerned about an attacker having access to should our backup become compromised.

Our initial research showed high-risk users were most concerned about loosing their contacts list, as it can often be very impractical or dangerous to regain contact by other means.

Message content however, seemed less appropriate to back up, as it is likely to contain sensitive details, and the other party of the conversation generally still has access to it. That is, re-gaining contact with someone gives you indirect access to your previous conversation with them, and all you really gain by including the messages in the backup is the convenience of having them displayed on your device without needing to ask.

Besides the clear disadvantage with regards to security, backing up content also has the disadvantage of being arbitrarily big. Since we are asking others to hold the backup on their own devices, it becomes less appealing for them the bigger the backup gets.

So we decided the backup should contain the account keys and the contacts list, as well as one other thing which we will discuss next.

3.3.4 Mutual interdependence of social backups

Adding the contacts list meant that we needed to be able to dynamically update the backup payload, because it changes over time as more contacts are added.

This required us to make an important change we made to the Dark Crystal protocol when integrating it to Briar - the shards and encrypted backups could not be treated as a single message.

So we decided to have one-time shard messages, which were shards of the key to decrypt a ‘backup’ message, and many ‘backup’ messages which are incrementally updated versions of the encrypted backup itself. When a backup message is received, its version number is checked. If it is greater than the version number of the backup currently held, the old backup is discarded and replaced with the new one.

Putting this system in place gave us the opportunity to easily add other kinds of data to the backup at a later stage. Most significantly, we add the shards one receives **from other people** to our own distributed backup.

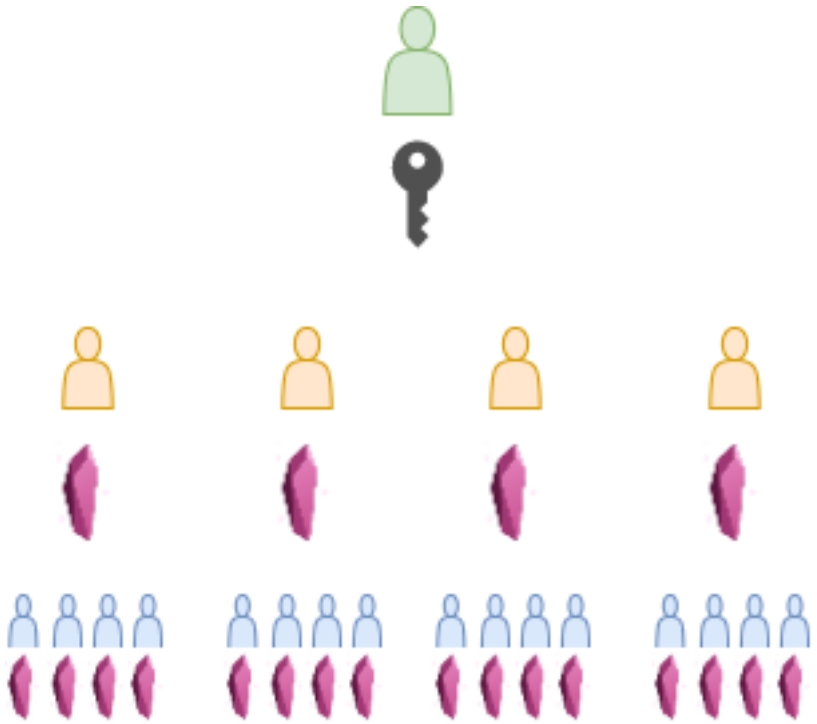

Figure 7: Recursive ‘shards of shards’ - a shard is a piece of an account backup, and an account backup itself contains shards.

So when you get a shard from someones else’s social backup, if you have a social backup yourself you add the shard to that backup, and send out the new version to your custodians. When you recover your account, you are still holding the shard. This creates mutual interdependence and makes this backup system very robust. Already we have the threshold mechanism which means we can recover backups even if some proportion of the custodians have lost their shards, and now, even if we cannot meet that threshold, there is the possibility to recover the missing shards from the custodians of the custodians.

3.3.5 Insights from user testing

3.3.5.1 Confusion around secret sharing

Users found the social backup concepts confusing. Many assumed that choosing more custodians would make the backup less secure, as the data would be in more places. But actually the opposite is true - the more custodians, the more places an adversary needs to go in order to recover the secret. In response to this we added an additional explainer screen. But the concept is undeniably a little unusual and unintuitive.

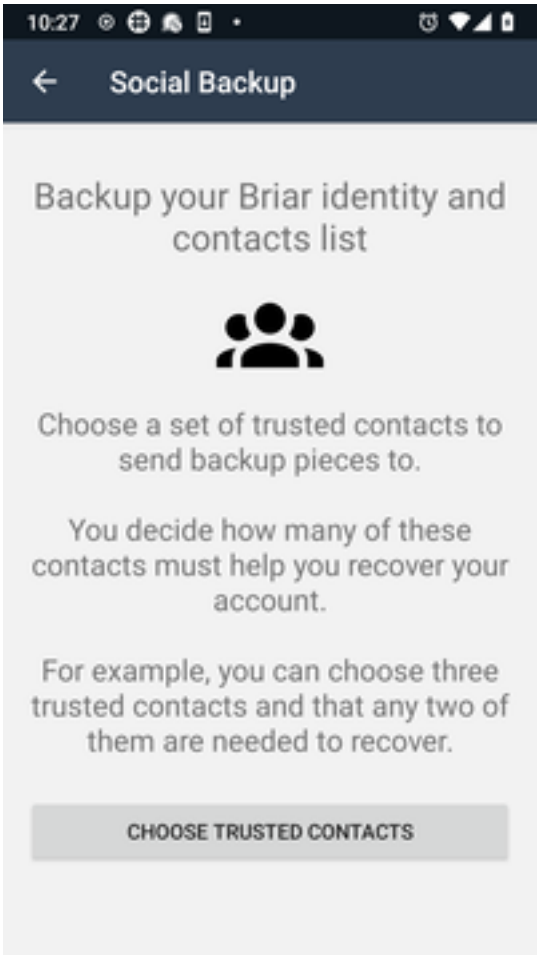

Figure 8: Screenshot showing an explainer screen for the Social Backup feature

Choosing the threshold was also a cause of confusion. We considered removing the option altogether, and simply setting a recommended threshold value. But it is anyway important that the secret owner understands how many contacts are needed in order to recover. We added some visual feedback to

the threshold screen to make this clear.

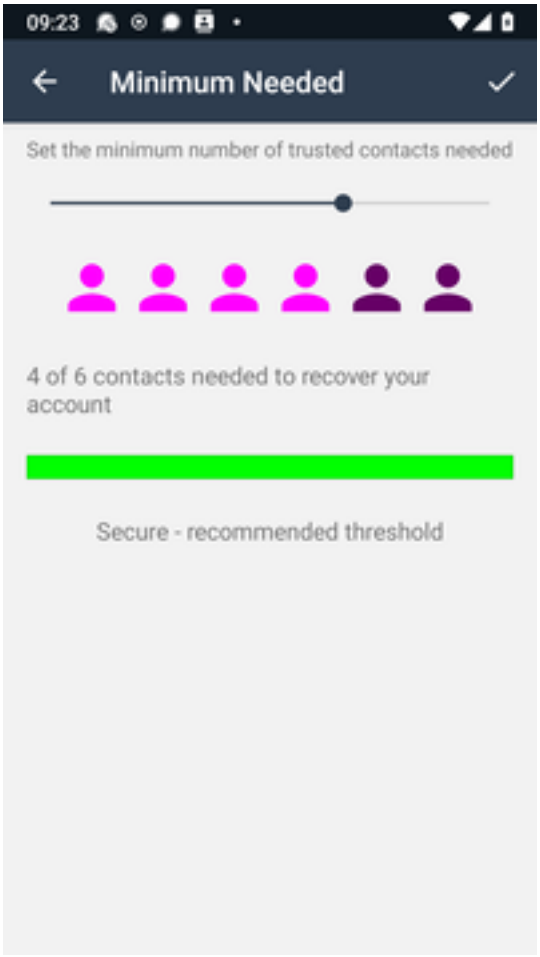

Figure 9: Screenshot showing setting the threshold for Social Backup

3.3.5.2 Consent for custodians

One participant said they expected to be asked whether they wanted to accept a backup piece, and in was noted that this is a desired feature. As discussed above, this is something we have given much consideration to when developing the key backup protocol, and it is a complex issue with no perfect solution.

