Private Timestamps and Selective Verification of Notarised Data on a Blockchain (Full Version)

ENRIQUE LARRAIA, OWEN VAUGHAN, nChain, UK

In this paper, we present a novel method for timestamping and data notarisation on a distributed ledger. The problem with on-chain hashes is that a cryptographic hash is a deterministic function that it allows the blockchain be used as an oracle that confirms whether potentially leaked data is authentic (timestamped or notarised by the user). Instead, we suggest using on-chain Pedersen commitments and off-chain zero-knowledge proofs (ZKP) for designated verifiers to prove the link between the data and the on-chain commitment. Our technique maintains the privacy of the data, and retains control of who can access it and when they can access it. This holds true even on a public blockchain, and even if the data is leaked by authorised parties. Indeed, an authorised data consumer (a designated-verifier for the ZKP), who discloses the data publicly, cannot convince anyone about the legitimacy of the data (in the sense that it is consistent with the information uploaded to the blockchain), because the ZKP proof is valid only for them. Our techniques can be used in scenarios where it is required to audit highly-sensitive data (e.g. application logs) by specific third parties, or to provide on-demand data certification by notaries

1 INTRODUCTION

Blockchains are rapidly gaining traction not just for cryptocurrencies but also as a vehicle for data integrity and decentralised databases. Two applications that are currently going mainstream are timestamping and notarisation (digitally signed data by a recognised authority). Trusted timestamping is straightforward due to data persistency and agreed order of insertion that a blockchain offers. There are multiple notary services that leverage the blockchain. These range from proving existence of data compliant with regulations, to contract liability and data legitimacy. They are achieved in conjunction with digital certificates. To some extent, one can even use it for proof of ownership [13]. However, most of the existing solutions resort to hashing the data and storing the digest on a blockchain. We identify three main problems when the hashed approach is used to timestamp or notarise data on-chain.

1.1 Problem Statement

We highlight several problems related to plain hashes on-chain.

Accessing timestamped data. Applications that require auditability (who can access the data and when) or need to grant access based on policies cannot rely solely on storing data on-chain. If we store the data in plain text on a public blockchain, then everyone can read it. If we instead store hashed data, the blockchain becomes an oracle to check consistency of shared data (by recomputing the hash and checking it against the one on-chain). This means that once a data owner gives access to her data to a single party, she loses control of who else can access her data. This is worrying as uploading data on-chain is the main approach taken by many applications. Permissioned ledgers do not seem to help much either: once granted access, nodes and users can download on-chain data at any moment, so in this case the data owner loses the ability of controlling when in time the data is accessible.

Lack of incentives for notaries. Notaries are required for authenticating data, guaranteeing compliance with laws, certifying conformity with contracts, and more. More often than not the notary offers his service upon user request and charges a fee either to the data owner or the data consumer. Signing the data and then uploading a hash is not incentive-driven for the notary. Indeed, the data can be freely shared by several parties and the notarisation checked multiple times at no extra cost. We need a mechanism to control who can verify the notarisation of on-chain data

Privacy of on-chain data. Last, simply storing a hash on-chain is not enough to guarantee privacy of the data. It may pose conflict with privacy regulations in some countries, especially if a salt is not used. For example, the GDPR regulations in Europe. This can be partially addressed using a permissioned ledger. But, even if one is willing to give up the public setting, the lack of standards to manage access to permissioned ledgers and clear guidelines in case of compromised keys [11] precludes their utilisation in many use cases.

1.2 Our solution

In our solution, the data is committed and hashed, we say it is 'obfuscated'. The distinction between commitment and hash is important. The data owner commits to the data using a private and random value that breaks the link between the data and its commitment (more technically, the commitment is hiding whereas the hash is just binding – see Section 2.1). Then, we use zero-knowledge proofs (ZKP) for designated verifiers [8] to prove knowledge of the randomness without revealing it. This is the same as proving the link between the data and its commitment. The proof is specially crafted for a designated verifier (a fixed data consumer), and he cannot use the proof to convince any other party of the link between off-chain and on-chain data. In contrast to publicly-verifiable SNARKs [14], the designated-verifier ZKP that we use does not require any trusted setup, it is fast, and straightforward to implement.

To privately timestamp, the data owner Alice commits and hashes the real data and uploads the hash of the commitment to the blockchain. Later, she proves to a designated data consumer Bob (and only to him), the link between the real data and the on-chain obfuscation as explained above. The ZKP and the immutability of the blockchain ensures timestamp for Bob. Indeed, if an obfuscation existed at a given time in the blockchain, so did the data committed in it. At the same time, data privacy cannot be undermined even after Alice shares the data with Bob. The obfuscation process and the non-transferability of the proof guarantees no one can tell the real off-chain data of Alice apart from some other potentially fake data presented by Bob. This is because the proof is meaningless to everyone but Bob, and the commitment is hiding, i.e. leaks no information about the data.

To selectively verify notarised data, we proceed as follows. The notary receives the data but only signs its obfuscation (either the

commitment or its hash). Since the obfuscation is binding, the notary is implicitly signing the real data too. Signing the obfuscation instead of the real data means that the data owner gets to decide to whom the signature is meaningful. Only a designated data consumer, who is convinced of the link between real and obfuscated data by virtue of the ZKP, concludes that the notary indeed signed the data. As in the previous application, no one without a designated proof can link the signature (stored on-chain) back to the real off-chain data.

The above applications are self-sovereign: the data owner need not trust nor delegate work to a trusted party. In some scenarios, however, the data owner may have little capability or willingness to interact with other parties or have limited connectivity. In return for a fee, she can simply send her data to a service provider (SP) and go offline. We explain how to outsource the applications to the SP ensuring the monetary incentive of the SP against colluding data consumers that try to re-use proofs addressed to just one of them.

