Timed Signatures and Zero-Knowledge Proofs –Timestamping in the Blockchain Era–

Aydin Abadi*[1], Michele Ciampi†[1], Aggelos Kiayias‡[2] and Vassilis Zikas§[2]

[1]The University of Edinburgh
[2]The University of Edinburgh and IOHK

Abstract

Timestamping is an important cryptographic primitive with numerous applications. The availability of a decentralized blockchain such as that offered by the Bitcoin protocol offers new possibilities to realise timestamping services. Nevertheless, to our knowledge, there are no recent blockchain-based proposals that are formally proved in a composable setting.

In this work, we put forth the first formal treatment of timestamping cryptographic primitives in the UC framework with respect to a global clock—we refer to the corresponding primitives as *timed* to indicate this association. We propose timed versions of primitives commonly used for authenticating information, such as digital signatures, non-interactive zero-knowledge proofs, and signatures of knowledge and show how those can be UC-securely implemented by a protocol that makes ideal (blackbox) access to a global transaction ledger based on the ledger proposed by Badertscher *et al.* [CRYPTO 2017] which is UC realized by the Bitcoin backbone protocol [Eurocrypt 2015]. Our definitions introduce a fine-grained treatment of the different timestamping guarantees, namely security against *postdating* and *backdating* attacks; our results treat each of these cases separately and in combination, and shed light on the assumptions that they rely on. Our constructions rely on a relaxation of an ideal beacon functionality, which we implement UC-securely assuming the ledger functionality. Given the many potential uses of such a beacon in cryptographic protocols this result may be of independent interest.

1 Introduction

Timestamping allows for a (digital) object—typically a document—to be associated with a creation time (interval), such that anyone seeing the timestamp can verify that the document was not created before or after this time. It has numerous applications from synchronizing asynchronous distributed systems to establishing originality of scientific discoveries and patents. In fact, the idea of timestamping has been implicit in science for centuries, with anagram-based instantiations being traced back to Galileo and Newton. The first cryptographic instantiation of timestamping was proposed by Haber and Stornetta [HS91].

*aydin.abadi@ed.ac.uk
†mciampi@ed.ac.uk
‡akiayias@inf.ed.ac.uk
§vassilis.zikas@ed.ac.uk

A cryptographic timestamping scheme involves a document creator (or client) and a verifier, where the document creator wishes to convince the verifier that a document was at his possession at time T. In typical settings, the aim is to achieve universal verification, where any party can verify the timestamp but one can also consider the simpler designated verifier-set version. Ideally, the protocol aims to protect against both *backdating* and *postdating* of a digital document. To define these two properties, let A be a digital document which was generated at time T. In backdating, an adversary attempts to claim that A was generated at time $T' < T$. In postdating, an adversary tries to claim that A was generated at time $T' > T$. We note that no existing solution achieves the above perfect form of timestamping. This would be feasible only by means of perfect synchrony and zero-delay channels. Instead, timestamping protocols, including those presented in this work, allow to prove backdating and postdating security for a sufficiently small time interval around T.

Haber *et al.* [HS91] achieve timestamping using a *hash-chain of documents*. In the plain, centralized version of their scheme the parties have access to a semi-trusted (see below) third party, called a *timestamping server* (TS). Whenever a client wishes to sign a document, he sends his ID and (hash of) his document to TS who produces a signed certificate, given the client's request. The certificate includes the current time (according to TS), the client's request, a counter, and a hash of the previous certification which links it to that certificate. The idea is that, assuming the TS processes the documents in the time and order they were received, if a document A appears in the hash chain before the hash of document B, then B must have been generated after A. If someone wants to check the order in which the two documents where generated, he can check the certificate, and assuming that he trusts TS's credentials, he can derive the order. The above solution suffers from the TS being a single point of failure. Concretely, the timestamping protocol is only effective if the TS is constantly online and responsive. This opens the possibility of denial-of-service attacks. Also, when used in the context of patents, in order to avoid the need to trust the TS from claiming the patent as its own, one needs to combine it with anonymity primitives, such as blind signatures [Cha88]. To circumvent such issues, [HS91] proposed a decentralized version of their scheme, where the clients interactively cooperate with each other to timestamp their documents. The efficiency and participation requirements of that scheme were later improved by Benaloh *et al.* [BdM91]. Later on, [BLSW05] formally models the timestamping mechanisms, previously proposed in [BdM91, HS91], using the UC model. Moreover, it provides a construction very similar to [BdM91, HS91] with the main difference that it utilises an additional trusted party, an auditor, who periodically verifies the TS.

More recently, [BLT14] proposes a protocol that requires multiple non-colluding servers who interactively time-stamp a document. Although such a level of decentralization eliminates the single-failure point issue, it brings additional complications. Concretely, first, it can only work if the servers are properly synchronized and their communication network is synchronous. Indeed, [BdM91, BLT14] have an implicit round structure where every server/client is always in the same round as all other servers/clients. Second, to avoid attacks by malicious servers that attempt to backdate or postdate a document (e.g., by creating a fork in the hash-chain) it seems necessary to assume that a majority of them are honest and will therefore keep extending the honest chain. Third, the identities and signature certificates of the servers and clients need to be public knowledge, leading to the *permissioned* model that often requires mechanisms for registering and deregistering (revoking) parties' certificates. We note in passing that the above issues are implicit in the treatment of [HS91, BdM91], and there is no known technique to mitigate them. We further remark that these issues are similar to the core problem treated by blockchains and their associated

cryptocurrencies [Nak08, Woo14, KRDO17]. Thus, one could use techniques from such primitives, e.g. relying on proofs of work or space, to develop a timestamping blockchain. In fact, there are existing commercial solutions, e.g., Guardtime[1], that use this idea to offer a blockchain-based timestampting system. Following this research line, very recently [LSS19] have presented a treatment of non-interactive timestamping schemes in the UC-model. The construction provided in [LSS19] is based on proofs of sequential work such as VDF's [BBBF18]. However, as the authors stated in [LSS19], the construction allows the adversary to pretend that a record was timestamped later than it actually was (i.e., it allows postdating attack). Also, even if the work of Landerreche et al. assumes the existence of a global clock, the timestamping service provides only ordering of events.[2]

Our Contributions. We put forth a formal composable treatment of timestamping of cryptographic primitives. Concretely, we devise a formal model of protocol execution for timestamping cryptographic primitives with respect to a global clock that parties have access to. We use the term *timed*, as in *timed (digital) signatures* to distinguish timestamping with respect to such a global clock from the guarantee offered by existing timestamping schemes [HS91, BdM91, LSS19], which only establishes causality of events—i.e., which of the hash-chained document was processed first—but does not necessarily link it to a global clock. We stress that although for simplicity our treatment assumes ideal access to a global clock—which is captured as in [BMTZ17] by a global clock functionality, it trivially extends to allow for parties having bounded-drift view of the clock [KMTZ13]—i.e. the adversary is allowed at time t to make a party think that the time is t' which might lie within a distance d from t for a known drift parameter d.

We then define *timed* versions of primitives commonly used for authenticating information, such as digital signatures, non-interactive zero-knowledge proofs [DMP88, BFM88], and signatures of knowledge [CL06] in Canetti's Universal Composition (UC) framework [Can01]. Our treatment explicitly captures security against *backdating* and *postdating* separately, and investigates the associated assumptions required to achieve each of these security notions.

Finally, we devise UC secure implementations of our timed primitives that use any ledger-based blockchain. Rather than building a new dedicated timestamping blockchain, our protocols take advantage of the recent composable treatment of ledger-based cryptocurrencies by Badertscher et al. [BMTZ17, BGK+18] to implement timed versions of these primitives while making blackbox (hybrid) access to a (global) transaction ledger functionality. This decouples the trust assumptions needed for secure timestamping from the ones needed for maintaining a secure ledger and makes the security of our protocols independent of the technology used to implement the ledger. In particular, our protocols can use any existing public blockchain to achieve backdating and/or postdating security. In fact, our protocols not only make blackbox use of the ledger functionality, but they also make blackbox use of the corresponding cryptographic primitive they rely on. For example, our timed signatures make blackbox use of a signature functionality [Can03] and no further cryptographic assumptions. This means that all our constructions can be instantiated with any protocols that UC securely realizes the underlying cryptographic primitives (ledger and signatures). Furthermore, our use of the ledger is *minimal* with postdating security requiring only read access to the ledger, while backdating security requiring only write access to the ledger. As a result it is readily compatible with Bitcoin or any other current permissionless distributed ledger.

[1] https://guardtime.com

[2] In [LSS19] the parties need to be synchronized via a global clock in order to keep track of the computation steps done by the adversary to compute the outputs of the verifiable delay function.

Our Techniques. A standard idea for achieving security against postdating attacks—i.e., thwarting an adversary that runs a cryptographic primitive and gets a bitstring as an output at time T but claims that it was created at time $T' > T$, is to embed in the cryptographic primitive's output evidence of an event (or just a value) which becomes publicly known at creation time and could not have been predicted in advance. A folklore use of this idea is for example to embed a newspaper article about an unexpected event. The main challenge with the above solution is that the unpredictable information needs to be verifiable (along with the time it became available) by anyone who attempts to verify the timestamp. In a cryptographic setting, this could be solved by assuming an unpredictable randomness beacon that generates a new value in every round, with the property that anyone can query it with a round index and receive the value that the beacon output in that round. Here we do not assume such a perfect beacon—as this would correspond to a strong trust assumption. So the main question to answer is:

> *How can we construct such a source of sufficiently unpredictable and publicly verifiable randomness?*

One might be tempted to think that the blockchain directly provides us with such a source. In fact, a number of proposals for a beacon based on Bitcoin exist [AD15, BCG15, BGZ16]. But, to our knowledge, none of these works has a formal specification of the beacon they achieve or a formal proof of its security based on standard cryptographic assumptions. In fact, as argued in [BGZ16], an unbiased beacon can not be constructed using such assumptions based on the Bitcoin protocol. In this work, we take a different path. We investigate how an ideal beacon as above can be weakened so that it is implementable by a protocol which uses the ledger functionality (and a random oracle). In particular, we specify a *weak beacon* functionality, denoted as $\mathcal{B}^{\mathsf{w}}$, which is sufficiently strong to be used for timestamping cryptographic primitives. In a nutshell, the beacon functionality is relaxed in the following way in order to obtain our weak version: First, the weak beacon is slower, and is only guaranteed to generate a new value every `MaxRound` many rounds, where `MaxRound` is a parameter that depends on the ledger's liveness parameter[3] (we discuss it in more details in Section 2).

Second, although the sequence of outputs of the beacon cannot be changed once set, instead of every party being able to learn this sequence at any time, the adversary is allowed to make different parties witness different prefixes of this sequence in any round; this can, however, happen only under the following two restrictions, which are derived from the properties of the ledger specified in [GKL15] (cf. Section 2): (1) the lengths of the prefixes seen by different parties do not differ by more than `WindowSize`, again a parameter which depends on the ledger, (2) the prefixes increase monotonically as the rounds advance (albeit not necessarily at the same rate), and most importantly, (3) the adversary has a limited capability of predicting the beacon's output. In a nutshell, this predictability will allow the adversary to be able to predict several future outputs, under the restriction that in every t outputs at least one of them could not have been predicted more than k rounds before it was generated by the beacon, where k is a parameter that will depend on the ledger's transaction liveness parameter. Interestingly, while the first two properties are captured in the composable treatment of [BMTZ17], the latter one is not. To address this, we introduce a simple *wrapper* functionality that upgrades the ledger functionality of [BMTZ17] to possess this weak unpredictability property while we show that the main result of [BMTZ17], namely that the Bitcoin backbone protocol of [GKL15] implements the ledger, can be strengthened accordingly.

[3] The ledger's liveness property from [BMTZ17] corresponds to the *chain growth* property from [GKL15].

We provide a formal description of the above sketched weak beacon, and prove that it can be implemented by a protocol which makes ideal access to any of the ideal ledger functionalities from the literature [BMTZ17, BGK+18] suitably augmented with our wrapper functionality. We believe that this result is of independent interest. Given the above beacon, we will show how it can be used to time(stamp) cryptographic primitives with respect to the global clock, the beacon (and the ledger) is connected to. We start with one of the most common primitives used in the timestamping literature, namely digital signatures. Note that the straightforward adaptation of digital signatures to their timed version—which only allows the adversary to register a signature at the right time—cannot be implemented given the above beacon. Instead, we devise a relaxation of such functionality which embraces the imperfections of the beacon, while preserving the security against postdating and backdating attacks.

To obtain postdate-security, we use the above idea of embedding in the signature the most recent value of the beacon. As the adversary cannot predict the output of the beacon for more than k rounds in the future, this already puts an upper-bound in his poststamping ability. Recall that in any timestamping scheme, the timestamp is associated with some time interval and the adversary can create valid timestamps within the interval. Note that our mechanism for postdate-security does not require writing anything on the ledger; instead, the signer and the verifier only need read-access. Obtaining backdate-security is trickier. First, we observe that if the signer has read-only access to the ledger, then the ledger cannot be used to counter backdating attacks. The reason is that an adversarial signer has full information on the history of the ledger, at a certain time T. So, it can always pretend the ledger is in a past state (e.g., use an old beacon output in the signature), and then issue the signature claiming it was created earlier. Nonetheless, if the signer can insert some data, via a transaction to the blockchain, then it is straightforward to guarantee protection against the backdating attack. Now, the signature is only considered validly timed after it appears on the ledger's state and it is posted within a predefined delay. Again, the formal guarantee needs to inherit the deficiencies of the ledger's output; in particular a verifier might in some round consider a signature accepting; whereas, another verifier does not, as the latter may have a shorter chain that does not contain the signature yet. But eventually every party will be able to check the timestamp. We view this separation between the timestamping abilities enabled by read/write vs read-only as an interesting feature which is exposed by our fine-grained treatment of timestamping. We note that this separation is not only theoretically interesting but has a clear implication in practice: unlike postdate security, backdate security using a cryptocurrency blockchain is not free of charge, since inserting information in the blockchain of any such cryptocurrencies has associated fees that the signer would need to pay.

Completing our treatment of timed signatures, we prove that combining the above two ideas, namely creating a signature with the beacon value and inserting it on the blockchain, yields a signature with both backdate and postdate security. We note that one might be tempted to assume that this level of security can be obtained without the use of the beacon, since the blockchain is immutable. Indeed, one can argue that postdate security is trivially solved by considering a signature valid once it is seen on the blockchain. This is however not the case, since a signer might generate the signature in the past with a future date, and only post it on the blockchain after that date (while using the signature in the meanwhile).

To see why the above makes a big difference, consider the following application scenario. A bank B has issued to Alice an electronic checkbook and wants to ensure that Alice cannot issue postdated signatures (e.g., to use them as collateral for a loan from another bank C). This cannot

be enforced by B by only requiring Alice to insert the signature on the blockchain, as Alice can issue the signature with a future date T, use with C at time $T' < T$ and only post it on the blockchain at time T. Bank C has no reason not to accept the signature as it knows that it will be considered valid at time T (even if Alice does not post it on the blockchain, the Bank C can do it for Alice). Mitigating a problem like this may be addressed by other techniques, e.g., by requiring the signer to post the transaction from the same public key as the one used for the signatures, however such workarounds would be using the ledger in a non-blackbox way. In any case this example demonstrates a delicate point in timestamping—namely the difference between the time object is created vs. when its timestamp becomes publicly valid—which highlights the usefulness of our fine-grained analysis.

The above issue becomes even more evident when considering timed signatures of knowledge, where we want to guarantee that the witness was known to the signer at the claimed time. We define a three-tier timed version of such signatures of knowledge analogously to the above time signatures, and show how these can be implemented by a timed version of non-interactive zero-knowledge proofs which we also introduce. We believe that both these primitives might have applications on autonomous and IoT systems where both the privacy and availability are of major concern. For instance, consider a case where a set of smart devices, in an IoT network, need to periodically prove their *availability* in zero-knowledge to a verifier, e.g. a smart contract. In this scenario, our timed NIZK proofs or signatures of knowledge (depending on a particular application) can be used by each device to prove that it knows the witness at a certain time, i.e. can prove it was available at a certain point in time.

Related Work. We have already reviewed the milestones in the timestamping literature and discussed its relation with the notions proposed in this paper. We have also discussed solutions using blockchain technologies, e.g. proofs of work and stake. For completeness we include a more detailed survey of that literature in Appendix A; in the same appendix, we discuss basic results in zero knowledge (including some recent attempts that use time [DNS98, LTCL07, EO95]). To our knowledge none of the existing blockchain-based solutions obtains timestamping with only ideal (blackbox) access to the ledger nor includes a formal composable proof of the claimed security. There is also literature on schemes called time-lock encryption and commitments, and time released signatures [RSW96, BN00, GJ03, LJKW18, LGR15]. Despite the similarity in the name, these works do not (aim to) achieve timestamping guarantees. For the sake of completeness, we also review these works in Appendix A. We also recall the work of Scafuro et al. [SSV19] in which the authors show how to obtain non-interactive witness-indistinguishable proof using the blockchain as a trusted-setup assumption.

Notation. We denote the security parameter by λ and use "||" as concatenation operator. For a finite set Q, $x \xleftarrow{\$} Q$ denotes a sampling of x from Q with uniform distribution. In this paper, PPT stands for probabilistic polynomial time. We use $\mathsf{poly}(\cdot)$ to indicate a generic polynomial function. Let $\mathbf{v}$ be a sequence of elements (vector); by $\mathbf{v}[i]$ we mean the i-th element of $\mathbf{v}$. Also, by $\mathbf{v}_{|i}$ and $\mathbf{v}_{|i,j}$ we mean the sequence of elements of $\mathbf{v}$ in the ranges $[1, \mathbf{v}[j]]$ and $[\mathbf{v}[i], \mathbf{v}[j]]$, respectively. Analogously, for a bi-dimensional vector M, we denote with $M[i, j]$ the element identified by the i-th row and the j-th column of M. Moreover, an adversary is denoted by $\mathcal{A}$. We assume the readers are familiar with standard notions such as *commitment* and UC-security. See Appendices C and D for their formal definitions.

Organization of Paper. The remainder of this paper is structured as follows. In Section 2 we put forth our execution modeling reviewing relevant aspects of the (G)UC framework. In Section 3 we provide an technical overview of all our results condensing the content of all the following sections. In Section 4 we provide the description of wrapper for the ledger functionality to capture the entropy contained in the blockchains. In Section 5 we describe our (weak) beacon functionality and demonstrate how it can be realized via the ledger functionality. Timed signatures are introduced in Section 6 together with constructions that realize them using the ledger as a resource as well as a standard digital signature functionality. In Section 7 we present timed proof of knowledge while in Section 8 we demonstrated the generalization to signatures of knowledge.

2 The Model

Following the recent line of works proving composable security of blockchain ledgers [BMTZ17, BGK+18] we provide our protocols and security proofs in Canetti's universal composition (UC) framework [Can01]. In this section we discuss the main components of our real-world model (including the associated hybrids). We assume that the reader is familiar with simulation-based security and has basic knowledge of the UC framework. We review all the aspects of the execution model that are needed for our protocols and proof, but omit some of the low-level details and refer the more interested reader to these works wherever appropriate. We note that for obtaining a better abstraction of reality, some of our hybrids are described as global (GUC) setups [CDPW07]. The main difference of such setups from standard UC functionalities is that the former is accessible by arbitrary protocols and, therefore, allow the protocols to share their (the setups') state. The low-level details of the GUC framework—and the extra points which differentiate it from UC—are not necessary for understanding our protocols and proofs; we refer the interested reader to [CDPW07] for these details.

Protocol participants are represented as parties—formally Interactive Turing Machine instances (ITIs)—in a multi-party computation. We assume a central adversary $\mathcal{A}$ who corrupts miners and uses them to attack the protocol. The adversary is *adaptive*, i.e., can corrupt (additional) parties at any point and depending on his current view of the protocol execution. Our protocols are synchronous (G)UC protocols [BMTZ17, KMTZ13]: parties have access to a (global) clock setup, denoted by $\mathcal{G}_{\mathtt{clock}}$, and can communicate over a network of authenticated multicast channels. We assume instant and *fetch-based* delivery channels [KMTZ13, CGHZ16]. Such channels, whenever they receive a message from their sender, they record it and deliver it to the receiver upon his request with a "fetch" command. In fact, all functionalities we design in this work will have such fetch-based delivery of their outputs. We remark that the instant-delivery assumption is without loss of generality as the channels are only used for communicating the timestamped object to the verifier which can anyway happen at any point after its creation. However, our treatment trivially applies also to the setting where parties communicate over bounded-delay channels as in [BMTZ17].

We adopt the *dynamic availability* model implicit in [BMTZ17] which was fleshed out in [BGK+18]. We next sketch its main components: All functionalities, protocols, and global setups have a dynamic party set. i.e., they all include special instructions allowing parties to register, deregister, and allowing the adversary to learn the current set of registered parties. Additionally, global setups allow any other setup (or functionality) to register and deregister with them, and they also allow other setups to learn their set of registered parties. For more details on the registration process we refer the reader to Appendix F. We conclude this section by elaborating on the hybrid functionali-

ties and global setups used by our protocol. These are standard functionalities from the literature; however, for self-containment we have included formal descriptions in the Appendices.

The Clock Functionality $\mathcal{G}_{\texttt{clock}}$ (cf. Fig. 14). The *clock functionality* was initially proposed in [KMTZ13] to enable synchronous execution of UC protocols. Here we adopt its global-setup version, denoted by $\mathcal{G}_{\texttt{clock}}$, which was proposed by [BMTZ17] and was used in the (G)UC proofs of the ledger's security.[4] $\mathcal{G}_{\texttt{clock}}$ allows parties (and functionalities) to ensure that the protocol they are running proceeds in synchronized rounds; it keeps track of round variable whose value can be retrieved by parties (or by functionalities) via sending to it the pair: `CLOCK-READ`. This value is increased when every honest party has sent to the clock a command `CLOCK-UPDATE`. The parties use the clock as follows. Each party starts every operation by reading the current round from $\mathcal{G}_{\texttt{clock}}$ via the command `CLOCK-READ`. Once any party has executed all its instructions for that round it instructs the clock to advance by sending a `CLOCK-UPDATE` command, and gets in an idle mode where it simply reads the clock time in every activation until the round advances. To keep more compact the description of our functionalities that rely on $\mathcal{G}_{\texttt{clock}}$, we implicitly assume that whenever an input is received the command `CLOCK-READ` is sent to $\mathcal{G}_{\texttt{clock}}$ to retrieve the current round. Moreover, before giving the output, the functionalities request to advance the clock by sending `CLOCK-UPDATE` to $\mathcal{G}_{\texttt{clock}}$.

The Random Oracle Functionality $\mathcal{F}_{RO}$ (see App. B Fig. 15). As typically in cryptographic proofs the queries to hash function are modeled by assuming access to a random oracle functionality: Upon receiving a query $(\texttt{EVAL}, sid, x)$ from a registered party, if x has not been queried before, a value y is chosen uniformly at random from $\{0,1\}^\lambda$ (for security parameter λ) and returned to the party (and the mapping (x, ρ) is internally stored). If x has been queried before, the corresponding ρ is returned.

