Ethereum Smart-Contracts for Digital Time Stamping
Using the Ethereum Network as a Trustless Time Stamping Authority

TYLER SZETO [ROUGH DRAFT]
tylerszeto@gmail.com
July 30, 2018

Authors note: This is a pre-print / working draft of the paper and is subject to change.

Abstract

A cryptographically secure decentralized financial network that allows for the use of "smart contracts" creates a platform which is capable of providing a trustless digital time stamping service. In this paper, we discuss the inherent advantages the Ethereum network has for housing a robust and dynamic digital time stamping protocol, as well as outlining our model of using smart-contracts for digital time stamping.

Introduction

Generally speaking, most cryptocurrencies have a well-tested track record of being secure. Currently, the Ethereum Network uses Proof-of-Work, in which a substantial amount of computing power is expended to produce every new block.[1]. Every new block introduces new Ether into the cryptosystem, which, at the moment, contains real-world trading value [1]. Because creating blocks requires energy, the longest-running chain is acknowledged as the most "valid" chain by participating nodes [13]. If a user wishes to modify the blockchain, the user would have to spend a substantial amount of computing power competing with all the other honest miners on the network. If a group of users wishes to collaborate together to modify a previous block, they would also have to compete against the network and discard any previous financial gains they previously made on the current chain.

What this outlines is a system in which, if a user wishes to modify a previous block, is met with computational discouragement. Not only does the blockchain guarantee that work was committed, but also time. With a predictable amount of time being generated between each block, one can relatively predict how far one must go back in time to reach a certain block. Blocks are time stamped, and a user can acknowledge that a block was created at a certain time due to the time required to make a new block.

[1]Ethereum Plans on Forking into "Casper", a Proof Of Stake security system[3]

This allows for a blockchain to also be used for digitally time stamping files. There is a very definitive cost to forge a time stamp when using block-chain technology. Contrasting to a trusted timestamping service, if incentivized with enough financial gain, may deceitfully sign a hash with a deceptive timestamp if the financial incentive is enough. No other participants would be able to sufficiently prove that the signature was deceitful.

In this paper, we outline the framework for using the Ethereum network for a robust signing platform, in which users can create hidden commitments, verify without trust that information was signed at a given time, and allowing multiple individuals to partake in a time stamp.

0.1 Previous Work

Using blockchain technology as a means for trustless digital timestamping is not new, albeit the concept has not been extensively explored. Origin Stamp outlines a contemporary use of digital time stamping - data is computed into a hash and is included within a Bitcoin transaction [4]. Another user may verify that the hash included within the transaction relates to a specific file. Receipts are generated in the form of a PDF, and the transaction can be broadcasted to twitter.

"Stampery" utilizes Merkle trees to aggregate multiple data sets to aid with the scaling issue of blockchain technology [5]. Stampery also submits copies of the time stamps to multiple blockchain services *(such as Etheruem, Litecoin, Ripple, Bitcoin)* as a part of their architecture, as well as emailing receipts.

Another similar project is Commit Coin, where commitments may be "carbon-dated", and revealed at a later time through the use of cryptographic puzzles [7].

0.2 The Etheruem Network

The Etheruem Network **(Etheruem)** describes itself as "..a decentralized platform for applications that run exactly as programmed without any chance of fraud, censorship or third-party interference" [2]. Utilizing blockchain technology very similar to Bitcoin, Etheruem allows for participants to create **smart contracts**: scripts that are computed on the Etheruem Virtual Machine *(EVM)* in exchange for Ether. Values submitted or produced in the script are stored in the blockchain similar to how a transaction would be documented.

0.3 Trusted Timestamping

In order to contrast decentralized time stamping with trusted time stamping, we briefly outline a trusted time stamping protocol. ANSI ASC x9.95 STANDARD specifies the requirements for a centralized trusted time stamping authority [8]. We superficially outline an example of a trusted time stamping protocol.

