Beeper: a p2p web chat

Noah Gallant (nag2145@columba.edu)
Undergraduate Computer Science Thesis
Columbia University
Advisor: Prof. Henning Schulzrinne (hgs10@columbia.edu)

Demo: https://beeper-chat.herokuapp.com/
GitHub: https://github.com/NoahGallant/beeper

Abstract

The goal of this thesis project is to explore the potential of building a peer-to-peer online web chat application whose architecture guarantees user data privacy through encryption. The application, called Beeper, is a client-side progressive web application (pwa) with user authentication and web chat functionality using the Dat peer-to-peer protocol as a data model. The project serves as a proof of concept for applications that do not employ dedicated servers for online data, but work within a shared public data network, the Dat. Beeper users manipulate 'their own' data in a javascript application, and they then seed encrypted versions of their data within the Dat which syncs to other users' applications.

Background

The client-server internet

Many commonly used online applications at present rely largely on what is known as the client-server model. In order for application data to stay persistent, always-available and consistent, dedicated or distributed servers or databases are employed to store the user's data. These application servers are configured, owned (or rented), and controlled by the application owner.

In order to ensure proper authentication and data storage practices on behalf of web users, the IEEE Computer Society provides best practices [Bedra, et. al]. These best practices highlight the most secure authentication structure. In modern authentication schemes, password are put through one-way “hash functions” and the result is stored. This hash can then be compared to password attempts run through the same function for user authentication [Krawczyk]. This methodology allows for secure storage of passwords in a database, even breached attackers cannot deduce the passwords from the hashes[1]. Wordpress applications, for example, store account passwords by default as MD5 hashes in a local MySQL database[2]. As perspective, VentureBeat reported in March 2018 that Wordpress powers 30% of websites. While that does not necessarily mean 30% of online accounts are stored authenticated in this manner, it shows the popularity of using this method of authentication [Sawers].

Trust

The IEEE best practices highlight the “authorize after you authenticate” strategy. In this strategy an application authenticates the identity of a user who has specific listed permissions according to application functions (see Role Based Action Control). Yet employing this alone does not really deal with the issues that seem to arise more fundamentally with attempting to employ proper data usage and authentication practices even at the highest level of software development. In 2016, hackers released the data of 500 million Yahoo! users [Perlroth]. In 2019, it was reported that Faecbook accidentally was storing hundreds of millions of user passwords in plain text for multiple years [Krebs]. These are among numerous data-privacy scandals in recent times, and consequently trust has become a growing issue between the user and application owner.

In response to these data mishaps, new technologies have recently emerged which enable normal online application behavior without having to “trust” the application owner with your data. Telegram, for example, seeks to offer encrypted chat application with the classic client-server model that is free (for the foreseeable future) due to a founding donation that funds the ongoing server costs[3]. Many others of these efforts exist in the blockchain space. For example, BlockStack provides an application structure with authentication that runs on its encrypted decentralized blockchain network [Ali, et al.]. Blockstack has successfully deployed numerous decentralized applications, but the operation of the network relies on incentivizing distributed computation with cryptocurrency tokens.

[1] See https://en.wikipedia.org/wiki/Rainbow_table
[2] https://codex.wordpress.org/Database_Description
[3] https://telegram.org/faq

Dat Protocol

Main principles

Dat is a community-based open protocol for data sharing[4]. The protocol was developed to enable better data usage in future web applications. As it stands it is a peer-to-peer data transfer network wherein public documents exist at “dat://” endpoints [Ogden, et al.]. Each endpoint contains a Dat, or “archive” which has a discovery key, a public key, and a private key. An archive can be thought of practically as a file folder which can be found at “dat://{public_key}”. Given access to the peer discovery network, one can use the discovery key to retrieve the Dat without exposing the public key. Given the private key for a Dat, you can write to that specific archive, which will update in “the Dat,” the overall shared appendable key network for dat://. Others can retrieve and watch a Dat so long as it being seeded.

Multi-writer and local keys

Recent developments to the Dat protocol has focused on improving the usability of the protocol through a concept called ‘multiwriter.’ ‘Multiwriter’ refers to the ability to authorize other peers on a Dat network to write to your archive. Each peer has a unique local key upon replicating the Dat they are attempting to write to. The owner (or a previously approved writer) can then approve that local key (which is the public key for their replication of the archive).

