Secure Collaborative Editing Without Central Servers

Benjamin Ylvisaker
Colorado College
bylvisaker@*

Michał Wiśniewski
Colorado College
michal.wis0@gmail.com

Jay Batavia
Galvanize, Inc.
Jay.Batavia@*

* - @coloradocollege.edu

1. INTRODUCTION

Collaborative editing applications like Google Docs and EtherPad are a popular way for teams to work together on data. The most common implementations of these tools involve clients sending users' edits to a central service provider to be integrated into a single consistent document/database. This architecture has obvious security and privacy issues: users' data are vulnerable to bad behavior by companies, rogue employees, outsider attacks on the servers, and snooping and/or litigious governments.

End-to-end encrypted (E2EE) architectures have been used in messaging applications (Signal, WhatsApp, Telegram) to provide protection against the attacks alluded to above. In E2EE applications, the service provider never has access to unencrypted user data or the keys needed to decrypt such data. There has been some research on bringing these kinds of E2EE architectures to collaborative editing applications, though we are not aware of any such applications that have reached the level of popularity achieved by messaging applications.

The current generation of E2EE architectures was designed to cryptographically protect users' data from the service provider. However, the functioning of these applications still relies on central communication and storage services. This means that malicious actors can relatively easily carry out denial of services attacks by blocking access to servers, or legally forcing organizations to shut down services. In the last couple of years there have been several instances of governments engaging in this kind of attack.

This proposal is about a new E2EE architecture for collaborative editing applications that has two goals, which are in some tension with each other:

1. Minimize the role of any central service. Once deployed on client machines, applications should continue to function even if all resources controlled by the application authors are blocked or taken down.

2. Require no specialized technical skill or resources on the parts of users.

The United We Stand (UWS) protocol is our attempt to achieve these goals. The central concepts in UWS are users and teams; users can participate in many teams. The components of the protocol are:

- A secure messaging protocol for communicating transactions to team members. The current prototype uses an adaptation of the Signal protocol.

- A mechanism for broadcasting messages to teammates. The current prototype uses commodity cloud storage accounts, where each user keeps a copy of each team's data in their own cloud storage account.

- An accumulate-only data model. The current prototype uses Datomic as inspiration for data modeling.

- A consistency protocol so all teammates can come to agreement on the state of the database. We currently use a relatively simple vector clock-based system for coming to agreement on the order of transactions.

Our work on the protocol is at a fairly early stage, but our prototype has validated the basic feasibility of the protocol.

2. BLOCKING E2EE SERVICES

The primary impetus behind this work is the (sometimes successful) attempts governments and law enforcement organizations have made to interfere with either the basic functioning or cryptographic integrity of E2EE services. We believe that secure and private communication and collaboration among individuals and groups is on balance a positive thing in the world, and we seek to strengthen the technologies that enable it. We begin with a quick recap of some of the high profile cases of governments interfering with E2EE services in recent years.

The Brazilian government and WhatsApp have been in conflict since 2015, when a prosecutor blocked the service nation-wide. More recently a Brazilian prosecutor froze millions of dollars worth of assets owned by Facebook, the owner of WhatsApp. The core of the conflict is WhatsApp's refusal to assist in criminal investigations. Of course, if WhatsApp has implemented the kind of E2EE system that they advertise, it is likely technically infeasible for them to provide the kind of data that the prosecutors want.

More recently, the Russian government has attempted to block the Telegram service. Telegram has attempted to work around the blocking by obscuring its services in various ways. But the Russian government has kept up with this cat and mouse game. In general, it will always be hard to prevent blocking of an open public service.

One technique that can be used to evade digital blockades is *domain fronting*. Domain fronting a trick in TLS that can be used to make the apparent and actual endpoint for a connection different, if both are host by the same large provider, like Google App Engine or Amazon Web Services. The Signal service has used this trick to avoid blockade in

countries like Iran, Egypt, Oman, Qatar, and UAE. Unfortunately, this technique requires at least passive acceptance by the large provider, and in May 2018 Amazon notified Signal that they should cease any domain fronting.

Many governments are currently considering *key disclosure laws*. These laws vary in their details, but in the most extreme cases they require any software or digital service provider that enables secure communication to hold copies of whatever keys are necessary to decrypt users' data.

Our goal is to completely eliminate these kinds of concerns by decentralizing the storage and communication for E2EE collaborative editing applications. If there is no central service provider, there is no target for the kinds of attacks described above.

3. WEAK POINT: CLOUD STORAGE

As mentioned in the introduction, the main storage and communication medium in UWS is commodity cloud storage accounts. We need to consider two kinds of attacks. The first is the storage service providers modifying files in some way. This could certainly be disruptive, but *all* uploaded files are signed by a private key, so making counterfeit files would be cryptographically infeasible.

The second kind of attack is denial of service from the storage service provider themselves, or externally via blocked access. It seems fairly unlikely that governments would completely block cloud storage systems like Dropbox or Google Drive. However, those storage providers may very well be motivated to block the storage of files for secure applications, especially if pressured to do so by governments.

As shown by the examples above, governments are certainly motivated to block secure communications applications. However, unlike many E2EE chat applications, in UWS there is no central service provider. So aspiring blockers could not block based solely on the traffic's destination.

Naturally, storage providers or external network service providers could examine files or network traffic for patterns that suggest it is UWS. If this ever became an issue we could investigate the kinds of steganographic techniques used by systems like Tor (for example we could make UWS files look like family photos). It is also conceivable to use decentralized peer-to-peer cloud storage systems. Several such systems have been designed, but none has achieved much popularity as far as the authors know. Perhaps the current anti-privacy mood in many governments will help these systems become more widely used.

4. TECHNICAL HIGHLIGHTS

One of the immediate challenges in the UWS protocol is that the storage and communication service is unreliable. Our prototype uses an approach inspired by journaling file systems to address this. A user's whole UWS collection of files is organized in a single tree structure. When a user wants to upload any changes, they first add new files that represent the changes, then make a single atomic modification to the root file pointer.

This means that the technical requirements for the storage service are quite modest: plain file uploads and download, and an atomic read-modify-write operation. Many cloud storage service providers support this kind of operation via the HTTP ETag headers.

An important issue we face is making the data access and consistency protocol fast enough for reasonable applications. There is no central server, so all work must be performed by clients. We take our inspiration for the database architecture part of the project from Datomic, a "functional" accumulate-only database. In preliminary testing, it seems that basic user to user communication with all the encryption and communication through database stored in cloud storage accounts can happen in less than a second. This should be low enough latency for many applications.

Our prototype implementation of UWS is quite preliminary, but all code is available on GitHub:
`https://github.com/benjaminy/ManyHands`