3.3.5.3 Recovery option when starting the app

We had added an additional screen when the Briar app is started for the first time, which asks the user

if they want to create a new account or restore an account from backup. This caused confusion right at the beginning of the process as participants were not yet aware of what is involved in restoring an account. In most cases, the user will want to create a new account, so we decided to make this the default path and make the 'restore account' option less prominent.

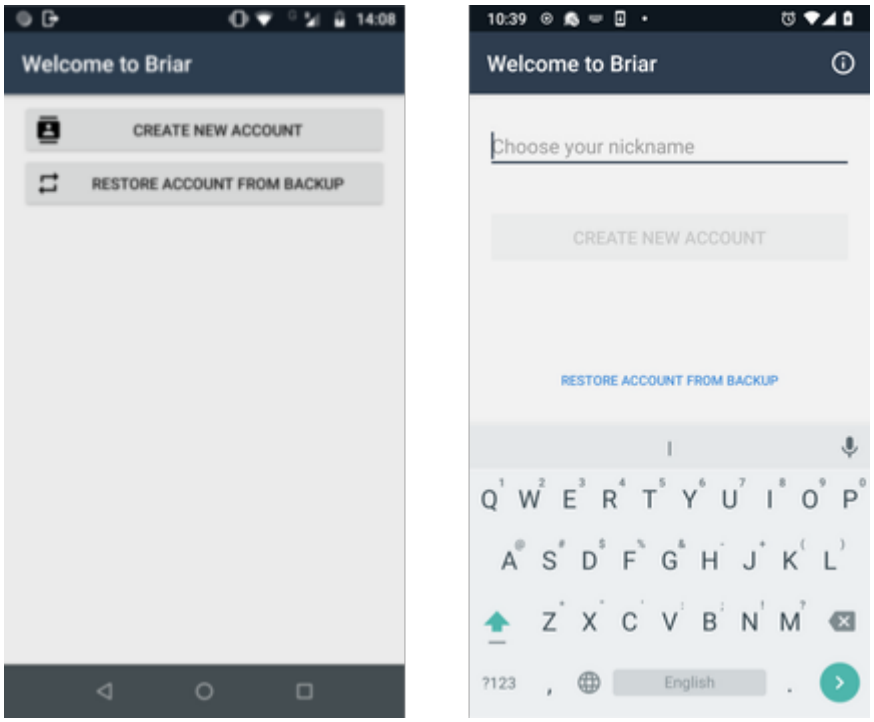

Figure 10: Welcome screen before and after improvements

3.4 Remote Wipe

3.4.1 Terminology

- *Wipee* - a person who appoints others to be able to remotely wipe their account.
- *Wipers* - trusted contacts who may remotely activate a wipe of the *wipee's* account.

3.4.2 Explanation

'Remote Wipe' allows a user to appoint trusted contacts with the ability to remotely activate the deletion of all Briar's application data. It is designed for situations where the user's device is assumed to be compromised by an adversary, for example when the user is arrested or captured. It works with a similar threshold principle to social backup - the user chooses a set of trusted contacts, and a critically-sized subset of these contacts may activate a remote wipe.

However, this feature does not use a secret-sharing algorithm. The ‘wipe’ signals do not contain any secret data, they are simply a particular type of message, and the user is able to verify whether it came from one of the trusted contacts using Briar’s transport protocol.

Figure 11: The explainer screen displayed to wipers when sending a wipe signal

3.4.3 How is the feature relevant to high-risk users?

Remote wipe is designed for situations where a user’s device is seized by an adversary. This could involve capture, arrest, or a raid of their home or workplace. Importantly, the feature is only useful if the appointed wipers are made aware of the situation before the attacker accesses the device.

Another restriction is that the remote-wipe signals can only be received when the device is signed-in to Briar, and has network connectivity. If they are sent when the device is not signed into Briar, the wipe will be activated as soon as both the sign-in password is entered and the device has network

connectivity. So it is possible that the attacker could hinder a remote wipe but putting the device into flight mode.

3.4.4 How social backup and remote wipe complement each other

The ‘Social Backup’ and ‘Remote Wipe’ features have some big technical differences, but they both essentially use the same principle - of assigning a special ability to some critically large subset of a trusted support group. The two features complement each other because social backup can be used to restore the account following a remote wipe. This takes the ‘dangerousness’ out of remote wipe, making the ‘wipers’ more ready to use it. This might mean the use it even when there is only a suspicion that the wipee’s device has been compromised, which makes the feature more powerful.

When we were implementing the user interface for Remote Wipe we realised how really similar the two features are in terms of user experience, and asked ourselves if it would actually make more sense to ‘roll the two features into one’ at the user interface level, making custodians for social backup and wipers be automatically the one and same thing.

This would reduce the need to set up both features, saving time and cognitive load. It also addresses the problem that a user might not feel the need to setup a remote wipe until its too late - making it implicit in setting up a social backup means it is automatically there if needed. But we need to assess whether there are some situations where you would want particular contacts to be custodians of your social backup, but not able to activate a remote wipe, or vice-versa.

3.4.5 Insights from user testing

Generally we found much less usability issues with Remote Wipe than with Social Backup. Participants understood the concepts well and had little difficulties when using the feature for the first time. This may have been partly due to our decision to use a fixed threshold of two, keeping the setup process simple.

3.4.5.1 Adding confirmation on for a wipe being activated

Participants who took the role of the wipers noted that they would expect to see some feedback as to whether a wipe had actually been activated, as otherwise they would worry that it had not. To make this clear, we added an additional ‘wiped’ message type, which is sent to all wipers just before a wipe is activated and displayed in the conversation view.

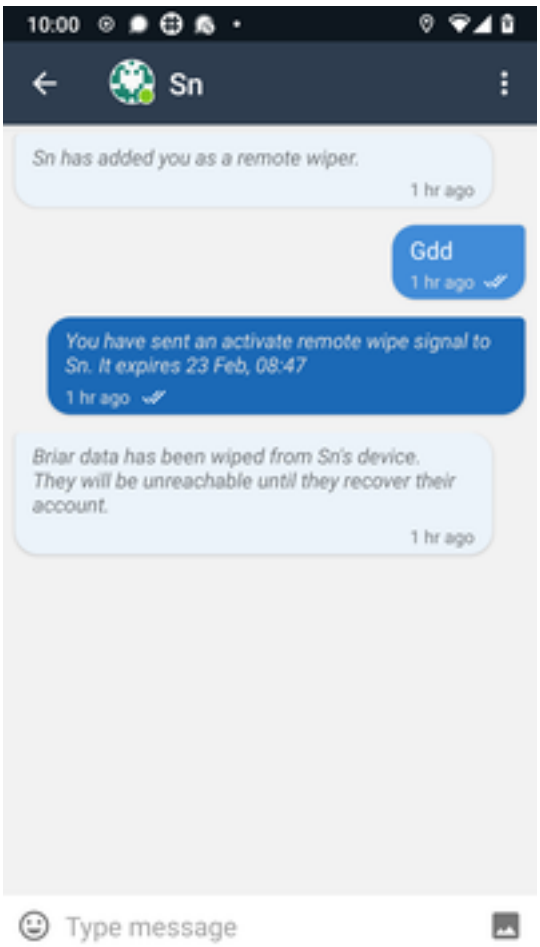

Figure 12: Screenshot showing the confirmation message that Briar data has been wiped from the remote contact

3.4.5.2 Merging the Social Backup and Remote Wipe features

When considering the possibility of merging the two features together, so that one set of trusted contacts are used for both features, there were several insights from feedback and discussion during the testing sessions which we discuss below. We eventually decided not to merge the two features based on these findings.