Introduction

Motivation

The goal of this research is to create an application with online behavior that employs *cryptographically secure* data-sharing between users *without* relying on servers. The application, a web chat app named Beeper, stands as a proof of concept for deploying secure web-like applications which use the Dat in a secure data model.

Core Requirements

We seek to fulfill the following user requirements with Beeper:

[4] https://datproject.org/

1. A user has an account which corresponds to a public and private key
    a. The public key serves as the network identifier for that user
    b. User can store public information at that endpoint, in a account.json for example
2. Entering the correct passphrase a user can retrieve their private key
    a. This enables identity-based signing and encrypting and login/logout
3. Only approved users of can read and send messages in the chat
4. Current members can add other users (by their public key) to the chat
5. The chat application should take the form of a client-side progressive web app

---

Implementation

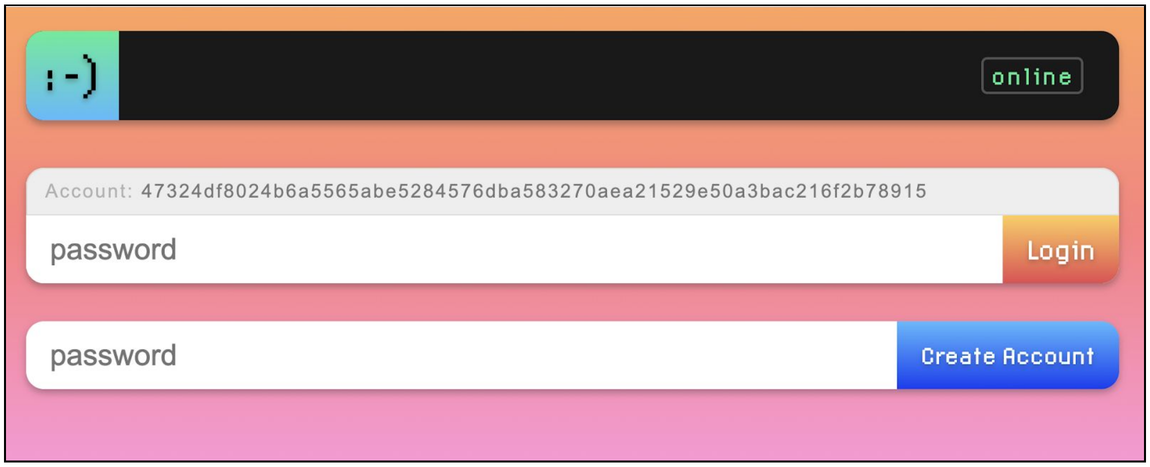
Libraries

Beeper was built using the front-end javascript framework Choo[5]. Choo provides routing, and a state and event emitter handler. In order to interface with Dat, the hyperdrive library is used[6]. Hyperdrive provides a prebuilt archive and database structure which Beeper syncs to the Dat network. Beeper also uses the tweetnacl javascript encryption library[7]. Tweetnacl is used in Beeper for encrypting and decrypting of json. We employ this library for both single secret key encryption (*secret box*) and public-private key pair encryption (*box*).

[5] https://choo.io
[6] https://github.com/mafintosh/hyperdrive
[7] https://github.com/dchest/tweetnacl-js