1. **Signing**
   (a) User has file `k` and computes `h = H(k)` where `H()` is some cryptographic hash function.
   (b) TSA creates a file `ts` *(time stamp)* such that `ts = h ‖ time` where ‖ is a concantonation and `time` is the time of signing.

(c) TSA signs `signing = Sign(H(ts), privateKey)` where `Sign()` is some signing function and `privateKey` is the TSA's private key.

2. **Verification**

   (a) Verifier checks that `ts = h ‖ time` and that `Ver(sign, H(ts), publicKey)` is true where `Ver()` is a signature verification function and `publicKey` is the public key of the TSA.

In summary, data is aggregated into a hash, which is sent to the TSA. The TSA creates a digital signature dependent on the received hash concatenated with the time of signing.

For verification, verifiers verify that the contents of the file, the time stamp generated by the TSA, and signature produced by the TSA match with the TSA's public key.

1 Weaknesses of Trusted Time Stamping

As is with any centralized authority, there are innate disadvantages in regards to trust.

1. **Security** If Alice has a file genuinely signed by a TSA at a given point, and if the TSA were to have their private key is stolen, her previous time stamps are effectively rendered void. Other actors cannot verify the legitimacy of the said time stamp. Therefore, Alice needs to place a certain degree of trust in the security of the TSA if her timestamps are to be respected over many years [6]. Stampery proposes using Transient Key Cryptography to solve this issue but identifies doing so is technically demanding and more difficult to implement [5]. "ProofMark" uses Transient Key Cryptography to solve this issue [9].
2. **Bribary** Authoritative systems are susceptible to bribes if the financial gain is lucrative enough. For example, if a TSA is only grossing $50,000 a year in revenue, and is offered $25,000 to falsify a time stamp, it may be in their best interest economically to maliciously commit forgeries.

2 Advantages of Digital Time-Stamping in Blockchain Based Architecture

Blockchain-based architecture solves the issues outlined in the previous section with a trustless, decentralized system.

2.1 Open Platform

Blockchain projects are open source and are therefore are openly auditable from anyone. If the Ethereum-Network is found to be insecure, there are more important factors than time-stamping being jeopardized. Currently, with a market cap of 45 billion USD, there is much more being incentivized to keep the Ethereum Network Secure than a TSA [10]. Stampery advertises the advantage of open-source projects a security feature, claiming that there is a 6 billion dollar bug

reward for bitcoin that is still unclaimed. While the Etheruem project is still in beta, there is an active community currently seeking bugs within the protocol itself [11]. As a consequence, when the Etheruem Network is audited for weaknesses, so too indirectly are the smart contracts running off the EVM.

Furthermore, blockchain based time-stamping protocols do not need a hidden private key to generate a time stamp. Everything is transparent within the network and is openly verifiable to any individual. As long as Etheruem maintains any intrinsical value, the time stamp is secure and permanently stored in the chain [4].

2.2 Financially Determanistic

Unlike a TSA, where a bribe may take place for a variety of reasons, (financial incentive, nepotism), the blockchain infrastructure has a deterministic price for how much it would cost to undo blocks. To commit a fraudulent timestamp in a blockchain system, an attacker may:

1. Bribe miners into undoing a chain to submit a time stamp earlier.
2. Compete against the honest chain in terms of mining to submit a time stamp earlier.

The challenge to forge a time-stamp is the same challenge for an individual to attempt to commit a double spend attack on the Ethereum Network - they are committing to the same problem of modifying a previous block.

