Two-tier blockchain timestamped notarization with incremental security

Alessio Meneghetti[1], Armanda Ottaviano Quintavalle[2], Massimiliano Sala[1], and Alessandro Tomasi[3]

[1] Department of Mathematics, University of Trento, Trento, Italy
[2] LDSD, Department of Physics and Astronomy, University of Sheffield, Sheffield, UK
[3] Security and Trust, Fondazione Bruno Kessler, Trento, Italy

Abstract

Digital notarization is one of the most promising services offered by modern blockchain-based solutions. We present a digital notary design with incremental security and cost reduced with respect to current solutions.

A client of the service receives evidence in three steps. In the first step, evidence is received almost immediately, but a lot of trust is required. In the second step, less trust is required, but evidence is received seconds later. Finally, in the third step evidence is received within minutes via a public blockchain.

1 Introduction

The solution here described was commissioned to provide a customer in the financial sector with evidence to corroborate its statement of the integrity, authenticity, and existence at a given time of its data. This is closely related to the problems known as *secure timestamping* and *notarization*. The commission was made with the very specific requirement that it should make use of a privately run blockchain-based service anchored to a public blockchain, but at the same time still be capable of working without the private ledger, if it should cease to operate. We find it interesting to discuss how this design challenge can be met, and what security guarantees it can offer.

Digital timestamping and blockchain have been linked from inception. The bitcoin whitepaper [14] explicitly cites the linked timestamping work by Haber and Stornetta [8] with Merkle trees [13] as efficiency improvement [4].

Other blockchain-based timestamping solutions have been recently proposed. The authors of [7] propose to commit aggregate data hashes to a bitcoin transaction. On the other hand, the authors of [5] report on existing solutions that use the data hash as a bitcoin address to which to spend a transaction (BTProof, now unavailable), or embed a custom string and a data hash in the transaction's data field (Proof of Existence [3]); the authors themselves extend the former by using three addresses per transaction: one encoding the name, one encoding metadata, and one encoding the data hash itself.

As pointed out in [12], there are several layers on which blockchain provides guarantees - of consistency, security, etc. - each with potentially different tools to check that these guarantees hold. The present solution does not offer a programming language for smart contracts and makes no specification as to the networking or consensus protocols.

The novel aspect of our solution is that a client of the service receives evidence in three steps. In the first step, evidence is received almost immediately, but a lot of trust is required. In the second step, less trust is required, but evidence is received seconds later. Finally, in the third step evidence is received within minutes via a public blockchain. We achieve our results thanks to the interaction between two blockchains, one of which is public.

After some notation and preliminaries in Section 2, we give a semi-formal specification of the solution in Section 3. Sections 4 and 5 contain our security proofs. Section 6 contains a discussion about possible DOS attacks, while Section 7 hosts our conclusions.

2 Notation and preliminaries

Let time be measured in intervals $[a, b) = \{t \in \mathbb{R} \mid a \leq t < b\}$, for any $a < b \in \mathbb{R}$. Let $\sigma_A()$ be a digital signature algorithm using the private key of data owner A; if the owner can be determined unambiguously, we may sometimes omit A for clarity. We assume σ to be resistant to impersonation attacks: an attacker D cannot obtain $\sigma_D(m)$ from $\sigma_A(m)$. We also assume σ to be unforgeable. For example, ECDSA satisfies both security properties under the usually-assumed hardness of the DLOG problem in (strong) elliptic curves.

Let $\|$ denote string concatenation. As a shorthand when dealing with containers of data with attached a signature of the contents, e.g. block headers, we will commonly employ expressions such as

$$s = T \parallel \sigma(s), \tag{1}$$

where T are the contents, the signature is computed as

$$\sigma(T \parallel \underline{0})$$

with $\underline{0}$ a string of zeroes of the same length as $\sigma(T \parallel \underline{0})$, and finally the container s is composed by replacing $\underline{0}$ by $\sigma(T \parallel \underline{0})$. The shorthand $\sigma(s)$ is employed to indicate that the signature refers to the entire contents, without writing them explicitly at length.