3.4.5.2.1 What happens if members of the support group have their devices compromised?

Generally this is less dangerous for the Remote Wipe 'wipers' than for the Social Backup 'custodians'. Some participants said that for Remote Wipe they would choose people actively involved in their

work, as being able to respond quickly outweighs the risk that these contacts might be targeted by an adversary themselves. For social backup they would rather choose people in a stable situation who were less likely to be targeted. So the roles are actually quite different, which speaks against merging the two features.

3.4.5.2.2 Are different levels of trust needed for Social Backup and Remote Wipe?

This depends on the circumstances, but generally participants said that if Remote Wipe is only set up when a backup is in place, Social Backup requires more trust than Remote Wipe.

3.4.5.2.3 Are different thresholds appropriate for Social Backup / Remote Wipe?

With remote wipe, a rapid response is important so a low threshold (two) is recommended, even with a large set of wipers. With social backup, it makes sense to have the threshold proportional to the number of custodians, and generally great than two. We generally recommend 75%, rounding down - for example 3 of 5. But having different thresholds does not mean we cannot use the same support group for each.

3.4.5.2.4 What are the implications of support group members knowing who each other are?

With Remote Wipe, it is important that the wipers know who each other are, as they may need to inform each other of the situation, or discuss whether activating a wipe is appropriate if they are unsure. With Social Backup there is a great security advantage in custodians not knowing who each other are, as an adversary cannot coerce them into giving up the identities of the others. This is another argument to not merge the two support groups.

3.4.5.2.5 Are there some cases where you would want somebody to be a backup custodian but not a remote wiper, or vice-versa?

Participants said that for social backup they would choose people in a ‘stable situation’ such as older relatives, who were unlikely to loose their device, and that for remote wipe they would rather choose people with whom they have most regular contact and who are most involved in their activities, as they were likely to know sooner when they are in danger.

4 Further use cases (beyond social backup and remote wipe)

4.1 Group governance

Threshold-based consensus mechanisms have the potential to be used for group governance or group management. Making group decisions or actions in this way is nothing new. Some legal structures for organisations require a majority vote for a certain decision to be made, or a group bank account might require at least two members to sign a cheque before the bank will accept it as valid. Importantly, these mechanisms are not supposed to replace the process of discussing the decision as a group and coming to a common agreement - they are additional measures. That is to say, just because only two members are needed to sign a cheque does not mean they don't need to consult the rest of the group or follow the group's guidelines - we use a combination of both social conventions and rules imposed 'mechanically'.

The popularity of the internet has meant that software is becoming increasingly collaborative. Groups using collaborative software often have actions that need to be made on behalf of the group. These might include allowing a particular member to join the group, blocking particular members or content, and publishing content on behalf of the group.

Much of the existing software which allows these kinds of collaborative actions is based on the client-server model. That is, the logic behind these decisions about user actions and privileges runs on a program on the server, and anyone who has administrator access to the server can essentially override these rules. Using cryptographic techniques, it is possible to implement these sorts of group governance mechanisms in such a way that this logic is implemented on the client (meaning each user's device) and that in order to participate one must run a client which abides to these rules.

This has the advantage of being more genuinely democratic, there are no 'superusers' or system administrators with special powers. Furthermore, the system cannot be undermined by an adversary who gains access to the server. But there are also some disadvantages - they are more complicated to implement, and there are always some edge-cases or particular circumstances where we might not want the rules to be applied.

4.1.1 Relevance to Internet Freedom

Methods of group governance are essential for communities or political groups to organise. As more and more group activities take place online, it is important that the collaboration software used offers a democratic way of managing the permissions and rights of group members. Often, ultimate control over the services used is implicitly given to the members with more technical abilities, or those with administrative access to the server which runs the software. These issues become more even prominent

for remote teams who have limited opportunity to gain an understanding of each others' viewpoints and feelings.

Furthermore, cryptographic techniques enable group governance systems which use peer-to-peer architectures. Such architectures help mitigate issues with privacy and censorship, as there are no servers which can be compromised or shut down. This makes the peer-to-peer model an attractive option for high-risk users. Unfortunately, developing peer-to-peer software is usually more difficult as so much of the existing software paradigms and protocols are based around the client-server model.

4.1.2 Cryptographic schemes for groups

4.1.2.1 Group threshold signatures

Group signatures, as described by Chaum in 1991 [8], allow a group member to make a signature on behalf of a group. A verifier with the group public key can determine whether a message was signed by a member of the group but not which member they are.

Threshold-based group signature schemes extend this notion to require consensus of a specific quorum of group members in order to produce a valid signature.

As well as having applications in electronic voting, group signatures can be used to make assertions on behalf of a group, for example a public statement which can be verified by anybody, or some internal group message representing an agreed action. Threshold group signatures make this practical as there is a specific degree of member agreement required to create a valid signature.

Some group encryption schemes rely on a trusted 'dealer' to coordinate the scheme, which is typically a program running on a server to which all group members have access. Although this often makes the schemes simpler, this is something we want to avoid when choosing a scheme as it introduces a security weakness.

4.1.2.1.1 Group Signatures or multiple signatures?

Before we look at schemes for group threshold signatures, we should ask ourselves whether we really need them as there is a very simple way of solving the same problem - by using multiple signatures:

- Take the 'group public key' to be the concatenation of all member's public keys, and the threshold value.
- All members who wish to sign a message do so, and publish their signature.
- Count how many of these signatures we are able to validate using the public keys contained in the 'group public key', allowing only one signature per public key.

- If we have met the threshold, we have consensus for this message.

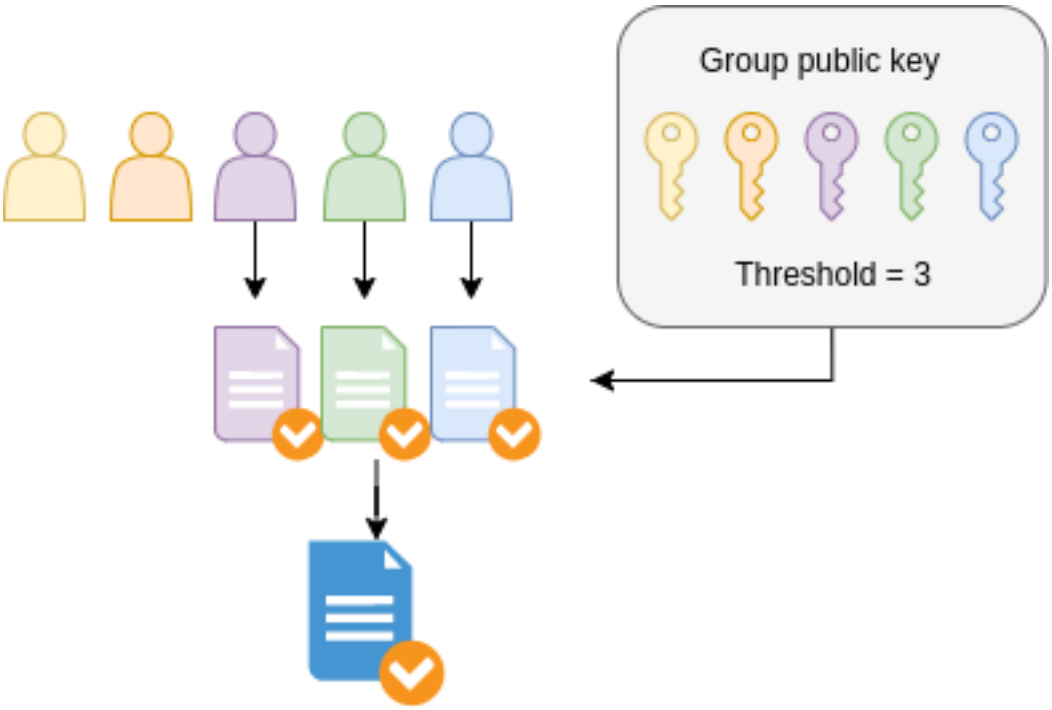

Figure 13: Multiple signatures: There is no aggregation of keys or signatures, we simply check if a set of signatures validate with a set of public keys

This system works, and it has some advantages: it is simple, it functions with any existing signature scheme and members can use the same keypair for signing messages as an individual or as a group member. But it does not keep private the identities of group members, or of who signed a particular message. In some cases this is a problem, but in some cases we actually prefer to have transparency. It is like voting on a decision by raising your hand in a group meeting. The lack of anonymity might be a problem in some cases, as we are afraid to reveal our position to the rest of the group. But in other cases we prefer that our voice can be heard as it helps us get an understanding of each others opinions.