3 Architecture

3.1 Smart Contract Diagram

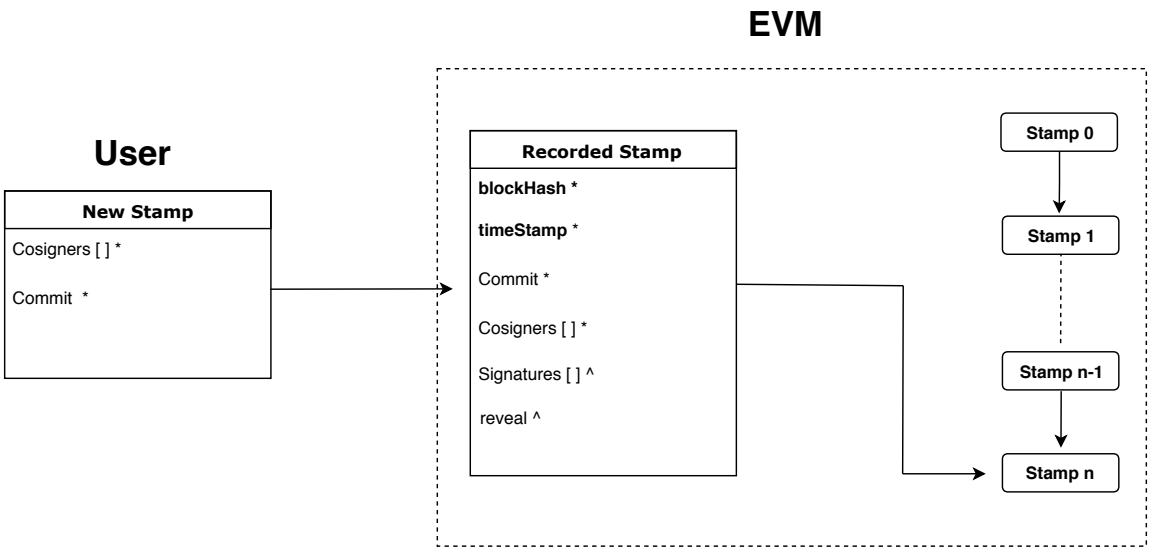

Figure 1: Sumbitting a new contract ot the EVM. See *remarks* for notation.

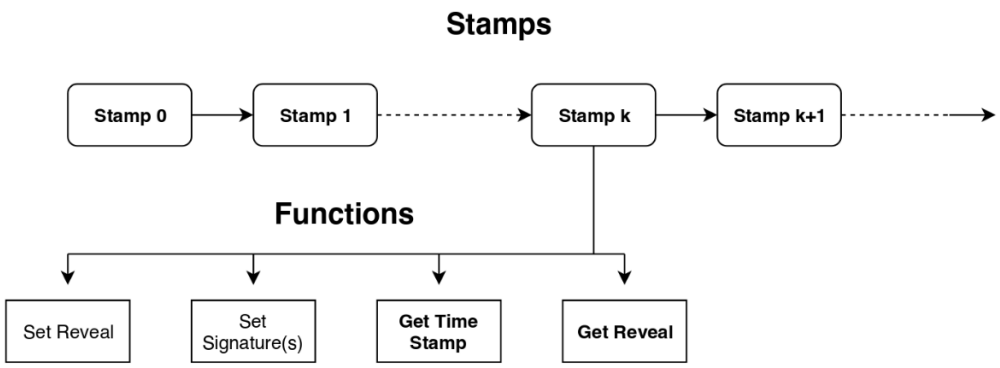

Figure 2: Accessing Stamp k within the smart contract.

3.2 Remarks

A flow chart showing how a new stamp is stored within the smart contract. We introduced the following notation:

1. * represents immutable information
2. ^ represents information, once set, is immutable.
3. **Bolded text** represents information generated by EVM

3.3 Publishing a Time Stamp to Smart Contract

Inputs. Publisher has the set of cosigners $[CS]$ and the hash `k`.

1. **Setup.**
   (a) Publisher securely chooses a value *salt* such that *salt* is derived from a secure random number generator, and computes `commit = hash(k, salt)`.
   (b) Publisher submits a new timestamp consisting of `commit` and the set $[CS]$ to the smart contract in the EVM.
   (c) EVM creates a new time stamp at index `x` where `x` $< 2^{256}$ and `x-1` was the last submitted time stamp (i.e incriments linearly) and returns `x` back to the user.
   (d) EVM computes `noncedHash = hash(commit, blockHash)` where `blockHash` is the previous block's hash. Publisher requests `noncedHash` from the smart contract at `x`.

