FreeJournal

Background Technologies & Intro

Description & Motivation

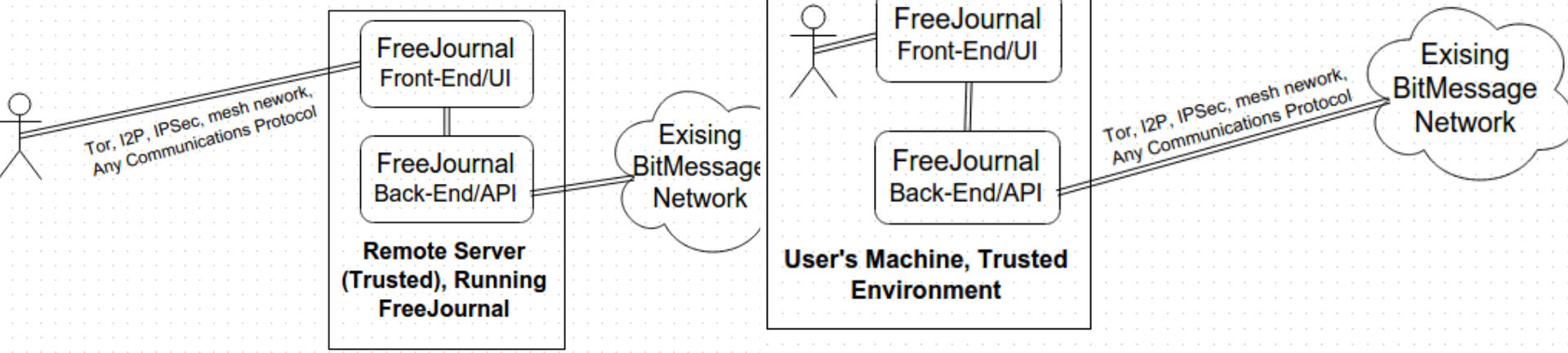
From Proposal: “Whistleblowers and journalists need a way to anonymously publish documents to a public or private audience of their choosing. Currently, they do so through complex anonymity software restricted to developers and cryptography experts, or through third-party institutions prone to internal politics and editorializing like Wikileaks. FreeJournal is a protocol and accompanying user-friendly front end application designed to assist in the anonymous and uncensorable release of documents to a public audience in a cryptographically secure manner.”

(read: https://wiki.cites.illinois.edu/wiki/pages/viewpage.action?title=FreeJournal&spaceKey=cs428sp15)

Project Architecture

Scenario 1: User Connects to Remote Service ("regular webapp"). Easy to use, less secure.

Scenario 2: User Runs Same Webapp As Server on **Local Trusted Machine**



Project Mockup: Submit

FreeJournal

View Your Uploaded Collections | Start a New Document Collection

Submit a New Document (text or PDF): **UPLOAD**
Document Title:
Document Keywords (Comma Separated):

SUBMIT

Project Mockup: View

FreeJournal

Select a document collection to view:

- Chase Consumer Banking Policies
- UIUC FOIA Request Responses
- Urbana Champaign Sewer Maps
- NYPD 2014 Arrest Statistics

Project Mockup: Collection

FreeJournal

You are viewing UIUC FOIA Requests (Go back)
Select a document below to view:

- Philip Daian FOIA request by FBI, 1/1/2005 *keywords*: student, FOIA, UIUC
  Documented uploaded before 5:00GMT, 1/1/2013 (trusted timestamp)
- Dennis Toeppen FOIA request for Sue Me, 1/1/2006 *keywords*: subex, FOIA
  Upload time unknown (no timestamp available)
- ….

Dependencies: Bitcoin

We will use Bitcoin to ***timestamp*** our documents. This allows us to distinguish old established leaks from new copies, providing a basic reputation system. This will be **optional for users**.

