Blockchain-Based Peer to Peer Secure Messaging

Eric Fleming Darin Smith Karen Tran Stephan Zharkov
University of California, Davis ECS 198 Spring 2016

Abstract: Messaging services such as Whatsapp, Slack, Kik and Facebook Messenger are widely used for both personal and work communication; Facebook Messenger has over 900 million monthly active users as of April 2016. All of these services share a defining characteristic of sending messages through a central server, which has security, privacy and trust issues. Measures such as end to end encryption of messages allow the creation of anonymous accounts and impose important security measures such as the deletion of messages. However, such measures either introduce other problems and are not guaranteed to be successful. Therefore, we propose creating a peer to peer messaging system based on the blockchain protocol. This would remove the massive point of vulnerability for attackers through the deficiency of the centralized server while eliminating the requirement of trusting that central authority. Further, it would provide irrefutable cryptographic evidence through the blockchain's property of non-repudiation, enabling the use of these messages for legal contracts or other professional services.

The problem we want to solve:

The main issues that a blockchain based Peer-to-Peer messaging app aims to resolve are the clear and inevitable security vulnerabilities of server-client based services. Server side messages allow for greater and more convenient access towards large numbers of users. It is no wonder as to why the messaging app WhatsApp had a recording breaking one billion users in February of 2016. The server-client based web model allows for fast, easy access between the client and the server, even with hundreds of thousands of interaction accessing the server daily. Needless to say, the large portion of users that access the server give fright towards extensive vulnerabilities of user information, compounded by further flaws that could potentially display messages between unwanted hackers and malware.

A common flaw of server side programming revolves around the usage of databases. SQL databases provide for comprehensive access towards making database queries, being used for everything revolving from user verification towards massive data lookups. However, if search queries to the server are unprotected, massive security flaws are at present as malicious

apps will be able to “inject” SQL code into the database system. Known commonly as SQL injection, this form of hacking and malware has the potential to destroy or replicate whole databases. On the other hand, a Peer-to-peer platform directly shares equal information between “nodes” or other computers that are part of the network, ensuring that only people who have access to the platform will be able to use the shared data. Considering that databases hold the number of communications between the users, with previous communications being easily accessible, a platform based on a blockchain creates a record of messages while ensuring the verification of the delivery of messages. Because hashing algorithms are considered to be “one-way”, there is no existing way of decrypting the messages.

Another important issue that compromises the security of common messaging apps is the rise of malware, or malicious software that commonly aims to collect private information, gain access to a computers or programs permissions, and lastly to disrupt computer operations. Malware can be injected to one’s internet browser, computer, and software if proper security measures are not taken. A download of a random software has the ability to access private information stored on ones computer, or being run constantly in one’s internet browser. While malicious files can still be sent via P2P networks, our average user holds privacy to the highest importance and will take proper precautions to ensure that their current machine will not be filled with malware. The current app idea will ensure only text-based messages are sent, until a secure and proper form of file distribution will be created to ensure a lack of malware.

Furthermore, a peer-to-peer system will ensure anonymity, as a client side system heavily relies upon user verification the creation of accounts that will be stored in more vulnerable databases. A system that solely communicates between a desired collection of computers will ensure privacy to the connected group, with encrypted and hashed messages that will ensure the safety and security of the shared messages.

Why it’s important:

Much of our messaging information, which includes text messaging, social media, and email, are stored in various centralized data centers throughout the globe. The advent of cloud technology further promotes such centralization, however this convenience can be of concern in regards to privacy and security. A block-chain messaging implementation could alleviate some of these issues, and provide decentralization.

Anonymous Addresses:

Text addresses can be used in a non-plain format. For example, users of Bitmessage, a peer to peer messaging app, would see an address similar to

- BM 2nTX1KchxgnmHvy9ntCN7sgKTraxczzyE

Which only includes information that allows other clients to communicate with your client if they were to send secure messages. There is no reason for one address limitation, as multiple addresses can be used and and generated

Encryption and Proof-of-Service:

Messages sent are encrypted with the public key from the recipient. Additionally, a proof-of-work system can be implemented to ensures a restriction on message sending, as seen with services such as Bitmessage which require the solving of a computational problem to send a message. This solving time limits the number of messages your computer can send in some given amount of time. A great advantage to this implementation would lead to the prevention of spam, a major problem in messaging. Cryptographically signed timestamps that are part of the block chain can serve as proof the message was written before the timestamp.