1. **Signings**
   (a) For all $cs \in [CS]$, cs verifies `hash(blockHash, hash(k, salt)) = noncedHash` in the smart contract at `x`.
   (b) For all $cs \in [CS]$, cs signs `cs.signature = Sign(cs.privateKey, noncedHash)` and submits `cs.signature` back to the smart contract at `x`.
   (c) EVM verifies `Ver(cs.signature, cs.publicKey, noncedHash)` in the smart contract at `x` is true.
   (d) When all $cs \in [CS]$ is validated, EVM documents `timeStamp` for the smart contract at `x`, where `timeStamp` is an unsigned integer representing EPOCH time.

1. **Reveals**
   (a) Any user that knows of `k` and `salt` submits both to smart contract at `x`.
   (b) EVM verifies that `commit = hash(k, salt)` for smart contract at `x`.
   (c) EVM updates `reveal = k` within the smart contract at `x`.

3.4 Receipt

Description Receipt of a time stamp

1. **Receipt Contents**
   (a) Smart Contract Address.
   (b) Time Stamp Index, `x`.

(c) `hashOfFile`.

(d) `salt`.

(e) Signers $cs \in [CS]$.

(f) `noncedHash`.

(g) Time stamp `datedStamp`.

3.5 Time Stamp Verification

Inputs. Verifier has receipt.

Goal. Client side validation of a time stamp

Owner:

1. **Setup.**

   (a) Verifier verifies that
   $\forall$ `y [y` $\in$ `Reciept.Signers` $\iff$ `y` $\in$ `SmartContract[x].Signers]`

   (b) Verifier checks that `Reciept.hash = SmartContract[x].hash`

   (c) Verifier checks that `Reciept.datedStamp = SmartContract[x].timeStamp`

4 Design Philosophy

The Ethereum Virtual Machine offers many advantageous in regards to user security, validity, and authenticity when compared to Bitcoin timestamping. In this chapter, we outline the issues inherent to simply publishing a hash to the network, and address how using smart contracts solves the said issue. This section is to explain the obscure design choices laid out in the previous section.

The ultimate goal is to reduce interactivity between prover(s) and validator(s).

4.1 User Privacy

Knowing an unsalted hash reveals *some* information about the contents of the file. This is understood in the context in which an actor has a rainbow table of hashes for specific documents [12].

In the context of a block-chain based TSA protocol, if another individual is scanning the blockchain for hashes of files they know the pre-images of, the said individual can become immediately aware if an individual published one of their hashes. Furthermore, it also possible to undo the publication in a double spend attack if one has the computing power.

In our protocol, an individual monitoring the Ethereum Network will have learned nothing about the published hashes, even if the two parties share the same file. However, when a user reveals the salt and hash, the information is publically verifiable. Even if another party attempts to censor a reveal, any individual can verify off chain that

```
commit = hash(fileHash, salt)
```

This allows for an individual to publish a time-stamp with confidence that the hash is initially kept private, but is still revealable at a later date.

4.2 Validity

Allowing users to sign information published to the network with a separate set of keys allows users to easily prove ownership of a time stamp. Consider a problem using naive blockchain time stamping - a user can easily claim that a time stamp published to the blockchain was theirs. For example, Alice publishes the hash *hash* of her file *marchMadnessPredictions.txt* to the blockchain as *tx1*. If Bob also has access to the same file, Bob can tell people that *tx1* was his time stamp, and people could believe him granted he reveals the pre-image. We refer to this as a "squatting" problem.