Much of the recent work on threshold group signature schemes cite blockchain applications as a primary use case. These are applications where there is an incentive to minimise how much data is published, and privacy of group members is important because blockchains are inherently publicly available. We need to keep in mind that the use-case for internal group governance is very different.

So we need to make a distinction between two kinds of use-cases. For internal group actions where anonymity is not important and published signatures are only accessible by group members, we can use multiple signatures. For anything involving open publication, if we want to hide the identities of

the individual signers, we should use a group threshold signature scheme.

4.1.2.1.2 Boneh-Lynn-Shacham

The Boneh-Lynn-Shacham (BLS) signature scheme is an elliptic curve scheme [5]. It was developed for its short signatures, but also has some desirable properties for building protocols. There are a number of operations which can be done on a private key, public key, signature tuple which produce another valid tuple. For example, a set of public keys can be 'added' to produce an aggregate public key [4]. If the corresponding signatures for a single message are also aggregated, we have an aggregate signature which can be validated with the aggregate public key.

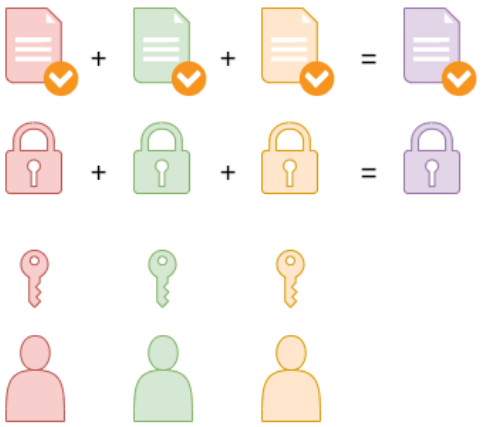

Figure 14: BLS Signature aggregation - if public keys and signatures are added together we get an aggregate signature with can be validated with the corresponding aggregate public key

Besides this 'addition' operation for aggregating, we can use Shamir's shares of a private key as individual private keys, sign a message with them, and 'combine' the signatures using Shamir's polynomial interpolation to get a group threshold signature which is valid for the corresponding 'combined' public key. Using distributed key generation, it is possible for a group to make such signatures without any member ever knowing the group private key.

Figure 15: BLS Threshold signature - Here, 2 of 3 members' signatures are combined to give a group signature. The group public key is the same regardless of which members signed

4.1.2.1.3 Distributed key generation and threshold signatures for ECDSA

BLS has some great properties, but in some cases we might be designing a system where we are restricted to using a particular signature scheme, such as Ed25519. There are a number of schemes which offer more generalisable methods of doing distributed key generation and threshold signatures [1], although often they involve many rounds of communication making them complicated to implement. Gennaro and Goldfeder's scheme [14] uses Paillier homomorphic encryption to allow members to add their share contributions together without revealing them to each other.

4.1.2.2 Group encryption

Another tool we have for group governance using cryptography, is to be able to specify who is able to access a particular message or piece of data. Very often we want to make some data only available to members of a group, and in order to do that we need agreement about who those members are. We need to decide carefully which mechanism we use for this, as being able to make decisions about who is or isn't a group member is fundamentally important to the existence of the group. We will look at some group encryption schemes and assess to what extent they require consensus of group members.

Group encryption is a hard problem in cryptography, particularly for dynamic groups where members join and leave. Many schemes require coordination by a trusted server, which is something we ideally want to avoid.

First lets think of the most simple way to do group encryption. We can encrypt a message separately for each group member, and sent it to them. This means the author of each message can decide who the group members are every time they write a message. It could be argued this does not really constitute a group at all. Depending on how messages are stored and sent, this might also mean some duplication of data, as the same message is encrypted many times. This duplication problem can be mitigated by encrypting the message once using a single-use symmetric key, and encrypting the key for each group member.

Another simple scheme is to use a symmetric key for all group messages. That is, anyone we give the group encryption key to is effectively a group member and can read the group messages. This is good because it requires means we always address all group members, we cannot exclude anyone. But it also means any member may distribute the key to someone else (willingly or unwillingly), and it is impossible to know how many individuals have the group key. The only way access can be revoked for future messages is by agreeing a new group key, and re-distributing it to the desired set of members. This scheme is only secure when used together with a public-key encryption scheme in order to distribute the group key.

Diffie-Hellman key exchange, which is the most common way to establish keys for encrypted communication between two individuals, can be extended for multi-party communication. It should be noted that we cannot mix the two. That is, is Alice, Bob and Carol may securely communicate as a group, but the key-exchange process involves revealing the shared secrets between any two of them, so they cannot use the same keypair for secure communication between only Alice and Bob. (add reference)

4.1.2.2.1 Tree-based Group Diffie-Hellman

Tree structures have been proposed to agree a group key composed of several members [26]. Each leaf of the tree is a member with a personal encryption keypair. They do a Diffie-Hellman key exchange with another member to get the secret key for their parent ‘node’ in the tree. They then compute the associated public key for this node, which is often referred to as ‘blinding’ the key so that it can be safely distributed to other members. Once enough ‘blinded’ node keys have been distributed, every member can compute a common group secret key. They must only know their own secret key, as well as the relevant ‘blinded’ public keys on their path to the root of the tree.