We note that the designated proof can be uploaded to the blockchain as well, dismissing the need of off-chain communication channels. Also, our techniques are blockchain agnostic. Throughout this paper we will present our arguments in terms of Bitcoin with the understanding that they can be extended to all blockchain settings.

1.3 Related Work

There are several applications that have been already deployed in real scenarios. Amazon Web Services timestamps documents on Ethereum publishing the document hash in a smart contract [10]. Blockademia [5] is a decentralised application run on Cardano that allows data owners to publish documents resulting in on-chain hashes; the app also searches the blockchain and returns the hash of the document (if any) allowing data consumers to verify consistency. 4ire [1] is deployed on Ethereum and allows the signing of documents between several parties. It stores the hash of the signed document on-chain and only allow parties to sign who have authenticated themselves and are approved by previous signers. A recent line of research focusses on secure information exchange among multiple parties. Mutually distrusting parties need access to information but not everyone is allowed to access all information. For example, Li et. al. in [4] also use ZKPs (and more building blocks) to stablish a market model where clients can choose different service providers in a fair and private way. More related to our work, Chowdhury et. al. in [9] propose to use permissioned ledger to restrict access to the hashed data. Data owners can push data, and only legitimate data consumers are granted access to the ledger. Di Ciccio et. al. [7] employ attribute-based encryption (ABE) storing encrypted data on a distributed file-system (e.g. IPFS); an on-chain hash points to the data location. A smart contract handling the ABE policy controls who can decrypt. Contrary to our solution, their approach cannot be turned self-sovereign since ABE needs a trusted party to generate the master key. Also, our solution uses an efficient zero-knowledge proof system, instead of heavy ABE.

1.4 Paper Organisation

The paper is organised as follows. Section 2 reviews background. Section 3 presents the designated verifier proof system. Section 4 elaborates on our applications: private timestamping and selective verification of notarised data. Section 5 explains how to delegate to a service provider. Section 6 presents benchmarks and concludes the paper.

2 PRELIMINARIES

2.1 Pedersen Commitments

Pedersen commitments [12] are defined over a mathematical group $\mathbb{G}_p$ of p elements. To commit to a vector $\mathbf{m} = (m_1, \ldots, m_n) \in \mathbb{Z}_p^n$ of integers modulo p we only need a single group element $C \in \mathbb{G}_p$. To commit to $\mathbf{m}$ using the commitment key $\mathbf{CK} = (G_1, \ldots, G_n, H) \in \mathbb{G}_p^{n+1}$, one computes the commitment:

$$C \leftarrow \mathrm{Comm}(\mathbf{m}, r, \mathbf{CK}) := m_1 \cdot G_1 + \cdots + m_n \cdot G_n + r \cdot H.$$

At a later time, to verify that $\mathbf{m}$ is committed in C the randomness r is revealed and the same process above recalculated and checked against C.

Pedersen commitments are *binding* and *hiding*. Informally, the former means that it is very unlikely to find two messages yielding the same commitment; whereas the later demands that nothing can be inferred about the committed message with the knowledge of the commitment.

Public-verifiably keys or keys with a trapdoor. To ensure the binding property, the commitment key must be generated in a verifiably way: no correlation between the group elements $\mathbf{G}, H$ is known, and this can be verified publicly.

Another possibility is to have H linearly dependent on $\mathbf{G}$. For example, for case $n = 1$ suppose $H = x \cdot G$ for a known x, and that $C = m \cdot G + r \cdot H$. It is possible to open a commitment C to an arbitrary value $m' \neq m$ setting $r' = r + x^{-1}(m - m') \mod p$. In other words, Pedersen commitments are not binding to the party knowing the *trapdoor* x.

2.2 Sigma Protocols

A sigma protocol [6]is a zero-knowledge proof system [16] to prove the veracity of a public statement without revealing anything about some private information (the witness).

More formally, let $\mathcal{R}$ be an NP-relation. That is, a subset of $\{0,1\}^* \times \{0,1\}^*$ such that membership $(st, w) \in \mathcal{R}$ can be checked in polynomial time in the length of st, and the length of w is also polynomial in the length of st.

The first element of the tuple is called the *statement* and it is public information. The second element is called the *witness* (to the statement) and it is private. There might be more than one witness for a given statement. The induced NP-language $\mathcal{L}_\mathcal{R}$ is the set of statements:

$$\mathcal{L}_\mathcal{R} = \{st \mid \exists\ w \text{ such that } (st, w) \in \mathcal{R}\}.$$

A sigma protocol is a three-round protocol between a prover and a verifier. Both parties receive as input the statement st. Additionally, the prover receives the witness w as an auxiliary input and the verifier may receive any arbitrary auxiliary input.

- First, the prover computes a commitment A, using randomness a, and sends A to the verifier.

- Next, the verifier randomly samples a challenge e and sends it to the prover,
- Last, the prover computes an answer z (using w and a) and sends it to the verifier.

The verifier based on the public transcript $\pi := (A, e, z)$ accepts the statement st as valid or not.

2.2.1 Properties. The properties of a sigma protocol are the following ones.

Completeness: If $(st, w) \in \mathcal{R}$ then the verifier accepts with probability one.

Special soundness: For any pair of accepting transcripts $\pi = (A, e, z)$, $\pi' = (A, e', z')$ with the same commitment A and distinct challenges $e \neq e'$, it is possible to reconstruct the witness w such that $(st, w) \in \mathcal{R}$.

Special honest verifier zero-knowledge (SHVZK) There exists a polynomial-time algorithm Sim which on input $(st, w) \in \mathcal{R}$ and random challenge e, it outputs an accepting transcript $\pi := (A, e, z)$ indistinguishable from a protocol's transcript.