The Global ledger Functionality $\mathcal{G}_{\texttt{ledger}}$**.** The last functionality (in fact, a global setup) is a cryptographic distributed transaction ledger, and is the main tool used in our constructions. We use the (backbone) ledgers proposed in the recent literature [BMTZ17, BGK+18] in order to describe a transaction ledger and its properties. As proved in [BMTZ17, BGK+18] such a ledger is implemented by known permissionless blockchains based on either proof-of-work (PoW), e.g., the Bitcoin, or poof-of-stake (PoS) e.g., Ouroboros Genesis. The ledger stores an immutable sequence of blocks—each block containing several messages typically referred to as *transactions* and denoted by `tx`—which is accessible from the parties under some restrictions discussed below. It enforces the following basic properties that are inspired by [GKL15, PSS17]:

- *Ledger's growth.* The size of the state of the ledger should be growing—by new blocks being added—as the rounds advance.
- (ℓ, μ)-*Chain quality.* Let $\ell \in \mathbb{N}$ be a number which is super-logarithmic in the security parameter and $\mu \in \mathbb{N}$. In any sequence of ℓ blocks, at least $\mu > 0$ of them have to be contributed by honest parties—in this context, parties are often referred to *miners*.[5]
- *Transaction liveness.* Old enough (and valid) transactions are included in the next block added to the ledger state.

[4] As a global setup, $\mathcal{G}_{\texttt{clock}}$ also exists in the ideal world and the ledger connects to it to keep track of rounds.

[5] Typically chain quality is specified by the ratio ℓ/μ, but it is useful for our description to break this into two parameters.

We next give a brief overview of the ledger functionality $\mathcal{G}_{\texttt{ledger}}$ proposed in [BMTZ17, BGK+18], focusing on the properties of $\mathcal{G}_{\texttt{ledger}}$ that are relevant for understanding our results (for self-containment we have included the formal description of the ledger functionality proposed in [BMTZ17] in Appendix G). Along the way we also introduce some useful notation and terminology. We refer the reader interested on the low-level details of the ledger functionality and its UC implementation to [BMTZ17, BGK+18]. We note that with minor differences related to the nature of the resource used to implement the ledger, PoW vs PoS, the ledgers proposed in these works are identical.

At a high-level anyone (honest miner or the adversary) might submit a transaction to $\mathcal{G}_{\texttt{ledger}}$ which is validated by means of a filtering predicate, and if it is found valid it is added to a *buffer*. The adversary $\mathcal{A}$ is informed that the transaction was received and is given its contents.[6] Periodically, $\mathcal{G}_{\texttt{ledger}}$ fetches some of the transactions in the buffer and creates a block including these transactions and adds this block to its permanent state, denoted as $\texttt{state}$, which is a data structure that includes the sequences of blocks that the adversary can no longer change. (In [GKL15, PSS17] this corresponds to the *common prefix*.)

Any miner or the adversary is allowed to request a read of the contents of the state and every honest miner will eventually receive $\texttt{state}$ as its output. However, as observed in [BMTZ17], it is not possible to achieve with existing constructions that at any given point in time all honest miners see exactly the same blockchain length, so each miner may have a different view of the state which is defined by the adversary. Therefore, the functionality $\mathcal{G}_{\texttt{ledger}}$ defines, for every honest miner p_i, a subchain $\texttt{state}_i$ of the state of length $|\texttt{state}_i| = \texttt{pt}_i$ that corresponds to what p_i gets as a response when it reads the state of the ledger. For convenience, we denote by $\texttt{state}_{\lfloor \texttt{pt}_i}$ the subchain of state that finishes in the $\texttt{pt}_i$-th block. Informally, the adversary can decide the value of the pointer $\texttt{pt}_i$ for each miner, with the following constraints: (1) he can only move the pointers forward; and (2) he cannot set pointers for honest miners to be too far apart, i.e., more than $\texttt{WindowSize}$ state blocks. The parameter $\texttt{WindowSize} \in \mathbb{N}$ reflects the similarity of the blockchain to the dynamics of a so-called *sliding window*, where the window of size $\texttt{WindowSize}$ contains the possible views of honest miners onto $\texttt{state}$ and where the head of the window advances with the head of $\texttt{state}$.

3 Technical Summary

In this section, for the sake of brevity and due to the page limit, we provide a technical summary of our contributions. In the later sections, we provide elaborated versions of them.

3.1 Beacon functionality and enhanced ledger

In order to construct a source of sufficiently unpredictable and publicly verifiable randomness, we design and use a blockchain-based beacon. We investigate how an ideal beacon can be weakened so that it is implementable by a protocol which uses the ledger functionality and a random oracle. In particular, we specify a weak beacon functionality which is sufficiently strong to be used for timestamping cryptographic primitives. Our beacon, similar to [BMTZ17, BGK+18, GKL15, PSS17], relies on the assumption that the blocks generated by the honest parties include at least $\hat{\lambda}$ bits entropy. However, this does not mean that it is possible to extract at least $\hat{\lambda}$-bit randomness from

[6] Taking a peak at the actual implementation of the ledger, this buffer contains transactions that, although validated, are either not inserted into a valid block yet, or are in a block which is not yet deep enough in the blockchain to be considered immutable for an adversary.

a sequence of blocks that contains an honestly generated entry. Generally speaking, the reason is that parties work in parallel to extend the chain, and there is a possibility that they collide which gives the adversary the choice between the colliding blocks. This gives the adversary a bit more power in guessing the output of the beacon. Informally, the entropy of the honest block can be reduced by a factor that depends on the number of honest blocks proposed within a small window from the round in which the beacon emits its value. Nevertheless, as we will argue later, this issue can at most eliminate a few bits of entropy from the beacon. Attempting to capture the above, we hit a shortcoming of the ledger from [BMTZ17]. The reason is that the current definition of the ledger does not account for the entropy of the honest blocks. A way to rectify that would be to change the ledger functionality in a *non-back-box manner* and reprove the its security. To resolve the aforementioned issue, we introduce an explicit wrapper called WBU-wrapper. In a nutshell, the WBU-wrapper wraps the ledger functionality, i.e., takes control of all its interfaces, and acts as an upper relayer. Together with the formal definition of the WBU-wrapper we also show that the (UC-abstraction of the) Bitcoin backbone protocol in [BMTZ17] emulates the wrapped ledger. As a next step we define a *weak* beacon functionality $\mathcal{B}^{\mathsf{w}}$ and provide an instantiation of it using the wrapped ledger functionality. This functionality can be queried with a round number ρ, and return the couple $(\eta, \mathtt{tsl})$, where η denotes the random value, and $\mathtt{tsl}$ represents the output number (i.e., there is not a one-to-one correspondence between the rounds of $\mathcal{G}_{\mathtt{clock}}$ and the output number of $\mathcal{B}^{\mathsf{w}}$). Note that any implementation of an ideal randomness beacon is expected to meet (at least) the following requirements:

Agreement on the Output: The output of the beacon can be verified by any party who has access to the beacon.

Liveness: The beacon generates new values as time advances. The output of the beacon can be verified (albeit at some point in the future) by any party who has access to the beacon.

Perfect Unpredictability: No one should be able to bias or even predict (any better than guessing) the outcome of the beacon before it has been generated.

Nevertheless, due to the adversarial influence on the contents of the ledger, we cannot obtain a perfect beacon from the ledgers that are implemented by common cryptocurrencies (cf. also [BGZ16] for an impossibility). Indeed, we will allow the adversary to predict the next Δ outputs of $\mathcal{B}^{\mathsf{w}}$, and to generate a new output after at most R rounds. At a high level, our beacon protocol works as follows. A party that wants to compute the latest beacon's output simply needs to compute the hash of the latest $\ell-\mu+1$ blocks of the ledger. This ensures that at least one hashed block is honest, and therefore that the adversary cannot predict more than the next Δ outputs, where $\Delta = \ell - \mu$. Moreover, $R = \mathtt{MaxRound}$, where $\mathtt{MaxRound}$ denotes the maximum number of rounds after that the state of the ledger has to be extended. We note passing that one might be tempted to implement the $\mathcal{B}^{\mathsf{w}}$ by hashing only the last (stable) block of the hash chain, which would yield to a more efficient construction. However, as we will argue later, this approach is not generic and is suitable only for a certain type of blockchains. For more details on the wrapper and on the weak beacon we refer the reader to Section 4 and 5 respectively.

3.2 Timed digital signature

In this section, we provide a technical overview of our timed signatures. Here, we extend the standard notion of the digital signature by different levels of timing guarantees. In our model, a timestamped signature σ for a message m is equipped with a time mark τ that contains information

about when σ was computed by the signer (σ corresponds to an output of $\mathcal{G}_{\mathtt{clock}}$). We refer to this special notion of signature as *Timed Signature (TSign)*. We define three categories of security for TSign: *backdate* security, *postdate* security, and their combination which we refer to just as *timed security*. Intuitively, backdate security guarantees that the signature σ time-marked with τ has been computed some time *before* τ; postdate security guarantees that the signature σ was computed some time *after* τ; and timed security provides to the party that verifies the signature σ a time interval around τ in which σ was computed. We will formally define these three new security notions with a single notion $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ parameterized by a flag $\mathtt{t} \in \{+,-,\pm\}$ where $\mathtt{t} = \text{“}-\text{”}$ indicates that the functionality guarantees backdate security, $\mathtt{t} = \text{“}+\text{”}$ indicates postdate security, and $\mathtt{t} = \text{“}\pm\text{”}$ indicates timed security. Analogously to the weak beacon, $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ and all parties that have access to this functionality, are registered to $\mathcal{G}_{\mathtt{clock}}$ which provides the notion of time inherently required by our model.

In a nutshell, and more formally, the functionality $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ provides to its registered parties a new time-slot $\mathtt{tsl} \in \mathbb{N}$ every R rounds (in the worst case). Once a time slot $\mathtt{tsl}$ is issued, it can be used to time(stamp) a signature σ. The meaning of $\mathtt{tsl}$ depends on the notion of security that we are considering. For backdate security (i.e., $\mathtt{t} = \text{“}-\text{”}$), a signature σ marked with $\mathtt{tsl}$ denotes that σ was computed during a time slot $\mathtt{tsl}' \leq \mathtt{tsl}$. For postdate security ($\mathtt{t} = \text{“}+\text{”}$) $\mathtt{tsl}$ denotes that σ was computed during a time slot $\mathtt{tsl}' \geq \mathtt{tsl}$. For timed security, the signature σ is equipped with two time-marks $\mathtt{tsl}_{\mathtt{back}}$ and $\mathtt{tsl}_{\mathtt{post}}$ that denote that σ was computed in a time-slot $\mathtt{tsl}'$ such that $\mathtt{tsl}_{\mathtt{post}} \leq \mathtt{tsl}' \leq \mathtt{tsl}_{\mathtt{back}}$. A new time-slot issued by $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ can be immediately seen and used by $\mathcal{A}$. However, the adversary can delay honest parties from seeing new time-slots—i.e., truncate the view that each honest party has of the available time-slots. That is, for each party p_i, $\mathcal{A}$ can decide to *hide* the most recent W-many available time-slots. There are also other subtleties that the ideal functionality $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ needs to capture and this makes the functionality more complicated that one might expect. We refer the reader to the formal definition of the functionality in Sec. 6.

To obtain postdate security we rely on the weak beacon and on signatures. The signer in our case queries the beacon thus obtaining the pair $(\eta, \mathtt{tsl})$ where η represents the $\mathtt{tsl}$-th output of $\mathcal{B}^{\mathsf{w}}$ (which is also the most recent) and sign the message together with with η. In order to obtain backdate-security, the signer inserts its signature, via a transaction to the blockchain. Now, the signature is only considered validly timed after it appears on the ledger's state and is posted within a predefined delay. Moreover, as we will prove, combining the above two ideas yields a signature with both backdate and postdate security. For the formal constructions and definition we refer the reader to Section 6.

3.3 Timed Zero-Knowledge PoK (TPoK) and Signature of Knowledge (TSoK)

TPoK. In this section we apply the same methodology used for timed signatures in the previous section to defined analogously timed versions of non-interactive zero-knowledge proofs of knowledge. The basis for our approach is the standard UC Non-Interactive Zero-Knowledge functionality proposed $\mathcal{F}_{\mathsf{NIZK}}$ in [GOS12]. Roughly, $\mathcal{F}_{\mathsf{NIZK}}$ considers two parties, a prover and a verifier. The prover provides as input to $\mathcal{F}_{\mathsf{NIZK}}$ an $\mathcal{NP}$-statement x. The functionality checks whether $(x, w) \in \mathsf{Rel}$ (where Rel is an $\mathcal{NP}$-relation) and if the check is successful then $\mathcal{F}_{\mathsf{NIZK}}$ stores (x, π) and sends a proof π to the prover. The verifier can query the functionality with a couple (x, π), and if $\mathcal{F}_{\mathsf{NIZK}}$ stores the couple (x, π) sends 1 to the verifier, 0 otherwise.

We extend $\mathcal{F}_{\mathsf{NIZK}}$ to consider different levels of timed-security, in the same way as we have done for signatures: A proof π generated with respect to an $\mathcal{NP}$-statement is equipped with a time-mark

$\mathtt{tsl}$ that gives some information about when π was computed. We refer to this notion of NIZK as TPoK and, also in this case, we consider three categories of security: *backdate*, *postdate* and *timed security*. The formalization of these notions is given by means of the UC-functionalities $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$, with $\mathtt{t} = \text{“}-\text{”}, \text{“}+\text{”}, \text{“}\pm\text{”}$. $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$ is formally described in the Sec 7. In this setting, intuitively, a prover can send to the functionality a couple (x, w), and if $(x, w) \in \mathsf{Rel}$ then $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$ returns a couple $(\pi, \mathtt{tsl})$, where $\mathtt{tsl}$ is a time-mark. A verifier that queries $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$ with a couple $(\pi, \mathtt{tsl})$ and gets 1 as the answer from the functionality has the guarantees that: 1) the prover knew the witness for the $\mathcal{NP}$-statement x (like in case of $\mathcal{F}_{\mathsf{NIZK}}$) and 2) the proof π was generated (using the witness) in some moment specified by $\mathtt{tsl}$. We also provide three instantiations, one for each of the three security notions mentioned above. That is, we show a protocol $\Pi_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$ that UC-realize $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$ for all $\mathtt{t} \in \{-, +, \pm\}$. $\Pi_{\mathsf{TPoK}}^{\mathsf{w},-}$ is similar to $\Pi_{\sigma}^{\mathsf{w},-}$, indeed the prover of $\Pi_{\mathsf{TPoK}}^{\mathsf{w},-}$, on input $(x, w) \in \mathsf{Rel}$, just needs to compute a NIZK proof (e.g. computed using $\mathcal{F}_{\mathsf{NIZK}}$) and store it into the ledger. $\Pi_{\mathsf{TPoK}}^{\mathsf{w},+}$ instead needs to use $\mathcal{B}^{\mathsf{w}}$. $\Pi_{\mathsf{TPoK}}^{\mathsf{w},+}$ follows the *commit-and-prove* paradigm in which the prover commits to the witness w for the $\mathcal{NP}$-statement x to be proven, and then proves to the verifier that the committed message corresponds to a valid witness for x. In our protocol we want to associate some time-stamp to the proof generated by the prover, so we slightly modify the above approach as follows. The prover obtains the pair $(\eta, \mathtt{tsl})$ by invoking the weak beacon $\mathcal{B}^{\mathsf{w}}$ with the current round ρ, where η represents the $\mathtt{tsl}$-th output of $\mathcal{B}^{\mathsf{w}}$ (which is also the most recent).

Then the prover computes a commitment $\mathtt{com}$ of $w||\eta$ and proves to the verifier that $\mathtt{com}$ contains a witness for x concatenated with η. The verifier, upon receiving the proof computed by the prover accepts it if and only if the following two conditions hold: 1) value η has been output by $\mathcal{B}^{\mathsf{w}}$ in some round τ; 2) the NIZK proof given by the prover is accepting. Since the NIZK that we use is a PoK and we assume that a malicious prover cannot predict the output of the weak beacon $\mathcal{B}^{\mathsf{w}}$ more than $\delta = \mathtt{MaxRound} \cdot (\mathtt{WindowSize} + \ell - \mu)$ rounds in advance then the verifier has the guarantee that the proof has been computed (and that the witness w was known by the prover) in some moment *subsequent* to $\tau - \delta$. The protocol $\Pi_{\mathsf{TPoK}}^{\mathsf{w},\pm}$ instead internally runs $\Pi_{\mathsf{TPoK}}^{\mathsf{w},+}$ and store the proof on the ledger, and this is sufficient to provide timed security.

TSoK. The work of Chase *et al.* [CL06] introduces the notion of *signature of knowledge (SoK)*. A signature of knowledge schemes allows to issue signatures on behalf of any $\mathcal{NP}$-statement. That is, receiving a valid signature for a message m with respect to an $\mathcal{NP}$-statement x means that the signer of m knew the witness w for the $\mathcal{NP}$-statement x. Exactly in the same spirit of signature and NIZK, we define the notions of backdate, postdate and timed SoK. That is, we define the UC-functionalities $\mathcal{F}_{\mathsf{TSoK}}^{\mathsf{w},\mathtt{t}}$ with $\mathtt{t} \in \{-, +, \pm\}$. Those functionalities are analogous to the functionalities for timed signatures and NIZKs we have just discussed (we refer the reader to Sec. 8 for the formal description of the functionalities and the protocols). It should be easy to see that postdate, backdate and timed secure NIZK implies respectively postdate, backdate and timed secure TSoK. Indeed we observe in the $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$-hybrid model it is possible to obtain a protocol that UC-realizes $\mathcal{F}_{\mathsf{TSoK}}^{\mathsf{w},\mathtt{t}}$. The approach is to construct a commitment of a witness contateneted with the message that we want to sign and then run $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$ to prove that the commitment actually contains the concatenation of the witness for x and the message m.

4 Weak Block Unpredictability (WBU)

A delicate point about the ledger from [BMTZ17, BGK+18] is the way it enforces the chain quality property from [GKL15]. Recall that this property requires that in every sequence of ℓ blocks put into the state, at least μ of them have to be associated with honest leaders. The ledger enforces this by the simulator declaring in a special field—corresponding to a coinbase transaction—the identity of the party who should be considered as having inserted each block; the extend-policy predicate will then ensure that the simulator has to declare blocks as created by honest parties with a sufficiently high frequency as above.

Our analysis—as well as the security analyses of the ledger [BMTZ17, BGK+18] and the backbone abstraction of the protocol [GKL15, PSS17]—uses the assumption that the coinbase transaction of such *honest* blocks includes at least $\hat{\lambda}$ bits randomly chosen by an honest party[7]. One might be tempted to deduce that it is possible to extract (at least) $\hat{\lambda}$ bits of randomness from each sequence of ℓ blocks. However, this is not the case. Informally, the reason is that parties are in parallel working to extend the chain, and there is a chance that they might collide, giving the adversary the choice between the colliding blocks. And, although, one can use the existence of uniquely successful rounds—i.e., rounds in which only one honest party succeeds in solving the PoW puzzle—guaranteed to exist by the analysis of [GKL15], this is not sufficient: The problem is that the most recent part of the blockchain is not stable (it is not part of the common prefix) so the adversary can, in principle overwrite it, potentially using alternative postfixes (which can include blocks even by honest parties that have inconsistent view of the blockchain's head). This gives the adversary a bit more slackness in guessing the output of the beacon. Informally, the entropy of the honest block can be reduced by a factor that depends on the number of honest blocks proposed within a small window from the round in which the beacon emits its value. However, as we will argue below, this grinding might at most eliminate a few bits of entropy from the beacon.

Attempting to capture the above, we hit a shortcoming of the ledger from [BMTZ17]. The reason is that in the current definition of the ledger, there is no way for an honest party to insert some random value into a block's content, as the ledger allows its simulator to have full control of the contents of the blocks inserted into the state. Note that the extend policy algorithm (responsible for enforcing the chain quality and liveness) in the ledger functionality does not account for the above property. A way to rectify that would be to adjust the extend policy, but this would then mean changing the ledger in a non-transparent manner.

Instead, here we choose to take the following approach, which was also proposed in [BMTZ17] for explicitly capturing assumptions—in the case of [BMTZ17] it was used for capturing honest majority of computing power: We introduce an explicit wrapper that exactly captures the property that yields the above entropic argument. We refer to this wrapper as WBU-wrapper, and to the corresponding property that it enforces as weak beacon unpredictability, and denote it as $\mathcal{W}_{\text{WBU}}$.

In a nuttshel, the WBU-wrapper wraps the ledger functionality, i.e., takes control of all its interfaces, and acts as a relayer except for the following behavior: It might accept a special input from the simulator in any round (even multiple times per round). Once it does, it returns a random nonce N and records the pair (N, ρ), where ρ is the current round. Furthermore, for each block inserted into the state, it records the block along with the round in which this insertion occurred (note that the wrapper can easily detect insertions by reading the state through all miner's

[7] Formally, in [BMTZ17, BGK+18] the ledger chooses the contents of the coinbase transactions of honest blocks, including the nonces and possible new keys/wallet-addresses, hence the simulator cannot predict them.

interfaces). If it observes that the simulator does not ask for a nonce for more than $\ell \cdot \texttt{MaxRound}$ rounds, or does not insert a block with its coinbase including a previously output nonce N within a δ-long time window from the creation of N, where $\delta = \texttt{MaxRound} \cdot \texttt{WindowSize} \cdot (\ell - \mu)$, then the wrapper halts. The formal definition of the weak block unpredictability wrapper is as follows.

Definition 1 (Weak Block Unpredictability Wrapper: $\mathcal{W}_{\text{WBU}}$) *A $\mathcal{W}_{\text{WBU}}$ is a functionality-wrapper (that wraps $\mathcal{G}_{\texttt{ledger}}$) and operates as follows:*

- *Upon receiving (new_nonce) from the simulator it returns random fresh $N \in \{0,1\}^\lambda$ to the simulator, and records (N, ρ), where ρ is the current round ($\mathcal{W}_{\text{WBU}}$ can get this round by querying the clock.)*
- *For any block proposed by the simulator that makes it into the ledger's state, which is flagged (via the coinbase transaction, by the simulator) as originating from an honest party ($\mathcal{W}_{\text{WBU}}$ can detect this as discussed above). If this block does not contain some N previously recorded, then halt; otherwise, if (N, ρ') has been recorded and the current round index is $\rho > \rho' + \delta = \rho' + \texttt{MaxRound} \cdot \texttt{WindowSize} \cdot (\ell - \mu)$ then halt. In any other case simply relay messages between the wrapped functionality and the entities it is connected to (i.e., the simulator, the environment, and the global setups it has registered with.)*

As an additional contribution of this work, which we believe might be of independent interest, we prove the following lemma which states that the (UC abstraction of the) Bitcoin backbone protocol from [BMTZ17] emulates the wrapped ledger $\mathcal{W}_{\text{WBU}}[\mathcal{G}_{\texttt{ledger}}]$, where, $\mathcal{G}_{\texttt{ledger}}$ is the ledger from [BMTZ17]. We prove that by showing that the blocks generated by the protocol satisfy the weak beacon unpredictability property. The lemma follows then directly by observing that the simulator of [BMTZ17] internally generates the coinbase for honest blocks by emulating the honest protocol. The detailed proof can be found in Appendix E.4.