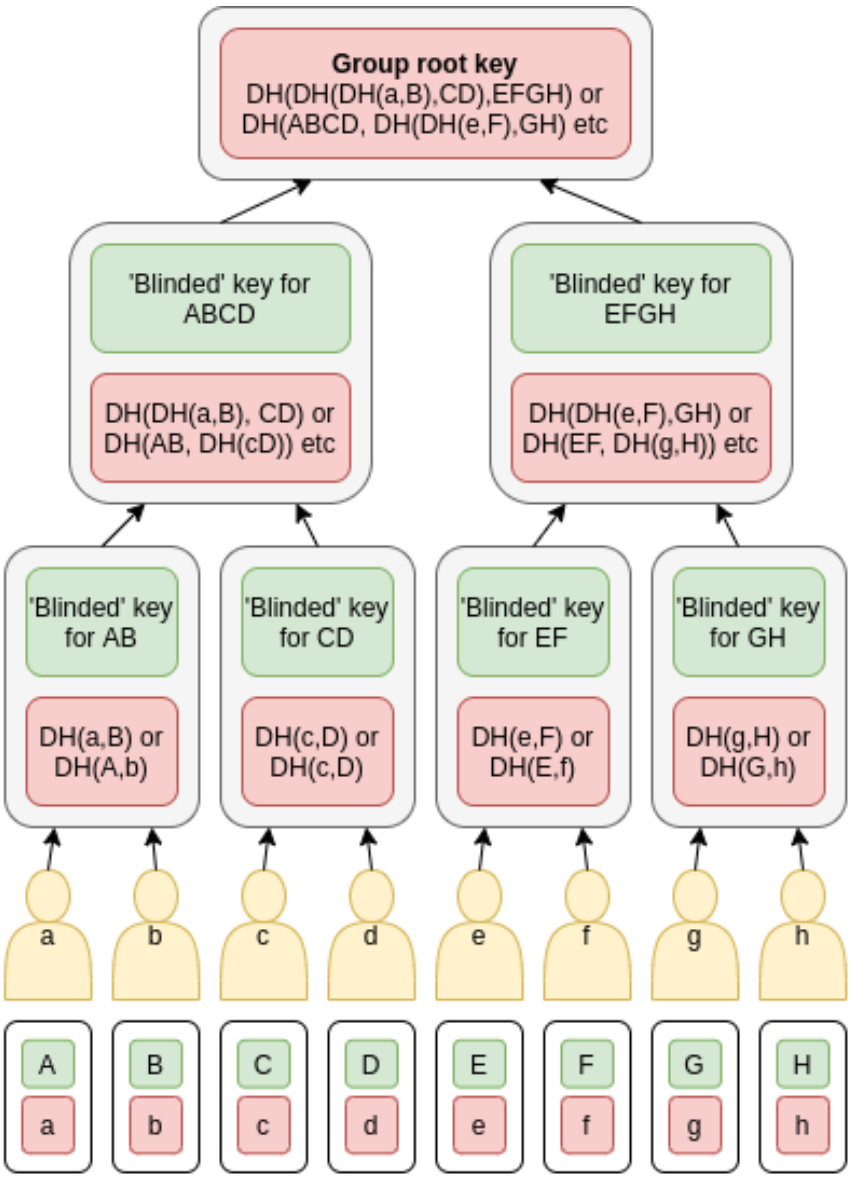

Figure 16: A key tree. Secrets are shown in red boxes, and only obtainable by members below them in the tree. The associated 'blinded' keys (green) are distributed to other group members such that all members can compute the root key

With this system, computing the group key requires consensus on who the members are. That is, we can only participate in the group (by reading or writing encrypted messages) if we agree on the set of members. A group member joining or leaving means a new group key will be computed, so if any member is unaware of the change, they will no longer be able to participate.

It should be noted however that the current group root key can be 'leaked' by a member, and anyone

with it will be able to participate for as long as the member set stays the same, unless there is some sort of re-keying protocol in place.

This system is described in the Internet Engineering Task Force (IETF) document RFC2627 [25], although it should be noted the proposed protocol involves a centralised server for coordinating the exchange. Many different protocols (eg: [6], [20]) have been published based on this idea, with different methods for handling group members joining and leaving.

To do this without a trusted server for coordination, there must be a way to deterministically decide which members share parent nodes in the tree. Kim, Perrig, and Tsudik [19] suggest a protocol where an 'insertion node' for a new member joining is chosen to be the shallowest, rightmost node where the join would not increase the depth of the tree. If the tree is equally balanced (like the one pictured above) the root node is taken to be the insertion node and the new tree will be deeper. The rightmost existing member who is a child of the insertion node takes the role of the 'sponsor' who computes the relevant blinded keys for the new version of the tree and broadcasts them to the other members. So a member's position in the tree is determined by the order in which they joined.

Another possible approach would be to order the list of members numerically by their public keys. Given the public keys of the other group members, we can deterministically find our position in that list and our adjacent partner with whom we 'pair with' to contribute to the group key. Any change to the set of members requires the same process of creating the new ordered list of members.

The process of setting up such a tree, or re-keying later, requires the participation of several group members. Even if we know the public keys of all prospective members of a group, we cannot begin encrypting messages to the group until we get know the blinded keys from the branches of the tree besides our own. This can be seen as a disadvantage, as group setup requires multiple rounds, or as an advantage, as a participatory setup process requires agreement within the group.

4.1.2.2.2 Messaging Layer Security

Messaging Layer Security (MLS) is a group encryption protocol proposed as a standard to the IETF. At the time of writing the protocol is a draft working document [2].

The protocol [10] is also based on a Diffie-Hellman tree structure, but additionally uses ratcheting to offer both forward and post-compromise security. It aims to scale well to large group sizes and can function asynchronously, meaning it does not require group members to actively interact in order to establish keys for each message, so that the system tolerates group members not being constantly online. It is possible to communicate securely as a group, even if no two group members are ever online at the same time. Furthermore, a peer may create a group and send the first encrypted message immediately even if every other member is online.

While these are very desirable properties for many use cases, for group governance we might actually want to impose the cooperation the other group members (or at least a critically-sized subset of them). MLS is designed to fit the needs of group communication for mobile instant messaging apps, whereas our use-case is consensual group governance.

Furthermore, MLS requires public server infrastructure to cache 'pre-keys' for the protocol in order to make these asynchronicity guarantees, which is something we want to avoid. However, note that the 'pre-key' cache system, as proposed by Marlinspike for TextSecure [22], does not require the server to be trusted. All pre-keys are signed with a long term signing key, and the client who generates them is maintains which key-ids have already been used, so if the server gives out the same key twice, the second one will be rejected.

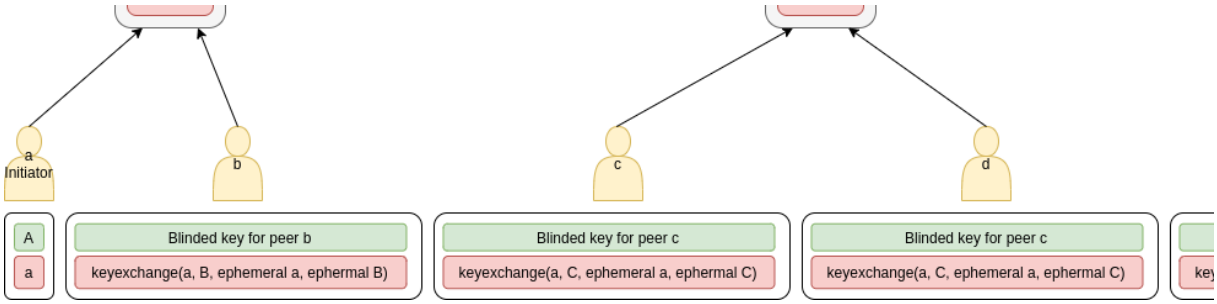

Figure 17: With MLS, we have a Diffie-Hellman key tree, but the person who sets up the group 'makes' keypairs for the other members' leaf nodes using a key exchange protocol. The initiator can then compute the whole tree and start encrypting messages without needing to wait for any other member to respond

With MLS, we have a tree-structure just as the one above, but instead of each leaf node being the member's long-term 'identity' keypair, the group initiator, Alice, derives new keypairs for each of the other members using a single-round key exchange protocol, such that only Alice and that member are able to compute the secret key. This process involves both the member's identity keypair and ephemeral public 'pre-keys' retrieved from the third party cache. Of course if we want to avoid this, members could generate these keys during the setup process, but then Alice must wait for a response from Bob before she can encrypt the first group message.

It should be noted that such groups are not composed deterministically. That is, if Alice and Bob both independently create groups with the same set of members, we will have two separate groups, as different ephemeral key material will have been used to create them.

To achieve post-compromise security, each group member periodically updates their leaf-node keypair and send the updated pubic keys along their co-path to the rest of the group. The members then compute the new root key, and hash in the previous one using a key derivation function, to give forward

secrecy. This means the current key for the group depends on both the 'chain' of previous group states and newly generated key material.

Previously, such ratcheting techniques which offer both forward and post-compromise secrecy have generally only been proposed for two-party encryption protocols [9]. MLS's 'Asynchronous racheting trees' extend this to groups.

For group governance applications, we want a protocol similar to MLS in that we use a ratcheting tree for security, but different in that we favour member participation over asynchonicity. That is, we are more interested in gaining consent through active participation of group members during the setup process than on being able to send messages instantly.