Throughout this paper we denote with $\eta_A(m)$ a public-key encryption of message m with public key of actor A; with $\mathcal{T}\{j\}$ a Merkle tree of a list of items j; and with $h(m)$ a cryptographic hash function, in the sense of [15], of message m. In particular, the collision resistance of $h()$ is required in order to deduce that the forgery of a path in the Merkle tree would imply a collision.

The seminal work on digital timestamping is due to Haber and Stornetta [8], on which widely used standards for trusted timestamping today are based, such as RFC 3161 [10], ANSI X9.95, and ISO 18014 [11].

We present the following two algorithms to recall these important methods and establish a common notation, though we do not make use of them directly in our solution.

Algorithm 1 (Trusted authority timestamping)**.** *In [8], the client holding data d sends its hash $h(d)$ to a trusted timestamping authority $\mathcal{A}$, which returns a signed statement τ of the time of receipt, t:*

$$\tau = h(d) \parallel t \parallel \sigma_{\mathcal{A}}(\tau). \tag{2}$$

RFC 3161 is substantially the same scheme, but an additional hashing step is introduced:

$$\tau = h(h(d) \parallel t) \parallel \sigma_{\mathcal{A}}(\tau) \tag{3}$$

These schemes place all trust in the hands of the authority $\mathcal{A}$, but in practice trust is distributed among several stakeholders with successive timestamps.

By using hash trees, more sophisticated timestamping algorithms have been developed (see e.g. [4, 9]).

Algorithm 2 (Tree-linked timestamping)**.** *At step k, define a time interval $[t_{k-1}, t_k)$. The server $\mathcal{A}$ collects all requests $\Theta_k = \{\theta_{k,i}\}_i$ received in the time interval, and builds the Merkle tree $\mathcal{T}(\Theta_k)$. We denote its root with r_k and we call it* interval root*.*

The interval roots are linked together by the following rule, thus forming a chain of hashes $\{R_\ell\}_{0 \le \ell \le k}$:

$$R_0 = 0$$
$$R_\ell = h\left(R_{\ell-1} \parallel r_\ell\right)$$

The values $\{R_\ell\}_{0 \le \ell \le k}$ are placed in a widely available repository. The server then returns to each requester a receipt with the time t_k, along with the path in the Merkle tree from the requester's leaf up to the value R_k.

The authors of [9] themselves point out that the integrity of the public repository of root hashes is the only requirement on which the authenticity of a document with receipt relies.

3 Solution description

We now describe our solution. We do not require that blocks are created at fixed-time intervals, but we require a time division in intervals. To be more precise, since each block hosts the time of its creation, we can consider time intervals using the index k ($k \ge 0$), so that interval k corresponds to $[t_{k-1}, t_k)$ for $k \ge 1$ and interval 0 corresponds to time $t < t_0$.

3.1 Architecture

The service handles interactions among the following participants:

- The clients of the service wish to provably record the existence of data d at a given time. C denotes an arbitrary client.
- Some service nodes check proposed transactions for validity, provide evidence of record to clients, and maintain a blockchain, which we will call *proxy blockchain*. N and M denote nodes. At block creation time, one of the service nodes acts as committing node and thus prepares a block, which is submitted to the other service nodes for acceptance.
- One auxiliary node A commits further evidence to a public ledger L, used as a reference clock and trusted timestamping service, and monitors client transaction activity. A never acts as a service node.

The service relies on a trusted certificate authority to provide the public key infrastructure necessary to both identify the nodes with permission to participate in the service and provide the ability to digitally sign documents. We assume that the proxy blockchain is at least permissioned if not private, and that it is a robust ledger in the sense of [6], i.e. guaranteeing persistence and liveness. We do not address the backbone protocol and the consensus algorithm, in particular we make no specification as to how the information of transactions propagates, or how the node that shall create the next block is chosen, or how the other service nodes accept its proposed block.