Lemma 1 *The (UC version of the) Bitcoin backbone protocol Π_{BB} [BMTZ17] realizes $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$.*

5 The (Weak) Beacon functionality

Before discussing the timed versions of the cryptographic primitives proposed in our work, we describe how to utilize the blockchain to derive a source of sufficiently unpredictable randomness, which we refer to as a *weak (randomness) beacon*. Note that any implementation of an ideal randomness beacon would be expected to satisfy (at least) the following properties:

Agreement on the Output: The output of the beacon can be verified by any party who has access to the beacon.

Liveness: The beacon generates new values as time advances. The output of the beacon can be verified (albeit at some point in the future) by any party who has access to the beacon.

Perfect Unpredictability: No one should be able to bias or even predict (any better than guessing) the outcome of the beacon before it has been generated.

However, as discussed in the introduction, due to the adversarial influence on the contents of the ledger, we cannot obtain such a perfect beacon from the ledgers that are implemented by common cryptocurrencies (cf. also [BGZ16] for an impossibility). Nonetheless, as it turns out, even under a worst-case analysis as in [GKL15, BMTZ17], the contents of the ledger are periodically updated

check_time_table($\mathcal{T}, \mathcal{T}', \mathcal{P}, R, \mathsf{max_{tsl}}$)

If all the following conditions are satisfied

1. $\mathcal{T}$ is a prefix of $\mathcal{T}'$;
2. For each corrupted party p_i $\mathcal{T}'[R, p_i] = \mathsf{max_{tsl}}$;
3. For all $j \in \{0, \ldots, R-1\}$, either $\mathcal{T}'[j+1, p_i] = \mathcal{T}'[j, p_i]$ or $\mathcal{T}'[j+1, p_i] = \mathcal{T}'[j, p_i] + 1$;
4. For all $l \in \{0, \ldots, R\}$, for all $p_i, p_j \in \mathcal{P}$, $|\mathcal{T}'[l, p_j] - \mathcal{T}'[l, p_i]| \leq \mathtt{WindowSize}$

then return 1, **else** return 0.

force_time_table($\mathcal{T}, \mathcal{P}, R, \mathsf{max_{tsl}}$)

Generate a random $\mathcal{T}'$ such that check_time_table($\mathcal{T}, \mathcal{T}', \mathcal{P}, R, \mathsf{max_{tsl}}$) $= 1$

Return $\mathcal{T}'$.

define_new_set($\mathtt{MaxSize}$)

Define $\mathcal{S} = \emptyset$. For $i = 1$ to $\mathtt{MaxSize}$ pick $\eta_i \xleftarrow{\$} \{0,1\}^\lambda$ and add η_i to $\mathcal{S}$. Return $\mathcal{S}$.

check_validity(j)

$c \leftarrow 0$

For $i = j - \ell, \ldots, j$

Parses $\mathcal{H}[i]$ as $(\eta_i, \mathtt{hflag}_i)$

If $\mathtt{hflag} = 1$ **then** set $c \leftarrow c + 1$

if $c \geq \mu$ **then** return 1, **else** return 0.

force_liveness($\mathsf{max_{tsl}}, \mathcal{T}, \mathcal{H}$)

$\mathsf{max_{tsl}} \leftarrow \mathsf{max_{tsl}} + 1$,

$\mathcal{T} \leftarrow$ force_time_table($\mathcal{T}, \mathcal{P}, R, \mathsf{max_{tsl}}$),

$\eta \xleftarrow{\$} \{0,1\}^\lambda$,

$\mathcal{H}[\mathsf{max_{tsl}}] \leftarrow (\eta, 1)$.

return $(\mathsf{max_{tsl}}, \mathcal{T}, \mathcal{H})$

check_liveness($\tau_{\mathtt{last}}, R$)

if $\tau_{\mathtt{last}} + \mathtt{MaxRound} = R$ **then** return 0 **else** return 1

Figure 1: Auxiliary Procedures.

with fresh unpredictable randomness. In the following, we provide a formal definition of a beacon satisfying a weaker notion of liveness and unpredictability, which as we will prove, can be constructed having blackbox access to the global functionality $\mathcal{W}_{\mathrm{WBU}}(\mathcal{G}_{\mathtt{ledger}})$. We refer to this beacon as a *weak beacon*. As we shall show, this beacon will be sufficient for our timestamping schemes.

In a nutshell, our weak beacon generates an unpredictable value η every Δ outputs. Concretely, we define our weak beacon as a UC-functionality $\mathcal{B}^{\mathsf{w}}$ in the $\mathcal{G}_{\mathtt{clock}}$-hybrid model. Note that an ideal beacon functionality is straightforward to define in this model as follows. It maintains a vector $\mathcal{H}$ of random values available to anyone upon request, and in each round it appends to this string a new uniformly random value. Before we formally define our weak beacon $\mathcal{B}^{\mathsf{w}}$, we review the ways in which our weak beacon relaxes the ideal-beacon properties, and the additional capabilities it offers to the adversary. $\mathcal{B}^{\mathsf{w}}$ is parameterized by a set of parameters $\mathsf{w} = ((\mu, \ell), \mathtt{MaxRound}, \mathtt{WindowSize}, \mathtt{MaxSize})$ whose role will become clear as we go over the adversary's capabilities:

Eventual Agreement on the Output: Similar to the ideal beacon, the functionality maintains an output sequence vector $\mathcal{H}$. However, instead of the parties guaranteed a consistent view of $\mathcal{H}$,

the adversary might choose a prefix of $\mathcal{H}$ that each party sees, with the restriction that length difference of the prefixes seen by any two parties in any round is upper bounded by a parameter $\texttt{WindowSize}$. More precisely, each party p_i can see only the first $\texttt{pt}_i$ elements of $\mathcal{H}$, where $\texttt{pt}_i$ is adversarially chosen in each round, with the restriction that the adversary is that $|\mathcal{H}| - \texttt{pt}_i \leq \texttt{WindowSize}$ for all p_i registered to $\mathcal{B}^{\mathsf{w}}$. In our weak beacon functionality this restriction will be enforced by means of a checking procedure, denoted as $\mathsf{check\_time\_table}$, which will be executed whenever the adversary attempts to rewrite indexes; if the check fails then another procedure, $\mathsf{force\_time\_table}$, is invoked which overwrites the adversary's choices with values of $\texttt{pt}_i$ that adhere to the above policy.

Slow Liveness: $\mathcal{B}^{\mathsf{w}}$ does not necessarily generate a new value in every round. Instead, the adversary can delay the generation of a new value but only by at most $\texttt{MaxRound}$ rounds.

Weak Unpredictability: The adversary has the following influence on the output of the beacon:

- The adversary can bias some of the beacon's outputs. More precisely, assume that $\mathcal{B}^{\mathsf{w}}$ is about to choose its ith value to be appended to its output vector $\mathcal{H}$. The adversary is given a set $\mathcal{S}_i$ of random values (where $|\mathcal{S}_i| \leq \texttt{MaxSize} = \mathsf{poly}(\lambda)$) and a choice: he can either allow the beacon to randomly choose the i-th output (in this case this output is considered honest), or he can decide on a value $\eta_i \in \mathcal{S}_i$ to append to the output vector. But, the restriction is that within every window of ℓ outputs, at least μ of them will be honest (looking ahead, this will be enforced by means of a procedure $\mathsf{check\_validity}$ described in the Fig.1.)
- The adversary can predict, in the worst case, the next $\ell - \mu$ outputs of the beacon. More precisely, let n be the size of $\mathcal{H}$; the adversary can ask $\mathcal{B}^{\mathsf{w}}$ to see $\ell - \mu$ sets $\mathcal{S}_{n+1}, \ldots, \mathcal{S}_{n+\ell-\mu}$ from which the next $\ell - \mu$ outputs will be chosen. In terms of rounds, this means that at any point the adversary might *predict* the output of a beacon for up to the next $\delta = (\ell - \mu + \texttt{WindowSize}) \cdot \texttt{MaxRound}$ rounds.

In the following, we elaborate on the exact power that each of the above properties yields to the adversary. For capturing eventual agreement on the output and slow liveness, we introduce the notion of a *time table* $\mathcal{T}$. It is a table with one column for each party that has ever been seen or registered with the beacon, indexed by the ID of the corresponding party (recall that we allow parties to register and deregister), and one row for each (clock) round. The table is extended in both dimensions as new parties register and as the time advances. For a party p_i and (clock-)round τ, the entry $\mathcal{T}[\tau, p_i]$ is an integer $\texttt{tsl}$ that we call *time-slot index*. This value $\texttt{tsl}$ defines the size of the prefix of the beacon's output $\mathcal{H}$ that p_i can see at round τ. That is, p_i at round τ can request any of the first $\texttt{tsl}$ outputs of $\mathcal{B}^{\mathsf{w}}$, denoted by $\mathcal{H}[1], \ldots, \mathcal{H}[\texttt{tsl}]$.

The adversary is allowed to instruct $\mathcal{B}^{\mathsf{w}}$ as to how $\mathcal{T}$ should be populated under the following restrictions: (1) for any party the values of its column, i.e., its time-slot indices, are monotonically non-decreasing and they are increasing by at least once in every $\texttt{MaxRound}$ rounds (this will enforce slow liveness), and (2) in any given round/row, no two time-slot indices (of two different parties) can be more than $\texttt{WindowSize}$ far apart (this together will enforce the eventual agreement property). These properties are formally enforced by two procedures, called $\mathsf{force\_time\_table}$ and

$\mathsf{check\_time\_table}$ that check if the adversary complies with the above policy as follows: The procedure $\mathsf{check\_time\_table}$ takes as input the current time table $\mathcal{T}$, a new table $\mathcal{T}'$ proposed by the adversary, the set of parties $\mathcal{P}$ registered to $\mathcal{B}^{\mathsf{w}}$, the current round R, $\mathsf{max}_{\mathsf{tsl}} = |\mathcal{H}|$; it outputs 0 if $\mathcal{T}'$ is invalid, and 1 otherwise. The procedure $\mathsf{force\_time\_table}$ is invoked to enforce the policy mandated by $\mathsf{check\_time\_table}$ in case the adversary is caught trying to violate it. In a nutshell, it generates a valid and randomly generated time table $\mathcal{T}'$ to be adopted instead of the adversary's

proposal. More concretely, force_time_table is invoked in the following two cases: 1) If $\mathcal{H}$ has not been extended in the last MaxRound rounds. In this case $\mathcal{B}^{\mathsf{w}}$ generates a random output, appends it to $\mathcal{H}$ and extends $\mathcal{T}$ using force_time_table. 2) If the adversary has not updated $\mathcal{T}$ in the last round, then a new $\mathcal{T}'$ (that extends the previous one) is generated via force_time_table.

The above two procedures are formally described in Fig. 1 and are, in fact, useful also for the definition of our other (timed) UC-functionalities described in the following section. The trickiest of the above properties to capture (and enforce in the functionality) is weak unpredictability. The idea is the following. Assume that the beacon has already generated outputs $\eta_1, \ldots, \eta_{i-1}$, where η_{i-1} was generated in round τ. Recall that, per the slow liveness property, the beacon does not generate outputs in every round. In every round after τ, the adversary is given a sequence of $\ell - \mu$ *output candidate sets* $\mathcal{S}_i, \ldots, \mathcal{S}_{i+\ell-\mu}$ sampled by $\mathcal{B}^{\mathsf{w}}$ and can do one of the following:

(1) decide to set the i-th beacon's output to a value from $\mathcal{S}_i$ of his choice. In this case, η_i is set to this value and flagged as dishonest (this is formally done by setting a flag $\mathtt{hflag}_i \leftarrow 0$ and storing the pair $(\eta_i, \mathtt{hflag}_i)$); the adversary is also given a next set $\mathcal{S}_{i+\ell-\mu+1}$ of size MaxSize sampled by the beacon by choosing MaxSize-many random values from $\{0,1\}^\lambda$ (cf. procedure define_new_set(MaxSize) in Fig. 1). Looking ahead in our beacon protocol, λ will correspond to the bits of entropy that are guaranteed to be included in an honestly generated ledger block. $\mathcal{S}_{i+\ell-\mu+1}$ will be the output candidate set for the $(i+\ell-\mu+1)$-th beacon output.

(2) instruct the beacon to ignore $\mathcal{S}_i$ and instead choose a uniformly random value for η_i. In this case, the beacon marks the i-th output as honest, i.e., sets $\mathtt{hflag}_i := 1$, informs the adversary about η_i, disposes of all existing output candidates sets, samples $\ell - \mu$ fresh candidates sets $\mathcal{S}_{i+1}, \ldots, \mathcal{S}_{i+\ell-\mu+1}$ and hands them to the adversary.

(3) instruct the beacon to not include any new output in the current round.

The choice (1) above captures the fact that the adversary can predict the next $\ell - \mu$ outputs of the beacon. However, to ensure that the above weakened unpredictability is meaningful, does not mess with liveness, and also achieves a guarantee similar to the chain quality property—i.e. that a truly random (honest) output of length λ is generated in sufficiently small intervals—the beacon enforces a policy on the adversary which ensures that the adversary's choices abide to the following restrictions: (A) any sequence of ℓ outputs of the beacon contains (at least) μ honest outputs, generated (randomly) by $\mathcal{B}^{\mathsf{w}}$, and (B) the adversary can leave the beacon without an output for at most MaxRound sequential rounds. Condition A is checked by the procedure check_validity whenever the adversary attempts to propose a new output from the corresponding candidate set, by taking choice (1) above; if the check fails the proposal of the adversary is ignored. Condition B is checked by procedure force_liveness($\mathsf{max}_{\mathsf{tsl}}, \mathcal{T}, \mathcal{H}$); if it fails, i.e., the adversary tries to delay the beacon's update by more than MaxRound rounds, then procedure force_liveness($\mathsf{max}_{\mathsf{tsl}}, \mathcal{T}, \mathcal{H}$) is invoked which forces the above policy in a default manner. The helper procedures and the formal description of our weak beacon functionality are included in Fig. 1 and 2, respectively.

5.1 Our weak beacon protocol

In this section, we propose a protocol that realizes the $\mathcal{B}^{\mathsf{w}}$ functionality in the $(\mathcal{W}_{\mathrm{WBU}}(\mathcal{G}_{\mathtt{ledger}}), \mathcal{F}_{RO})$-hybrid model. We first recall some of the properties that $\mathcal{G}_{\mathtt{ledger}}$ (similarly $\mathcal{W}_{\mathrm{WBU}}(\mathcal{G}_{\mathtt{ledger}})$) enjoys, and will be useful here.

1. The chain quality property of $\mathcal{G}_{\mathtt{ledger}}$ guarantees that any portion of state of length ℓ contains, at least, a portion of μ blocks originated by some honest parties. This means that the

The functionality is parametrized by the algorithms $\mathsf{check\_time\_table}(\mathcal{T}, \mathcal{T}', \mathcal{P}, R, \mathsf{max_{tsl}})$, $\mathsf{define\_new\_set}(\mathtt{MaxSize})$, $\mathsf{force\_time\_table}(\mathcal{T}, \mathcal{P}, R, \mathsf{max_{tsl}})$, $\mathsf{check\_validity}(\mathtt{tsl})$ along with the parameters $\mathtt{MaxSize}$, $\mathtt{MaxRound}$, (ℓ, μ), $\tau_{\mathtt{last}} \leftarrow 0$ $\mathsf{max_{tsl}} \leftarrow 0$, the time table $\mathcal{T}$, a set of parties $\mathcal{P}$ and adversary $\mathcal{A}$. $\mathcal{T}[0, p_i] = 0$ for all $p_i \in \mathcal{P}$. We assume the functionality to be registered to $\mathcal{G}_{\mathtt{clock}}$. The functionality manages the output vector $\mathcal{H}$ that records the random values issued by the functionality. $\mathcal{H}$ is initially empty. The functionality is also initialized with the sets $\mathcal{S}_1 \leftarrow \mathsf{define\_new\_set}(\mathtt{MaxSize}), \ldots, \mathcal{S}_{\ell-\mu} \leftarrow \mathsf{define\_new\_set}(\mathtt{MaxSize})$ and $\mathtt{p} \leftarrow 0$. Let R be the the response obtained by querying $\mathcal{G}_{\mathtt{clock}}$, upon receiving any input I from any party or from the adversary act as follows:

- If $\mathsf{check\_liveness}(\tau_{\mathtt{last}}, R) = 0$ then $(\mathsf{max_{tsl}}, \mathcal{T}, \mathcal{H}) = \mathsf{force\_liveness}(\mathsf{max_{tsl}}, \mathcal{T}, \mathcal{H})$
- If $\mathsf{check\_time\_table}(\mathcal{T}, \mathcal{T}, \mathcal{P}, R, \mathsf{max_{tsl}}) = 0$ then $\mathcal{T} \leftarrow \mathsf{force\_time\_table}(\mathcal{T}, \mathcal{P}, R, \mathsf{max_{tsl}})$.

Fetch

- If $I = (\mathtt{FETCH}, \tau_{\mathtt{req}}, sid)$ is received from party p_i or from $\mathcal{A}$ (on behalf of a corrupted party p_i) check if $\tau_{\mathtt{req}} \leq R$. If it is not then ignore I; otherwise, do the following:
  - $\mathtt{tsl} \leftarrow \mathcal{T}[\tau_{\mathtt{req}}, p_i]$;
  - parses $\mathcal{H}[\mathtt{tsl}]$ as $(\eta, \mathtt{hflag})$;
  - returns $(\mathtt{FETCH}, sid, \mathtt{tsl}, \eta)$ to p_i.

Sampling

- If $I = (\mathtt{READ\_SETS}, sid)$ is received from $\mathcal{A}$, then send $(\mathtt{READ\_SETS}, sid, \mathcal{S}_1, \ldots, \mathcal{S}_{\ell-\mu})$ to $\mathcal{A}$.
- If $I = (\mathtt{SET}, sid, \eta)$ is received from $\mathcal{A}$, then check if $\mathcal{H}[\mathsf{max_{tsl}} + 1]$ has not been set yet, $\mathsf{check\_validity}(\mathsf{max_{tsl}}) = 1$ and $\eta \in \mathcal{S}_{\mathtt{p}}$. If it is not, then ignore I; otherwise, do the following: $\tau_{\mathtt{last}} \leftarrow R$, $\mathsf{max_{tsl}} \leftarrow \mathsf{max_{tsl}} + 1$, $\mathcal{H}[\mathsf{max_{tsl}}] \leftarrow (\eta, 0)$, $\mathtt{p} \leftarrow \mathtt{p} + 1$ and send $(\mathtt{OK}, sid, \eta)$ to $\mathcal{A}$.
- If $I = (\mathtt{SET\_RANDOM}, sid)$ is received from $\mathcal{A}$, then check if $\mathcal{H}[\mathsf{max_{tsl}} + 1]$ has not been set yet. If it is not, then ignore I; otherwise, do the following:
  - $\tau_{\mathtt{last}} \leftarrow R$, $\mathsf{max_{tsl}} \leftarrow \mathsf{max_{tsl}} + 1$, $\eta \xleftarrow{\$} \{0,1\}^{\lambda}$, $\mathcal{H}[\mathsf{max_{tsl}}] \leftarrow (\eta, 1)$, $\mathtt{p} \leftarrow 0$.
  - For $i = 1, \ldots, \ell - \mu$
    $\mathcal{S}_i \leftarrow \mathsf{define\_new\_set}(\mathtt{MaxSize})$
  - send $(\mathtt{OK}, sid, \eta)$ to $\mathcal{A}$.

Set delays

If $I = (\mathtt{SET\text{-}DELAYS}, sid, \mathcal{T}')$ is received from $\mathcal{A}$, then set $b \leftarrow \mathsf{check\_time\_table}(\mathcal{T}, \mathcal{T}', \mathcal{P}, R, \mathsf{max_{tsl}})$. If $b = 1$, then set $\mathcal{T} = \mathcal{T}'$; ignore I otherwise.

Figure 2: The $\mathcal{B}^{\mathsf{w}}$ functionality.

remaining $\ell - \mu$ blocks might be chosen by the adversary and the content of these blocks are under full control of the adversary.

2. Since honest blocks include at least λ bits of entropy, the output of a hash function modeled as a RO with security parameter λ on input an honest block represents a uniform random value in $\{0,1\}^{\lambda}$.

At a high level, our beacon protocol works as follows. A party that wants to compute the beacon's output reads `state` from $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$ and outputs the hash of the latest $\ell - \mu + 1$ blocks of `state`. At first glance, as any chunk of $\ell - \mu + 1$ blocks of state contains (at least) an honestly generated block, the output of the beacon is an unpredictable random value. However, this is not the case. The first observation is that, using the technique described above, an adversary can predict the next $\ell - \mu$ outputs of the beacon in advance. In particular, the adversary first allows a sequence of μ honestly generated blocks to be added to the chain and then it inserts its own $\ell - \mu$ pre-computed adversarial blocks after those μ blocks. But, the *prediction power* of the adversary is not limited to $\ell - \mu$ blocks. We recall that the view that an honest party has of the ledger state could differ of at most `WindowSize` blocks. Therefore, in the worst case, the adversary sees `WindowSize` blocks in advance with respect to an honest party, thus giving an additional prediction power to him. In conclusion we can claim that, given a ledger $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$ with chain quality parameters (μ, ℓ) and window size `WindowSize`, it is possible to construct a weak beacon $\mathcal{B}^{\mathsf{w}}$ in which an adversary can predict, with respect to an honest party, the next $\Delta = \ell - \mu + \texttt{WindowSize}$ outputs. This means that an adversary, in the worst case, can predict the outcome of the beacon $\delta = \texttt{MaxRound} \cdot \Delta$ rounds in advance with respect to an honest party, where `MaxRound` represents the liveness parameter of $\mathcal{G}_{\texttt{ledger}}$. The only thing left to argue is how the output of the beacon are distributed. In the above scenario, not only the adversary can predict the next $\ell - \mu$ outputs, but can also bias those outputs since he can decide to extend `state` with any sequence of $\ell - \mu$ blocks. Since we model the hash function as a random oracle, it is easy to see that the bias of the output of the beacon depends on the randomness inside the honest blocks and on the hashing power of the adversary. Indeed, a powerful adversary can always decide what the next $\ell - \mu$ of `state` will be in the worst case. In this work we denote with `MaxSize` the maximum number of queries that the adversary can ask the RO. We observe that our instantiation of the weak beacon only needs to read information from the ledger. In Fig. 3 we provide a formal construction of the weak beacon protocol Π^{w} that UC-realizes $\mathcal{B}^{\mathsf{w}}$ with $\mathsf{w} = ((\mu, \ell), \texttt{MaxRound}, \texttt{WindowSize}, \texttt{MaxSize})$. The steps described in Fig. 3 follows the description given here with the exception that all the parties invoke the procedure update_time_table() every time that an input is received (see Fig. 4). This procedure helps a party to keep track of the size of the ledger (i.e. the size of `state`) at any round.