4.1.3 Existing group governance tools

Here we look at some collaboration software and assess their methods of administrating the group.

4.1.3.1 Loomio

Loomio [21] is decision-making software and web service designed to assist groups with the collaborative consensus focused decision-making processes. It is a free software web application, where users can initiate discussions and put up proposals.

Figure 18: Loomio, a voting-based decision making tool

Loomio provides a platform for both casting votes on an issue and discussing it, which is important particularly when group members are unable to meet face to face. Discussing the issue and gaining an understanding of other members' opinions is perhaps the most important part of the consensus process.

However, Loomio assists in making group decisions, but not in upholding the decided action. For example, if group members vote on whether to allow a new person to join the group, it is still the responsibility of an administrator to ultimately decide whether the chosen action is carried out. Furthermore since it is web-based software, the administrators of the server have ultimate control over the service.

So Loomio is a great tool for facilitating discussion and decision making, and perhaps it could be used in combination with cryptographic group governance techniques.

Figure 19: Annotated screenshot of Loomio

4.1.3.2 Keybase

Keybase [27] is a secure messaging and file-sharing app with support for private groups. It uses a centralised client-server architecture with client-side encryption. The client app is open-source but the server code is not.

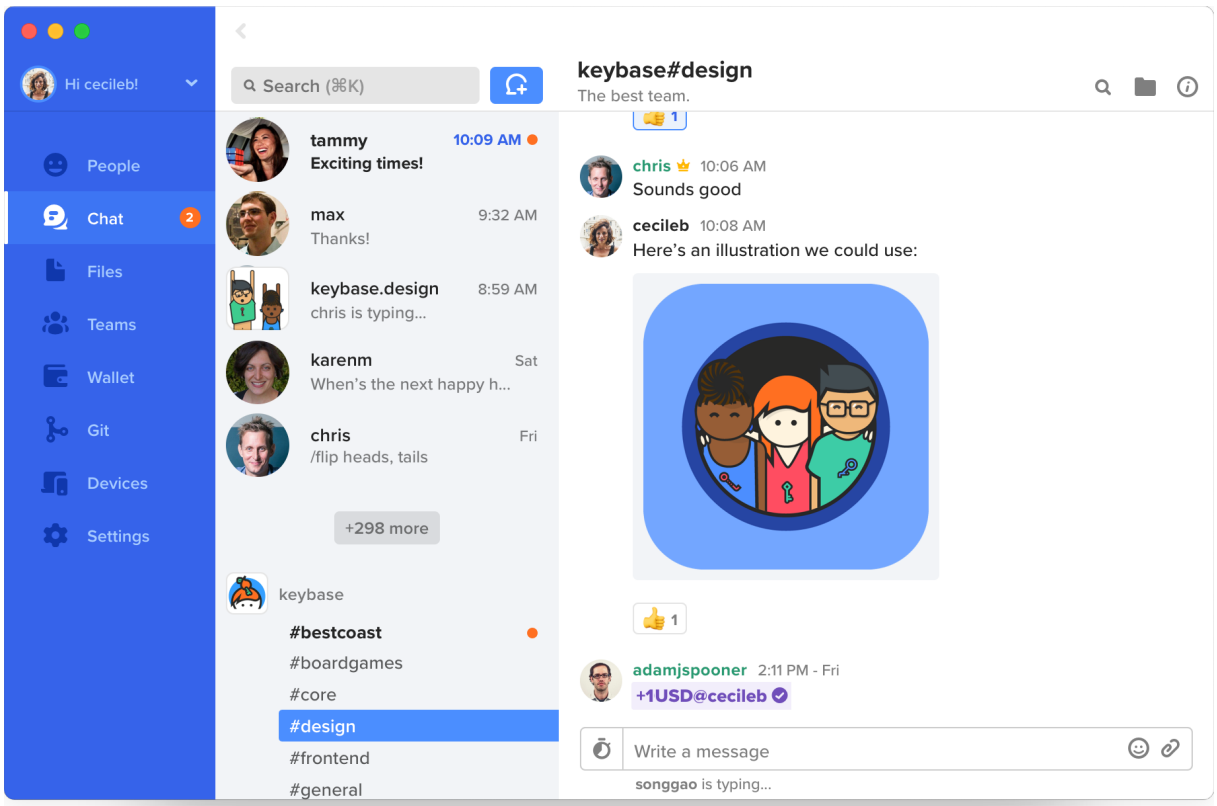

Figure 20: Screenshot of Keybase

Both files and chat channels are encrypted with granular control over permissions - individual channels or folders can be restricted to particular members. Identities on Keybase can be 'proved' by linking them with other social network platforms by publishing a signature on those platforms. PGP keys and control of a specific website can also be proved in this way.

Group members may have an 'admin' role which allows them to add or remove members, as well as give or revoke admin status to other members. The initiator of a group is automatically an admin. That is, there is no inherent consensus mechanism for a member joining or leaving the group. It is possible to give all members the admin role, but then any member may add or remove anyone else without consulting any other member.

Being a client-server application, the company providing the Keybase software have ultimate control over the service. They are unable to decrypt content, but they have access to identities of group members as well as other metadata, and may remove content or revoke the service entirely.

4.1.3.3 Nextcloud

Nextcloud [16] is a free and open-source file-hosting and cloud application platform. Organisations may run the software on their private servers, giving them a lot of autonomy in comparison to using cloud services hosted by a third party. Besides file-hosting, Nextcloud offers many collaboration applications useful to organisations, such as calendars, chat, and office software.

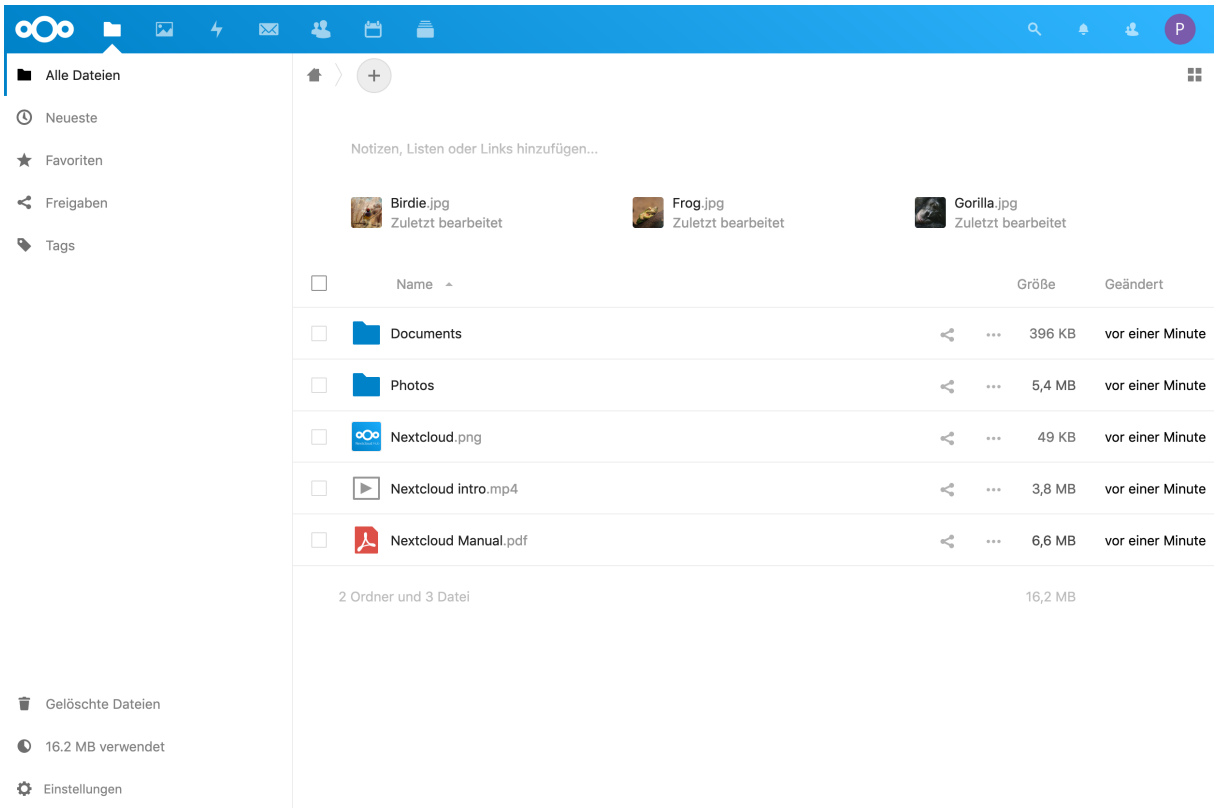

Figure 21: Screenshot of Nextcloud

Transport encryption and server-side encryption are offered by default but client-side encryption is an opt-in feature for specific folders. When using the client app rather than the web interface, folders are synced with a local copy, meaning each group member has their own copy of their group files, which remains available should the server go down.