We remark that special soundness implies a stronger property: sigma protocols are also proof of knowledge (of a witness).

Also, SHVZK implies the standard notion of (honest-verifier) zero-knowledge, where the simulator is tasked with simulating transcripts on receiving only the statement as input. In other words, SHVZK guarantees that no information about the witness is leaked from the exchanged messages assuming the verifier behaves as prescribed.

2.2.2 An example: Schnorr Protocol. Given two group elements $G, H \in \mathbb{G}_p$, the Schnorr protocol [15] proves knowledge of the discrete logarithm $x = dlog_G(H) \in \mathbb{Z}_p$. The steps are:

- The prover computes samples $a \in \mathbb{Z}_p$ at random and computes $A = a \cdot G$. It sends A to the verifier.
- The verifier randomly samples a challenge $e \in C \subseteq \mathbb{Z}_p$ and sends it to the prover.
- The prover computes $z = a + exmodp$ and sends it to the verifier. The verifier accepts if and only if $z \cdot G = A + e \cdot H$.

The Schnorr protocol is complete and SHVZK. For special soundness, observe that from two accepting transcripts (A, e, z), (A, e', z') with different challenges $e \neq e'$ we can write $z \cdot G = A + e \cdot H$ and $z' \cdot G = A + e' \cdot H$. Subtracting the second equation from the first one, and using that $e - e'$ has (multiplicative) inverse in $\mathbb{Z}_p$ we can write $dlog_G(H) = (z - z)/(e - e')$.

2.2.3 Fiat-Shamir Heuristic. Sigma protocols are examples of public-coin interactive proof systems. The challenge sent by the (honest) verifier is random and independent from the prover's messages. Exploiting this feature, an interactive sigma protocol can be turned non-interactive (just one message from prover to verifier) by emulating the verifier's entropy used to sample the challenge with a cryptographic hash function, which is seen as a truly random function [2, 3]. Namely, setting $e = \mathsf{Hash}(st, A)$. Observe that $tr := (st, A)$ is the (public) transcript occurring right before the challenge e is generated by the verifier in the interactive case.

The assumption on the hash function ensures two things. First, the challenge e is randomly distributed, and therefore, the interactive protocol only needs to satisfy zero-knowledge against honest verifiers (or SHVZK). Second, the prover is unable to calculate the challenge before calculating the commitment A (and the statement st), so the order of execution of the protocol cannot be inverted. The latest is true provided the challenge space C is large enough so that trying with different commitments A does not ever hit the (unique) challenge that would allow simulation. A medium/conservative choice is to use challenges of size 80 or 128 bits respectively.

2.3 Sigma Protocols with Designated Verifier.

Let $\mathcal{L}_\mathcal{R}$ be an NP-language that admits a sigma protocol. Jakobsson et. al. [8] design a framework in which a prover Alice is able to prove a statement $x \in \mathcal{L}_\mathcal{R}$ only to a designated verifier Bob (who is in possession of a secret x), and no one else. The proof is non-interactive, and the conviction is *non-transferable*. The latter means that Bob is not able to convince a third party Charlie of the veracity of the statement, even if Charlie gets the secret x.

Intuitively, as the authors in [8] put it, the idea is that Alice generates a proof π_{Bob} to prove the modified statement:

$$\text{“}st x \in \mathcal{L}_\mathcal{R} \text{ or I know the secret } x\text{”}.$$

The proof π_{Bob} is crafted specially for Bob. If Bob is confident that his secret x has not been compromised, then π_{Bob} is indeed a convincing proof to himself. However, Bob is unable to convince Charlie that st is valid using π_{Bob}, because the very same proof could have been generated by Bob (proving that he knows trapdoor x instead that the statement $st \in \mathcal{L}_\mathcal{R}$ is valid).

More technically, what Jakobsson et. al. [8] propose is to use a trapdoor commitment scheme (such as Pedersen with non-verifiable commitment keys –see Section 2.1), where the trapdoor x is known to the designated verifier. The prover will commit to a random value v in a commitment C, and use C, (along with the statement st and the first message A of the sigma protocol) to generate 'half' of the challenge with the hash function. Namely, the generation of the non-interactive challenge is

$$e := h + v \text{ where } h = \mathsf{Hash}(st, C, A).$$

The designated verifier Bob besides the checks of the sigma protocol will also check that C opens to v. Now, since Bob can open C to any value he likes (using the trapdoor x), he can fake proofs π_{Bob} running the simulator Sim on a random challenge e to obtain $\pi := (A, e, z)$ and then open commitment C to $v^* := e - \mathsf{Hash}(st, C, A)$.

3 THE UNDERLYING PROOF SYSTEM

We design a proof system allowing a data owner to control who can be convinced of the link between her data and a hashed commitment of it – what we call the 'obfuscation' of the data.

3.1 The non-interactive designated-verifier sigma protocol

The data owner first commits to her data m (interpreted as an integer modulo p) in a hiding way using a Pedersen commitment key $\mathbf{CK}_{DO} := (G_{DO}, H_{DO})$. The public statement is the tuple:

$$st := (m, C, \mathbf{CK}_{DO}, obf).$$

Prove:
Inputs :
- Statement $st = (m, C, \mathbf{CK}_{DO}, obf)$
- Witness $r \in \mathbb{Z}_p$ // s.t. $C = m \cdot G_{DO} + r \cdot H_{DO}$
- Context information $\mathbf{CK}_{DC}$

Steps :
(1) $a \leftarrow \mathbb{Z}_p, A := a \cdot H_{DO}$
(2) $v, s \leftarrow \mathbb{Z}_p, D := v \cdot G_{DC} + s \cdot H_{DC}$
(3) $h := \mathsf{Hash}(\mathbf{CK}_{DC}, st, A, D)$
(4) $e := h + v$
(5) $z := a + er$
(6) Output $\pi_{DC} := (D, A, z, v, s)$