Additional Real World Prospects and Disadvantages:

The DoD (Department of Defense) is a real world example of where the benefits of blockchain messaging can be of great importance.

"Legacy messaging and back office infrastructures, traditionally based on centralized, unencrypted hub-and spoke database architecture, are expensive, inefficient, brittle and subject to cyber attack. The overhead costs of maintaining such architectures is rising rapidly. Many organizations unknowingly keep duplicate information and fail to ensure synchronization thus amplifying the potential for data theft and data corruption/rot," - DARPA (Defense Advanced Research Projects Agency)

Speed is one concern of the DoD, and a peer-to-peer system that relies on multiple connections could potentially be faster than its centralized counterpart. Security is also of big concern, so the reasons are clear as to why they considered the need of block-chain messaging platform as a secure messaging system. A record of all transactions, be it military contracts to troop movements or testimony to law enforcement, would soundly be kept as a record on this block-chain. More-so, many regions of the world have “denied communication environments,”

or regions where direct communication to centralized servers is not possible. This platform then offers a secure communication method in these zones back to headquarters in a timely and efficient manner, with very low risk of hackers identifying the messages.

No system is foolproof, as block-chain messaging has its fair share of disadvantages. For one, these systems are not as convenient to the novice user, as more knowledge of a block-chain client is generally required compared to a direct messaging counterpart. Sending large messages can also be problematic when considering the proof-of-work boundary that is proportional to the message size, as systems such as Bitmessage mentioned earlier require more computation time to send them. This is a substantial limiting factor on slower computers.

Implementation:

As mentioned above, the two primary goals of our messaging service are security (through removing the central server from the system) and permanent, irrefutable messaging. Therefore, a blockchain-based system seemed perfect, but how would that work? One option we considered was to base our service on the Bitcoin blockchain protocol, sending a minuscule amount of currency along with a large message field. This however seems clumsy, potentially expensive for heavy users, and Bitcoin's scripting language is intentionally not very full featured. More promising is the Ethereum generalized blockchain system, which is designed with creation of third party blockchain apps as a central, core function. These programs, called DApps by the Ethereum community, can be built in a variety of languages, such as C++, Python, Javascript, Go, Java and two custom DApp languages, Solidity and Serpent.

Ethereum also has a built in communication protocol for DApps called Whisper, which was designed to enable chat room apps that are not real time, or to use an example from the Whisper wiki, a service for whistleblowers to send information to journalists. That latter example was one usecase we envisioned for our service, so it seems perfect. In fact, the Ethereum team began development on a Whisper-based chat client, but never fully tested or launched it. Further, Ethereum is already built around the idea of running third party applications, so after development our messaging app could be run through their network, with no server required. Of course, that does require payment to the miners running the app, so a monetization model to compensate for that may be needed.

Our team experimented with Solidity, the custom DApp language, and considered Python, but have decided that Javascript with the Ethereum API is more practical for our purposes. This is for a variety of reasons: Firstly, Javascript (and web languages in general) run on nearly every device available, at least if you design with compatibility in mind. Messaging services are typically composed of a web client and mobile apps, which can both be built with Javascript, the latter via means of a web wrapper. Secondly, while Solidity is similar to Javascript, there are sufficient differences that it introduces both an unnecessary learning curve for development. That's particularly undesirable as if this project were to be open source, reduces the viability of that model, since far fewer people outside of our team would know how to contribute. Finally, with Javascript the user interface design would be done through Cascading Style Sheets, which are very user friendly. This is useful as good designers aren't always advanced programmers and vice versa.