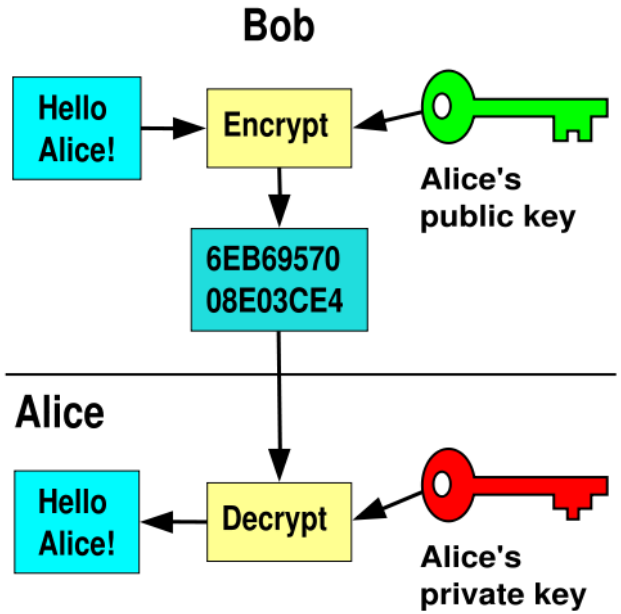
Bitcoin

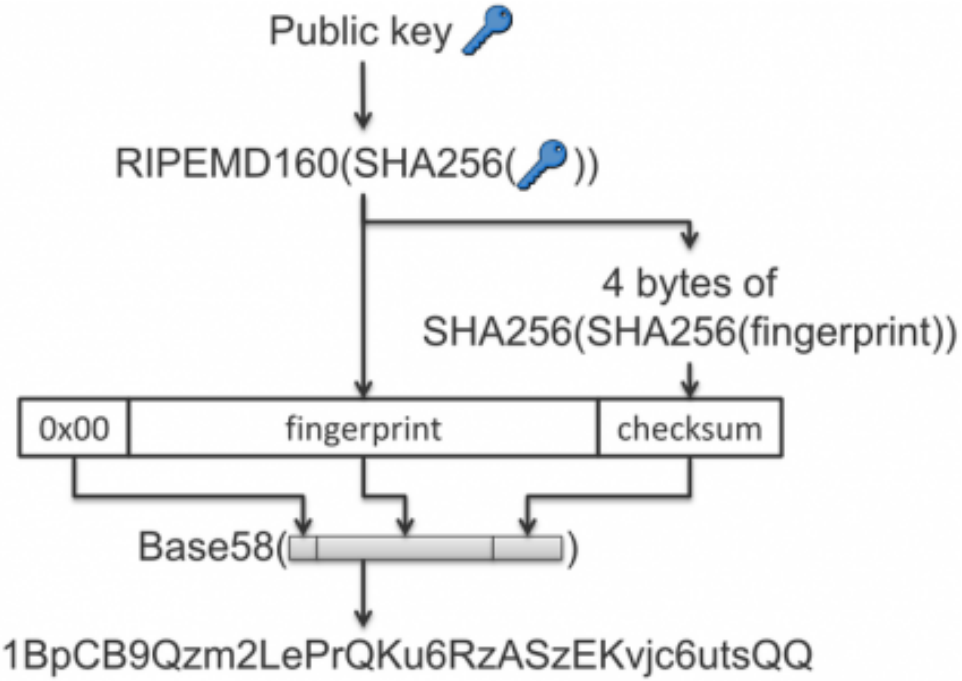
Bitcoin: What to know for this project

- What is a hash function and what does it give us? What are public and private keys and what do they give us? How do Bitcoin addresses relate to these concepts?
- What is a Merkle tree and why is it used?
- What is proof of work and why it's important
- Why and how can the Bitcoin network give us trusted timestamps for 40 bytes of data?