Verify:
Inputs :
- Statement $st = (m, C, \mathbf{CK}_{DO}, obf)$
- Proof $\pi_{DC} := (D, A, z, v, s)$
- Context information $\mathbf{CK}_{DC}$

Steps :
(1) If $D \neq w \cdot G_{DC} + s \cdot H_{DC}$ reject
(2) $h := \mathsf{Hash}(\mathbf{CK}_{DC}, st, A, D)$
(3) $e := h + v$
(4) If $z \cdot H_{DO} \neq A + e \cdot (C - m \cdot G_{DO})$ reject
(5) If $obf \neq \mathsf{sha256}(C)$ reject
(6) Else accept

Fig. 1. Prove and Verify algorithms. Hash is a cryptographic hash function that maps bitstrings to $\mathbb{Z}_p$

She will prove knowledge of the randomness used to commit to m. Concretely, she proves knowledge of an element in the set:

$$ZKP(st) := \{r \in \mathbb{Z}_p \mid C = m \cdot G_{DO} + r \cdot H_{DO} \wedge obf = \mathsf{sha256}(C)\}$$

The commitment key $\mathbf{CK}_{DO}$ must be generated in a verifiable way ensuring the group elements G_{DO}, H_{DO} are independent. This key can be shared across multiple data owners, and it can be widely available by e.g. storing it on the blockchain; for example embedding it an unspendable OP_RETURN output in Bitcoin.

Before the data owner can generate a proof π_{DC}, the designated data consumer chooses a random secret trapdoor $x \in \mathbb{Z}_p$ and sets up a new trapdoor commitment key as $\mathbf{CK}_{DC} := (G_{DC}, H_{DC} := x \cdot G_{DC})$. In Figure 1 we detail the non-interactive prover and verifier of the sigma protocol.

3.2 Faking proofs

Let a valid statement $st = (m, C, \mathbf{CK}_{DO}, obf)$. The designated data consumer, armed with the knowledge of the trapdoor x, can generate valid proofs for any data $m^* \neq m$ even if he does not know an opening r for C. This is possible because the designated data consumer can open commitments under his key $\mathbf{CK}_{DC}$ to any value he likes, and therefore choose in advance the challenge e that purportedly proves knowledge of $DLOG_{H_{DO}}(C - m^* \cdot G_{DO})$. For completeness, we detail how to fake proofs in Figure 2.

FakeProof:
Inputs :
- Statement $st = (m, C, \mathbf{CK}_{DO}, obf)$
- $m' \in \mathbb{Z}_p$ // any data, possibly $m' \neq m$
- Trapdoor $x \in \mathbb{Z}_p$ // s.t. $H_{DC} = x \cdot G_{DV}$
- Context information $\mathbf{CK}_{DC}$

Steps :
(1) $z, e \leftarrow \mathbb{Z}_p, A := z \cdot H_{DO} - e \cdot (C - m' \cdot G_{DO})$
(2) $d \leftarrow \mathbb{Z}_p, D := d \cdot G_{DV}$
(3) $h := \mathsf{Hash}(\mathbf{CK}, st, A, D)$
(4) $w := \cdot e - h$
(5) $s := \frac{d - e + h}{x}$ // compute opening with trapdoor
(6) Output $\pi_{DC} := (D, A, z, w, s)$

Fig. 2. Faking proofs by a designated verifier

4 SELF-SOVEREIGN APPLICATIONS

4.1 Timestamping Data On-chain Privately

The first application of the proof system described in Section 3 is to timestamp data in a private manner. The data owner gets to decide who can audit the timestamps at a later time. The steps of the interaction are as follows (see also Figure 3 for an illustration).

(1) The data owner Alice uploads the obfuscated data $obf := \mathsf{sha256}(C)$ to the blockchain, where $C = \mathsf{Comm}(C, r, \mathbf{CK}_{DO})$, while keeping the commitment opening r private.

The obfuscation is logged in the blockchain to ensure its existence at a given time. However, due to the hiding property of Pedersen commitments no one can infer which data m has been timestamped implicitly.

Any data consumer Bob that is given the data and wants to verify the link with the on-chain obfuscation will act as the designated verifier. Bob will be convinced that indeed m is consistent with obf. Alice and Bob proceed as follows.

(2) Bob downloads obf from the blockchain.
(3) Bob generates a trapdoor commitment key $\mathbf{CK}_{DC} := (G, H) \in \mathbb{G}_p^2$. Recall the trapdoor is $x \in \mathbb{Z}_p$ such that $H = x \cdot G$. It sends $\mathbf{CK}_{DC}$ to Alice.
(4) Alice runs algorithm Prove from Figure 1 on inputs $st = (m, C, \mathbf{CK}, obf)$, r, and obf. It sends m, C, π_{DC} to Bob.
(5) Bob runs algorithm Verify from Figure 1 on inputs m, C, π_{DC}. If the output is accepting, Bob deems the data m correct.

Privacy of the timestamps via Non-Transferable Proofs. After receiving m, C, π_{DC} with $obf = \mathsf{sha256}(C)$ from the data owner Alice, the data consumer Bob can produce for any data $m^* \neq m$ of his choice a convincing proof π^*_{DC}, that is, a proof that passes the verification algorithm (see Figure 2). With such power, a third data consumer Charlie cannot ever be sure if the data that Bob forwards comes from Alice or from Bob himself. Put differently, Charlie cannot tell whether the data was created before uploading obf to the blockchain by Alice or afterwards when Bob sees the commitment C. This means that any data coming from Bob is not bounded to the timestamp obf, maintaining the privacy of m to anyone but Bob.