Now that we have a platform (Ethereum) and a language (Javascript, with an accompanying HTML and CSS interface) picked out, there's the question of precisely what features should be implemented. Some of this has been described earlier in this paper but as a formal definition our service will consist of: 1) The capability to send private messages between two or several users, with anonymity if desired by the users. 2) A private group chat board, in the style of Slack/HipChat, again with optional anonymity. 3) Web, Android and Apple iOS clients, and potentially desktop applications as well for the main 3 platforms. 4) A permanent record or "ledger" of the messages, with the capability to access and share this ledger.

References and other related systems:

PAuth:

One example of a peer-to-peer authentication protocol is called PAuth. This security protocol has users' identities verified by their peers through zero-knowledge proofs; the results are then reported to a centralized server, and lessens the servers need for power to perform the computational verification for a password. Their goal is to prevents DoS (denial-of-service) attacks by overwhelming the attacker with an extensive amount of hash performances and computational requirement of password verification. The use of zero-knowledge proofs to verify a user's identity increases the communicative responsibility of a network. Thus, the concept of PAuth decreases computational responsibility of a centralized server and increases communicative responsibility for the peers involved in the network. In contrast to our blockchain

system, PAuth focuses less on protecting individual messages, and instead, relies on security in numbers similarly to Bitcoin. Just as Bitcoin is contingent on preventing attackers from gaining a majority of computational power, PAuth's security depends on attackers unable to control substantial amounts of centralized server connection.

The PAuth process is as follows: 1) The authenticating user provides the central server with their public key. 2) The server randomly selects k (number of packets the server must process for each authentication request) of peers and provides the user with a list of addresses currently linked to the network. 3) Authenticating user contacts each peer with public key to begin a zero-knowledge proof. 4) A nonce (random string) is generated, encrypted by a public key, and the results are sent back to the authenticating user. 5) User replies to each peer with decryption. 6) The user's response and original nonce are checked. YES or NO responses are sent to the server accordingly. 7) If the server receives k' number of YES replies within a certain timeout period, the authentication attempt is a success. Otherwise, the attempt fails.

BitMessage:

Another peer-to-peer authentication protocol closely analogous to our blockchain system is called BitMessage. BitMessage seeks to hide user identities, displacing the server connection with peer-to-peer messaging and concealing IP addresses that are otherwise open for "man-in-the-middle" attacks (or government controlled attacks by ostensibly secure communication through trusted web browsers). BitMessage guarantees the verification of original senders by authenticating complex addresses, such as the one mentioned above in the "Why It's Important" section under *Anonymous Addresses*. Similarly to Bitcoin, proof-of-work is required for each message. The time which the message is conceived is included to prevent malicious rebroadcasting of old messages. If a message is too old, it cannot be relayed. However, if the sender decides to rebroadcast the message, it may be done so using the private key of the user that it is bound to. Contrary to PAuth, BitMessage creates a hierarchy of nodes, each self segregating into child streams. For example, the Root Stream is the 1st parent stream. This stream is connected to a 2nd left child stream and a 3rd right child stream. The 2nd left child stream would have a 4th left and 5th right child stream, etc. This assists in saving hard drive space and reducing the processing power, as sending a message to all nodes directly would be inefficient.

One of the major differences between BitMessage and Ethereum is that Ethereum has a ledger which allows users to reference back to past messages. BitMessage is also only accessible to a limited subset of desktop computers, while Ethereum can be accessed on any device using a web browser because we implement our protocol using Javascript rather than Python.

Works cited:

Gao, Zijing, Thomas Lu, and Anand Srinivansan. “PAuth: Peer-to-peer Authentication Protocol.” *Courses.csail.mit.edu.* 20 Mar. 2012. Web. 29 May 2016.

Rao, Leena. "Facebook Messenger Hits 900 Million Users." *Fortune News.* Fortune, 2016. Web. 26 May 2016.

Yeung, Ken. "WhatsApp Passes 1 Billion Monthly Active users." *VentureBeat*. 1 Feb. 2016. Web. 29 May 2016.

Warren, Jonathan. "Bitmessage: A Peer-to-Peer Message Authentication and Delivery System." *Bitmessage.org*. 27 Nov. 2012. Web. 29 May 2016.