All are answered in https://bitcoin.org/en/developer-documentation and the Bitcoin Wiki

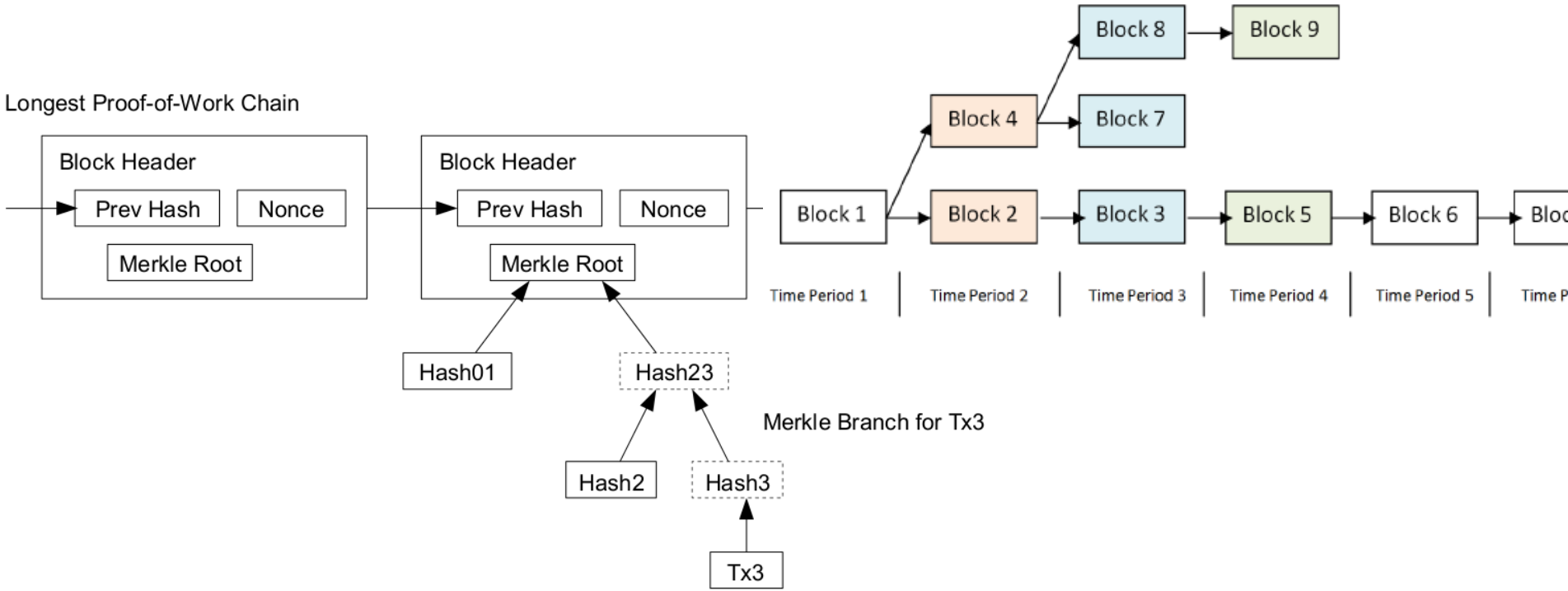
Bitcoin - addresses



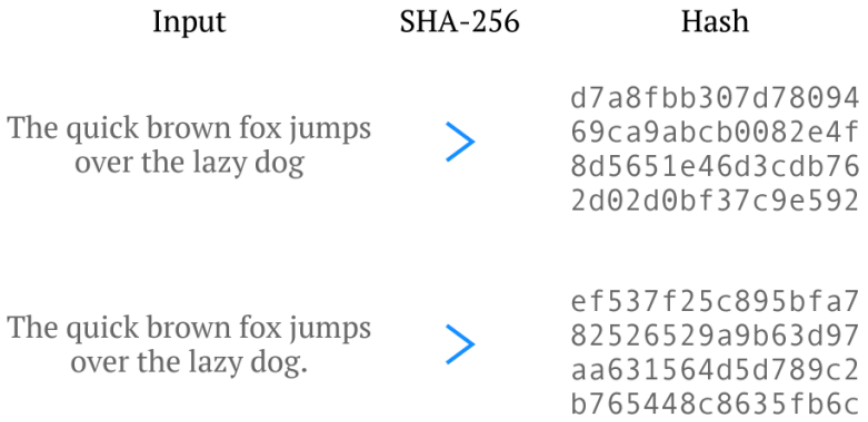


Read: https://en.bitcoin.it/wiki/Address

Bitcoin - the blockchain



Bitcoin - proof of work



x is valid iff hash(x)< target (hard to find such x)

Read: https://en.bitcoin.it/wiki/Proof_of_work

Bitcoin - trusted timestamp

- Inclusion in blockchain provides **history**. Blockchain is hard to modify and as transaction gets older - existence deep in blockchain can confirm **date of transaction** by providing resistance to attacks.
- So, we can include arbitrary data in the Bitcoin blockchain. This is **timestamped**.