We assume that a set of parties $\mathcal{P}$ are registered to $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$, $\mathcal{G}_{\texttt{clock}}$ and $\mathcal{F}_{RO}$. Every time that a party $p_i \in \mathcal{P}$ receives an input it invokes the procedure update_time_table(). Let $\texttt{cq} \leftarrow \ell - \mu + 1$. The party p_i on input $(\texttt{FETCH}, \tau, sid)$ proceeds as follows.

1. Send $(\texttt{READ}, sid)$ to $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$ and wait for an answer.
2. Upon receiving $(\texttt{READ}, sid, \texttt{state})$ from $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$ set $\texttt{tsl} \leftarrow \mathcal{T}_{p_i}^{\texttt{local}}(\tau)$ and send $(\texttt{EVAL}, sid, \texttt{tsl}||\texttt{state}_{|\texttt{tsl}-\texttt{cq},\texttt{tsl}})$ to $\mathcal{F}_{RO}$.
3. Upon receiving the answer $(\texttt{EVAL}, sid, \eta)$ from $\mathcal{F}_{RO}$ output $(\texttt{FETCH}, sid, \texttt{tsl}, \eta)$.

Figure 3: The weak beacon protocol in $(\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}}), \mathcal{G}_{\texttt{clock}}, \mathcal{F}_{RO})$-hybrid model.

Theorem 1 *Let $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$ be the wrapper functionality for $\mathcal{G}_{\texttt{ledger}}$ defined in Def. 1 parametrized by $((\mu, \ell), \texttt{MaxRound}, \texttt{WindowSize})$, then protocol Π^{w} described in Fig. 3 securely realizes $\mathcal{B}^{\mathsf{w}}$ in the $(\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}}), \mathcal{G}_{\texttt{clock}}, \mathcal{F}_{RO})$-hybrid model with $\mathsf{w} = ((\mu, \ell), \texttt{MaxRound}, \texttt{WindowSize}, \texttt{MaxSize})$ where $\texttt{MaxSize} = \mathsf{poly}(\lambda)$.*

update_time_table()
When the procedure is invoked by $p_i \in \mathcal{P}$ the query $(\texttt{CLOCK-READ}, sid_C)$ is sent to $\mathcal{G}_{\texttt{clock}}$. Upon receiving the answer $(\texttt{CLOCK-READ}, sid_C, R)$ from $\mathcal{G}_{\texttt{clock}}$, send $(\texttt{READ}, sid)$ to $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$. Upon receiving receiving $(\texttt{READ}, sid, \texttt{state})$ from $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$ set $\mathcal{T}_{p_i}^{\texttt{local}}(R) := |\texttt{state}|$.

Figure 4: The procedure update_time_table().

Intuitively, in the proof of this theorem the simulator populates $\mathcal{T}$ and $\mathcal{H}$ by reading from $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$ the view that each party has of the ledger state. We note that it would be easy to also implement alternative ledger that manages the view that the parties have of the ledger state using $\mathcal{T}$ by means of a protocol that uses the $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$ (this new ledger manages a time-table $\mathcal{T}$ that is updated by asking the size of the state on the underlying ledger). We refer the reader to Appendix E for the formal proof of the theorem.

5.2 Discussion on alternative constructions

By looking at some real blockchains such as the Bitcoin blockchain, one would be tempted to implement $\mathcal{B}^{\mathsf{w}}$ by hashing only the last (stable) block of the hash chain. This might actually be a more efficient way to implement our weak-beacon functionality than the one in this work. However, it would not be a *generic* approach and is suitable for a certain type of blockchains, e.g., Bitcoin. In particular, the alternative approach would require some properties of blockchains not captures by $\mathcal{G}_{\texttt{ledger}}$. Indeed, for the case of Bitcoin we have that the blocks are organised in a hash-chain—where the hash function behaves as a random oracle—and those aspects are not exported to the UC Ledger. Our construction works for arbitrary blockchains, as it uses the UC ideal ledger functionality by Badertscher et al. in a black-box (ideal) manner wrapped with $\mathcal{W}_{\text{WBU}}$. We use $\mathcal{W}_{\text{WBU}}$ to only capture the fact that honest blocks (which appear frequently according to chain quality) have sufficient entropy, which we prove is the case using only the basic properties of the Backbone protocol; for adversarial blocks the Ledger offers no unpredictability guarantees. Hence, the alternative construction cannot work unless we make extra assumptions on the structure of the blockchain and hence on the entropy of the maliciously generated blocks. We remark that even though one might be able to prove that Bitcoin's output does have such extra properties—sufficient for the above alternative construction—the resulting statement would not be stronger: one would still need to rely on the unpredictability of the next honest block and on chain quality; hence it would at most give a slightly more efficient solution (hashing one block instead of $\ell - \mu$) at the cost of a more involved analysis with extra assumptions.

6 Timed Signatures (TSign)

In this section, we extend the standard notion of the digital signature by different levels of timing guarantees. For self-containment, we have included the standard signatures functionality proposed by Canetti [Can03] in Fig. 17 of Appendix B.

In our model, a timestamped signature σ for a message m is equipped with a time mark τ that contains information about when σ was computed by the signer. We refer to this special notion

of signature for a time mark τ that is associated with the global clock $\mathcal{G}_{\mathtt{clock}}$ as *Timed Signature (TSign)*. We define three categories of security for TSign: *backdate*, *postdate* security, and their combination which we refer to just as *timed security*. Intuitively, backdate security guarantees that the signature σ time-marked with τ has been computed some time *before* τ; postdate security guarantees that the signature σ was computed some time *after* τ; and timed security provides to the party that verifies the signature σ a time interval around τ in which σ was computed.

We formally define these three new security notions by means of a single UC-functionality $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ (see Fig. 5.a and 5.b for a detailed description.) $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ is parameterized by a flag $\mathtt{t} \in \{+,-,\pm\}$ where $\mathtt{t} = \text{“}-\text{”}$ indicates that the functionality guarantees backdate security, $\mathtt{t} = \text{“}+\text{”}$ indicates postdate security, and $\mathtt{t} = \text{“}\pm\text{”}$ indicates timed security. Analogously to the weak beacon, $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ and all parties that have access to this functionality, are registered to $\mathcal{G}_{\mathtt{clock}}$ which provides the notion of time inherently required by our model. For generality, we parametrize $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ with $\mathsf{w} = (\Delta, \mathtt{MaxRound}, \mathtt{WindowSize}, \mathtt{waitingTime})$, where the meaning of these parameters is discussed below.

In a nutshell, the functionality $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ provides to its registered parties a new time-slot $\mathtt{tsl} \in \mathbb{N}$ every $\mathtt{MaxRound}$ rounds (in the worst case). The exact moment in which each such time slot is issued is decided by the adversary $\mathcal{A}$ via the input $(\mathtt{NEW\_SLOT}, \mathit{sid})$. Once a time slot $\mathtt{tsl}$ is issued, it can be used to time(stamp) a signature σ. The meaning of $\mathtt{tsl}$ depends on the notion of security that we are considering. For backdate security (i.e., $\mathtt{t} = \text{“}-\text{”}$), a signature σ marked with $\mathtt{tsl}$ denotes that σ was computed during a time slot $\mathtt{tsl}' \leq \mathtt{tsl}$. For postdate security ($\mathtt{t} = \text{“}+\text{”}$) $\mathtt{tsl}$ denotes that σ was computed during a time slot $\mathtt{tsl}' \geq \mathtt{tsl}$. For timed security, the signature σ is equipped with two time-marks $\mathtt{tsl}_{\mathtt{back}}$ and $\mathtt{tsl}_{\mathtt{post}}$ that denote that σ was computed in a time-slot $\mathtt{tsl}'$ such that $\mathtt{tsl}_{\mathtt{post}} \leq \mathtt{tsl}' \leq \mathtt{tsl}_{\mathtt{back}}$. A new time-slot issued by $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ can be immediately seen and used by $\mathcal{A}$. However, $\mathcal{A}$ can delay honest parties from seeing new time-slots—i.e., truncate the view that each honest party has of the available time-slots. That is, for each party p_i, $\mathcal{A}$ can decide to *hide* the most recent $\mathtt{WindowSize}$-many available time-slots. This means that, for example, in any round R the party p_1 could see (and use) the most recent time-slot $\mathtt{tsl}$, whereas p_2's view might have $\mathtt{tsl} - \mathtt{WindowSize}$ as the most recent time-slot.

To keep track of the association between rounds and time-slots, $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ manages a time table $\mathcal{T}$ in the same way as $\mathcal{B}^{\mathsf{w}}$. That is, an entry $\mathcal{T}[\tau, p_i]$ is an integer $\mathtt{tsl}_{p_i}$, where p_i represents a party registered to $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ and τ represents round number. The value $\mathtt{tsl}_{p_i}$ defines the view that the party p_i has of the available time-slots in round τ. In particular, at round τ party p_i can access and use the time slots $1, \ldots, \mathtt{tsl}_{p_i}$. The time table $\mathcal{T}$ is controlled by $\mathcal{A}$ but it is limited to change the content of $\mathcal{T}$ according to the parameter $\mathtt{WindowSize}$ as we discussed above. More formally, $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ checks that the changes made by $\mathcal{A}$ to $\mathcal{T}$ are valid using the procedures check_time_table and force_time_table (the same ones that were used by our weak beacon, see Fig. 1.)

Note that the way to obtain postdate security is by relying on the unpredictability of the beacon. However, this creates the following subtlety. As the adversary is able to predict future values of our (weak) beacon he can attempt to postdate signatures as far in the future as his prediction reaches. To capture this behaviour, our functionality is parameterized by a value $\Delta \in \mathbb{N}$, which we call the *prediction parameter*. This parameter is only relevant when $\mathtt{t} \in \{\text{“}+\text{”}, \text{“}\pm\text{”}\}$. With this parameter we allow the adversary to use, before of any honest party, Δ new time-slots. This means that, for the case of postdate and timed security, an adversary can compute a signature σ marked with a time slot $\mathsf{max}_{\mathsf{tsl}} + \Delta$, where $\mathsf{max}_{\mathsf{tsl}}$ denotes the most recent time-slot. However this creates a new issue, this time with the security proof: when the simulator receives from its adversary a

signature timed with a presumably predicted beacon value, it cannot be sure whether the adversary will indeed instruct the beacon to output this value when its time comes. To resolve that, the functionality allows its simulator/adversary to withdraw signatures which refer to a *future* time slot $\mathtt{tsl} > \mathsf{max}_{\mathtt{tsl}}$ via the command $(\mathtt{DELETE}, sid, \cdot)$. We also introduce a parameter `waitingTime`, which is relevant when $\mathtt{t} \in \{-, \pm\}$ and allows the following adversarial interference: Whenever an honest party wants to time-mark a signature, $\mathcal{A}$ can decide to delay the marking operation until that `waitingTime` time-slots have been issued by $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$. This means that an honest party that requests to time-mark σ in round R has to wait, in the worst case, $\mathtt{waitingTime} \cdot \mathtt{MaxRound}$ rounds in order to see σ time-marked. To guarantee that a new time-slot is available every `MaxRound` (at least) rounds, any time that an input is received the functionality checks that a new time-slot has been issued using the procedure check_liveness following exactly the same approach of $\mathcal{B}^{\mathsf{w}}$ (see. Sec 5 for more details on how the liveness is enforced).

The formalization of our functions (proposed in Fig. 5.a and 5.b) extends the Canetti's standard signature functionality $\mathcal{F}_{\mathtt{SIGN}}$ that we recall in Fig. 17). Roughly, $\mathcal{F}_{\mathtt{SIGN}}$ stores all the signatures that are issued, and when a verification request for a message m occurs then $\mathcal{F}_{\mathtt{SIGN}}$ checks whether or not she is storing a signature for m. In the description of $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ we make explicit the data structure, that we call *signature-table*, that stores the signature (with the corresponding time-stamping) by denoting it with Tab_σ. In the formal description of this and the other *timed* functionalities considered in this work, we always use `t` as a parameter of the inputs. The reason to use `t` also as a part of the input given to the timed functionalities is that there might be protocols that use multiple timed functionality (e.g. the $\mathcal{F}_\sigma^{\mathsf{w},-}$ and $\mathcal{F}_\sigma^{\mathsf{w},\pm}$), and therefore it could be useful to discriminate the inputs of $\mathcal{F}_\sigma^{\mathsf{w},-}$ from the input of $\mathcal{F}_\sigma^{\mathsf{w},+}$ by means of the parameter `t`. Finally, to keep the description of the functionality as compact as possible, $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ is also equipped with procedure standard_verification (see Fig. 6) that abstracts the checks for completeness, unforgeability and consistency performed by Canetti's standard signature verification phase in $\mathcal{F}_{\mathtt{SIGN}}$, Fig. 17).

6.1 Our constructions

We provide a scheme $\Pi_\sigma^{\mathsf{w},\mathtt{t}}$ that UC-realizes the functionality $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ for $\mathtt{t} = \text{“}-\text{”}, \text{“}+\text{”}, \text{“}\pm\text{”}$. The backdate secure TSign scheme $\Pi_\sigma^{\mathsf{w},-}$ showed in Fig. 12 realizes $\mathcal{F}_\sigma^{\mathsf{w},-}$ in the $(\mathcal{F}_{\mathtt{SIGN}}, \mathcal{G}_{\mathtt{ledger}}, \mathcal{G}_{\mathtt{clock}})$-hybrid model. In this, informally, the signer signs a message m using a standard signature scheme, creates a transaction that contains the signature of the message and then asks the ledger to store the transaction permanently into `state`. Intuitively, the construction is secure as of the security of the signature scheme and because the history of the ledger cannot be changed. Also, $\Pi_\sigma^{\mathsf{w},-}$ realizes $\mathcal{F}_\sigma^{\mathsf{w},-}$ with $\mathsf{w} = (\bot, \mathtt{MaxRound}, \mathtt{WindowSize}, \mathtt{waitingTime})$ where `MaxRound`, `WindowSize` and `waitingTime` are the parameters of $\mathcal{G}_{\mathtt{ledger}}$. In this construction, as for the weak beacon protocol, every honest party p_i maintains a table $\mathcal{T}_{p_i}^{\mathtt{local}}$ that is updated on any input received by p_i according to the procedure update_time_table(). The aim of $\mathcal{T}_{p_i}^{\mathtt{local}}$ is to associate each round of $\mathcal{G}_{\mathtt{clock}}$ to the size of the state of $\mathcal{G}_{\mathtt{ledger}}$. The postdate secure TSign scheme $\Pi_\sigma^{\mathsf{w},+}$, showed in the Fig. 8, realizes $\mathcal{F}_\sigma^{\mathsf{w},+}$ in the $(\mathcal{F}_{\mathtt{SIGN}}, \mathcal{B}^{\mathsf{w}}, \mathcal{G}_{\mathtt{clock}})$-hybrid model. In this protocol the signer, on input a message m, first invokes the weak beacon $\mathcal{B}^{\mathsf{w}}$ thus obtaining the most recent output (in his view) η and then signs $m||\eta$ using a standard signature scheme. Intuitively, this scheme is secure due of the unforgeability of the signature scheme and because an adversary can, in the worst case, predict only the future Δ outputs of the beacon. More precisely, $\Pi_\sigma^{\mathsf{w},+}$ realizes $\mathcal{F}_\sigma^{\mathsf{w},+}$ with $\mathsf{w} = (\Delta = (\ell, \mu), \mathtt{MaxRound}, \mathtt{WindowSize}, \bot)$ where `MaxRound`, `WindowSize` are the parameters of $\mathcal{B}^{\mathsf{w}}$.

Initialization. The main parameter of $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ is $\mathtt{t} \in \{-,+,\pm\}$. The functionality is also parametrized by the algorithms $\mathsf{check\_time\_table}(\mathcal{T},\mathcal{T}',\mathcal{P},R,\mathsf{max_{tsl}})$, $\mathsf{force\_time\_table}(\mathcal{T},\mathcal{P},R,\mathsf{max_{tsl}})$, $\mathsf{check\_liveness}(\tau_{\mathtt{last}},R)$ along with the parameters `WindowSize`, `MaxRound`, Δ, $\mathsf{max_{tsl}} \leftarrow 0$, `waitingTime`, $\tau_{\mathtt{last}} \leftarrow 0$ the time-table $\mathcal{T}$, the signature-table Tab_σ and running with parties $\mathcal{P}$ and adversary $\mathcal{A}$. $\mathcal{T}[0,p_i] \leftarrow 0$ for all $p_i \in \mathcal{P}$. We assume that the parties are registered to $\mathcal{G}_{\mathtt{clock}}$. Let R be the the response obtained by querying $\mathcal{G}_{\mathtt{clock}}$, upon receiving any input I from any party or from the adversary act as follows:

- If $\mathsf{check\_liveness}(\tau_{\mathtt{last}},R) = 0$ then $\mathsf{max_{tsl}} \leftarrow \mathsf{max_{tsl}} + 1$, $\tau_{\mathtt{last}} \leftarrow R$, $\mathcal{T} \leftarrow \mathsf{force\_time\_table}(\mathcal{T},\mathcal{P},R,\mathsf{max_{tsl}})$.
- If $\mathsf{check\_time\_table}(\mathcal{T},\mathcal{T},\mathcal{P},R,\mathsf{max_{tsl}}) = 0$ then $\mathcal{T} \leftarrow \mathsf{force\_time\_table}(\mathcal{T},\mathcal{P},R,\mathsf{max_{tsl}})$.

Key Generation.
Upon receiving a value $(\mathtt{KEY\_GEN}, sid)$ from some party $S \in \mathcal{P}$, verify that $sid = (S, sid')$ for some sid. If not, then ignore the request. Else, hand $(\mathtt{KEY\_GEN}, sid)$ to the $\mathcal{A}$. Upon receiving $(\mathtt{VERIFICATION\_KEY}, sid, v)$ from the $\mathcal{A}$, output $(\mathtt{VERIFICATION\_KEY}, sid, v)$ to S, and record the pair (S, v).

Signature.

- If $I = (\mathtt{TIMED\_SIGN}, sid, \mathtt{t}, m, \tau_{\mathtt{req}})$ is received from party S, verify that $sid = (S, sid')$ for some sid'. If not, then ignore the request. If $\tau_{\mathtt{req}} \leq R$ then
  - if $\mathtt{t} =$ " $+$ " then record $(\mathtt{t}, \perp, m, S, \tau_{\mathtt{req}})$ and send $(\mathtt{TIMED\_SIGN}, sid, \mathtt{t}, m, \tau_{\mathtt{req}})$ to $\mathcal{A}$;
  - if $\mathtt{t} \in \{-,\pm\}$ then record $(\mathtt{t}, \mathsf{max_{tsl}}, m, S, \tau_{\mathtt{req}})$ and send $(\mathtt{TIMED\_SIGN}, sid, \mathtt{t}, \mathsf{max_{tsl}}, m, \tau_{\mathtt{req}})$ to $\mathcal{A}$;
  - else output an error message to S and halt.
- If $I = (\mathtt{TIMED\_SIGNATURE}, sid, \mathtt{t}, m, (\mathtt{tsl_{back}}, \sigma, \mathtt{tsl_{post}}))$ is received from $\mathcal{A}$, verify that no entry $((m,\sigma,0), \mathtt{tsl_{post}})$ has been stored in Tab_σ. If it is not then send an error message to S and halt. If an entry $(\mathtt{t}, \mathsf{oldmax}, m, S, \tau_{\mathtt{req}})$ has been recorded, then do the following.
  - if $\mathtt{t} \in \{-,\pm\}$ then
    - if (S is honest and $\mathtt{tsl_{back}} \leq \mathsf{oldmax} + \mathtt{waitingTime}$) or ($S$ is corrupted) and $\mathtt{tsl_{back}} > \mathsf{max_{tsl}}$) then $\mathtt{tsl_{back}}' \leftarrow \mathtt{tsl_{back}}$
    - else output an error message to p_i and halt.
  - if $\mathtt{t} \in \{+,\pm\}$ then
    - if ($\mathcal{T}[\tau_{\mathtt{req}}, S] = \mathtt{tsl_{post}}$ and S is not corrupted) or ($\mathtt{tsl_{post}} \leq \mathcal{T}[\tau_{\mathtt{req}}, S] + \Delta$ and S is corrupted) then $\mathtt{tsl_{post}}' \leftarrow \mathtt{tsl_{post}}$
    - else output an error message to S and halt.

  Add $(\mathtt{tsl_{back}}', (m,\sigma,v,1), \mathtt{tsl_{post}}')$ to Tab_σ and send $(\mathtt{TIMED\_SIGNATURE}, sid, \mathtt{t}, (\mathtt{tsl_{back}}', m, \sigma, \mathtt{tsl_{post}}'))$ to S.

Figure 5.a: The $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ functionality.

The timed secure TSign scheme $\Pi_\sigma^{\mathsf{w},\pm}$ showed in Fig. 9 realizes $\mathcal{F}_\sigma^{\mathsf{w},\pm}$ in the $(\mathcal{F}_\sigma^{\mathsf{w},+}, \mathcal{B}^{\mathsf{w}}, \mathcal{G}_{\mathtt{clock}})$-hybrid model. In the construction we promote the postdate security of $\mathcal{F}_\sigma^{\mathsf{w},+}$ to timed security by simply storing the signature obtained via $\mathcal{F}_\sigma^{\mathsf{w},+}$ into the ledger. The security of this scheme follows immediately from the postdate security of $\mathcal{F}_\sigma^{\mathsf{w},+}$ and from the immutability of $\mathcal{G}_{\mathtt{ledger}}$.

Verification

- If $I = (\texttt{TIMED\_VERIFY}, sid, \mathtt{t}, (\mathtt{tsl}_{\mathtt{back}}, m, \sigma, v', \mathtt{tsl}_{\mathtt{post}}))$ is received then hand $(\texttt{TIMED\_VERIFIED}, sid, \mathtt{t}, (\mathtt{tsl}_{\mathtt{back}}, m, \sigma, v', \mathtt{tsl}_{\mathtt{post}}))$ to $\mathcal{A}$. Upon receiving $(\texttt{TIMED\_VERIFIED}, sid, m, \phi)$ from $\mathcal{A}$ invoke $f = \mathsf{standard\_verification}(m, \sigma, v', \phi, \mathtt{tsl}_{\mathtt{post}})$ and check if $f = 1$. If it is not, then send $(\texttt{TIMED\_VERIFIED}, sid, \perp, m, 0, \perp)$ to p_i, otherwise set $\tau_{post} \leftarrow \perp, \tau_{back} \leftarrow \perp$ and do the following:
  - If $\mathtt{t} \in \{+, \pm\}$ and $\mathcal{T}[R, p_i] \geq \mathtt{tsl}_{\mathtt{post}}$ then find the smallest τ_{post} such that $\mathcal{T}[\tau_{post}, p_i] = \mathtt{tsl}_{\mathtt{post}}$ else send $(\texttt{TIMED\_VERIFIED}, sid, \mathtt{t}, m, (\perp, \pi, 0, \perp))$ to p_i.
  - If $\mathtt{t} \in \{-, \pm\}$ and $\mathcal{T}[R, p_i] \geq \mathtt{tsl}_{\mathtt{back}}$ then find the smallest τ_{back} such that $\mathcal{T}[\tau_{back}, p_i] = \mathtt{tsl}_{\mathtt{back}}$ else send $(\texttt{TIMED\_VERIFIED}, sid, \mathtt{t}, m, (\perp, \pi, 0, \perp))$ to p_i.