A single Nextcloud instance can have multiple groups using it with different permissions. Because of this, Nextcloud has two types of administrators: Super Administrators and Group Administrators. Group administrators have the rights to create, edit and delete users in their assigned groups. Group administrators cannot access system settings, or add or modify users in the groups that they are not Group Administrators for. Super Administrators can access all settings and create or delete users in all

groups.

As well as the administrator role system within the Nextcloud software, being a client-server application, those who have administrator access to the server have ultimate control. Since Nextcloud is self-hosted, this means that at least these administrators are members of the group. But this means that the more technically able group members have the most power, and that administrators may take actions such as revoking the service without consulting other group members. The physical location of the server can also create a power imbalance, if there is not a collective space to keep it.

Self-hosted client-server platforms like Nextcloud are undoubtedly very empowering tools for groups of high-risk users, and arguably the best thing we have right now. But we think better still would be a peer-to-peer architecture combined with cryptographic techniques to ensure administrative decisions are made democratically.

4.2 Design for a Key Re-issuance Protocol

The Dark Crystal Key-backup protocol addresses only one component of the key custody problem: key loss. The other component, key compromise, requires a way of proving or re-establishing our identity in the cryptographic system we are using, such that the compromised key is made effectively useless.

We propose a design for a new protocol for establishing a new key, so that an identity is not defined by a particular key, but by a series of keys, where the final key is the active key we currently use. This specifically addresses the cases where keys might have been compromised. Our design uses the BLS signature scheme.

It works by each peer assigning a 'support group' who are empowered to make assertions on their behalf. This is similar to the group of custodians for social backup. The support group can announce a new key when the old one is assumed to be compromised, and any other peer can validate these announcement messages and determine the current key for a particular identity.

This is done is such a way that the identities of the members of the support group are kept private, and that there is specific degree of tolerance to a particular group members being unavailable or uncooperative. The basic process is as follows:

1. The identifier holder, Alice, chooses a set of trusted contacts as her support group, and a threshold amount needed to make an assertion on her behalf.
2. Each trusted contact makes a contribution to a distributed key generation, collectively producing an aggregated group public key.
3. Alice publishes the support group's public key signed with her current identity keypair.
4. In the event of key loss or compromise, Alice generates a new keypair, effectively making a fresh account on the system, finds the contact information for her support group members, and sends them a signed, timestamped message containing her new public key. She also contacts them out of band to confirm it was her. By 'out of band' we mean using some communication system where authentication does not rely on the compromised key - such as a telephone call or personal meeting.
5. Group members who are convinced of Alice's new identity respond by signing her message and Alice aggregates their signatures to produce a single signature. If enough members have signed, this signature can be validated with the group public key. Alice publishes this message, which serves to assert her new public key, and also to revoke the old one.
6. Any client software which seeks to validate messages from Alice must resolve her current public key by looking for these messages which are signed by her trusted group.

7. If support group members change, the new group's aggregate public key announcement must be signed by both Alice and the old group.

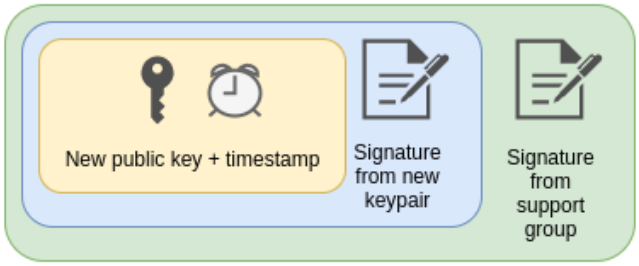

Figure 22: A key re-issuance message

Importantly, anonymity of support group members is maintained. Nobody but Alice can see which group members created the signature or who the group members are. This makes it difficult for an adversary to target the support group in order to impersonate Alice.

The scheme looks promising, although we glossed over the most complicated part of the process - the distributed key generation in step 2. There are already several protocols and implementations of this process, eg: [3], [15], but it is important to note that this process involves multiple rounds of communication between members of the support group.

Specifically, each member needs to send each other member a public 'verification vector' as well as a secret key contribution which should be encrypted specifically for the recipient member. This might not sound so difficult, but since we cannot guarantee that all members are simultaneously online, the setup process could take some time, which is going to effect usability. [17] offers some possible optimisations to reduce complexity.

Another major disadvantage of this scheme is that unlike the Key Backup protocol, this protocol cannot be used with existing systems which have not implemented this mechanism.

4.3 Inheritance

Returning to our three key loss scenarios, ‘inheritance’ describes the case where we actually want a key to be able to be compromised in a specific situation. The secret owner might have a wish for what happens to sensitive data of theirs in the case that they captured, imprisoned, or killed.

Our key backup protocol is useful for these situations, but this is a very different use case to that of the secret owner recovering the data themselves, and so requires very different usability considerations. Most importantly, the heirs do not only need to be able to access the secret data, they also need to know what to do with it. For example if it is an encryption key, they need to know where is the encrypted data, what software can be used to decrypt it, and what should be done with the secret documents when they are recovered.

4.3.1 User story for Inheritance case

Carol is a journalist investigating corruption in a near-east country under a repressive regime. She has recently received some sensitive documents from an anonymous source, and is concerned that this makes her a target. If she is captured, this information may never come to light. She is careful with her handling of the documents, avoiding that surveillance would reveal that they exist, or that others who discover them could reveal the information in an inappropriate or untimely manner, damaging the investigation. She therefore stores the files on a remote server, encrypted with her private key, which she accesses through Tor.

- In this case, Carol chooses to shard her key using a secure communications app that has implemented the Dark Crystal key backup protocol, allowing for any data of the user’s choice to be sharded.
- Carol opens the app, opens the options menu and chooses the ‘share a secret’ option from the list.
- The app asks Carol to enter the secret which she wishes to share. She does this by clicking on the ‘add from device’ option. This allows her to access her device’s file system, where she locates her private key and clicks ‘add’.
- The next screen asks Carol if she would like to attach a note or label to her secret. She decides to provide brief instructions on where her custodians can find the sensitive files, and what they should do with them. She clicks ‘done’.
- Carol is then asked to select her trusted custodians from her list of contacts. She chooses 6 trusted contacts. The custodians she chooses are all contacts from her professional network - a combination of lawyers, reporters and campaigners - and are all located abroad, outside

of the jurisdiction she is residing within and reporting on. This further secures the data from compromise.