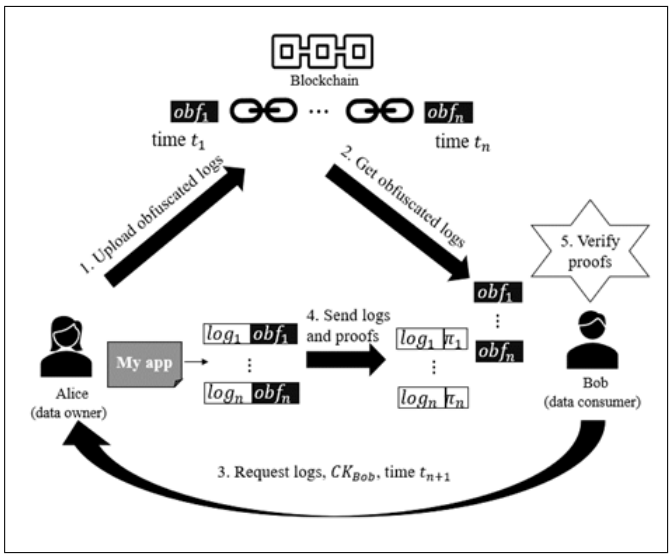

Fig. 3. Example of timestamping application logs privately in a blockchain.

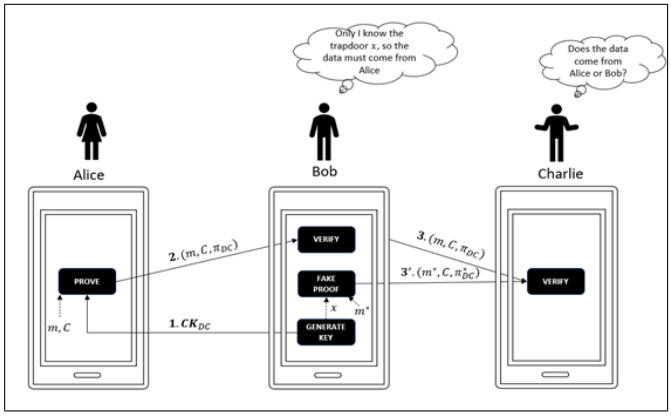

Fig. 4. Controlling who can verify the link between data and its on-chain obfuscation.

4.2 Notarisation of Data

In this scenario the data owner takes an active role in the process of notarisation (whereby the data is signed by a notary) and uploads the signed data to the blockchain herself. It is a two-phase protocol between the data owner, the designated data consumer, and the notary. The parameters of the scheme are preconfigured and already stored on the blockchain. They include the verification key PK of the notary for signatures as well as the commitment key $\mathbf{CK}_{DO}$ of the data owners.

- **Phase 1: Notarisation of data.** In the first phase, the data owner commits to her data m and sends it along with the commitment C to the notary, who signs the commitment. The data owner subsequently stores the obfuscated and signed data $D = (obf, \sigma)$ in the blockchain, where $obf = \text{sha256}(C)$.
- **Phase 2: Verification of notarised data.** In the second phase, a designated data consumer retrieves obf from the blockchain. It also requests the data m and the commitment C to the data owner, along with the proof π_{DC} proving knowledge of the commitment opening r. The designated verifier checks the proof and the signature σ, and also that obf is the hash of C.

Signing business-compliant data. The notary receives the data m so he can enforce its compliance before signing the commitment C. We emphasize that he does not know the private randomness r, and therefore he cannot establish the link between m and C. In order to implicitly sign m and not something else, the data owner proves the link in zero-knowledge to the notary, who acts as a designated verifier.

5 OUTSOURCING THE APPLICATIONS

Unlike in the previous use cases, now the data owner trusts a service provider (SP) with her private randomness r. In return, the SP generates the proof π_{DC} for the data consumer and interacts with the blockchain on behalf of the data owner. Further, the SP lets parties register as data consumers (which may involve authentication and compliance with specific policies), and prove to them the correct timestamping/notarisation of the requested data that is stored in the blockchain. These two services are offered in exchange for a fee. The data owner interacts just once with the SP to send her data. After that she remains completely oblivious to the process.

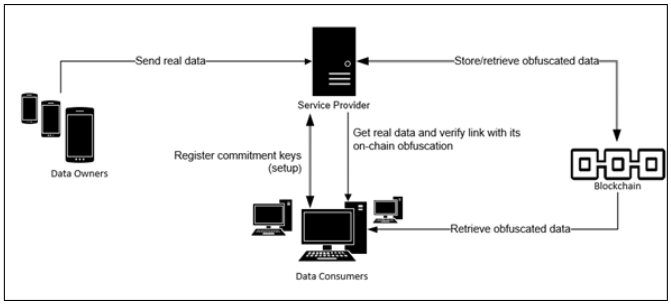

Fig. 5. Architecture of the outsourced application. Data owners send their data to the SP. Data consumers register their keys and verify correctness of data interacting with the SP.

5.1 Signing Notarised Data

When the outsourced service is notarisation, the signature that the notary (SP) puts on the data can be done in two ways.

Explicit signature. The notary signs the data commitment C and sets the obfuscated data to $obf := \text{sha256}(C, \sigma)$. Thus, the concatenation of the commitment and its signature. We assume the data consumer has the correct verification key of the notary (e.g. via a standard PKI infrastructure). He receives the signed commitment, and it checks that the signature is correct and that its hash matches the on-chain obfuscation obf.