  Send $(\texttt{TIMED\_VERIFIED}, sid, \tau_{back}, m, f, \tau_{post})$ to p_i.

Set delays and new time-slots

If $I = (\texttt{NEW\_SLOT}, sid)$ is received from $\mathcal{A}$ then $\tau_{\mathtt{last}} \leftarrow R$, $\mathsf{max}_{\mathsf{tsl}} \leftarrow \mathsf{max}_{\mathsf{tsl}} + 1$.
If $I = (\texttt{SET-DELAYS}, sid, \mathcal{T}')$ is received from $\mathcal{A}$, then $b \leftarrow \mathsf{check\_time\_table}(\mathcal{T}, \mathcal{T}', \mathcal{P}, R, \mathsf{max}_{\mathsf{tsl}})$. If $b = 1$ then $\mathcal{T} \leftarrow \mathcal{T}'$ else ignore I.

Withdraw future signature If $I = (\texttt{DELETE}, sid, \mathtt{t}, (\mathtt{tsl}_{\mathtt{back}}, m, \sigma, v', \mathtt{tsl}_{\mathtt{post}}))$ then check if $\mathtt{tsl}_{\mathtt{post}} > \mathsf{max}_{\mathsf{tsl}}$. If it is not then ignore I otherwise delete the entry (if it exists) $(\mathtt{tsl}_{\mathtt{back}}, (m, \sigma, v, 1), \mathtt{tsl}_{\mathtt{post}})$ from the storage.

Figure 5.b: The $\mathcal{F}_\sigma^{\mathsf{w},\mathtt{t}}$ functionality (cont'd).

$\mathsf{standard\_verification}(m, \sigma, v', \phi, \mathtt{tsl}_{\mathtt{post}})$

- If $v' = v$ and the entry $((m, \sigma, v, 1), \mathtt{tsl}_{\mathtt{post}})$ is stored in Tab_σ then set $f = 1$.
- Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1, \mathtt{tsl}_{\mathtt{post}})$ for any σ' is stored in Tab_σ, then set $f = 0$ and record the entry $((m, \sigma, v, 0), \mathtt{tsl}_{\mathtt{post}})$.
- Else, if there is an entry $(m, \sigma, v', f', \mathtt{tsl}_{\mathtt{post}})$ stored in Tab_σ then let $f = f'$.
- Else let $f = \phi$ and store the entry $(m, \sigma, v', \phi, \mathtt{tsl}_{\mathtt{post}})$ in Tab_σ.

Return f

Figure 6: The Procedure $\mathsf{standard\_verification}$

Theorem 2 *The protocol $\Pi_\sigma^{\mathsf{w},-}$ described in Fig. 7 realizes with perfect security $\mathcal{F}_\sigma^{\mathsf{w},-}$ in the $(\mathcal{G}_{\mathtt{ledger}}, \mathcal{G}_{\mathtt{clock}}, \mathcal{F}_{\mathtt{SIGN}})$-hybrid model where $\mathcal{G}_{\mathtt{ledger}}$ is parametrized by $((\mu, \ell), \mathtt{MaxRound}, \mathtt{WindowSize}, \mathtt{waitingTime})$.*

Theorem 3 *Let $\mathcal{F}_\sigma^{\mathsf{w},+}$ be the functionality parametrized by $\mathsf{w} = (\Delta, \mathtt{MaxRound}, \mathtt{WindowSize}, \perp)$, then the protocol $\Pi_\sigma^{\mathsf{w},+}$ described in Fig. 8 realizes with perfect security $\mathcal{F}_\sigma^{\mathsf{w},+}$ in the $(\mathcal{B}^{\mathsf{w}'}, \mathcal{G}_{\mathtt{clock}}, \mathcal{F}_{\mathtt{SIGN}})$-hybrid model where $\mathcal{B}^{\mathsf{w}'}$ is parametrized by $\mathsf{w}' = ((\mu, \ell), \mathtt{MaxRound}, \mathtt{WindowSize}, \mathtt{MaxSize})$ with $\Delta = \ell - \mu$.*

We refer the reader to Appendix E for the formal proofs of the above theorems. The proof for the following theorem follows the arguments given in the proof of Theorem 2. Indeed we observe that the only difference here is that into the ledger we store a signature issued by $\mathcal{F}_\sigma^{\mathsf{w},+}$ instead of a signature computed by $\mathcal{F}_{\mathtt{SIGN}}$.

We assume that the parties $\mathcal{P}$ are registered to $\mathcal{G}_{\texttt{ledger}}$, $\mathcal{G}_{\texttt{clock}}$ and $\mathcal{F}_{\texttt{SIGN}}$. Each party $p_i \in \mathcal{P}$ manages a local time table $\mathcal{T}_{p_i}^{\texttt{local}}$ that is updated any time that p_i receives an input by invoking the procedure update_time_table().

Initialisation. The signer $S \in \mathcal{P}$ sends $(\texttt{KEY\_GEN}, sid)$ to $\mathcal{F}_{\texttt{SIGN}}$ thus obtaining $(\texttt{VERIFICATION\_KEY}, sid, v)$.

Signature. The signer $S \in \mathcal{P}$ on input $(\texttt{TIMED\_SIGN}, sid, -, m, \tau_{\texttt{req}})$ executes the following steps.

1. Send $(\texttt{SIGN}, sid, m)$ to $\mathcal{F}_{\texttt{SIGN}}$ and upon receiving the answer $(\texttt{SIGNATURE}, sid, m, \sigma)$, create a transaction $\texttt{tx} := (m, \sigma)$ and send $(\texttt{SUBMIT}, sid, \texttt{tx})$ to $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$.
2. Wait until $\texttt{tx}$ is added to the state of $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$. Let $\texttt{tsl}_{\texttt{back}}$ be the block of $\texttt{state}$ that contains $\texttt{tx}$, output $(\texttt{TIMED\_SIGNATURE}, sid, -, (\texttt{tsl}_{\texttt{back}}, m, \sigma, \bot))$.

Verification. The party $p_i \in \mathcal{P}$ on input $(\texttt{TIMED\_VERIFY}, sid, -, (\texttt{tsl}_{\texttt{back}}, m, \sigma, v', \bot))$ proceeds as follows.

1. Send $(\texttt{VERIFY}, sid, m, \sigma, v')$ to $\mathcal{F}_{\texttt{SIGN}}$.
2. Upon receiving $(\texttt{VERIFIED}, sid, m, b)$ from $\mathcal{F}_{\texttt{SIGN}}$, if $b = 1$ then send $(\texttt{READ}, sid)$ to $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$ else output $(\texttt{TIMED\_VERIFIED}, sid, -, (\bot, m, 0, \bot))$.
3. Upon receiving the answer $(\texttt{READ}, sid, \texttt{state})$ from $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$ check if the transaction $\texttt{tx} = (m, \sigma)$ is stored in the $\texttt{tsl}_{\texttt{back}}$-th block of $\texttt{state}$. If it is, then find the smallest τ_{back} such that $\mathcal{T}_{p_i}^{\texttt{local}}[\tau_{back}] = \texttt{tsl}_{\texttt{back}}$ and output $(\texttt{TIMED\_VERIFIED}, sid, -, (\tau_{back}, m, 1, \bot))$ otherwise output $(\texttt{TIMED\_VERIFIED}, sid, -, (\bot, m, 0, \bot))$.

Figure 7: The protocol $\Pi_{\sigma}^{\mathsf{w},-}$ in the $(\mathcal{F}_{\texttt{SIGN}}, \mathcal{G}_{\texttt{ledger}}, \mathcal{G}_{\texttt{clock}})$-hybrid model.

We assume that the parties $\mathcal{P}$ are registered to $\mathcal{F}_{\texttt{SIGN}}$, $\mathcal{B}^{\mathsf{w}}$ and $\mathcal{G}_{\texttt{clock}}$.

Initialisation. The signer $S \in \mathcal{P}$ sends $(\texttt{KEY\_GEN}, sid)$ to $\mathcal{F}_{\texttt{SIGN}}$ thus obtaining $(\texttt{VERIFICATION\_KEY}, sid, v)$.

Signature. The signer $S \in \mathcal{P}$ on input $(\texttt{TIMED\_SIGN}, sid, +, m, \tau_{\texttt{req}})$ executes the following steps.

- Send $(\texttt{FETCH}, \tau, sid)$ to $\mathcal{B}^{\mathsf{w}}$ and upon receiving $(\texttt{FETCH}, sid, \texttt{tsl}_{\texttt{post}}, \eta)$ send $(\texttt{SIGN}, sid, \texttt{tsl}_{\texttt{post}}||m||\eta)$ to $\mathcal{F}_{\texttt{SIGN}}$.
- Upon receiving $(\texttt{SIGNATURE}, sid, \texttt{tsl}_{\texttt{post}}||m||\eta, \sigma)$, $\sigma_t \leftarrow (\sigma, \eta)$ and output $(\texttt{TIMED\_SIGNATURE}, sid, +, (\bot, m, \sigma_t, \texttt{tsl}_{\texttt{post}}))$.

Verification. The party $p_i \in \mathcal{P}$ on input $(\texttt{TIMED\_VERIFY}, sid, +, (\bot, m, \sigma_t, v', \texttt{tsl}_{\texttt{post}}))$ parses σ_t as (σ, η) and executes the following steps.

- Parses σ_t as (σ, η) and query $\mathcal{B}^{\mathsf{w}}$ to check when (and if) the value η was issued by $\mathcal{B}^{\mathsf{w}}$. If η has never been issued by $\mathcal{B}^{\mathsf{w}}$ then output $(\texttt{TIMED\_VERIFIED}, sid, +, (\bot, m, 0, \bot))$. Else, let τ_{post} be the round in which η has been issued, send $(\texttt{VERIFY}, sid, \texttt{tsl}_{\texttt{post}}||m||\eta, \sigma, v')$ to $\mathcal{F}_{\texttt{SIGN}}$.
- Upon receiving $(\texttt{VERIFICATION}, sid, m, b)$ from $\mathcal{F}_{\texttt{SIGN}}$, if $b = 1$ then output $(\texttt{TIMED\_VERIFIED}, sid, +, (\bot, m, 1, \tau_{post}))$ else output $(\texttt{TIMED\_VERIFIED}, sid, +, (\bot, m, 0, \bot))$

Figure 8: The protocol $\Pi_{\sigma}^{\mathsf{w},+}$ in the $(\mathcal{F}_{\texttt{SIGN}}, \mathcal{B}^{\mathsf{w}}, \mathcal{G}_{\texttt{clock}})$-hybrid model.

Theorem 4 *Let $\mathcal{F}_{\sigma}^{\mathsf{w},\pm}$ be the functionality parametrized by $\mathsf{w} = (\Delta, \texttt{MaxRound}, \texttt{WindowSize}, \texttt{waitingTime})$,*

We assume that the parties $\mathcal{P}$ are registered to $\mathcal{G}_{\texttt{ledger}}$, $\mathcal{G}_{\texttt{clock}}$ and to the functionality $\mathcal{F}_{\sigma}^{\mathsf{w},+}$. Each party $p_i \in \mathcal{P}$ manages a local time table $\mathcal{T}_{p_i}^{\texttt{local}}$ that updates on any input he receives by invoking the procedure update_time_table().

Initialisation. The signer $S \in \mathcal{P}$ sends $(\texttt{KEY\_GEN}, sid)$ to $\mathcal{F}_{\sigma}^{\mathsf{w},+}$ thus obtaining $(\texttt{VERIFICATION\_KEY}, sid, v)$.

Signature. The signer $S \in \mathcal{P}$ on input $(\texttt{TIMED\_SIGN}, \pm, sid, m, \tau_{\texttt{req}})$ execute the following steps.

1. Forward $(\texttt{TIMED\_SIGN}, sid, +, m, \tau_{\texttt{req}})$ to $\mathcal{F}_{\sigma}^{\mathsf{w},+}$ and upon receiving the answer $(\texttt{TIMED\_SIGNATURE}, sid, +, (\perp, m, \sigma, \texttt{tsl}_{\texttt{post}}))$, create a transaction $\texttt{tx} \leftarrow (m, \sigma, \texttt{tsl}_{\texttt{post}})$ and send $(\texttt{SUBMIT}, sid, \texttt{tx})$ to $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$.
2. Wait until $\texttt{tx}$ is added to the state of $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$. Let $\texttt{tsl}_{\texttt{back}}$ be the block of $\texttt{state}$ that contains $\texttt{tx}$, output $(\texttt{TIMED\_SIGNATURE}, sid, \pm, (\texttt{tsl}_{\texttt{back}}, m, \sigma, \texttt{tsl}_{\texttt{post}}))$.

Verification. The party $p_i \in \mathcal{P}$ on input $(\texttt{TIMED\_VERIFY}, sid, \pm, (\texttt{tsl}_{\texttt{back}}, m, \sigma, v', \texttt{tsl}_{\texttt{post}}))$ proceeds as follows.

1. Send $(\texttt{TIMED\_VERIFY}, sid, +, (\perp, m, \sigma, v', \texttt{tsl}_{\texttt{post}}))$ to $\mathcal{F}_{\sigma}^{\mathsf{w},+}$.
2. Upon receiving $(\texttt{TIMED\_VERIFIED}, sid, +, \perp, m, b, \tau)$ from $\mathcal{F}_{\sigma}^{\mathsf{w},+}$, if $b = 1$ then send $(\texttt{READ}, sid)$ to $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$ else output $(\texttt{TIMED\_VERIFIED}, sid, \pm, \perp, m, 0, \perp)$.
3. Upon receiving the answer $(\texttt{READ}, sid, \texttt{state})$ from $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$ check if the transaction $\texttt{tx} = (m, \sigma, \texttt{tsl}_{\texttt{post}})$ is stored in the $\texttt{tsl}_{\texttt{back}}$-th block of $\texttt{state}$. If it is, then find the smallest τ_{back} such that $\mathcal{T}_{p_i}^{\texttt{local}}[\tau_{back}] = \texttt{tsl}_{\texttt{back}}$ and output $(\texttt{TIMED\_VERIFIED}, sid, \tau_{back}, m, 1, \tau_{post})$ otherwise output $(\texttt{TIMED\_VERIFIED}, sid, \perp, m, 0, \perp)$.

Figure 9: The protocol $\Pi_{\sigma}^{\mathsf{w},\pm}$ in the $(\mathcal{F}_{\sigma}^{\mathsf{w},+}, \mathcal{G}_{\texttt{ledger}}, \mathcal{G}_{\texttt{clock}})$-hybrid model.

then the protocol $\Pi_{\sigma}^{\mathsf{w},\pm}$ described in Fig. 9 realizes with perfect security $\mathcal{F}_{\sigma}^{\mathsf{w},\pm}$ in the $(\mathcal{F}_{\sigma}^{\mathsf{w},+}, \mathcal{G}_{\texttt{ledger}}, \mathcal{G}_{\texttt{clock}})$-hybrid model-hybrid model.

7 Timed Zero-Knowledge Proof of Knowledge (TPoK)

The NIZK Functionality. In this section we apply the same methodology used for timed signatures in the previous section to defined analogously timed versions of non-interactive zero-knowledge proofs of knowledge. The basis for our approach is the standard UC Non-Interactive Zero-Knowledge functionality proposed in [GOS12].[8] In order to use the functionality from [GOS12] as basis for our fetch-based delivery (cf. Section 2) timed NIZK functionality, we first turn it into a fetch-based delivery version: instead of waiting for the adversary to deliver the proof to the honest verifier, we allow the verifier to request for the proof, a request which is answered directly if the adversary has allowed the proof to be generated. This is a simple syntactic modification of the functionality from [GOS12]; for self-containment we have included the resulting functionality, denoted by $\mathcal{F}_{\mathsf{NIZK}}$, in Fig. 16 in Appendix B.

Here is how we extend $\mathcal{F}_{\mathsf{NIZK}}$ to consider different levels of timed-security, in the same way as we have done for signatures: A proof π generated with respect to an $\mathcal{NP}$-statement is equipped with a time-mark $\texttt{tsl}$ that gives some information about when π was computed. We refer to this

[8] Although [GOS12] focuses on constructing UC perfect NIZK argument of knowledge the corresponding functionality is the same for UC NIZK proof of knowledge.

notion of NIZK as TPoK and consider three categories of security: *backdate*, *postdate* and *timed security*. The formalization of these notions is given by means of the UC-functionalities $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$, with $\mathtt{t} = \text{“}-\text{”}, \text{“}+\text{”}, \text{“}\pm\text{”}$. $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$ is formally described in the Fig. 10.a and 10.b. In this, intuitively, a prover P that generates an accepting proof π equipped with a time-mark $\mathtt{tsl}$ gives to the verifier V the guarantee that: 1) he knew the witness for the $\mathcal{NP}$-statement that he is proving; 2) the proof π was generated (using the witness) in some moment specified by $\mathtt{tsl}$. We also provide three instantiations, one for each of the three security notions mentioned above. That is, we show a protocol $\Pi_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$ that UC-realize $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$ for all $\mathtt{t} \in \{-,+,\pm\}$. We show the details of $\Pi_{\mathsf{TPoK}}^{\mathsf{w},+}$ and $\Pi_{\mathsf{TPoK}}^{\mathsf{w},\pm}$ only since $\Pi_{\mathsf{TPoK}}^{\mathsf{w},-}$ is similar to $\Pi_{\sigma}^{\mathsf{w},-}$. Indeed the prover of $\Pi_{\mathsf{TPoK}}^{\mathsf{w},-}$, on input $(x,w) \in \mathsf{Rel}$, just needs to compute a NIZK proof (e.g. computed using $\mathcal{F}_{\mathsf{NIZK}}$) and store it into the ledger.

Initialization. The main parameter of $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$ is $\mathtt{t} \in \{-,+,\pm\}$. The functionality is also parametrized by the algorithms $\mathsf{check\_time\_table}(\mathcal{T},\mathcal{T}',\mathcal{P},R,\mathsf{max}_{\mathsf{tsl}})$, $\mathsf{force\_time\_table}(\mathcal{T},\mathcal{P},R,\mathsf{max}_{\mathsf{tsl}})$, $\mathsf{check\_liveness}(\tau_{\mathtt{last}},R)$ along with the parameters $\mathtt{WindowSize}$, $\mathtt{MaxRound}$, Δ, $\mathsf{max}_{\mathsf{tsl}} \leftarrow 0$, $\mathtt{waitingTime}$, $\tau_{\mathtt{last}} \leftarrow 0$ the time-table $\mathcal{T}$ and running with parties $\mathcal{P} = \{p_1,\dots\}$ and adversary $\mathcal{A}$. $\mathcal{T}[0,p_i] \leftarrow 0$ for all $p_i \in \mathcal{P}$. We assume that the parties are registered to $\mathcal{G}_{\mathtt{clock}}$. Let R be the the response obtained by querying $\mathcal{G}_{\mathtt{clock}}$, upon receiving any input I from any party or from the adversary act as follows:

- If $\mathsf{check\_liveness}(\tau_{\mathtt{last}},R) = 0$ then $\mathsf{max}_{\mathsf{tsl}} \leftarrow \mathsf{max}_{\mathsf{tsl}} + 1$, $\tau_{\mathtt{last}} \leftarrow R$, $\mathcal{T} \leftarrow \mathsf{force\_time\_table}(\mathcal{T},\mathcal{P},R,\mathsf{max}_{\mathsf{tsl}})$.
- If $\mathsf{check\_time\_table}(\mathcal{T},\mathcal{T},\mathcal{P},R,\mathsf{max}_{\mathsf{tsl}}) = 0$ then $\mathcal{T} \leftarrow \mathsf{force\_time\_table}(\mathcal{T},\mathcal{P},R,\mathsf{max}_{\mathsf{tsl}})$.

Proof.

- If $I = (\mathtt{TIMED\_PROVE}, sid, (x,w), \tau_{\mathtt{req}})$ is received from party p_i or from $\mathcal{A}$ (on behalf of a corrupted party p_i) check if $(x,w) \in \mathsf{Rel}$ and $\tau_{\mathtt{req}} \leq R$. If it is not then ignore I otherwise do the following.
  - if $\mathtt{t} \in \{-,\pm\}$ then record $(\mathtt{t}, \mathsf{max}_{\mathsf{tsl}}, x, p_i, \tau_{\mathtt{req}})$ and send $(\mathtt{TIMED\_PROVE}, sid, \mathtt{t}, \mathsf{max}_{\mathsf{tsl}}, x, p_i)$ to $\mathcal{A}$;
  - if $\mathtt{t} = \text{“}+\text{”}$ then record $(\mathtt{t}, \bot, x, p_i, \tau_{\mathtt{req}})$ and send $(\mathtt{TIMED\_PROVE}, sid, \mathtt{t}, x, p_i)$ to $\mathcal{A}$;
- If $I = (\mathtt{TIMED\_PROOF}, sid, (\mathtt{t}, \mathtt{tsl}_{\mathtt{back}}, x, p_i, \pi, \mathtt{tsl}_{\mathtt{post}}))$ is received from $\mathcal{A}$ then check if $(\mathtt{t}, \mathsf{oldmax}, x, p_i, \tau_{\mathtt{req}})$ was recorded. If it is not then ignore I otherwise do the following.
  - if $\mathtt{t} \in \{-,\pm\}$ then
    - if (p_i is honest and $\mathtt{tsl}_{\mathtt{back}} \leq \mathsf{oldmax} + \mathtt{waitingTime}$) or ($S$ is corrupted) and $\mathtt{tsl}_{\mathtt{back}} > \mathsf{max}_{\mathsf{tsl}}$) then $\mathtt{tsl}_{\mathtt{back}}{}' \leftarrow \mathtt{tsl}_{\mathtt{back}}$ else ignore I.
  - if $\mathtt{t} \in \{+,\pm\}$ then
    - if ($\mathcal{T}[\tau_{\mathtt{req}}, p_i] = \mathtt{tsl}_{\mathtt{post}}$ and p_i is not corrupted) or ($\mathtt{tsl}_{\mathtt{post}} \leq \mathcal{T}[\tau_{\mathtt{req}}, p_i] + \Delta$ and p_i is corrupted) then $\mathtt{tsl}_{\mathtt{post}}{}' \leftarrow \mathtt{tsl}_{\mathtt{post}}$ else ignore I.

  Store $(\mathtt{tsl}_{\mathtt{back}}{}', (x,\pi), \mathtt{tsl}_{\mathtt{post}}{}')$ and send $(\mathtt{TIMED\_PROOF}, sid, (\mathtt{tsl}_{\mathtt{back}}{}', x, \pi, \mathtt{tsl}_{\mathtt{post}}{}'))$ to p_i.