The main function of our service is delivered by enabling a client C to prove existence of some data at commit time t, based on some evidence. We provide evidence in the form of

digitally signed receipts and blockchain and C receives evidence with *incremental trust*, in three successive steps, as sketched below:

1. Evidence issued by the service node receiving data.
2. Evidence issued by the service node that create blocks in the proxy blockchain.
3. Evidence issued by the auxiliary node A and hosted by public blockchain L: A issues some evidence, which is partially stored in a public repository L, such as the bitcoin network.

In the first step, C receives a first signed receipt. In the second step, C receives a second receipt and is able to access some evidence on the proxy blockchain. In the last step, C receives a final receipt and is able to access some evidence on a *public* blockchain. We speak of incremental trust because the probability of C colluding with the other actors decreases significantly from one step to the next.

3.2 Block creation and Node receipt issuance

We now describe in detail the first phases of our system. We assume that all clients are operating in time interval k.

Transactions
Each client C, identified by a digital identity ι_C, creates a self-signed statement τ and sends it to one of the nodes M for validation, which becomes in notation (1)

$$\tau = \sigma_C(d) \parallel t \parallel \iota_C \parallel \sigma_C(\tau) \tag{4}$$

This statement τ is analogous to a transaction in popular blockchain solutions, so we will call it *transaction*. M checks the signature in τ for validity and the claimed time t (i.e. $t > t_{k-1}$). If the check is successful, M broadcasts τ to the other nodes and sends $\sigma_M(\tau)$ back to C, which acts as the *first receipt*. We call M the *validator* of transaction τ.

Block creation
The next committing Node N then constructs a block by the following procedure. We assume that k blocks have already been created, with block $k-1$ being the last created[1], meaning that all the following actions by N are implicitly related to time interval $[t_{k-1}, t_k)$.

Let T_C be the set of valid transactions originated by C and received[2] by N (via validator nodes), ordered according to a predefined rule[3]. Let $\mathcal{T}(T_C)$ be the Merkle tree of T_C, and let R_C be its root.
Let $\mathcal{C}$ be the set of all Clients that originated transactions received by N, which we call *transacting Clients*. Node N calculates the Merkle tree $\mathcal{T}\left(\{R_C\}_{C\in\mathcal{C}}\right)$, whose root is called P. We refer to R_C as the *Client root* and P as the *block root*.

We now define block B_k constructed by Node N. The block will contain a *header*, a list of *summary transactions* (one per transacting Client), and a *phantom part*.
The summary transaction for C contains a public encryption and is as follows:

$$\eta_A(\iota_C) \parallel R_C.$$

[1] Block 0 can be made with obvious modifications.
[2] This set might be smaller than the set of all transactions issued by C
[3] For example, according to the order defined by the integer representation of $\sigma_C(d)$.

The header $\mathcal{H}_k$ of B_k is, in notation (1),

$$\mathcal{H}_k = h\left(\mathcal{H}_{k-1}\right) \parallel k \parallel t_k \parallel \mathrm{P} \parallel \iota_N \parallel \sigma_N\left(\mathcal{H}_k\right), \tag{5}$$

where t_k is stated by N as the creation time of B_k.

The phantom part is the list of all transactions referred by the summary transactions, that is $\cup_{C\in\mathcal{C}} T_C$. This transaction list is called *phantom part* because it is a part of the block which is visible only to the Nodes, and so invisible to the Clients.

Receipts
Once N creates the block B_k, N issues a receipt ρ_C to each transacting Client C, which in notation (1) is written

$$\rho_C = T_C \parallel R_C \parallel \pi_C \parallel \sigma_N\left(\rho_C\right), \tag{6}$$

where π_C is the shortest path in the Merkle tree from R_C to P.