Implicit signature. The notary embeds the obfuscated data in an unspendable OP_RETURN output of a Bitcoin transaction that spends a pay-to-public-key-hash (P2PKH) output. The data consumer receives the P2PKH address that is known to belong to the notary, the commitment C, and it checks that its hash is embedded

in the transaction that spends the notary P2PKH address. This has the effect of delegating signature verification to blockchain nodes (miners).

5.2 Protecting the Service Provider: Key Registration

A coalition of mutually distrusting data consumers may interact to pay the fee of the SP just once and re-use the same proof for all of them. This is possible if none of the parties knows the secret trapdoor explicitly, but instead is jointly held by all of them. We protect SPs against this type of coalitions without resorting to trusted third party or authenticated channels as in [8].

In our solution, the data consumer proves in zero-knowledge that he knows the trapdoor x of a given commitment key $\mathbf{CK}_{DC} = (G, H)$ as part of the key registration. The difficulty resides in proving *explicit* knowledge of the trapdoor x. For example, the standard Schnorr protocol [15] to prove knowledge of the logarithm of H in base G does not suffice here. Indeed, a coalition of data consumers that share the trapdoor x can collaborate to prove joint knowledge of x without any of them knowing x explicitly.

5.2.1 Solution 1: Outsourcing Generation of the Trapdoor. A straightforward solution to avoid malicious coalitions of verifiers is to outsource the generation of the trapdoor commitment key $(x, \mathbf{CK}_{DC})$ to a trusted third party – the trapdoor generator. This party is trusted to not collude with the SP to share the trapdoor x, and to not generate simulated proofs. On a generation request from a designated data consumer, the trapdoor generator will issue the signed pair $(x, \mathbf{CK}_{DC})$. Note it includes the trapdoor x explicitly. Any data consumer can register a commitment key $\mathbf{CK}_{DC}$ with the SP, who would check such key is signed by the trapdoor generator. If that is the case, the SP is convinced that the knowledge of the trapdoor x of $\mathbf{CK}_{DC}$ is known by at least one party – now the de facto designated data consumer – namely the party that requested the generation of the key to the trapdoor generator. This would make the proof π_{DC} convincing only to such designated consumer but not the other members of the coalition.

5.2.2 Solution 2: Proving Knowledge of the Trapdoor in Zero-knowledge. The above solution introduces a trusted party (the trapdoor generator), which depending on use cases might not be desirable. Instead, the designated consumer can prove (in zero-knowledge) to the SP that he knows the trapdoor x of a given commitment key $\mathbf{CK}_{DC}$.

The difficulty resides in proving *explicit* knowledge of the trapdoor x. For example, the standard Schnorr protocol to prove knowledge of the logarithm of H_{DC} in base G_{DC} does not suffice here. A coalition of verifiers as above can collaborate to prove joint knowledge of x assuming each of them knows an additive share x_i only.

The Underlying Idea. To register the commitment key $\mathbf{CK}_{DC}$, the data consumer plays the role of the prover, and the SP the role of the verifier. As we shall see below, provided the proof verifies successfully, the SP is convinced that the data consumer has used the trapdoor x explicitly in the generation of the proof with overwhelming probability.

We use the standard Schnorr protocol (see Section 2.3) but demand the data consumer (the prover) to commit in advance to all possible answers. More concretely, on common input $\mathbf{CK}_{DC} = (G, H := x \cdot H)$ the protocol is as follows:

- the data consumer commits in advance to the two possible answers $z_0 := a, z_1 := a + x$. He commits by hashing: $c_i = \mathsf{Hash}(z_i)$. Here $a \in \mathbb{Z}_p$ is the randomness used to generate the first message of the Schnorr protocol. The data consumer sends $A := a \cdot G$, and c_0, c_1 to the verifier.
- the SP issues a challenge bit $e \in \{0, 1\}$
- the data consumer sends answer z_e, and the SP checks correct decommitment $c_e = \mathsf{Hash}(z_e)$ and correct answer $z_e \cdot G = A + e \cdot H$.

The above modified Schnorr protocol gives soundness error $p = 1/2$. To amplify soundness to $p = 2^{-s}$, they repeat the protocol s times sequentially.

Reducing the number of repetitions. We can increase the size of the challenge to d bits. Each execution of the protocol with d-bit challenges gives soundness error 2^{-d} . To achieve soundness error 2^{-s} where s is a fixed security parameter we need to repeat it a total of $r = s/d$ times.

However, now the prover needs to commit to 2^d different answers $z_i = r + e_i x$. This can be done efficiently using a Merkle tree of depth d , where the i-th leaf is set to z_i. The prover sends the root c of the Merkle tree before the verifier issues the challenge e , and it answers with $z = r + ex$ along with the Merkle proof $\mathbf{p}$ for it.

Security Analysis – Why the Data Consumer Knows the Trapdoor Explicitly? The protocol sketched above has special soundness. This means that there exists a polynomial-time extractor algorithm $\mathcal{E}$, that from two different accepting transcripts (with the first message fixed) can extract the trapdoor x. More specifically, from two different challenges $e \neq e'$ and two answers z, z', extracts by computing $x := (z - z')/(e - e') \mod p$.

Now, assume the data consumer knows all possible answers. In particular, he can always generate two accepting transcripts (for two different challenges) and run the extractor $\mathcal{E}$ on them to output the trapdoor x. In other words, if the prover knows all possible answers, then he knows the trapdoor x *explicitly*.

The probability of not knowing the answers but providing a valid Merkle proof $\mathbf{p}$ for c, z_e is at most $d\epsilon$, where ϵ is the probability of finding a collision of the hash function used in the Merkle tree of depth d.

We conclude that a convincing prover knows x *explicitly* with probability at least $1 - d\epsilon$. Last, observe that ϵ is negligible in the size of c assuming collision resistance of the hash function.