Figure 10.a: The $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$ functionality.

TPoK with Postdate Security via UC-NIZK. Our protocol $\Pi_{\mathsf{TPoK}}^{\mathsf{w},+}$ UC-realizes $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},+}$ in the $(\mathcal{F}_{\mathsf{NIZK}}, \mathcal{B}^{\mathsf{w}}, \mathcal{G}_{\mathtt{clock}})$-hybrid model (with $\mathsf{w} = ((\mu,\ell), \mathtt{MaxRound}, \mathtt{WindowSize}, \mathtt{MaxSize})$). $\Pi_{\mathsf{TPoK}}^{\mathsf{w},+}$ follows the *commit-and-prove* paradigm in which the prover commits to the witness w for the $\mathcal{NP}$-

Verification

- If $I = (\texttt{TIMED\_VERIFY}, sid, \mathtt{t}, x, (\mathtt{tsl}_{\mathtt{back}}, \pi, \mathtt{tsl}_{\mathtt{post}}))$ then check if $\mathtt{tsl}_{\mathtt{back}} \leq \mathsf{max}_{\mathsf{tsl}}$ or $\mathtt{tsl}_{\mathtt{back}} = \perp$. If it is not then ignore I. Otherwise set $\tau_{post} \leftarrow \perp$, $\tau_{back} \leftarrow \perp$, $\mathtt{b} = 0$ and continue as follows.
  If $\mathtt{t} \in \{-, \pm\}$ then
  - If $(\mathtt{tsl}_{\mathtt{back}}, x, \pi, \mathtt{tsl}_{\mathtt{post}})$ is stored and $\mathcal{T}[R, p_i] \geq \mathtt{tsl}_{\mathtt{back}}$ then find the smallest τ_{back} such that $\mathcal{T}[\tau_{back}, p_i] = \mathtt{tsl}_{\mathtt{back}}$ and set $\mathtt{b} \leftarrow 1$.

  If $\mathtt{t} \in \{+, \pm\}$ then
  - If $(\mathtt{tsl}_{\mathtt{back}}, x, \pi, \mathtt{tsl}_{\mathtt{post}})$ is stored and $\mathcal{T}[p_i, R] \geq \mathtt{tsl}_{\mathtt{post}}$ then find the smallest τ_{post} such that $\mathcal{T}[\tau, p_i] = \mathtt{tsl}_{\mathtt{post}}$ and set $\mathtt{b} \leftarrow 1$.

  If $\mathtt{b} = 0$ and $\mathtt{t} = +$ then record $(x, p_i, \mathtt{tsl}_{\mathtt{post}})$ and send $(\texttt{VERIFY}, sid, (x, \pi, \mathtt{tsl}_{\mathtt{post}}))$ to $\mathcal{A}$.
  If $\mathtt{b} = 1$ then send $(\texttt{TIMED\_VERIFICATION}, \mathtt{t}, sid, \tau_{back}, 1, \tau_{back})$.
  If $\mathtt{b} = 0$ then send $(\texttt{TIMED\_VERIFICATION}, \mathtt{t}, sid, \perp, 0, \perp)$.
- If $I = (\texttt{WITNESS}, w, x, \mathtt{tsl}_{\mathtt{post}})$ is received from $\mathcal{A}$, $(x, w) \in \mathsf{Rel}$ and $(x, p_i, \mathtt{tsl}_{\mathtt{post}})$ was recorded, then then store $(\perp, (x, \pi), \mathtt{tsl}_{\mathtt{post}})$, ignore I otherwise. If there exists a value τ such that $\mathcal{T}[\tau, p_i] = \mathtt{tsl}_{\mathtt{post}}$ then return $(\texttt{TIMED\_VERIFICATION}, \mathtt{t}, sid, \tau, 1)$ to p_i, send $(\texttt{TIMED\_VERIFICATION}, \mathtt{t}, sid, \perp, 0)$ otherwise.

Set delays and new time-slots

If $I = (\texttt{NEW\_SLOT}, sid)$ is received from $\mathcal{A}$ then $\tau_{\mathtt{last}} \leftarrow R$, $\mathsf{max}_{\mathsf{tsl}} \leftarrow \mathsf{max}_{\mathsf{tsl}} + 1$.
If $I = (\texttt{SET-DELAYS}, sid, \mathcal{T}')$ is received from $\mathcal{A}$, then $b \leftarrow \mathsf{check\_time\_table}(\mathcal{T}, \mathcal{T}', \mathcal{P}, R, \mathsf{max}_{\mathsf{tsl}})$. If $b = 1$ then $\mathcal{T} \leftarrow \mathcal{T}'$ else ignore I.

Withdraw future proof If $I = (\texttt{DELETE}, sid, \mathtt{t}, (\mathtt{tsl}_{\mathtt{back}}, x, \pi, \mathtt{tsl}_{\mathtt{post}}))$ then check if $\mathtt{tsl}_{\mathtt{post}} > \mathsf{max}_{\mathsf{tsl}}$. If it is not then Ignore I otherwise delete the entry (if it exists) $(\mathtt{tsl}_{\mathtt{back}}, x, \pi, \mathtt{tsl}_{\mathtt{post}})$ from the storage.

Figure 10.b: The $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},\mathsf{t}}$ functionality (cont'd).

statement x to be proven, and then proves to the verifier that the committed message corresponds to a valid witness for x. In our protocol we want to associate some time-stamp to the proof generated by the prover, so we slightly modify the above approach as follows. The prover obtains (η, τ) by invoking the weak beacon $\mathcal{B}^{\mathsf{w}}$ and computes a commitment $\mathtt{com}$ of $w||\eta$ (see Appendix C.1 for a formal definition of commitment). Then the prover proves to the verifier that $\mathtt{com}$ contains a witness for x concatenated with η. More precisely, let L be an $\mathcal{NP}$-language and Rel be the corresponding $\mathcal{NP}$-relation; to compute a proof the prover uses a NIZK PoK for the following $\mathcal{NP}$-relation

$$\mathsf{Rel}' = \{(\mathtt{com}, x, \eta), (\mathtt{dec}, w) \; s.t. \; \mathsf{Dec}(\mathtt{com}, \mathtt{dec}, \eta||w)) = 1 \text{ AND } (x, w) \in \mathsf{Rel}\}.$$

where x is the statement being proved and w is the corresponding witness (i.e. $(x, w) \in \mathsf{Rel}$). The verifier, upon receiving the proof computed by the prover accepts it if and only if the following two conditions hold: 1. value η has been output by $\mathcal{B}^{\mathsf{w}}$ in some round τ; 2. the NIZK proof given by the prover is accepting.

Since the NIZK that we use is a PoK and we assume that a malicious prover cannot predict the output of the weak beacon $\mathcal{B}^{\mathsf{w}}$ more than $\delta = \mathtt{MaxRound} \cdot (\mathtt{WindowSize} + \ell - \mu)$ rounds in advance then the verifier has the guarantee that the proof has been computed (and that the witness w was known by the prover) in some moment *subsequent* to $\tau - \delta$.

In Fig. 11 we show the details of our protocol. Our protocol uses $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{B}^{\mathsf{w}}$ and $\mathcal{G}_{\mathtt{clock}}$. We

recall that $\mathcal{B}^{\mathsf{w}}$ can be implemented by having having read-only access to $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$.

We assume that the parties $\mathcal{P}$ are registered to $\mathcal{F}_{\mathsf{NIZK}}$ for the $\mathcal{NP}$-relation Rel', $\mathcal{B}^{\mathsf{w}}$ and $\mathcal{G}_{\texttt{clock}}$.

Proof. The party $p \in \mathcal{P}$ on input $(\texttt{TIMED\_PROVE}, sid, (x, w), \tau)$ checks if $(x, w) \notin \mathsf{Rel}$. If it is, then ignores I else executes the following steps.

- Send $(\texttt{FETCH}, \tau, sid)$ to $\mathcal{B}^{\mathsf{w}}$ and upon receiving $(\texttt{FETCH}, sid, \texttt{tsl}, \eta)$, compute $(\texttt{com}, \texttt{dec}) \xleftarrow{\$} \mathsf{Com}(w||\eta)$ and define $x_t \leftarrow (\texttt{com}, x, \eta)$ and $w_t \leftarrow (w, \texttt{dec})$ such that $(x_t, w_t) \in \mathsf{Rel}'$.
- Send $(\texttt{PROVE}, sid, (x_t, w_t))$ to $\mathcal{F}_{\mathsf{NIZK}}$ and upon receiving the answer $(\texttt{PROOF}, sid, \pi)$ output $(\texttt{PROOF}, sid, x_t, (\pi, \texttt{tsl}))$.

Verification. The party p on input $(\texttt{TIMED\_VERIFY}, sid, +, x_t, (\perp, \pi, \texttt{tsl}))$ parses x_t as $(\texttt{com}, x, \eta)$ and executes the following steps.

- Query the beacon to see in which round τ the value η was issued.
- If this τ does not exist then output $(\texttt{TIMED\_VERIFICATION}, sid, +, \perp, 0, \perp)$, otherwise send $(\texttt{VERIFY}, sid, x_t, \pi)$ to $\mathcal{F}_{\mathsf{NIZK}}$.
- Upon receiving $(\texttt{VERIFICATION}, sid, b)$ from $\mathcal{F}_{\mathsf{NIZK}}$ output $(\texttt{TIMED\_VERIFICATION}, sid, \texttt{t}, \perp, b, \tau)$.

Figure 11: The protocol $\Pi^{\mathsf{w},+}_{\mathsf{TPoK}}$ in the $(\mathcal{F}_{\mathsf{NIZK}}, \mathcal{B}^{\mathsf{w}}, \mathcal{G}_{\texttt{clock}})$-hybrid model.

Theorem 5 *Let $\mathcal{F}^{\mathsf{w},+}_{\mathsf{TPoK}}$ be the functionality parametrized by $\mathsf{w} = (\Delta, \texttt{MaxRound}, \texttt{WindowSize}, \perp)$ then the protocol described in Fig. 11 realizes with perfect security $\mathcal{F}^{\mathsf{w},+}_{\mathsf{TPoK}}$ in the $(\mathcal{B}^{\mathsf{w}'}, \mathcal{F}_{\mathsf{NIZK}}, \mathcal{G}_{\texttt{clock}})$-hybrid model where $\mathcal{B}^{\mathsf{w}'}$ is parametrized by $\mathsf{w}' = ((\mu, \ell), \texttt{MaxRound}, \texttt{WindowSize}, \texttt{MaxSize})$ with $\Delta = \ell - \mu$.*

From Postdate Secure TPoK to Timed TPoK. In this section we show how to obtain a protocol $\Pi^{\mathsf{w},\pm}_{\mathsf{TPoK}}$ that UC-realizes $\mathcal{F}^{\mathsf{w},\pm}_{\mathsf{TPoK}}$ in the $(\mathcal{F}^{\mathsf{w},+}_{\mathsf{TPoK}}, \mathcal{G}_{\texttt{ledger}}, \mathcal{G}_{\texttt{clock}})$-hybrid model. $\Pi^{\mathsf{w},\pm}_{\mathsf{TPoK}}$ is the first protocol described in this work that uses $\mathcal{G}_{\texttt{ledger}}$ to store some information. Informally, every time that it is required to generate a proof for an $\mathcal{NP}$-statement x, $\Pi^{\mathsf{w},\pm}_{\mathsf{TPoK}}$ queries $\mathcal{F}^{\mathsf{w},+}_{\mathsf{TPoK}}$ thus obtaining a proof $(x, \pi, \texttt{tsl}_{\texttt{post}})$.

Then the prover of $\Pi^{\mathsf{w},\pm}_{\mathsf{TPoK}}$ generates a transaction that contains $(x, \pi, \texttt{tsl}_{\texttt{post}})$ and asks $\mathcal{G}_{\texttt{ledger}}$ to add the transaction to $\texttt{state}$. The parameter $\texttt{waitingTime}$ of $\Pi^{\mathsf{w},\pm}_{\mathsf{TPoK}}$ is the same parameter that in $\mathcal{G}_{\texttt{ledger}}$ defines the upper bound on the number of blocks that needs to be added to $\texttt{state}$ before that a transaction submitted from an honest party gets into $\texttt{state}$. Intuitively, the *postdate security* of $\Pi^{\mathsf{w},\pm}_{\mathsf{TPoK}}$ comes immediately from the postdate security of $\mathcal{F}^{\mathsf{w},+}_{\mathsf{TPoK}}$ and the *backward security* comes from the fact that once that a transaction is part of $\texttt{state}$ it cannot be removed. The formal construction of $\Pi^{\mathsf{w},\pm}_{\mathsf{TPoK}}$ for the $\mathcal{NP}$-Relation Rel is shown in Fig. 12. In this, every honest party p_i manages a table $\mathcal{T}^{\texttt{local}}_{p_i}$ that is updated on any input received by p_i according to the procedure $\mathsf{update\_time\_table}()$, see Sec. 5.1 for more details. The proofs that our constructions implement the different notions of TPoK are very similar to the proof of Sec. 6.1. We provide formal proofs in the full version of the paper.

8 Timed Signatures of Knowledge

In [CL06] Chase *et al.* introduce the notion of *signature of knowledge (SoK)*. A signature of knowledge schemes allows to issue signatures on behalf of any $\mathcal{NP}$-statement. That is, receiving a valid signature for a message m with respect to an $\mathcal{NP}$-statement x means that the signer of m knew the witness w for the $\mathcal{NP}$-statement x. In there, the authors propose a UC-definition of

We assume that the parties $\mathcal{P}$ are registered to $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$ and to the functionality $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},+}$ that is parametrized with the $\mathcal{NP}$-relation Rel. Every time that a party $p_i \in \mathcal{P}$ receives ad input she invokes the procedure update_time_table() (see Fig. 4 fore more details on this procedure).

Proof. The party $p_i \in \mathcal{P}$ on input $(\texttt{TIMED\_PROVE}, sid, \pm, (x, w), \tau)$ check if $(x, w) \in \mathsf{Rel}$. If it is not, then ignore I, proceeds as follows otherwise.

1. Forward $(\texttt{TIMED\_PROVE}, sid, +, (x, w), \tau)$ to $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},+}$ and upon receiving the answer $(\texttt{TIMED\_PROOF}, sid, +, \bot, \pi, \texttt{tsl})$, create a transaction $\texttt{tx} \leftarrow (x, \pi, \texttt{tsl})$ and send $(\texttt{SUBMIT}, sid, \texttt{tx})$ to $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$.
2. Wait until $\texttt{tx}$ is added to the state of $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$. Let $\texttt{tsl}_{\texttt{back}}$ be the block of $\texttt{state}$ that contains $\texttt{tx}$, output $(\texttt{TIMED\_PROOF}, sid, \pm, x, (\texttt{tsl}_{\texttt{back}}, \pi, \texttt{tsl}_{\texttt{post}}))$.

Verification. The party $p_i \in \mathcal{P}$ on input $(\texttt{TIMED\_VERIFY}, sid, \pm, x, (\texttt{tsl}_{\texttt{back}}, \pi, \texttt{tsl}_{\texttt{post}}))$ proceeds as follows.

1. Send $(\texttt{TIMED\_VERIFY}, sid, +, x, (\bot, \pi, \texttt{tsl}_{\texttt{post}}))$ to $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},+}$.
2. Upon receiving $(\texttt{TIMED\_VERIFICATION}, sid, +, \bot, b, \tau)$ from $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},+}$, if $b = 1$ then send $(\texttt{READ}, sid)$ to $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$ else output $(\texttt{TIMED\_VERIFICATION}, sid, \pm, \bot, 0, \bot)$.
3. Upon receiving the answer $(\texttt{READ}, sid, \texttt{state})$ from $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$ check if the transaction $\texttt{tx} := (x, \pi, \texttt{tsl}_{\texttt{post}})$ is stored in the $\texttt{tsl}_{\texttt{back}}$-th block of $\texttt{state}$. If it is, then find the smallest τ_{back} such that $\mathcal{T}_{p_i}^{\texttt{local}}[\tau_{back}] = \texttt{tsl}_{\texttt{back}}$ and output $(\texttt{TIMED\_VERIFICATION}, sid, \pm, \tau_{back}, 1, \tau_{post}, 1)$ otherwise output $(\texttt{TIMED\_VERIFICATION}, sid, \pm, \bot, 0, \bot)$.

Figure 12: The protocol $\Pi_{\mathsf{TPoK}}^{\mathsf{w},\pm}$ in the $(\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},+}, \mathcal{G}_{\texttt{ledger}}, \mathcal{G}_{\texttt{clock}})$-hybrid model.

signature of knowledge and provide a construction for it. Moreover, they propose a game-based definition of signature of knowledge and show that the UC and game-based definitions are indeed equivalent. Chase *et al.* also provide a construction for a signature of knowledge scheme assuming NIZK and encryption scheme[9]. The construction provided is very intuitive. The signer, on input a witness w for an $\mathcal{NP}$-statement x and the message m to be signed, encrypts $w||m$ thus obtaining the ciphertext c. Then, the signer computes a NIZK proof π to demonstrate that: "c contains the concatenation of the message m with a valid witness for x" and sends $(\sigma = (c, \pi), m)$ to the verifier. The verifier, upon receiving $(\sigma = (c, \pi), m)$, checks the validity of the NIZK by running the NIZK verifier on input (c, π, x, m). In this part of the paper we extend the definition of signature of knowledge by requiring the signatures to be time marked. That is, a signature for a message m with respect to an $\mathcal{NP}$-statement x has the form $(\sigma, m, \texttt{tsl})$ where $\texttt{tsl}$ gives an indication about when the σ was actually computed.

Exactly in the same spirit of signature and NIZK, in this section we define the notions of backdate, postdate and timed SoK. That is, we define the UC-functionalities $\mathcal{F}_{\mathsf{TSoK}}^{\mathsf{w},\mathtt{t}}$ with $\mathtt{t} \in \{-, +, \pm\}$. Those functionalities are analogous to the functionalities for timed signatures and NIZKs proposed in Sections 7 (see Fig. 13.a and 13.b). It should be easy to see that postdate, backdate and timed secure NIZK implies respectively postdate, backdate and timed secure TSoK. Indeed we observe in the $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$-hybrid model it is possible to obtain a protocol that UC-realizes $\mathcal{F}_{\mathsf{TSoK}}^{\mathsf{w},\mathtt{t}}$ following the approach proposed in [CL06] that we mentioned above. The only difference is that instead of using NIZK we rely on $\mathcal{F}_{\mathsf{TPoK}}^{\mathsf{w},\mathtt{t}}$ to prove that the ciphertext c contains the concatenation of the witness for x and the message m.

[9]The construction of [CL06] requires the NIZK to be simulation sound and the encryption scheme to be *dense*.

Initialization. The main parameter of $\mathcal{F}_{\mathsf{TSoK}}^{\mathsf{w},\mathtt{t}}$ is $\mathtt{t} \in \{-,+,\pm\}$. The functionality is also parametrized by the algorithms $\mathsf{check\_time\_table}(\mathcal{T},\mathcal{T}',\mathcal{P},R,\mathsf{max_{tsl}})$, $\mathsf{force\_time\_table}(\mathcal{T},\mathcal{P},R,\mathsf{max_{tsl}})$, $\mathsf{check\_liveness}(\tau_{\mathtt{last}},R)$ along with the parameters $\mathtt{WindowSize}$, $\mathtt{MaxRound}$, Δ, $\mathsf{max_{tsl}} \leftarrow 0$, $\mathtt{waitingTime}$, $\tau_{\mathtt{last}} \leftarrow 0$ the time-table $\mathcal{T}$ and running with parties $\mathcal{P} = \{p_1, \dots\}$ and adversary $\mathcal{A}$. $\mathcal{T}[0,p_i] \leftarrow 0$ for all $p_i \in \mathcal{P}$. We assume that the parties are registered to $\mathcal{G}_{\mathtt{clock}}$. Let R be the the response obtained by querying $\mathcal{G}_{\mathtt{clock}}$, upon receiving any input I from any party or from the adversary act as follows:

- If $\mathsf{check\_liveness}(\tau_{\mathtt{last}},R) = 0$ then $\mathsf{max_{tsl}} \leftarrow \mathsf{max_{tsl}} + 1$, $\tau_{\mathtt{last}} \leftarrow R$, $\mathcal{T} \leftarrow \mathsf{force\_time\_table}(\mathcal{T},\mathcal{P},R,\mathsf{max_{tsl}})$.
- If $\mathsf{check\_time\_table}(\mathcal{T},\mathcal{T},\mathcal{P},R,\mathsf{max_{tsl}}) = 0$ then $\mathcal{T} \leftarrow \mathsf{force\_time\_table}(\mathcal{T},\mathcal{P},R,\mathsf{max_{tsl}})$.

Proof.

- If $I = (\mathtt{TIMED\_SIGN}, sid, (x,w), m, \tau_{\mathtt{req}})$ is received from party p_i or from $\mathcal{A}$ (on behalf of a corrupted party p_i) check if $(x,w) \in \mathsf{Rel}$ and $\tau_{\mathtt{req}} \leq R$. If it is not then ignore I otherwise do the following.
  - if $\mathtt{t} \in \{-,\pm\}$ then record $(\mathtt{t}, \mathsf{max_{tsl}}, x, p_i, \tau_{\mathtt{req}})$ and send $(\mathtt{TIMED\_SIGN}, sid, \mathtt{t}, \mathsf{max_{tsl}}, x, m, p_i)$ to $\mathcal{A}$;
  - if $\mathtt{t} =$ " $+$ " then record $(\mathtt{t}, \perp, x, m, p_i, \tau_{\mathtt{req}})$ and send $(\mathtt{TIMED\_SIGN}, sid, \mathtt{t}, x, m, p_i)$ to $\mathcal{A}$;
- If $I = (\mathtt{TIMED\_SIGNATURE}, sid, (\mathtt{t}, \mathtt{tsl_{back}}, x, m, p_i, \sigma, \mathtt{tsl_{post}}))$ is received from $\mathcal{A}$ then check if $(\mathtt{t}, \mathsf{oldmax}, x, m, p_i, \tau_{\mathtt{req}})$ was recorded. If it is not then ignore I otherwise do the following.
  - if $\mathtt{t} \in \{-,\pm\}$ then
    - if (p_i is honest and $\mathtt{tsl_{back}} \leq \mathsf{oldmax} + \mathtt{waitingTime}$) or ($S$ is corrupted) and $\mathtt{tsl_{back}} > \mathsf{max_{tsl}}$) then $\mathtt{tsl_{back}}' \leftarrow \mathtt{tsl_{back}}$ else ignore I.
  - if $\mathtt{t} \in \{+,\pm\}$ then
    - if ($\mathcal{T}[\tau_{\mathtt{req}}, p_i] = \mathtt{tsl_{post}}$ and p_i is not corrupted) or ($\mathtt{tsl_{post}} \leq \mathcal{T}[\tau_{\mathtt{req}}, p_i] + \Delta$ and p_i is corrupted) then $\mathtt{tsl_{post}}' \leftarrow \mathtt{tsl_{post}}$ else ignore I.