3.3 Public ledger and Auxiliary receipt issuance

Let m be a fixed number $m \geq 2$. Every m blocks, node A interacts with the public blockchain. Let k_0 be the last time interval in which this happened, and call *anchorage block* a block corresponding to one of these interactions. At the end of time interval k_0+m, another anchorage block is created, therefore A collects the ordered list of Merkle roots and block hashes of blocks $k_0+1,\ldots,k_0+m$ and uses these as $2m$ leaves of an auxiliary Merkle tree $\mathcal{T}_A$,

$$\mathcal{T}_A = \mathcal{T}\left(\left\{\mathrm{P}_k, h\left(\mathcal{H}_k\right)\right\}_{k_0+1\leq k\leq k_0+m}\right) \tag{7}$$

with root $\mathcal{R}_{k_0+m}$, referred to as the *auxiliary root*.

The data committed to the public ledger[4] will be

$$\texttt{pub\_data} = \iota_A \parallel k_0 \parallel m \parallel \mathcal{R}_{k_0+m}\,. \tag{8}$$

Finally, the auxiliary node issues an auxiliary receipt $\underline{\rho}_C$ to every client transacting in the intervals $k_0+1 \leq k \leq k_0+m$. Each such Client will already be in possession of a set of receipts (6), which contains a set of blocks roots inside the paths π_C's. This set of block roots is a subset of the leaves of the tree with root $\mathcal{R}_{k_0+m}$. Therefore, $\underline{\rho}_C$ will contain the shortest path $\underline{\pi}_C$ in $\mathcal{T}_A$ required for C to recompute $\mathcal{R}_{k_0+m}$:

$$\underline{\rho}_C = k_0+1 \parallel k_0+m \parallel \texttt{pub\_data} \parallel v \parallel \underline{\pi}_C \parallel \sigma_A\left(\underline{\rho}_C\right), \tag{9}$$

where v is the address in the public blockchain of the transaction containing `pub_data`.

4 Proof of notarization

The system that we have described in the previous section may appear unnecessarily complicated. After all, if there exists a "good" timestamping server, people could just use it and get its signature in return. But herein lies the problem, because for a timestamping server to be

[4]The commitment of these data to the public ledger may require more than one transaction, according to the public-blockchain transaction format.

"good" it needs to be secure, reliable, approachable - in the sense that it is easy to communicate with it, both in bandwidth and in permissions - and cheap to use. While features like reliability, connectivity, and cost can be relatively easy to estimate, security remains much more difficult to evaluate. Indeed, we are not aware of any timestamping service on the Internet that presently satisfies all these properties, especially security.

The only system that might provide reasonable security is a public blockchain, such as the Bitcoin, and it would easily provide also approachability and reliability (in particular, avoiding the risk of a single point of failure). However, at present, the cost of transactions on a public blockchain is very high, making its direct use for storing proofs of documents infeasible. Therefore, many competing solutions have been proposed, whose general aim is to collect information on many documents - typically in hash form - and create paths of hashes linking each document to the final digest released on the public blockchain, e.g., Eternity Wall [1], Factom [16], and Guardtime [2]. These solutions must give their users some sort of receipt, allowing them to reconstruct the hash path and prove the existence of their documents.

Although our solution may appear similar, we aim at something more: we want to give our users incremental security. In the next sections we will describe our security claims and provide proofs.

4.1 Security claims

The solution in Section 3 builds on the basic premise of digital notarization of a document. Each client correctly interacting with our previous system (in a correct implementation) may be seen as a notary making a statement of the existence of data d by adding their digital signature, $\sigma_C\left(h\left(d\right) \parallel t\right)$. We are not concerned with the kind of information carried by d. Indeed, the data may or may not be a document signed by parties entering a contract, and their handwritten signatures may have been added on a paper or digital copy; we here emphasize that the only digital statement of *authenticity* by digital signature is the Client's.

We assume throughout that the blockchain is a *robust ledger* in the sense of [6], i.e. guaranteeing persistence and liveness. Since no specification of consensus mechanism is made here by design, we do not consider the question of whether the blockchain offers sufficient guarantees of consistency, crash fault tolerance, or security against collusion.