Architecture

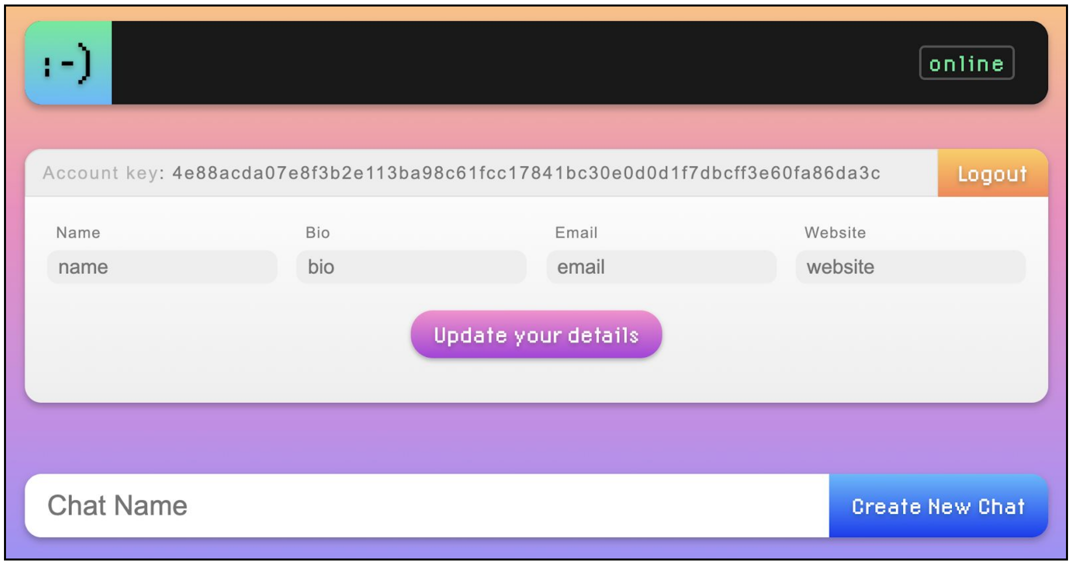
Application Views and Operation



Account: 47324df8024b6a5565abe5284576dba583270aea21529e50a3bac216f2b78915

Home page ('/')

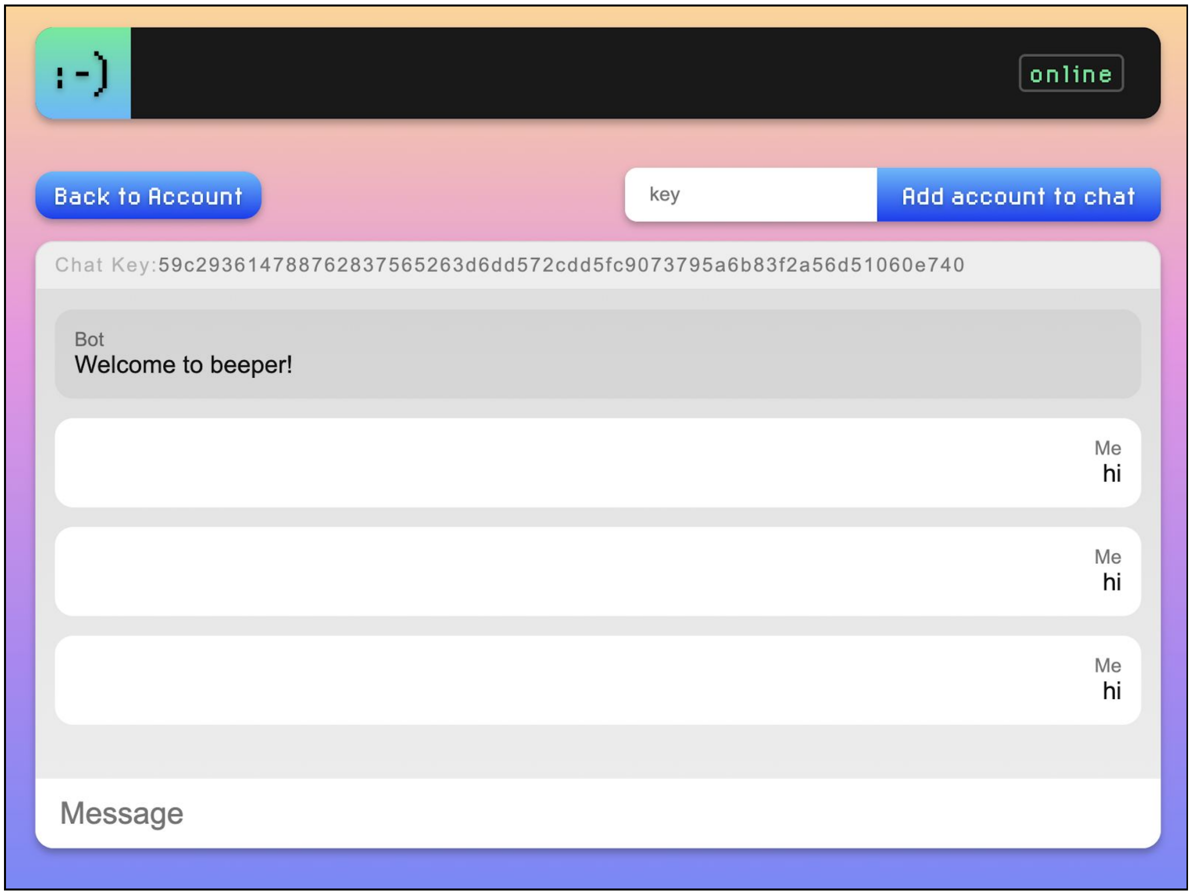
On the home page the user can enter a password and a new account Dat will be created. On this action, a secret box containing the secret key is encrypted using an MD5 hash of the password and stored in the account Dat. The user is then forwarded to the account page. The home page also includes a sign-in which attempts to unlock the secret box within that account Dat.