Implementing the Protocol in Practice. Figure 6 defines the non-interactive version (using the Fiat-Shamir transform) of the protocol sketched above.

We observe that to preserve zero-knowledge (of the trapdoor) the Prover executes all rounds sequentially. This affects how the challenges are derived from the transcript. Specifically, let

$$\pi_i := (A_i, c_i, e_i, z_i, \mathbf{p}_i)$$

be the transcript generated in the i-th round. The $(i + 1)$-th challenge is the hash of $(\pi_i, A_{i+1}, c_{i+1})$. Also, the prover generates the

ProveKeyRegistration:

Inputs :

- Statement $st = \mathbf{CK}_{DV} = (G, H)$
- Witness $x \in \mathbb{Z}_p$ s.t. $H = x \cdot G$

Steps :

(1) $\pi_0 \leftarrow st$
(2) For $i = 1$ to r do:
  (a) $a_i \leftarrow \mathbb{Z}_p, A_i = a_i \cdot G$
  (b) For $j = 0$ to $2^d - 1$: $z_{i,j} = a + jx \mod p$
  (c) $c_i \leftarrow \text{GenMerkleRoot}(\mathbf{z}_i)$
  (d) $(e_i, n_i) \leftarrow \text{GenChallenge}(\pi_{i-1}, A_i, c_i)$ `// see below`
  (e) $z_i = z_{i,e_i}$
  (f) $\mathbf{p}_i \leftarrow \text{GenMerkleProof}(\mathbf{z}_i, e_i)$
  (g) $\pi_i = (A_i, c_i, e_i, n_i, z_i, \mathbf{p}_i)$
(3) Output $\boldsymbol{\pi} = (c_i, e_i, n_i, z_i, \mathbf{p}_i)_{i=1}^r$

VerifyKeyRegistration:

Inputs :

- Statement $st = \mathbf{CK}_{DV} = (G, H)$
- Proof $\boldsymbol{\pi} = (c_i, e_i, n_i, z_i, \mathbf{p}_i)_{i=1}^r$

Steps :

(1) $\pi_0 \leftarrow st$
(2) For $i = 1$ to r do:
  (a) $b_i \in \{0, 1\} \leftarrow \text{CheckMerkleProof}(c_i, \mathbf{p}_i, z_i)$
  (b) If b_i reject
  (c) $Y_i = e_i \cdot H$
  (d) $Z_i = z_i \cdot G$
  (e) $A_i = Z_i - Y_i$ `// reconstruct first message`
  (f) $(e_i^*, n_i^*) \leftarrow \text{GenChallenge}(\pi_{i-1}, A_i, c_i)$
  (g) If $(e_i^*, n_i^*) \neq (e_i, n_i)$ reject
  (h) $\pi_i = (A_i, c_i, e_i, n_i, z_i, \mathbf{p}_i)$
(3) Accept if no rejections in Step 2

GenChallenge(π_{i-1}, A_i, c_i):

(1) $e_i = 2^d, n_i = 0$
(2) while $e_I \geq 2^d$ do: `// rejection sampling`
  (a) $n_i = n_i + 1$
  (b) $h_i = \text{Hash}(\pi_{i-1}, A_i, c_i, n_i)$
  (c) $e_i \leftarrow \text{Trunc}(h_i)$ `// Truncate first` $\lceil \frac{d}{8} \rceil$ `bytes`
(3) Output (e_i, n_i)

Fig. 6. NIZK proof system to prove explicit knowledge of the trapdoor x. It is parameterized with the bitsize d of the challenges and the number of iterations r. The hash function outputs bitstrings of length k.

challenge with e.g. rejection sampling to not introduce bias when reducing mod 2^d for arbitrary d.

Complexity and Choice of Parameters. We instantiate Pedersen commitments over any elliptic curve with order p of 256 bits and the hash function used to derive the challenge and to construct the Merkle tree with sha256. These choices yield proofs $\pi := (c_i, \mathbf{p}_i, e_i, z_i, n_i)_{i=1}^r$ of bitsize roughly $r(512 + 257d)$ where $d < 256$ is the bitsize of the challenges and r is the number of iterations (rounds).

The prover needs to perform r scalar multiplications and the verifier twice as many. We therefore seek to minimize as much as possible the number of iterations r. This is to minimize both, the computational and communication complexity. However, we cannot set r to small (e.g., $r = 1$), as this would yield a Merkle tree excessively large (e.g., 2^s leaves) to compute on Prover's side.

Concrete values of r and d should be determined empirically having in mind that $r = s/d$, and that we have fixed the soundness security parameter to $s \in \{80, 128, 256\}$.

6 CONCLUSIONS AND RECOMMENDATIONS

In this paper, we have explained how to achieve private timestamping and selective verification of notarised data in a blockchain. The techniques use non-interactive sigma protocols for designated verifiers and only upload (hashes of) committed data to the blockchain. Due to the hiding property of the Pedersen commitments and non-transferability of the proofs, we can guarantee privacy and control who can verify the proofs. Our applications are straightforward to implement, do not assume trust in third parties, and are blockchain-agnostic. Although not stated explicitly, the designated proof can be uploaded to the blockchain as well, dismissing the need of off-chain communication channels.

6.1 Benchmarks

We have implemented a proof of concept in Rust programming language. Pedersen commitments and the prover and designated-verifier (Figure 1) are instantiated over the elliptic curve Curve2559. We have benchmarked the combined time it takes to commit and generate a designated-verifier proof, and also the time it takes to verify the proof for data varying up to 4KB of size. The tests have run in a laptop with processor AMD Ryzen 7 PRO 4750U at 1.70 GHz and 32GB RAM. All timings are under a fraction of a second. See Figure 7.