Bitcoin - OP_RETURN

- How to include arbitrary data?
- Special type of transaction called **OP_RETURN** - input Bitcoins, output arbitrary data (up to 40 bytes). Enough to store single SHA-256 sum with short prefix.
- We can put the **merkle hash** of a collection of documents in OP_RETURN to timestamp.

Bitcoin - API

We will be (tentatively) using the **python-bitcoinlib** project to interface with the Bitcoin blockchain. An example of an OP_RETURN transaction is here:

https://github.com/petertodd/python-bitcoinlib/blob/master/examples/timestamp-op-ret.py

BitMessage

BitMessage attempts to build a secure system for peer to peer messaging.

Doesn’t use blockchain but adopts PoW, all nodes see all principles from Bitcoin.

Won’t go into full detail - please see https://bitmessage.org/bitmessage.pdf and https://bitmessage.org/wiki/Protocol_specification and especially https://www.bitmessage.org/wiki/API_Reference

BitMessage - Broadcast

BitMessage uses addresses that are public key hashes like Bitcoin - one of the message types you can send is a **broadcast** from an address to all *subscribers* of that address.

This is the feature we will leverage in our design.

BitMessage - DML

- Another interesting use of BitMessage - not needed for process but elaboration is here:

  https://bitmessage.org/wiki/Decentralized_Mailing_List

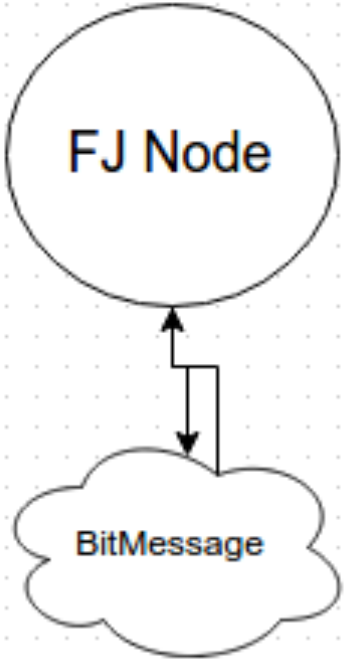
BitMessage - API

We will use **pybitmessage** - Python BitMessage API using RPC to communicate to the core BitMessage client.