  Store $(\mathtt{tsl_{back}}', (x,\sigma), m, \mathtt{tsl_{post}}')$ and send $(\mathtt{TIMED\_SIGNATURE}, sid, (\mathtt{tsl_{back}}', (x,\sigma), m, \mathtt{tsl_{post}}'))$ to p_i.

Figure 13.a: The $\mathcal{F}_{\mathsf{TSoK}}^{\mathsf{w},\mathtt{t}}$ functionality.

9 Acknowledgments

This research was partially supported by H2020 project PRIVILEDGE #780477 and OxChain project, EP/N028198/1, funded by EPSRC.

Verification

- If $I = (\texttt{TIMED\_VERIFY}, sid, \mathtt{t}, (\mathtt{tsl}_{\mathtt{back}}, (x, \sigma), m, \mathtt{tsl}_{\mathtt{post}}))$ then check if $\mathtt{tsl}_{\mathtt{back}} \leq \mathsf{max}_{\mathsf{tsl}}$ or $\mathtt{tsl}_{\mathtt{back}} = \perp$. If it is not then ignore I. Otherwise set $\tau_{post} \leftarrow \perp$, $\tau_{back} \leftarrow \perp$, $\mathtt{b} = 0$ and continue as follows.

  If $\mathtt{t} \in \{-, \pm\}$ then

  - If $(\mathtt{tsl}_{\mathtt{back}}, (x, \sigma), m, \mathtt{tsl}_{\mathtt{post}})$ is stored and $\mathcal{T}[R, p_i] \geq \mathtt{tsl}_{\mathtt{back}}$ then find the smallest τ_{back} such that $\mathcal{T}[\tau_{back}, p_i] = \mathtt{tsl}_{\mathtt{back}}$ and set $\mathtt{b} \leftarrow 1$.

  If $\mathtt{t} \in \{+, \pm\}$ then

  - If $(\mathtt{tsl}_{\mathtt{back}}, (x, \sigma), m, \mathtt{tsl}_{\mathtt{post}})$ is stored and $\mathcal{T}[p_i, R] \geq \mathtt{tsl}_{\mathtt{post}}$ then find the smallest τ_{post} such that $\mathcal{T}[\tau, p_i] = \mathtt{tsl}_{\mathtt{post}}$ and set $\mathtt{b} \leftarrow 1$.

  If $\mathtt{b} = 0$ and $\mathtt{t} = +$ then record $(x, p_i, \mathtt{tsl}_{\mathtt{post}})$ and send $(\texttt{VERIFY}, sid, ((x, \sigma), m, \mathtt{tsl}_{\mathtt{post}}))$ to $\mathcal{A}$.

  If $\mathtt{b} = 1$ then send $(\texttt{TIMED\_VERIFICATION}, \mathtt{t}, sid, \tau_{back}, 1, \tau_{back})$.
  If $\mathtt{b} = 0$ then send $(\texttt{TIMED\_VERIFICATION}, \mathtt{t}, sid, \perp, 0, \perp)$.
- If $I = (\texttt{WITNESS}, w, x, \mathtt{tsl}_{\mathtt{post}})$ is received from $\mathcal{A}$, $(x, w) \in \mathsf{Rel}$ and $(x, p_i, \mathtt{tsl}_{\mathtt{post}})$ was recorded, then then store $(\perp, (x, \sigma), m, \mathtt{tsl}_{\mathtt{post}})$, ignore I otherwise. If there exists a value τ such that $\mathcal{T}[\tau, p_i] = \mathtt{tsl}_{\mathtt{post}}$ then return $(\texttt{TIMED\_VERIFICATION}, \mathtt{t}, sid, \tau, 1)$ to p_i, send $(\texttt{TIMED\_VERIFICATION}, \mathtt{t}, sid, \perp, 0)$ otherwise.

Set delays and new time-slots

If $I = (\texttt{NEW\_SLOT}, sid)$ is received from $\mathcal{A}$ then $\tau_{\mathtt{last}} \leftarrow R$, $\mathsf{max}_{\mathsf{tsl}} \leftarrow \mathsf{max}_{\mathsf{tsl}} + 1$.
If $I = (\texttt{SET-DELAYS}, sid, \mathcal{T}')$ is received from $\mathcal{A}$, then $b \leftarrow \mathsf{check\_time\_table}(\mathcal{T}, \mathcal{T}', \mathcal{P}, R, \mathsf{max}_{\mathsf{tsl}})$. If $b = 1$ then $\mathcal{T} \leftarrow \mathcal{T}'$ else ignore I.

Withdraw future SoK If $I = (\texttt{DELETE}, sid, \mathtt{t}, (\mathtt{tsl}_{\mathtt{back}}, (x, \sigma), m, \mathtt{tsl}_{\mathtt{post}}))$ then check if $\mathtt{tsl}_{\mathtt{post}} > \mathsf{max}_{\mathsf{tsl}}$. If it is not then Ignore I otherwise delete the entry (if it exists) $(\mathtt{tsl}_{\mathtt{back}}, (x, \sigma), m, \mathtt{tsl}_{\mathtt{post}})$ from the storage.

Figure 13.b: The $\mathcal{F}_{\mathsf{TSoK}}^{\mathsf{w},\mathsf{t}}$ functionality.

References

[AD15] Marcin Andrychowicz and Stefan Dziembowski. PoW-based distributed cryptography with no trusted setup. pages 379–399, 2015.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018.

[BCG15] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source. *IACR Cryptology ePrint Archive*, 2015:1015, 2015.

[BdM91] Josh Benaloh and Michael de Mare. Efficient broadcast time-stamping. Technical report, 1991.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). pages 103–112, 1988.

[BGK+18] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. *IACR Cryptology ePrint Archive*, 2018:378, 2018 (Proceedings version to appear at ACM CCS 2018).

[BGZ16] Iddo Bentov, Ariel Gabizon, and David Zuckerman. Bitcoin beacon. *CoRR*, abs/1605.04559, 2016.

[BLSW05] Ahto Buldas, Peeter Laud, Märt Saarepera, and Jan Willemson. Universally composable time-stamping schemes with audit. pages 359–373, 2005.

[BLT14] Ahto Buldas, Risto Laanoja, and Ahto Truu. Efficient quantum-immune keyless signatures with identity. *IACR Cryptology ePrint Archive*, 2014:321, 2014.

[BLT17] Ahto Buldas, Risto Laanoja, and Ahto Truu. A server-assisted hash-based signature scheme. In *Secure IT Systems - 22nd Nordic Conference, NordSec 2017, Tartu, Estonia, November 8-10, 2017, Proceedings*, pages 3–17, 2017.

[BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. pages 324–356, 2017.

[BN00] Dan Boneh and Moni Naor. Timed commitments. In *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, pages 236–254, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. pages 136–145, 2001.

[Can03] Ran Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003. http://eprint.iacr.org/2003/239.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. pages 61–85, 2007.

[CE12] Jeremy Clark and Aleksander Essex. Commitcoin: Carbon dating commitments with bitcoin - (short paper). In *FC 2012*, pages 390–398, 2012.

[CGHZ16] Sandro Coretti, Juan A. Garay, Martin Hirt, and Vassilis Zikas. Constant-round asynchronous multi-party computation based on one-way functions. pages 998–1021, 2016.

[Cha88] David Chaum. Blind signature systems. U.S. Patent #4,759,063, July 1988.

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. pages 78–96, 2006.

[DMD04] Michael De Mare and Lincoln DeCoursey. A survey of the timestamping problem. Technical report, Citeseer, 2004.

[DMP88] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge proof systems. pages 52–72, 1988.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 409–418, New York, NY, USA, 1998. ACM.

[EO95] Tony Eng and Tatsuaki Okamoto. Single-term divisible electronic coins. In Alfredo De Santis, editor, *Advances in Cryptology — EUROCRYPT'94*, 1995.

[GJ03] Juan A. Garay and Markus Jakobsson. Timed release of standard digital signatures. In Matt Blaze, editor, *Financial Cryptography*. Springer Berlin Heidelberg, 2003.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. pages 281–310, 2015.

[GMG15] Bela Gipp, Norman Meuschke, and André Gernandt. Decentralized trusted timestamping using the crypto currency bitcoin. *CoRR*, abs/1502.04015, 2015.

[GN17] Yuefei Gao and Hajime Nobuhara. A decentralized trusted timestamping based on blockchains. *IEEJ Journal of Industry Applications*, 6(4):252–257, 2017.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3):11:1–11:35, 2012.

[HS91] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, Jan 1991.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. pages 477–498, 2013.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. pages 357–388, 2017.

[LGR15] Jia Liu, Flavio Garcia, and Mark Ryan. Time-release protocol from bitcoin and witness encryption for sat. *IACR Cryptology ePrint Archive*, 2015:482, 2015.

[Lin10] Yehuda Lindell. Foundations of cryptography 89-856. http://u.cs.biu.ac.il/~lindell/89-856/complete-89-856.pdf, 2010.

[LJKW18] Jia Liu, Tibor Jager, Saqib A. Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, 2018.

[LSS19] Esteban Landerreche, Marc Stevens, and Christian Schaffner. Non-interactive cryptographic timestamping based on verifiable delay functions. IACR, 2019.

[LTCL07] TC Lam, Cheng-Chung Tan, Yu-Jen Chang, and Jyh-Charn Liu. Timed zero-knowledge proof (tzkp) protocol. In *submitted to IEEE Real-Time and Embedded Technology and Application Symposium*, 2007.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. pages 643–673, 2017.

[RSW96] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.

[SSV19] Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Publicly verifiable proofs from blockchains. *PKC 2019*, 2019.

[Woo14] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

A A Survey of Related Work

Time-stamping digital documents. The idea of time-stamping a digital document was introduced in [HS91]. They provide two schemes, centralised and decentralised based on hash-chain and digital signatures. The former scheme includes a centralised time-stamping server and a set of clients (each of them given a unique ID). Each time a client wants to sign a document, it sends its ID and hash of its document to the time-stamping server who produce a signed certificate, given the clients request, where the certificate includes time, client's request, a counter, as well as information that links it to a certificate issued for the previous client. In this scheme, in order for a party to verify the correctness of the time-stamp included in the certificate of client i^{th}, it has to ask client $i-1^{th}$ to provide its certificate. However, this scheme is susceptible to forward-date attack by mounting a Sybille attack. In particular, the server can create a set of ID's and a set of dummy documents. Then, it computes certificates for the dummy documents. This allows it to forward-date the next document given by a legitimate client. Furthermore, for the scheme to support a fine-grained time-stamping mechanism, sufficiently high number of requests has to be sent frequently to the server. The second scheme, i.e. decentralised one, utilises a pseudorandom generator too. In this scheme, there is no time-stamping server and clients interactively cooperate with each other to time-stamp a document. As explained in [DMD04], an issue with this scheme is that many participants must engage in the process in order to time-stamp an individual document; however, in the absence of enough incentive they may participate that leads to denial of service. A collection of schemes proposed in [BdM91], that improve the efficiency of [HS91]. The schemes allow clients to interactively time-stamp their documents in rounds, such that in every round all documents and previous round time-stamp are combined together and a hash value of the combination is produced. To improve the scheme efficiency, in every round a Merkle tree is built on top of the documents and the previous round time-stamp. Then, the root of the tree is considered as the documents' time-stamp of the round. Moreover, [BLT14] proposes a protocol that requires multiple non-colluding servers who interactively time-stamp a document. Many variants of time-stamping mechanism have been proposed ever since. Recently, Buldas *et al.* [BLT17] proposed an efficient server-aided hash-based time-stamped signature scheme based on a Merkle tree and hash function. The scheme associates a time-slot to each signing key that can be used only for the related time period to sign a message. In particular, the signer first generates a set of signing keys, one for each time-slot, and commits to them. To sing a message at a certain time period, it hashes the related key along with the message and sends the result to the server who time-stamps it, using the tree, and returns the time-stamped value to the signer. When the time-slot has passes the signer releases the signature and the associated signing key. In the verification phase, the verifier accepts the signature if (a) the signing key and the time-stamp, provided by the server, belong to the same slot, and (b) the server has time stamped the same signature. In this scheme, it is assumed that the time stamping server is fully trusted and does not collude with a signer; otherwise, a signer can back/post-date a signature.

Time-lock encryptions and timed signatures. The idea to send information into the future, i.e. time-lock encryption, was first put forth by Timothy C. May. A time-lock encryption allows one to encrypt a message such that it cannot be decrypted (even the sender) until a certain amount of time has passed. The time-lock encryption scheme May proposed lies on a trusted agent. However, later on, [RSW96] proposes a protocol that does not require a trusted agent, and is secure against a receiver who may have access to many computation resources. The scheme is based on Blum-Blum-Shub

pseudorandom number generator that relies on modular repeated squaring, believed to be sequential. Later on, [BN00, GJ03] improves the previous work and proposed timed commitment and timed signature schemes. The former scheme allows a party to commit to a value in a way that if, later on, it avoids opening the commitment, it can be forced-open by anyone after a certain time period. The proposed timed-signature scheme lets two mutually distrustful parties sign a message, and prove to each other that they have signed a right message and it can be extracted after a certain time period, without disclosing their signature or decrypting it at the proving phase. The signature scheme is based on the repeated modular double-squaring, the timed-commitment and zero-knowledge proof systems. Very recently, after the appearance of Bitcoin and the blockchain technology, a set of blockchain-based time-lock encryptions have been proposed, e.g. [LJKW18, LGR15]. Intuitively, in these protocols, a sender encrypts a message using a witness encryption scheme whose secret key is a hash of future block, in the blockchian, that will appear after a certain time period. Note that the main difference between the witness encryption-based and the repeated squaring-based (or in general puzzle-based) approaches is that in the former one, when a secret key appears the recipient starts decrypting the ciphertext; whereas, in the latter one, the receiver has to start decrypting it, as soon as the ciphertext is provided, and the decryption process has to be run for the whole lifetime of the ciphertext.

Blockchain-based time-stamping service. In the same line of work, there are schemes, such as Origin-Stamp[10] or [GMG15, CE12, GN17], in which clients simply store the hash value (or commitment) of their data in a form of a transaction into a blockchian to time-stamp them. These schemes, unlike the previous ones, do not require a centralised trusted party or a collaboration of clients to time-stamp data, at the cost of blockchain transaction fee.

Time and zero-knowledge proofs. Lam *et al.* in [LTCL07] propose a protocol, called timed zero-knowledge proof, that supports secure access of shared computing resources for anonymous clients. The protocol is mainly based on a centralised e-cash scheme proposed in [EO95]. In the former protocol, in order for a client to obtain a timed zero-knowledge proof/token, it first receives a token from a central authority, and sends the token to a server who time-stamps it and returns the result to the client. The client can use the time-stamped token anonymously for a certain session only once and if the client double spends the token, it can be de-anonymised. We highlight that the notion of time has been also considered in concurrent zero-knowledge protocols, e.g. [DNS98], to enable them to be simulated in a concurrent setting. In these protocols, timing constraints are used to impose: (a) delays at the prover-side on sending some messages, and (b) time-outs at the verifier-side on accepting some messages. Thus, in the concurrent zero-knowledge protocols, the time constraints are imposed locally to the protocol's participants and are not embedded in a proof to time-stamp it.

Concluding Remarks. Among all schemes we studied above, only the server-aided schemes support a fine-grained and accurate time-stamping (at the cost of fully trusting the servers), as there is only one clock that belongs to the server who accurately time-stamp all data it receives. However, in the other schemes, the time is approximate, e.g. relative to the growth of a blockchain's length (in the blockchain-based protocols) or the computational power of the recipients (in the puzzle-based approaches).

[10] http://originstamp.org

B Additional UC Functionalities and Global Setups

The functionality is available to all participants. It is parametrized with variable τ, a set of parties $\mathcal{P}$, and a set $\mathcal{F}$ of functionalities. For each party $p \in \mathcal{P}$ it manages variable d_p. For each $f \in \mathcal{F}$ it manages variable d_f. Initially, $\tau := 0$, $\mathcal{P} := \emptyset$, $\mathcal{F} = \emptyset$

Synchronization

- Upon receiving $(\texttt{CLOCK-UPDATE}, sid_C)$ from some party $p \in \mathcal{P}$, set $d_p := 1$; execute *Round-Update()* and forward $(\texttt{CLOCK-UPDATE}, sid_C, p)$ to $\mathcal{A}$.
- Upon receiving $(\texttt{CLOCK-UPDATE}, sid_C)$ from some functionalities $f \in \mathcal{F}$ set $d_f := 1$; execute *Round-Update()* and forward $(\texttt{CLOCK-UPDATE}, sid_C, f)$ to $\mathcal{A}$.
- Upon receiving $(\texttt{CLOCK-READ}, sid_C)$ from any participant (including the environment, the adversary, or any ideal-shared or local-functionality) return $(\texttt{CLOCK-READ}, sid_C, \tau)$ to the requestor.

Procedure: Round-Update() if $d_f = 1$ for all $f \in \mathcal{F}$ and $d_p = 1$ for all honest $p \in \mathcal{P}$, then set $\tau = \tau + 1$ and reset $d_f := 0$ and $d_p := 0$ for all parties in $\mathcal{P}$.

Figure 14: The Global Clock Setup $\mathcal{G}_{\texttt{clock}}$

The functionality is parametrized by a security parameter λ. It maintains a set of registered parties $\mathcal{P}$ (initially set to $\emptyset$) and a (dynamically updatable) function table T (initially $T = \emptyset$).

- Upon receiving $(\texttt{EVAL}, sid, x)$ from $p \in \mathcal{P}$ or from $\mathcal{A}$, if $(x, \rho) \in T$, then return ρ to the requestor. Otherwise, if no entry for x is in T, then choose $\rho \leftarrow \{0,1\}^\lambda$, add (x, ρ) in T, and return ρ to the requestor.

Figure 15: The Random Oracle functionality $\mathcal{F}_{RO}$

B.1 The basic signature functionality

In Fig. 17, we provide the basic UC signature functionality proposed in [Can03].

C Additional Tools

C.1 Commitment Schemes

Definition 2 (Commitment Scheme) *Given a security parameter 1^λ, a commitment scheme* $\mathsf{CS} = (\mathsf{Sen}, \mathsf{Rec})$ *is a two-phase protocol between two* PPT *interactive algorithms, a sender* Sen *and a receiver* Rec. *In the commitment phase* Sen *on input a message m interacts with* Rec *to produce a commitment* com, *and the private output* d *of* Sen.

The functionality is parametrized by an $\mathcal{NP}$-relation Rel and running with parties $\mathcal{P} = \{p_1, \ldots, p_n\}$ and adversary $\mathcal{A}$.

Proof

- On input $I = (\texttt{PROVE}, sid, (x, w))$ from party $p_i \in \mathcal{P}$, ignore I if $(x, w) \notin \mathsf{Rel}$, record (x, p_i) and send $(\texttt{PROVE}, x)$ to $\mathcal{A}$ otherwise.
- On input $I = (\texttt{PROOF}, \pi, (x, p_i))$ from $\mathcal{A}$, if (x, p_i) is recorded then store (x, π) and send $(\texttt{PROOF}, sid, \pi)$ to p_i, ignore I otherwise.

Verification

- On input $I = (\texttt{VERIFY}, sid, x, \pi)$ is received from a party $p_i \in \mathcal{P}$ check whether (x, π) is stored. If not then record (x, π, p_i) and send $(\texttt{VERIFY}, x, \pi)$ to $\mathcal{A}$, otherwise return $(\texttt{VERIFICATION}, sid, 1)$ to p_i.
- On input $I = (\texttt{WITNESS}, w, (x, \pi, p_i))$ from $\mathcal{A}$, (x, π, p_i) is recorded and if $(x, w) \in \mathsf{Rel}$ then store (x, π) and return $(\texttt{VERIFICATION}, sid, 1)$ to p_i, return $(\texttt{VERIFICATION}, sid, 0)$ to p_i otherwise.

Figure 16: The NIZK functionality $\mathcal{F}_{\mathsf{NIZK}}$

In the decommitment phase, Sen *sends to* Rec *a decommitment information* $(m, \mathtt{d})$ *such that* Rec *accepts* m *as the decommitment of* $\mathtt{com}$.

Formally, we say that $\mathtt{CS} = (\mathsf{Sen}, \mathsf{Rec})$ *is a perfectly binding commitment scheme if the following properties hold:*

Correctness:

- *Commitment phase. Let* $\mathtt{com}$ *be the commitment of the message* m *given as output of execution of* $\mathtt{CS} = (\mathsf{Sen}, \mathsf{Rec})$ *where* Sen *runs on input a message* m. *Let* $\mathtt{d}$ *be the private output of* Sen *in this phase.*
- *Decommitment phase*[11]. Rec *on input* m *and* $\mathtt{d}$ *accepts* m *as decommitment of* $\mathtt{com}$.

Statistical (resp. Computational) Hiding([Lin10]): *for any adversary (resp.* PPT *adversary)* $\mathcal{A}$ *and a randomly chosen bit* $b \in \{0, 1\}$, *consider the following hiding experiment* $\mathsf{ExpHiding}^b_{\mathcal{A},\mathtt{CS}}(\lambda)$*:*

- *Upon input* 1^λ, *the adversary* $\mathcal{A}$ *outputs a pair of messages* m_0, m_1 *that are of the same length.*
- Sen *on input the message* m_b *interacts with* $\mathcal{A}$ *to produce a commitment of* m_b.
- $\mathcal{A}$ *outputs a bit* b' *and this is the output of the experiment.*

For any adversary (resp. PPT *adversary)* $\mathcal{A}$, *there exist a negligible function* ν, *s.t.:*

$$\left|\text{Prob}\left[\,\mathsf{ExpHiding}^0_{\mathcal{A},\mathtt{CS}}(\lambda) = 1\,\right] - \text{Prob}\left[\,\mathsf{ExpHiding}^1_{\mathcal{A},\mathtt{CS}}(\lambda) = 1\,\right]\right| < \nu(\lambda).$$

Statistical (resp. Computational) Binding: *for every commitment* $\mathtt{com}$ *generated during the commitment phase by a possibly malicious unbounded (resp. malicious* PPT*) sender* $\mathsf{Sen}^\star$ *there exists a negligible function* ν *such that* $\mathsf{Sen}^\star$, *with probability at most* $\nu(\lambda)$, *outputs two decommitments* $(m_0, \mathtt{d}_0)$ *and* $(m_1, \mathtt{d}_1)$, *with* $m_0 \neq m_1$, *such that* Rec *accepts both decommitments.*

We also say that a commitment scheme is perfectly binding *iff* $\nu(\lambda) = 0$.

[11] In this paper we consider only non-interactive commitment and decommitment phase.