Let us consider the first step of our incremental security, which is the first interaction of a client C with our system. Let us call "Tom" someone who will come and will not believe in C's claims about the existence of its claimed documents at the claimed time. C hopes to be able to convince Tom by using our protocol.

At the start, C sends the signature of a document to the node network, encapsulated in the transaction τ (4), and a node M receives it. When M sends back to C the signature $\sigma_M\left(\tau\right)$, M is actually giving C the first evidence that C can show to Tom about its good faith. C is therefore immediately - say, in a few seconds - in possession of a receipt claiming the existence of its documents at the claimed time.

Whether Tom trusts this claim depends on whether Tom trusts M, specifically. This is equivalent to trusting a single timestamping server and can be modeled simply as follows.

Theorem 4.1. *If M is trusted, then the data claimed by C existed at the claimed time and were known to C.*

Proof. This comes from the unforgeability property of the signature σ_C and σ_M. □

In the second step, to increase trust we need to increase the number of entities working for the system, but making sure they arrive at an agreement on the documents. Nowadays, this

can be achieved with a private blockchain. We require it to be private so that Tom knows all service nodes and can decide whether he trusts them.

Observe that we have not specified how consensus is reached in the proxy blockchain. It could be that a majority of nodes is needed, or that all nodes must confirm N's proposed block, or some other more complex strategy. It does not matter, as long as Tom agrees that the consensus algorithm is trustworthy. What matters is that C has collected the new block header and that he has received a receipt ρ_C (6) from N. With this second evidence, Tom will agree on the following

Theorem 4.2. *If the new block has been generated with a trusted consensus algorithm, then (at least at time t_k) the data claimed by C existed and were known to C.*

Proof. All the service nodes that reached consensus have seen T_C, so by signalling their agreement to the new block they agreed that the transactions from C were valid and, in particular, that the transactions were sent before the creation time t_k of the block. Crucially, Tom has not seen the phantom part of the block, but he does not need it. Indeed, from $\mathcal{H}_k$ Tom can get t_k and P. From ρ_C he gets T_C and π_C, which shows the validity of the hash path.
C could not have forged the hash path π_C due to the hash security properties. □

An obvious question might arise when looking at the above proof: why keep a phantom part? This need arises from privacy reasons: we do not want any client C to see the transactions coming from another client C'. Of course, this goal could be reached with e.g. cryptography, but we take advantage of the use of a private blockchain to avoid more complicated features.

Thus C obtains within a short period of time some evidence on its documents that provides a much higher confidence for Tom: it is one thing to compromise or collude with a single participant M, and another to organize a collusion among the service nodes, including the miner N.

In the third and last step, if Tom does not trust the proxy blockchain, we will assume that he trusts the anchored public blockchain. Again, this trust means that, whatever consensus algorithm employed and whatever participants involved, the anchored public block was issued at a time prior to $\bar{t}$.

Assuming that C has collected the new public block and the receipts $\underline{\rho}_C$ and ρ_C (ρ_C refers to time interval k), Tom will then agree on the following.

Theorem 4.3. *If a new public block has been generated in a public blockchain by a trusted consensus algorithm, then (at least at time $\bar{t}$) the data claimed by C existed and were known to C.*

Proof. The new public block contains a public transaction containing `pub_data`. Since the public nodes have reached consensus and Tom trusts the public blockchain, he will trust that `pub_data` existed before $\bar{t}$. Indeed, from `pub_data` Tom can get $\mathcal{R}_{k_0+m}$. From $\underline{\rho}_C$ he gets $\underline{\pi}_C$, which shows the validity of the hash path from R_{k_0+m} to P_K. From ρ_C he gets π_C, which shows the validity of the hash path from P_K to R_C. From ρ_C he also extracts T_C, which contains the transaction with the claimed data, and can check its validity by recomputing R_C.

C could not have forged the hash paths $\underline{\pi}_C, \pi_C$ or the tree root R_C, without incurring in a hash collision. □

Observe that in this last step Tom does not need to put any trust in the proxy blockchain, because he can verify by himself the related hash chains.