- When asked to select a quorum of these who will be required to recombine the secret, she chooses 4. The user interface provides visual guidance relating to the security provided by the number she chooses for each.
- The app appends the label to the secret and executes the sharding algorithm. It then offers Carol the option to add additional information to the shards before encrypting them with the custodian's public keys.
- In this specific case, as she needs the custodians to know who one another are so that they can recombine the secret in her absence, Carol chooses to append the custodian's identities to the shards. This means that the identities are encrypted in transit and at rest on the custodians' devices, but that the custodians can discover one another once they decrypt their shards should the need arise.
- Alternately, if she decides that despite their locations, she would prefer not to encrypt the shards together with this list, she can simply contact the custodians out of band to let them know who one another are, ensuring that this information is never stored together with the shards.
- She then clicks 'send shards' and the app sends the shards to her custodians.
- As she proceeds with her work, Carol continues to update the files encrypted with her key. Some months later, while still investigating the case, she disappears on her way to work one morning, never arriving at the office. Realising this, her colleague sends a message to their professional network and begins publicising her disappearance through media channels.
- Eventually one of her custodians decides that it is an appropriate time to recombine the shards, find out what Carol was working on and execute her instructions.
- He contacts the other custodians to let them know. The other custodians independently confirm that she has in fact disappeared, and that the initiator is who he claims to be. Once they are confident that the situation is authentic, they collectively choose one of them to whom they will all forward their shards.
- On receiving the shards, the recipient is able to recombine them, revealing both the encryption key and the instructions that Carol has provided. They are then able to execute her wishes in relation to the data she was protecting.

It should be noted that in this scenario, as indicated above, it may be desirable to implement a mechanism to ensure that the custodian who initiates the recombining of the shards is not the one who gains possession of the secret. This provides an extra layer of protection in the case that one of the custodians is or becomes malicious.

5 Conclusion

Our Key Backup protocol as well as our work with Briar have shown a concrete use-case for threshold-based secret sharing in the context of internet freedom. We have discussed the associated usability and security issues in order to best inform anyone considering adopting this technology.

In terms of wider use-cases, we have focussed on group governance in collaborative software, as this seems the most relevant and applicable. We looked at existing models in popular collaborative software, and the potential power imbalances from both within the group and external actors.

We have also proposed a design for a new protocol for 'Key Re-issuance' which, in combination with our existing Key Backup protocol, aims to fully address the problem of key custody described in this report.

Throughout this report we advocate a peer-to-peer architecture combined with cryptographic techniques to ensure administrative decisions are made democratically, as well as key management techniques to ensure responsibility is distributed amongst the collective.

References

[1] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. *A Survey of ECDSA Threshold Signing*. Cryptology ePrint Archive, Report 2020/1390. https://ia.cr/2020/1390. 2020.

[2] R. Barnes et al. *Messaging Layer Security Draft IETF proposal*. 2021. URL: https://messaginglayersecurity.rocks/mls-protocol/draft-ietf-mls-protocol.html.

[3] Alexander Block. *BLS M-of-N Threshold Scheme and Distributed Key Generation*. 2018. URL: https://github.com/dashpay/dips/blob/master/dip-0006/bls_m-of-n_threshold_scheme_and_dkg.md#distributed-key-generation-dkg-protocol.

[4] Dan Boneh, Manu Drijvers, and Gregory Neven. "Compact Multi-signatures for Smaller Blockchains". In: *Advances in Cryptology – ASIACRYPT 2018*. Ed. by Thomas Peyrin and Steven Galbraith. Cham: Springer International Publishing, 2018, pp. 435–464. ISBN: 978-3-030-03329-3.

[5] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *Advances in Cryptology — ASIACRYPT 2001*. Ed. by Colin Boyd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 514–532. ISBN: 978-3-540-45682-7.

[6] Emmanuel Bresson et al. "Provably Authenticated Group Diffie-Hellman Key Exchange". In: *Proceedings of the 8th ACM Conference on Computer and Communications Security*. CCS '01. Philadelphia, PA, USA: Association for Computing Machinery, 2001, pp. 255–264. ISBN: 1581133855. DOI: 10.1145/501983.502018.

[7] Briar. *Briar, Secure Messaging Anywhere*. 2021. URL: https://briarproject.org.

[8] David Chaum and Eugène van Heyst. "Group Signatures". In: *Advances in Cryptology — EUROCRYPT '91*. Ed. by Donald W. Davies. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 257–265. ISBN: 978-3-540-46416-7. URL: https://link.springer.com/chapter/10.1007%2F3-540-46416-6_22.

[9] Katriel Cohn-Gordon et al. "A Formal Security Analysis of the Signal Messaging Protocol". In: *2017 IEEE European Symposium on Security and Privacy (EuroS P)*. 2017, pp. 451–466. DOI: 10.1109/EuroSP.2017.27.

[10] Katriel Cohn-Gordon et al. "On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 1802–1819. ISBN: 9781450356930. DOI: 10.1145/3243734.3243747. URL: https://eprint.iacr.org/2017/666.pdf.

[11] Magma Collective. *Dark Crystal - Briar Project Case Report*. 2021. URL: https://darkcrystal.pw/assets/briar-case-project-report.pdf.

[12] Magma Collective. *Dark Crystal Key Backup Protocol Specification*. 2021. URL: https://darkcrystal.pw/protocol-specification.

[13] Magma Collective. *Dark Crystal Key Backup Threat Model*. 2021. URL: https://darkcrystal.pw/threat-model.

[14] Rosario Gennaro and Steven Goldfeder. "Fast Multiparty Threshold ECDSA with Fast Trustless Setup". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 1179–1194. ISBN: 9781450356930. DOI: 10.1145/3243734.3243859. URL: https://dl.acm.org/doi/pdf/10.1145/3243734.3243859.

[15] Rosario Gennaro et al. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: *Journal of Cryptology* 20 (2006), pp. 51–83.

[16] Nextcloud GmbH. *Nextcloud - The self-hosted productivity platform that keeps you in control*. 2021. URL: https://nextcloud.com.

[17] Kobi Gurkan et al. "Aggregatable Distributed Key Generation". In: *Advances in Cryptology – EUROCRYPT 2021*. Ed. by Anne Canteaut and François-Xavier Standaert. Cham: Springer International Publishing, 2021, pp. 147–176. ISBN: 978-3-030-77870-5.

[18] The Ponemon Institute. *Global Encryption Trends Study 2018*. 2018. URL: https://go.ncipher.com/rs/104-QOX-775/images/2018-nCipher-Ponemon-Global-Encryption-Trends-Study-es.pdf.

[19] Yongdae Kim, Adrian Perrig, and Gene Tsudik. "Simple and fault-tolerant key agreement for dynamic collaborative groups". In: *Proceedings of the 7th ACM Conference on Computer and Communications Security*. 2000, pp. 235–244.

[20] Sangwon Lee et al. "An Efficient Tree-Based Group Key Agreement Using Bilinear Map". In: *Applied Cryptography and Network Security*. Ed. by Jianying Zhou, Moti Yung, and Yongfei Han. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 357–371. ISBN: 978-3-540-45203-4.

[21] Loomio Cooperative Limited. *Loomio - Where collaborative organizations share, discuss and decide*. 2021. URL: https://www.loomio.com.

[22] Moxie Marlinspike. *Forward Secrecy for Asynchronous Messages*. 2013. URL: https://signal.org/blog/asynchronous-security (visited on 11/30/2021).

[23] Simply Secure. *The Limits to Digital Consent*. 2021. URL: https://simplysecure.org/resources/The_Limits_to_Digital_Consent_FINAL_Oct2021.pdf.

[24] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176.

[25] D. Wallner, E. Harder, and R. Agee. *RFC2627 - Key Management for Multicast: Issues and Architectures*. 1999. URL: https://datatracker.ietf.org/doc/html/rfc2627.

[26] Chung Kei Wong, M. Gouda, and S.S. Lam. "Secure group communications using key graphs". In: *IEEE/ACM Transactions on Networking* 8.1 (2000), pp. 16–30. DOI: 10.1109/90.836475.

[27] Inc Zoom Video Communications. *Keybase - End-to-end encryption for things that matter*. 2021. URL: https://keybase.io.