Alice could use the same private key involved in the transaction to prove ownership, but doing so is interactive and highly not recommended. [2]

Requiring two separate private keys, one exclusively meant for the Ethereum Network (signing transactions) and one for proving validity (from here on known as validity keys), allows one to be more liberal with time stampings. Users may publish their validity keys to various third-party social media websites, and other users may verify they are the same keys within a time stamp.

This solves the "squatting" problem, in which there is a definitive public key meant exclusively for file signing. Owners may sign custom messages on third-party sites with their validity keys to prove they are the true owner.

4.3 Coosigning Information

It is uncommon for a document in the real world to only be signed by one individual. In many cases where files need to be time-stamped and signed, there are usually multiple actors involved.

As a result, the protocol allows for the signings between multiple parties. One user may broadcast a time stamp to the network with multiple validity keys. The other signers sign the relevant hash necessary and submit it back to the contract. The dating of the time stamp is only recorded when all parties successfully sign the relevant hash.

4.4 Replay Prevention

Suppose in the previous example, cosigners simply signed the hash published to the network. Another user may collect the signatures and re-submit it at a later date, to try and deceitfully claim the hash was submitted later. Though the original owner(s) can prove that the hash was published earlier, doing so would be interactive.

Using the current block hash as a means of noncing information offers a clever way of preventing replay attacks. Information published to a smart contract is ultimately left out of control to the user - it is impossible for a user to predict what information the user will then have to sign later.

Intuitively, this means that any information signed by a user cannot be used in a separate block, mitigating the potential of a replay attack.

[2] It is highly unrecommended to use Bitcoin private keys outside of the context of traditional spending. Traditionally the private keys are kept encrypted, and only momentarily decrypted to sign a transaction. We see this in services such as electrum. Exporting private keys to clear text is not recommended. It is recommended to change public keys whenever possible.

5 Conclusion

We discussed the inherent advantages in using the Etheruem network as a trustless time-stamping authority. The protocol allows for secure and verifiable interaction directly between the participant and the blockchain, in a way that reduces interaction between provers and validators, and is also trustless. The protocol allows for three major features that cannot be solved with traditional blockchain time stamping: hashes are allowed to be hidden with the use of commitments, hashes may be signed in such a way to prevent replay attacks, and hashes are signable between multiple parties. This allows for a more robust and decentralized time stamping platform such that interaction is minimalized and is difficult to censor.

References

1. A Next-Generation Smart Contract and Decentralized Application Platform
   `https://github.com/ethereum/wiki/wiki/White-Paper`
2. Etheruem Website
   `https://ethereum.org/`
3. Casper
   `https://blockonomi.com/ethereum-casper/`
4. Decentralized Trusted Timestamping using the Crypto Currency Bitcoin
   `https://www.gipp.com/wp-content/papercite-data/pdf/gipp15a.pdf`
5. Stampery BLock Chain Time Stamping Architecture
   `https://s3.amazonaws.com/stampery-cdn/docs/Stampery-BTA-v6-whitepaper.pdf`
6. Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)
   `https://www.ietf.org/rfc/rfc3161.txt`
7. Commit Coin
   `https://eprint.iacr.org/2011/677.pdf`
8. American National Standards Institute (ANSI) (2005).
   `X9 x9.95-2005 trusted time stamp management and security. Standard.`
9. ProofSpace White Paper.
   `http://fios.com/proofmarksystemtech.pdf`
10. Etheruem Market Cap (taken June-26-2018)
    `https://coinmarketcap.com/currencies/ethereum/`
11. ETHEREUM Bounty Program
    `https://bounty.ethereum.org/`
12. Rainbow table to crack password using MD5 hashing algorithm
    `https://ieeexplore.ieee.org/abstract/document/6558135/`
13. Bitcoin: a peer to peer electronic cash system
    `http://bitcoin.org/bitcoin.pdf`

14. ELECTRONIC SIGNATURES IN GLOBAL AND NATIONAL COMMERCE ACT
https://www.gpo.gov/fdsys/pkg/PLAW-106publ229/html/PLAW-106publ229.htm