Obviously, in this third step of incremental trust Tom does feel very sure about C's claim, but C obtains all the needed third evidence only *after* the public block creation, which will probably last some minutes and its timestamping claim can only be up to $\bar{t}$ rather than its claimed t.

5 Attacks with widespread collusion

In Section 4 we showed how Tom can be convinced by C in different trust scenarios. To convince Tom, C needed some valid evidence from the system. But the system might decide not to release such evidence and try to obtain some advantage for itself. We will consider now attack scenarios where the system does not interact correctly with C. We will assume that all node services (including miners and auxiliary nodes) are malevolent.

The three scenarios we are considering are: "Fake Owner", "Ghost document: proxy version", "Ghost document: public version". In the "Ghost document" scenarios also the client C is malevolent.

Fake Owner
Client C sends a transaction τ (4) at time t. The system does not acknowledge τ, and instead provides a colluding client D with evidence enough to claim that D knew data d at time t.

Theorem 5.1. *The Fake Owner attack fails.*

Proof. First, the malevolent nodes need to create a fake transaction τ'. We do not need to model what they will do next, because we claim it is impossible to create it. Indeed, τ' would have the form in notation (1)

$$\tau' = \sigma_D(d)||t||\iota_D||\sigma_D(\tau')$$

In particular, it would contain $\sigma_D(d)$. However, we assumed that our signature algorithm was resistant to impersonation attacks, and so this cannot happen. □

Ghost document: proxy version
After a new proxy block $\mathcal{H}_k$ is created at time t_k - but before an anchorage block, i.e. $k \neq m\lambda$ for any λ - C colludes with all service nodes and inserts a new transaction claiming that C knew data d' at time $t' < t_k$.

This attack may work, because the nodes can decide to:

- discard block k and all existing successive blocks in their blockchain,
- recompute T_C (to include τ') and all relevant information in $\mathcal{H}_k$,
- recompute ρ_C (to include the new T_C), recompute all receipts for the other clients (and send them);
- recompute the headers of the successive blocks (to have valid hash pointers) and resend them to the clients.

The other clients might complain at seeing the change in past block headers, but they may be satisfied when they receive valid blocks and valid receipts. If this attack is performed rarely, the clients (and Tom) may be induced into believing that the block updates are due to some software problems rather than malice.

Ghost document: public version
After a new proxy block $\mathcal{H}_k$ is created at time t_k (an anchorage block, so $k = m\lambda$ for some λ)

and the anchor has been created at time t', C colludes with all service nodes and inserts a new transaction in $\mathcal{H}_k$ claiming that C knew data d' at time $t' < t_k$.

Theorem 5.2. *A ghost document in public version cannot be created.*

Proof. Since the anchor has been created, `pub_data` is in the public blockchain. To be able to claim knowledge of data d', C needs a valid hash path pointing to the root corresponding to the new T_C, so that he could use it to replace π_C. However, this path must end in $\mathcal{R}_k$, which is immutable in the public blockchain, and so it is impossible to forge another path due to security properties of the hash function. □

6 Some comments on DOS attacks

In the past section we do not investigate scenarios when malevolent actors of the system want to mount a DOS (Denial Of Service). We now examine this situation. There are three possible attackers: a validator node, the auxiliary node A and the PKI's CA.

Validator node If a malevolent node M receives a transaction τ from a client, M can decide to ignore it. In this case, C would notice that something went amiss and C would try to contact another node N. This kind of DOS is dangerous only if the client's communication with the system is limited to a group of colluding nodes.
A malevolent M could do worse than simply dropping τ: M could send the first receipt $\sigma_M(\tau)$ back to C and avoid broadcasting it to the other service nodes. In this way, C is tricked into thinking that M is behaving honestly. However, C is expecting to receive also the second receipt ρ_C in a while. If this does not happen, C will know something is wrong and it will then interact with other nodes. On the other hand, if C receives ρ_C and τ has not been used to construct the relevant hashes, then again C will notice something is wrong.