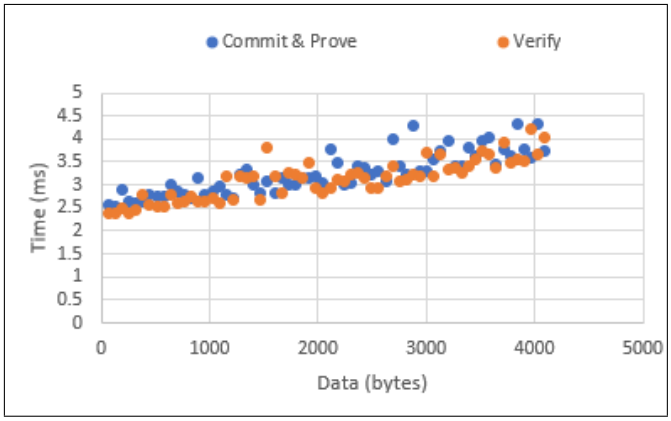

Fig. 7. Times to commit and prove (combined) and verify.

For reference, we have also compared the time to commit and prove data with the time it takes to hash the same data with SHA512 (Figure 8). We have observed that the performance overhead decreases when data size is increased. We believe this is because we encode data as points in the curve via hashing. The overhead when committing 4KB of data is roughly a 7x factor.

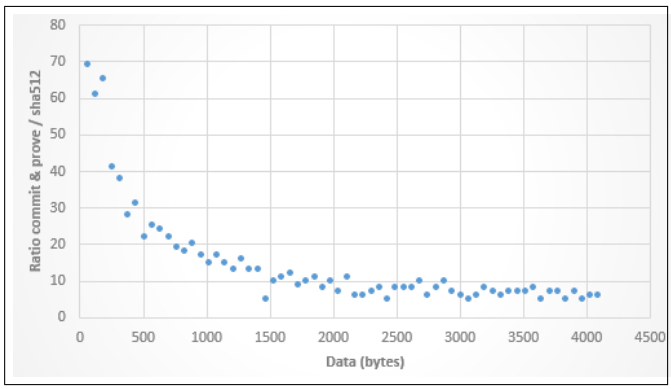

Fig. 8. Overhead of commit & prove w.r.t. SHA512.

6.2 Future work

There are two avenues we can pursue as future work. First, develop a proof-of-concept implementation with benchmarks for the service provider described in Section 5.2. Second, we can explore how to commit to large data sets and only reveal certain parts of it. This will allow data owners to disclose a specific data segment to one consumer, and a different data segment to a different consumer. To achieve this, the commitment scheme and its associated designated-verifier should be massaged accordingly.

REFERENCES

[1] 4irelabs. [n. d.]. Blockchain Use Case for Notary. https://4irelabs.com/cases/notarization-in-blockchain/.
[2] Adi Shamir Amos Fiat. 1986. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology CRYPTO, Santa Barbara, California, USA, 1986, Proceedings.*
[3] M. Bellare and P. Rogaway. 1993. Random oracles are practical: A paradigm for designing efficient protocols. In *In ACM CCS 93, pages 62–73. ACM Press, November 1993.*
[4] Huan Chen Li Cheng Bin Li Yijie Wang, Peichang Shi. 2018. A Fast and Privacy-Preserving Method Based on the Permissioned Blockchain for Fair Transactions in Sharing Economy. In *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE).*
[5] Blockademia. [n. d.]. Blockademia white paper. .https://blockademia.com/images/whitepaper-en.pdf.
[6] Ivan Damgard. [n. d.]. On Σ-protocols. Lecture notes available at https://www.cs.au.dk/~ivan/Sigma.pdf.
[7] Ingo Weber Edoardo Marangone, Claudio Di Ciccio. 2018. Fine-grained Data Access Control for Collaborative Process Execution on Blockchain. In *BPM 2022: Business Process Management: Blockchain, Robotic Process Automation, and Central and Eastern Europe Forum.*
[8] Kazue Sako Markus Jakobsson and Russell Impagliazzo. 1996. Designated verifier proofs and their applications. In *Advances in Cryptology - EUROCRYPT 1996.* Springer.
[9] Muhammad Ashad Kabir Jun Han Paul Sarda Mohammad Jabed Morshed Chowdhury, Alan Colman. 2018. Blockchain as a Notarization Service for Data Sharing with Personal Data. In *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE).*
[10] Christoph Niemann. 2022. Notarize documents on the Ethereum Blockchain. https://aws.amazon.com/blogs/database/notarize-documents-on-the-ethereum-blockchain/.
[11] Marta Poblet Oleksii Konashevych. 2018. Is Blockchain Hashing an Effective Method for Electronic Governance?. In *31st International Conference on Legal Knowledge and Information Systems (JURIX 2018) Groningen (The Netherlands).*
[12] Torben P. Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO.* Springer.
[13] Rohan Pinto. 2022. A Blockchain-Based Digital Notary: What You Need To Know. https://www.forbes.com/sites/forbestechcouncil/2019/11/12/a-blockchain-based-digital-notary-what-you-need-to-know.
[14] Bryan Parno Marian Raykova Rosario Gennaro, Craig Gentry. 2013. Quadratic Span Programs and Succinct NIZKs without PCPs. In *Advances in Cryptology - EUROCRYPT 2013.* Springer.
[15] C P Schnorr. 1990. Efficient identification and signatures for smart cards. In *G Brassard, ed. Advances in Cryptology – Crypto '89, 239–252, Springer-Verlag, 1990. Lecture Notes in Computer Science, nr 435.*
[16] Charles Rackoff Shafi Goldwasser, Silvio Micali. 1989. The Knowledge Complexity of Interactive Proof Systems.. In *SIAM J. Comput., volume 18, pages 186-208. 1989.*