AP Computer Science Final Project

William GOODALL

June 1, 2017

1 Motivations

Social networking sites, like Twitter, are vital to many of our most important conversations. It's vital to make sure that these conversations are displayed as we've intended them. By using cryptography, it is possible to make a social network which can't be artificially manipulated, even by its owners.

This project sets out to do just that. It uses cryptography to ensure that posts are exactly as their authors wrote them.

2 Methods

The general idea of the application is that every object must have its contents signed by its creator's private key. Every post, to be valid, must contain a valid signature, and every user record must contain a signature of its fields to be legitimate.

To accomplish this, every user generates an RSA keypair. This is stored in LocalStorage on the client. Then, to make a post, they sign it with their keypair, and send it to the server. The server then verifies that the post has a valid signature, and adds it to the database. In response to queries for user feeds, bios, or for individual posts, the server will include signatures for all data so the client can verify the authenticity of what it receives.

User information is also cryptographically verified. Using their account keypair, each user signs any updates to their username and bio. The server keeps the most recent of these updates, and returns it with any requests involving that user.

This has some interesting properties—account and post creation can happen entirely offline, and the user doesn't need to fill out any information to create an account. Just click a button, generate a keypair, and the app is fully functional. The server also doesn't need to perform any authentication of the requests it receives, other than checking that the content signature matches the content owner.

I implemented the app in two parts: the server, written in Node, and the client, written as a React SPA. The server provides an API to the client to create/read posts and change user information. The server, before it commits any changes to the database, verifies that all objects have the correct signatures. Conversely, the client verifies the signatures of any data it receives from the server. In the future, alternate server implementations (or some kind of federation) would be fairly easy additions.

The database user is Google Cloud's Datastore—a distributed hierarchical object store. Each user can be identified by the hash of their public key, and each post can be identified by the hash of its public key and contents. The schema can be loosely represented as follows:

```
User(<fingerprint>):
    username,
    id,
    signature,

    Post(<post hash>):
        contents,
        pubkey,
        timestamp,
        signature
```

The client is implemented as an SPA, using React for the view layer and Redux for state management. It is entirely static, and only communicates with the server through the server's API. This, coupled with offline post creation and profile updates, would make it fairly easy to modify the frontend to run offline, with a ServiceWorker for instance.

References

[1] *Crypto — Node.js v8.0.0 Documentation*. URL: https://nodejs.org/api/crypto.html.

[2] *Express 4.x - API Reference*. URL: https://expressjs.com/en/api.html.

[3] *How to hash a list of multiple items*. URL: https://crypto.stackexchange.com/questions/10058/how-to-hash-a-list-of-multiple-items.

[4] *Node.js on Google Cloud*. URL: https://cloud.google.com/nodejs/.

[5] *React Reference*. URL: https://facebook.github.io/react/docs/react-api.html.

[6] *RSA (Cryptosystem)*. URL: https://en.wikipedia.org/wiki/RSA_(cryptosystem).