Auxiliary node A cannot modify the H_k's to construct its Merkle tree (and thus compute a valid `pub_data`), because A cannot sign impersonating one of the service nodes. However, A may avoid to insert some of the H_k's, effectively removing from the auxiliary tree all the transactions received by the clients in the chosen intervals. This DOS attack by A is easily spotted by all nodes (and all clients) when the anchoring happens, because it will be impossible to reconclie the issued auxiliary receipts and the other receipts held by the clients.

Certification Authority We assume in our system that the CA is trusted, because if the CA were to issue certificates to malicious peers, and if it failed to revoke them, the system would be vulnerable to a majority attack. However, it could be that the CA itself is flooded by packets sent by DOS attackers. In this scenario, it may impossible for the system participants to check the validity of new data coming into the system, depending on the public keys held by each participant. Yet, assuming that honest peers will not validate transactions without a certificate revocation list being available, the validity of past transactions remains perfectly checkable by anyone having the relevant receipts (including Tom, if C gives them to him), since they are enough to validate the data inserted in the public blockchain.

7 Conclusions

We have shown a two-tiered system of independent blockchains for secure timestamping that offers incremental levels of evidence to clients. We have examined under what assumptions the system may be deemed secure; in particular, we have seen that under the assumption of an honest certification authority, only denial of service attacks are feasible, and they are also

immediately noticeable. The two-tiered system is designed to reduce the cost and increase efficiency of commitments to a slow and costly public blockchain, while at the same time still enabling clients to use their past evidence even if the intermediate blockchain solution were to cease being operational.
While we are satisfied with our finding, we notice that our results hold in a blockchain having an indefinite but supposedly robust consensus algorithm. It would be interesting to investigate how our system could be effectively integrated in a blockchain enjoying a specific consensus algorithm, such as proof-of-work or proof-of-stake.

Acknowledgments

Some partial results of this paper have been presented at the Euregio Blockchain Conference in 2018. The more advanced results presented here have been funded by the project MIUR PON "Distributed Ledgers for Secure Open Communities".

References

[1] Eternity wall. https://eternitywall.it/.

[2] Guardtime. https://guardtime.com/.

[3] Proof of existence. https://poex.io/.

[4] Dave Bayer, Stuart Haber, and W. Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In Renato Capocelli, Alfredo De Santis, and Ugo Vaccaro, editors, *Sequences II - Methods in Communication, Security, and Computer Science*, pages 329–334. Springer, New York, NY, 1991.

[5] Yuefei Gao and Hajime Nobuhara. A decentralized trusted timestamping based on blockchains. *IEEJ Journal of Industry Applications*, 6(4):252–257, 2017.

[6] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.

[7] Bela Gipp, Norman Meuschke, and André Gernandt. Decentralized trusted timestamping using the crypto currency bitcoin. In *iConference*, 2015.

[8] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, 1991. Presented at CRYPTO 1990.

[9] Stuart Haber and W. Scott Stornetta. Secure names for bit-strings. In *4th ACM conference on Computer and communications security (CCS)*, pages 28–35. ACM New York, NY, USA, 1997.

[10] IETF RFC 3161. *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*, 08 2001.

[11] ISO 18014-4:2015. *Time-stamping services – Part 4: Traceability of time sources*, 04 2015.

[12] Shin'ichiro Matsuo. How formal analysis and verification add security to blockchain-based systems. In *Formal Methods in Computer Aided Design (FMCAD)*, pages 1–4, 2017.

[13] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, Berlin, Heidelberg, 1987.

[14] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf, 2008.

[15] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics. In Bimal Kumar Roy and Willi Meier, editors, *Fast Software Encryption (FSE), 11th International Workshop on*, pages 371–388, 2004.

[16] Paul Snow, Brian Deery, Jack Lu, David Johnston, and Peter Kirby. Factom white paper v1.2. https://www.factom.com/assets/docs/Factom_Whitepaper_v1.2.pdf, 04 2018.