More info here: https://www.bitmessage.org/wiki/API_Reference

FreeJournal Node Intro

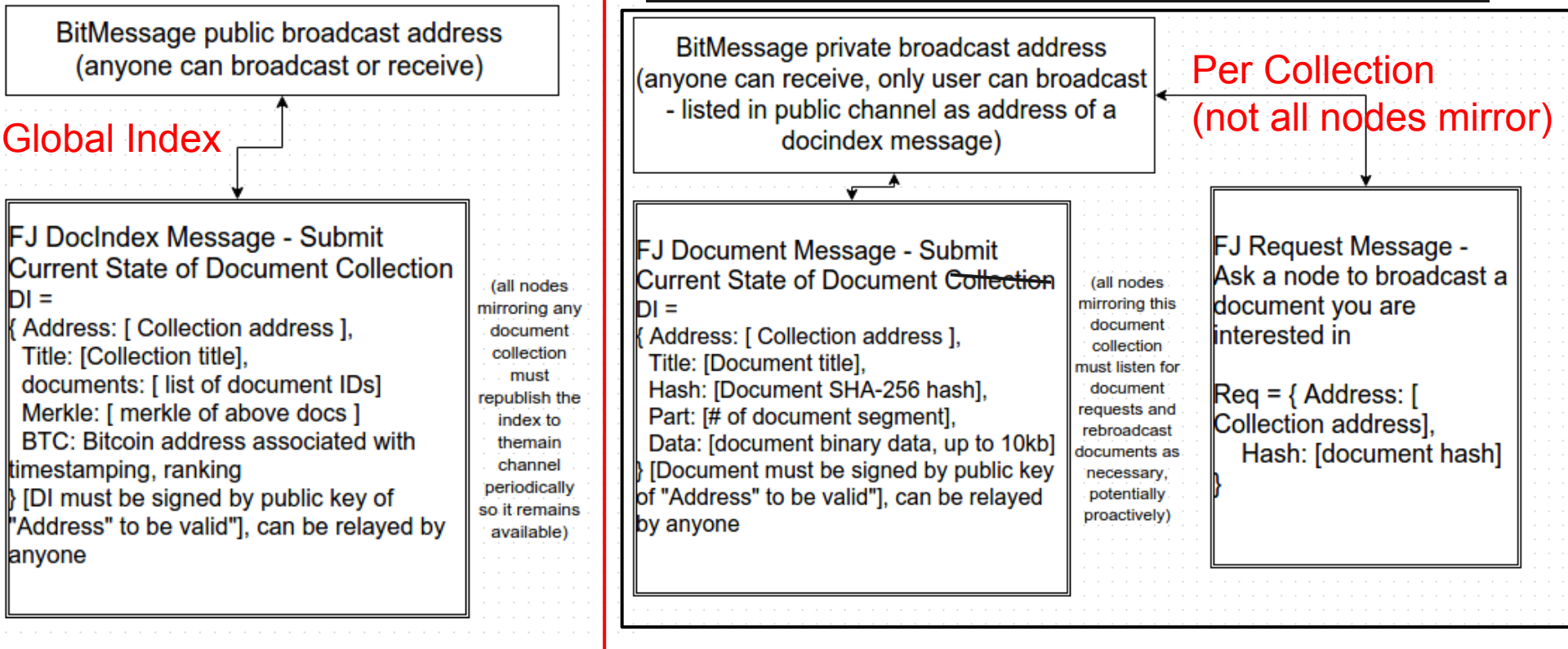
- Every user running the FJ backend API library is a **node**
- Nodes communicate with each other through BitMessage only, **never directly** - all FJ nodes are BitMessage nodes and indistinguishable from outside



Library/API Architecture

- One **global BitMessage broadcast address**, shared publicly (write/read access) coordinates/indexes **document collections**
- Each document collection has own BitMessage broadcast system, public read but potentially private write. Publishers and mirroring nodes publish documents to this channel, readers subscribe.

Library/API Architecture



Bitcoin Timestamping

- Any user (not necessarily publisher) can add trusted timestamp for any DocIndex after the fact by simply creating a Bitcoin transaction with the Merkle hash (“Merkle” field) in the OP_RETURN of a transaction to the listed BTC address. Users can vote on whether a leak is good or bad by sending even or odd valued transactions to that same address. Integrity of both features is protected by the Bitcoin network (strong anti-fragile guarantees).

Commenting

- Comments to be added after intial prototype finished.

  Theoretically, comments also published to each **“document collection”** and can also be checkpointed through Bitcoin blockchain for anti-fragile guarantees.

Logistics - Code & Docs

- https://github.com/FreeJournal
- Please sign up! We will use Github Wiki for internal developer and Scrum documentation. We will use MarkDown files in Github for user documentation.
- **TODO**: Document Scrum process, create testing infrastructure, document spec

Deliverables - Project

- FreeJournal protocol specification
- FreeJournal example API implementation
- Operational FreeJournal network
- FreeJournal web frontend
- Stretch Goal: FreeJournal desktop app

Use Cases: See proposal

Deliverables - Iteration 1

All:

- Familiarize with background technologies & API
- Prepare development environment
- Ask questions about anything unfamiliar

Individual:

- Prerequisites/Wiki, Testing & CI infrastructure setup (1 person)
  (Dan)
- Document API/library specification (3)
  (Phil, Wenxue, Fernando)
- Create frontend mockups and document requirements (1) (Walker)
- Setup Scrum & dev documentation, style guidelines,coding rules (1) (Walker)
- Document uploading & receiving on Bitmessage (2)
  (Brian, Drew)