Account key: 4e88acda07e8f3b2e113ba98c61fcc17841bc30e0d0d1f7dbcff3e60fa86da3c

Account Page ('/account/:key')

The account page is only accessible upon successful decryption of the login secret box. This page contains a form for editing the public information corresponding to the account Dat. The logout button will remove the account secret key from the browser's local storage and redirect the user to the home page. The account page also contains a form for creating a chat which constructs a new chat Dat, storing the secret key of the chat in a box using the user's own public key and the chat private key. In chat creation, the public key of the chat is also stored in an encrypted secret box file within the user's account Dat using their private key.



Chat Key:59c293614788762837565263d6dd572cdd5fc9073795a6b83f2a56d51060e740

Bot
Welcome to beeper!

Me
hi

Me
hi

Me
hi

Chat Page ('/chat/:key')

This page contains the chat view and operation. When a user goes to send a message the message is added as a secret box file in the chat/ folder of the chat Dat. By adding files to the archive rather than writing to a single chat log file, Beeper avoids possibly complex conflict resolution with unsync-ed peers. To load messages the chat/ folder is read and each message is decrypted using the chat secret key. Upon adding a friend to the chat, the secret key is written to the boxes/

folder in the chat Dat encrypted in a box using the public key of the friend and the private key of the chat. The user is then taken to the ‘Adding friend page’

Adding friend page ('/addFriendToChat/:key/:chatKey')

This page has the generated for a friend to join the chat. The application waits on this page during the adding members to chat process until the new member has written their chat local key to their dat. The user is then forwarded to the addToChat endpoint with the local key of the friend.

Endpoints

‘/addToChat/:key/:writerKey/:key32’

Accessing this endpoint authorizes new local key to the chat Dat.

‘/loadLocal/:key’

Accessing this endpoint finds my local key for the chat Dat (for the user accessing)

‘/writeLocal/:key/:chatKey’

Accessing this endpoint writes local key to a file within my account Dat (for the user accessing)

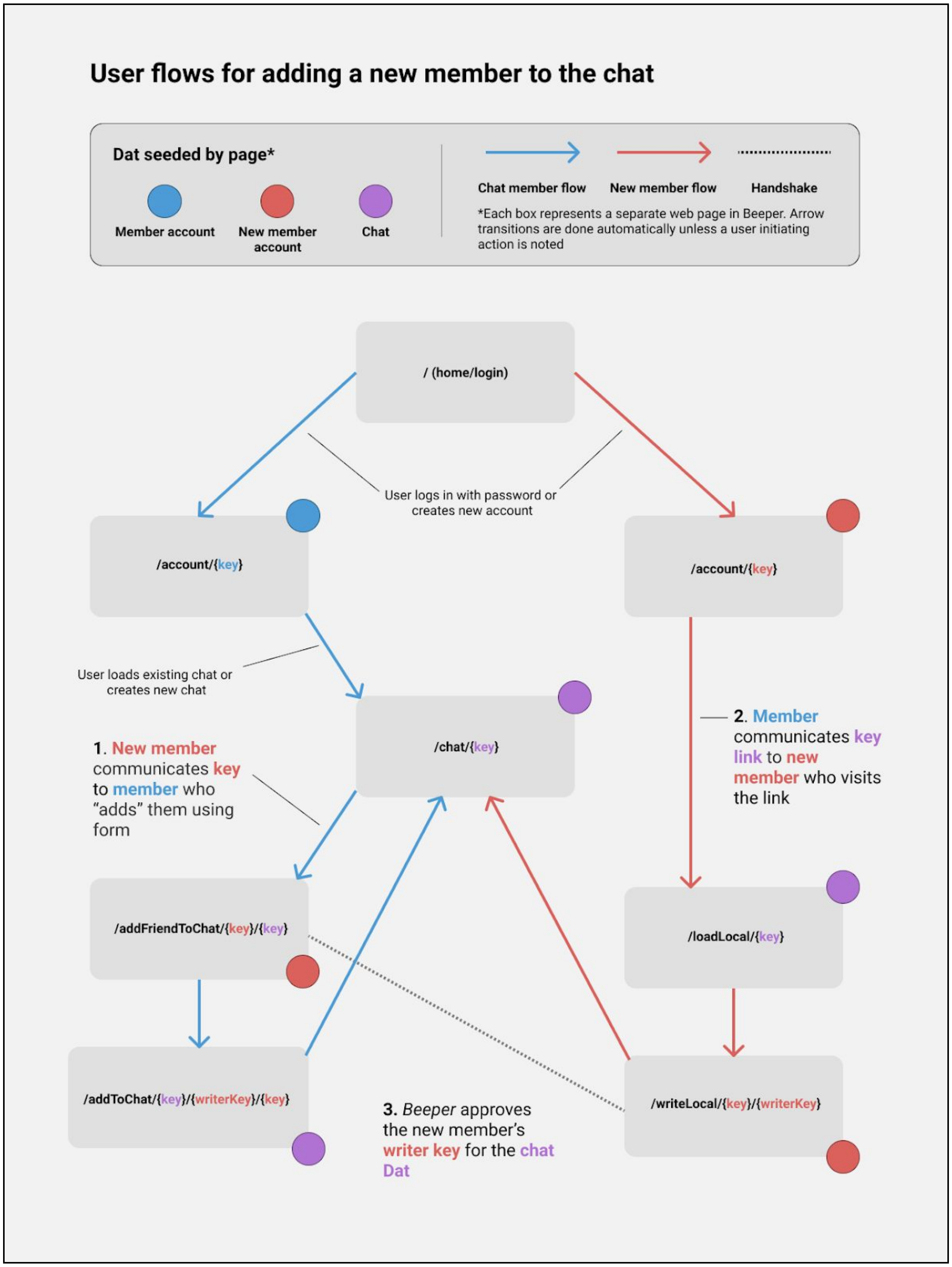

User flows for adding a new member to the chat
Dat seeded by page*
Member account
New member account
Chat
Chat member flow
New member flow
Handshake
*Each box represents a separate web page in Beeper. Arrow transitions are done automatically unless a user initiating action is noted
/ (home/login)
User logs in with password or creates new account
/account/{key}
/account/{key}
User loads existing chat or creates new chat
/chat/{key}
1. New member communicates key to member who "adds" them using form
2. Member communicates key link to new member who visits the link
/addFriendToChat/{key}/{key}
/loadLocal/{key}
/addToChat/{key}/{writerKey}/{key}
/writeLocal/{key}/{writerKey}
3. Beeper approves the new member's writer key for the chat Dat

Diagram of the user flow for adding a new member to a chat

Software

Bugs

There are some known bugs with the current version of beeper as it stands:

- Sometimes you need to reload the chat to see messages
- Upon successfully adding a new member to a chat there is a repeated display of messages (fixed upon refresh)
- The handshake process can fail at times due to choo emitter errors

Easily implemented

In its current form Beeper has the groundwork for easily extending it to use of the data and encryption model for many improvements:

- Adding a new device to access your account would use the same logic already implemented in adding a writer to the chat Dat but instead adds the writer to the account Dat
- In order to make it easier to use, the interface could display account info and chat names in the links on the home and account page (instead of keys). Implementation would mean replacing the key stored in the encrypted account file with a stringified JSON object.
- In order to implement attachments-like functionality, Beeper could also allow users to put encrypted files and images in the chat Dat using the chat secret key. Users could then choose from these files as attachments.
- Each message object already contains a signed version of the message but no verification is done when loading the messages.

Data Persistence

As the Dat protocol stands there are some limitations with access to up-to-date data in a multi-writer context. Taking chat at a point in time, the following is a ruleset for determining if user X has the most up-to-date messages from any user Y.

- X can always see their own messages
- X can always see the most up to date messages from online Ys
- X can see offline Y up to latest sync
    - Latest sync may not come from Y but could have synced with Y
- X is not always completely up to date, but can always view a local version of the chat which is up-to-date according with other chat members according to the above rules

These data limitations could be circumvented if someone were constantly seeding the chat. Beeper is currently limited to seeding a single Dat at a time.

Discussion

Requirements

Beeper fulfills the original requirements of the project in creating a secure application in which users handle their own data. The applications demonstrates the potential to use the Dat in a smart data sharing mechanism with full encryption. This is enabled by the multi-writer feature within the Dat protocol. In some ways the web application is limited due to the limits of connecting to the Dat through standardly available javascript functions in-browser. Due to this-- without members of the chat always being online, a user is not guaranteed to receive the most up-to-date messages from everyone.

Beeper has some limits to its practical use as an application. Firstly, the necessity of users to communicate long strings for keys can definitely be confusing and in some ways prevents the key-logic functionality from being intuitive to a user. Additionally, a user of an app like Beeper, while they might have to trust the application developer less, are still susceptible to phishing attacks. In its current state there is no account recovery or two-factor authentication within Beeper-- which are important to modern account usage and security. Account recovery might be possible through simple putting your account secret key in an email to yourself, but that's essentially emailing yourself your password. In all, Beeper demonstrates the potential for Dat to be a data model for a peer-to-peer application, but there are definite hurdles to making it viable for a production application.

Future work

There is a lot of opportunity for future development in Dat-powered web applications and Beeper specifically. For Beeper, the immediate future work focuses specifically on the improvements listed in **Implementation**. In order to make it more functional as an application, methods for improving the seed/sync process between known friends should also be focused on.

Furthermore, there exist a plethora of use-cases for web apps with peer-to-peer data models. Some examples include a multi-writer Wiki or possibly a micro-blogging platform (one with multi-device accounts[8]). Currently there are a multitude of efforts to write applications which leverage Dat.

[8] Fritter is a version of Twitter which uses Dat but is limited in functionality

The Project

I did not at all envision this being the project I would completed when I started, but I am quite satisfied with how it came out. Beeper had originally been just a part of a larger research project in user preferences, but due in part the difficulty of making it, I realized that this needed to be the focus of the project. I owe a lot to Jim Pick who created the first demo of a multi-writer application for Dat called Dat Shopping List[9], a collaborative shopping list. I used the application as a starting frame for the Beeper thinking it would be simple to add the necessary operation (it was not as simple as I anticipated). This really taught me the difference in understanding a technology, and understanding how to build (properly) with a technology. In order for me to actually understand how the project worked with archives and the Dat network took quite some time and furthermore in that process I understood better what Beeper would establish as a proof of concept. I hope to be able to continue to work on this application and within the Dat ecosystem as a whole.

[9] https://blog.datproject.org/2018/05/14/dat-shopping-list/

Works Cited

Ali, Muneeb, et al. *Blockstack Technical Whitepaper*. 2017, *Blockstack Technical Whitepaper*, blockstack.org/whitepaper.pdf.

Bedra, Aaron, et al. "Design Best Practices for an Authentication System." *IEEE Cybersecurity*, 2 June 2016, cybersecurity.ieee.org/blog/2016/06/02/design-best-practices-for-an-authentication-system/.

Krawczyk, Hugo, and Shai Halevi. "Randomized Hashing and Digital Signatures." *Randomized Hashing and Digital Signatures*, 2006, webee.technion.ac.il/~hugo/rhash/.

Krebs, Brian. "Facebook Stored Hundreds of Millions of User Passwords in Plain Text for Years." *Krebs on Security*, 2019, krebsonsecurity.com/2019/03/facebook-stored-hundreds-of-millions-of-user-passwords-in-plain-text-for-years/.

Ogden, Maxwell, et al. *Dat - Distributed Dataset Synchronization And Versioning*. Code for Science, 2018, *Dat - Distributed Dataset Synchronization And Versioning*, www.datprotocol.com/.

Perlroth, Nicole. "Yahoo Says Hackers Stole Data on 500 Million Users in 2014." *The New York Times*, The New York Times, 22 Sept. 2016, www.nytimes.com/2016/09/23/technology/yahoo-hackers.html.

Sawers, Paul. "WordPress Now Powers 30% of Websites." *VentureBeat*, VentureBeat, 5 Mar. 2018, venturebeat.com/2018/03/05/wordpress-now-powers-30-of-websites/.