Key Generation.
Upon receiving a value $(\texttt{KEY\_GEN}, sid)$ from some party $S \in \mathcal{P}$, verify that $sid = (S, sid')$ for some sid. If not, then ignore the request. Else, hand $(\texttt{KEY\_GEN}, sid)$ to the $\mathcal{A}$. Upon receiving $(\texttt{VERIFICATION\_KEY}, sid, v)$ from the $\mathcal{A}$, output $(\texttt{VERIFICATION\_KEY}, sid, v)$ to S, and record the pair (S, v).
Signature. If $I = (\texttt{SIGN}, sid, m)$ is received from party S, verify that $sid = (S, sid')$ for some sid'. If not, then ignore the request, else send $(\texttt{SIGN}, sid, m)$ to $\mathcal{A}$. Upon receiving $I = (\texttt{SIGNATURE}, sid, m, \sigma)$ from $\mathcal{A}$, verify that no entry $(m, \sigma, v, 0)$ is stored. If it is, then output an error message to S and halt. Else, send $(\texttt{SIGNATURE}, sid, m, \sigma)$ to S, and store the entry $(m, \sigma, v, 1)$.
Verification
Upon receiving a value $(\texttt{VERIFY}, sid, m, \sigma, v')$ from some party p_i, hand $(\texttt{VERIFY}, sid, m, \sigma, v')$ to the adversary. Upon receiving $(\texttt{VERIFIED}, sid, m, \phi)$ from the adversary do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $f = 1$. (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds.)
2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma, v, 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, v, 0)$. (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
3. Else, if there is an entry (m, σ, v', f') stored, then let $f = f'$. (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
4. Else, let $f = \phi$ and record the entry (m, σ, v', ϕ)

Send $(\texttt{VERIFIED}, sid, m, f)$ to p_i.

Figure 17: The $\mathcal{F}_{\texttt{SIGN}}$ functionality of [Can03]

When a commitment scheme $(\mathsf{Com}, \mathsf{Dec})$ *is non-interactive, to not overburden the notation, we use the following notation.*

- *Commitment phase.* $(\mathtt{com}, \mathtt{dec}) \xleftarrow{\$} \mathsf{Com}(m)$ *denotes that* com *is the commitment of the message* m *and* dec *represents the corresponding decommitment information.*
- *Decommitment phase.* $\mathsf{Dec}(\mathtt{com}, \mathtt{dec}, m) = 1$.

D Universal Composability (UC)

The Universal Composability (UC) framework introduced by Canetti in [Can01] is a security model capturing the security of a protocol Π under the concurrent execution of arbitrary other protocols. All those other protocols and processes not related to the protocol Π go through an environment $\mathcal{Z}$. The environment has the power to decide the input that the parties should use to run the Π, and to see the output of these parties. In this framework there is also an adversary $\mathcal{A}$ for the protocol Π that decides the parties to be corrupted and can communicate with $\mathcal{Z}$ (who knows which parties have been corrupted by $\mathcal{A}$). The security in this model is captured by the simulation-based paradigm. Let $\mathcal{F}$ be the ideal functionality that should be realized by Π. The ideal functionality $\mathcal{F}$

can be seen as a trusted party that handles the entire protocol execution and tells the parties what they would output if they executed the protocol correctly. We consider the ideal process where the parties simply pass on inputs from the environment to $\mathcal{F}$ and hand what they receive to the environment. In the ideal process, we have an ideal process adversary Sim. Sim does not learn the content of messages sent from $\mathcal{F}$ to the parties, but is in control of when, if ever, a message from $\mathcal{F}$ is delivered to the designated party. Sim can corrupt parties and at the time of corruption it will learn all inputs the party has received and all outputs it has sent to the environment. As the real world adversary, Sim can freely communicate with the environment. We compare running the real protocol with running the ideal process and say that Π UC-realizes $\mathcal{F}$ if no environment can distinguish between the two worlds. This means that the protocol is secure, if for any polynomial time $\mathcal{A}$ running in the real world with Π, there exists a polynomial time Sim running in the ideal process with $\mathcal{F}$, so no non-uniform polynomial time environment can distinguish the two worlds.

E Formal Proofs

E.1 Proof of Theorem 3

Following the approach proposed in the proof of Theorem 2, we summarize the behaviour of Sim as follows.

- If $(\texttt{KEY\_GEN}, sid)$ is received then forward it to $\mathcal{F}_{\texttt{SIGN}}$ upon receiving the answer $(\texttt{VERIFICATION\_KEY}, sid, v)$ sent it back to $\mathcal{F}_{\sigma}^{\mathsf{w},+}$.
- If the input $(\texttt{TIMED\_SIGN}, sid, +, m, \tilde{S}, \tau_{\texttt{req}})$ is received it means that a dummy party $\tilde{S}$ has received the input $(\texttt{TIMED\_SIGN}, sid, +, m, \tau_{\texttt{req}})$. Therefore, send $(\texttt{FETCH}, sid, \tau_{\texttt{req}})$ to $\mathcal{B}^{\mathsf{w}}$, and upon receiving $(\texttt{FETCH}, \eta)$ send $(\texttt{SIGN}, sid, \texttt{tsl}_{\texttt{post}}||m||\eta)$ to $\mathcal{F}_{\texttt{SIGN}}$. Upon receiving $(\texttt{SIGNATURE}, sid, m', \sigma)$, define $\sigma_t \leftarrow (\sigma, \eta)$ send $(\texttt{TIMED\_SIGNATURE}, sid, +, m, (\perp, \sigma_t, v, \texttt{tsl}_{\texttt{post}}))$ to $\mathcal{F}_{\sigma}^{\mathsf{w},+}$.
- If the input $(\texttt{TIMED\_VERIFIED}, sid, +, (\perp, m, \sigma, v', \texttt{tsl}_{\texttt{post}}))$ is received it means that a dummy party $\tilde{p}_i$ has received the input $(\texttt{TIMED\_VERIFY}, sid, +, (\perp, m, \sigma_t, v', \texttt{tsl}_{\texttt{post}}))$. Therefore parses σ_t as (σ, η) and send $(\texttt{VERIFY}, sid, \texttt{tsl}_{\texttt{post}}||m||\eta, \sigma, v')$ to $\mathcal{F}_{\texttt{SIGN}}$ and upon receiving the answer $(\texttt{VERIFIED}, sid, m, \phi)$ query the $\mathcal{B}^{\mathsf{w}}$ to check if the value η is the $\texttt{tsl}_{\texttt{post}}$-th output of $\mathcal{B}^{\mathsf{w}}$. If it is not then set $\phi \leftarrow 0$ and send $(\texttt{TIMED\_VERIFIED}, sid, m, \phi)$ to $\mathcal{F}_{\sigma}^{\mathsf{w},+}$.
- At any round Sim updates $\mathcal{T}$ according to the time table that is managed by $\mathcal{B}^{\mathsf{w}}$. That is, let $\mathcal{T}'$ be the time table of $\mathcal{B}^{\mathsf{w}}$, then Sim sends $(\texttt{SET-DELAYS}, sid, \mathcal{T}')$ to $\mathcal{F}_{\sigma}^{\mathsf{w},+}$ at any round.
- If the adversary obtains a valid signature σ for the message $i||m||\eta$ by querying $\mathcal{F}_{\texttt{SIGN}}$ with $(\texttt{SIGN}, sid, i||m||\eta)$ then send $(\texttt{TIMED\_SIGN}, sid, +, m, \tau_{\texttt{req}})$ to $\mathcal{F}_{\sigma}^{\mathsf{w},+}$. Upon receiving $(\texttt{TIMED\_SIGN}, sid, +, \mathsf{max}_{\mathsf{tsl}}, m, S, \tau_{\texttt{req}})$, check if $\mathsf{max}_{\mathsf{tsl}} + \delta \leq i$. If it is not, then ignore, otherwise set $\texttt{tsl}_{\texttt{post}} \leftarrow i$ and send $(\texttt{TIMED\_SIGNATURE}, sid, \texttt{t}, m, (\perp, \sigma, v, \texttt{tsl}_{\texttt{post}}))$ to $\mathcal{F}_{\sigma}^{\mathsf{w},+}$. We refer to a signature $(m, (\perp, \sigma, \texttt{tsl}_{\texttt{post}}))$ computed in the way we just described as *predicted signature*. Any time that a predicted signature is catch by Sim, he stores it together with η.
- At any round Sim updates $\mathcal{T}$ according to the time table that is managed by $\mathcal{B}^{\mathsf{w}}$. That is, let $\mathcal{T}'$ be the time table of $\mathcal{B}^{\mathsf{w}}$, then then Sim sends $(\texttt{SET-DELAYS}, sid, \mathcal{T}')$ to $\mathcal{F}_{\sigma}^{\mathsf{w},+}$ at any round.

- Any time that $\mathcal{B}^{\mathsf{w}}$ issues a new value Sim checks if the signatures predicted by the adversary are actually valid. That is, for any recorded entry with the form $(m, (\bot, \sigma, \mathtt{tsl}_{\mathtt{post}}), \eta)$ that have $\mathtt{tsl}_{\mathtt{post}} = |\mathcal{H}|$ (we recall that $\mathcal{H}$ contains all the output issued by $\mathcal{B}^{\mathsf{w}}$), Sim checks if $\mathcal{H}[\mathtt{tsl}_{\mathtt{post}}] = \eta$. If it is not then Sim sends $I = (\mathtt{DELETE}, sid, \mathtt{t}, (\mathtt{tsl}_{\mathtt{back}}, x, \pi, \mathtt{tsl}_{\mathtt{post}}))$. In the end Sim aligns the size of the time-slots with the size new size of $\mathcal{H}$ by sending $(\mathtt{NEW\_SLOT}, sid)$ to $\mathcal{F}_{\sigma}^{\mathsf{w},+}$.

E.2 Proof of Theorem 2

Let $\mathcal{A}$ be an arbitrary polynomial time adversary. We will describe a corresponding polynomial time ideal process adversary Sim such that no non-uniform polynomial time environment can distinguish whether $\Pi_{\sigma}^{\mathsf{w},-}$ is running in the $(\mathcal{W}_{\mathrm{WBU}}(\mathcal{G}_{\mathtt{ledger}}), \mathcal{G}_{\mathtt{clock}}, \mathcal{F}_{\mathtt{SIGN}})$-hybrid model with parties $p_1, \ldots, p_n$ and adversary $\mathcal{A}$ or the ideal process is running with $\mathcal{F}_{\sigma}^{\mathsf{w},-}$, Sim and dummy parties $\tilde{p}_1, \ldots, \tilde{p}_n$. Sim starts by invoking a copy of $\mathcal{A}$. It will run a simulated interaction of $\mathcal{A}$, the parties and the environment. In particular, whenever the simulated $\mathcal{A}$ communicates with the environment, Sim just passes this information along. And whenever $\mathcal{A}$ corrupts a party p_i, Sim corrupts the corresponding dummy party $\tilde{p}_i$. We summarize the behaviour of Sim as follows.

- If $(\mathtt{KEY\_GEN}, sid)$ is received then forward it to $\mathcal{F}_{\mathtt{SIGN}}$ upon receiving the answer $(\mathtt{VERIFICATION\_KEY}, sid, v)$ sent it back to $\mathcal{F}_{\sigma}^{\mathsf{w},-}$.

- If the input $(\mathtt{TIMED\_SIGN}, sid, -, m, \tilde{S}, \tau_{\mathtt{req}})$ is received it means that a dummy party $\tilde{S}$ has received the input $(\mathtt{TIMED\_SIGN}, sid, -, m, \tau_{\mathtt{req}})$. Therefore, send $(\mathtt{SIGN}, sid, m)$ to $\mathcal{F}_{\mathtt{SIGN}}$, and upon receiving $(\mathtt{SIGNATURE}, sid, m, \sigma)$ generate the transaction $\mathtt{tx} \leftarrow (m, \sigma)$ and send $\mathtt{tx}$ to $\mathcal{W}_{\mathrm{WBU}}(\mathcal{G}_{\mathtt{ledger}})$. When $\mathtt{tx}$ is added to the state $\mathtt{state}$ of $\mathcal{W}_{\mathrm{WBU}}(\mathcal{G}_{\mathtt{ledger}})$, take the index of the block $\mathtt{tsl}_{\mathtt{back}}$ that contains $\mathtt{tx}$ and send $(\mathtt{TIMED\_SIGNATURE}, sid, -, m, (\mathtt{tsl}_{\mathtt{back}}, \sigma, v, \bot))$ to $\mathcal{F}_{\sigma}^{\mathsf{w},-}$.

- If the input $(\mathtt{TIMED\_VERIFIED}, sid, -, (\mathtt{tsl}_{\mathtt{back}}, m, \sigma, v', \bot))$ is received it means that a dummy party $\tilde{p}_i$ has received the input $(\mathtt{TIMED\_VERIFY}, sid, -, (\mathtt{tsl}_{\mathtt{back}}, m, \sigma, v', \bot))$. Therefore Send $(\mathtt{VERIFY}, sid, m, \sigma, v')$ to $\mathcal{F}_{\mathtt{SIGN}}$ and upon receiving the answer $(\mathtt{VERIFIED}, sid, m, \phi)$ send $(\mathtt{READ}, sid)$ to $\mathcal{W}_{\mathrm{WBU}}(\mathcal{G}_{\mathtt{ledger}})$. Upon receiving the answer from $(\mathtt{READ}, sid, \mathtt{state})$ check if the block in the $\mathtt{tsl}_{\mathtt{back}}$-th position of $\mathtt{state}$ contains the transaction $\mathtt{tx} = (m, \sigma)$. If it is not, then $\phi \leftarrow 0$. Send $(\mathtt{TIMED\_VERIFIED}, sid, m, \phi)$ to $\mathcal{F}_{\sigma}^{\mathsf{w},-}$.

- At any round Sim updates $\mathcal{T}$ according to the slackness values of $\mathcal{W}_{\mathrm{WBU}}(\mathcal{G}_{\mathtt{ledger}})$. That is, Sim reads the values $\mathtt{pt}_{i_1}, \ldots, \mathtt{pt}_{i_n}$ from $\mathcal{W}_{\mathrm{WBU}}(\mathcal{G}_{\mathtt{ledger}})$, defines a time-table $\mathcal{T}'$ that extends the previous one ($\mathcal{T}$) by setting $\mathcal{T}'[R, p_{i_1}] = \mathtt{pt}_{i_j}$ for all the honest parties p_{i_j}, and set $\mathcal{T}'[\tau, p_{i_k}] = |\mathtt{state}|$ for all the corrupted parties p_{i_k}. Then Sim sends $(\mathtt{SET\text{-}DELAYS}, sid, \mathcal{T}')$ to $\mathcal{F}_{\sigma}^{\mathsf{w},-}$.

We recall that the maxim number of blocks after that an honest transaction is added to the state of $\mathcal{W}_{\mathrm{WBU}}(\mathcal{G}_{\mathtt{ledger}})$ is $\mathtt{waitingTime}$. Therefore, if the environment can distinguish between the ideal and the real execution it means that the security of either $\mathcal{F}_{\mathtt{SIGN}}$ or $\mathcal{G}_{\mathtt{ledger}}$ has been compromised.

E.3 Proof of Theorem 1

Let $\mathcal{A}$ be an arbitrary polynomial time adversary. We will describe a corresponding polynomial time ideal process adversary Sim such that no non-uniform polynomial time environment can distinguish

whether Π^{w} is running in the $(\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}}), \mathcal{G}_{\texttt{clock}}, \mathcal{F}_{RO})$-hybrid model with parties $p_1, \ldots, p_n$ and adversary $\mathcal{A}$ or the ideal process is running with $\mathcal{B}^{\mathsf{w}}$, Sim and dummy parties $\tilde{p}_1, \ldots, \tilde{p}_n$. Sim starts by invoking a copy of $\mathcal{A}$. It will run a simulated interaction of $\mathcal{A}$, the parties and the environment. In particular, whenever the simulated $\mathcal{A}$ communicates with the environment, Sim just passes this information along. And whenever $\mathcal{A}$ corrupts a party p_i, Sim corrupts the corresponding dummy party $\tilde{p}_i$.

In the following description of Sim, we denote with $\texttt{head}$ the current size of the state $\texttt{state}$ of $\mathcal{G}_{\texttt{ledger}}$. The behaviour of Sim can be summarized as follows.

1. Whenever $\mathcal{A}$ sends $(\texttt{EVAL}, sid, i||x)$ to $\mathcal{F}_{RO}$, Sim does the following.

   - If $i > \texttt{head} + \ell - \mu$, then pick a random value $\rho \in \{0,1\}^\lambda$ and instruct $\mathcal{F}_{RO}$ to reply with $(i||x, \rho)$ any time that $(\texttt{EVAL}, sid, i||x)$ is received.
   - If $\texttt{head} < i \leq \texttt{head} + \ell - \mu$, then send $(\texttt{READ\_SETS}, sid)$ to $\mathcal{B}^{\mathsf{w}}$. Upon receiving $(\texttt{READ\_SETS}, sid, \mathcal{S}_1, \ldots, \mathcal{S}_{\ell-\mu})$, take randomly a new element η from $\mathcal{S}_{i-\texttt{head}} - \tilde{\mathcal{S}}_{i-\texttt{head}}$, $\tilde{\mathcal{S}}_{i-\texttt{head}} \leftarrow \tilde{\mathcal{S}}_{i-\texttt{head}} \cup \{\eta\}$ and instruct $\mathcal{F}_{RO}$ to reply with η on the query $(\texttt{EVAL}, sid, i||x)$.

2. Any time that $\texttt{state}$ is extended with a new block $\texttt{Block}$, Sim checks if $\texttt{Block}$ is generated honestly by reading the value $\texttt{hflag}$. Let $\texttt{cq} \leftarrow \ell - \mu + 1$. Sim computes $x \leftarrow \texttt{state}_{|\texttt{head},\texttt{head}-\texttt{cq}}$ and does the following.

   - If $\texttt{hflag} = 1$ then send $I = (\texttt{SET\_RANDOM}, sid)$ to $\mathcal{B}^{\mathsf{w}}$. If $\mathcal{B}^{\mathsf{w}}$ replies with $(\texttt{OK}, sid, \eta)$, then instruct $\mathcal{F}_{RO}$ to reply with η on the query $(\texttt{EVAL}, sid, \texttt{head}||x)$ and set $\tilde{\mathcal{S}}_1 = \emptyset, \ldots, \tilde{\mathcal{S}}_{\ell-\mu} = \emptyset$
   - if $\texttt{hflag} = 0$ check if $\mathcal{F}_{RO}$ has been queried with $(\texttt{EVAL}, sid, \texttt{head}||x)$. If it is, then send $(\texttt{SET}, sid, x)$ to $\mathcal{B}^{\mathsf{w}}$ else send $I = (\texttt{SET\_RANDOM}, sid)$. We observe that if $(\texttt{EVAL}, sid, \texttt{head}||x)$ has been queried before by the adversary, then $\eta \in \mathcal{S}_{(\texttt{head} \mod \ell-\mu)}$.

3. At any round, Sim updates $\mathcal{T}$ according to the slackness values of $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$. That is, Sim reads the values $\texttt{pt}_{i_1}, \ldots, \texttt{pt}_{i_n}$ from $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$, defines a time-table $\mathcal{T}'$ that extends the previous one $(\mathcal{T})$ by setting $\mathcal{T}'[R, p_{i_1}] = \texttt{pt}_{i_j}$ for all the honest parties p_{i_j}, and set $\mathcal{T}'[\tau, p_{i_k}] = |\texttt{state}|$ for all the corrupted parties p_{i_k}. Then Sim sends $(\texttt{SET-DELAYS}, sid, \mathcal{T}')$ to $\mathcal{B}^{\mathsf{w}}$.

We now observe that Sim in step 3 keeps consistent the view that each party has of $\texttt{state}$ with the view of $\mathcal{H}$. That is, each party that in the ideal model can see the i-th value of $\mathcal{H}$ can see also the i-th block of $\texttt{state}$ in the real world. This is done in step 3 by updating the time-table $\mathcal{T}$ consistently with the slackness of $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$. Moreover, any time that $\texttt{state}$ is extended with a new block, $\mathcal{H}$ is extended as well. As showed in step 2, $\mathcal{H}$ is extended via adding a random value decided by the functionality if $\texttt{state}$ was extended honestly, or by using a value taken from the set $\mathcal{S}_i$ with $i = \texttt{head} \mod \ell - \mu$. The crucial observation here is that, because honest blocks include at least λ bits of fresh entropy, if $\texttt{state}$ is extended with an honest block then the output of the RO on input $(\texttt{EVAL}, sid, \texttt{state}_{(\texttt{head}-\ell+\mu+1,\texttt{head})})$ is a uniform in $\{0,1\}^\lambda$. Moreover, since any consecutive blocks of $\texttt{state}$ contain, at least, μ honestly generated blocks, this guarantee that the adversary can only predict the next $\ell - \mu$ outputs. In step 1 Sim takes care of the latter aspect by keeping consistent the content of the sets $\mathcal{S}_1, \ldots, \mathcal{S}_{\ell-\mu}$ with the queries made by the real world adversary to the RO. In more details, once that Sim gets the sets $\mathcal{S}_1, \ldots, \mathcal{S}_{\ell-\mu}$ from $\mathcal{B}^{\mathsf{w}}$ then he instructs the

RO to reply to the query $(\texttt{EVAL}, sid, j||x)$ with η, where η is randomly chosen from $\mathcal{S}_{j \mod \ell-\mu}$. We recall that these sets have size $\texttt{MaxSize}$ where $\texttt{MaxSize}$ denotes an upper bound on the number of queries that can be made by the adversary to the RO. We also observe that, in order to reply with different values of $\mathcal{S}_{j \mod \ell-\mu}$ to the queries $(\texttt{EVAL}, sid, j||x)$, $(\texttt{EVAL}, sid, j||x')$ with $x \neq x'$, Sim keep track of all the values of $\mathcal{S}_{j \mod \ell-\mu}$ that have been already used to program the RO in a special set $\tilde{\mathcal{S}}_{j \mod \ell-\mu}$.

E.4 Proof of Lemma 1

Let κ be the security parameter of Π_{BB}, $\delta = \texttt{MaxRound} \cdot \texttt{WindowSize} \cdot (\ell - \mu)$ and $\lambda = \hat{\lambda} - \log(\kappa)$.

We start the proof by showing that in Π_{BB}, under the assumption that the honest parties include a random value of length $\hat{\lambda}$ in each coinbase transaction, no adversary can predict at round ρ, with probability greater than $2^{-\lambda}$, the nonce η' that is added by an honest party to a block that becomes part of $\texttt{state}$ at round $\rho + \delta$. Let ρ_1 be the round in which an honest party p_i sees $\mathsf{st} = \texttt{state}_{|p_i}$, then the block that extends st could either be a block generated by an honest party, or a block generated by a malicious party. Moreover, this new block has to be added to st after at most $\texttt{MaxRound} \cdot \texttt{WindowSize}$ rounds. In the case that the block that extends st is malicious, we cannot say much on the entropy since, without loss of generality, we assume that the content of the block is in full control of the adversary. Let us now consider the case in which the added block is honest. Let ρ_2 be the round in which the block becomes part of the ledger state. We note that all honest parties that generate candidate blocks[12] for $\texttt{state}$ at round ρ_2 could see $\texttt{state}$ already at round ρ_1 with $\mathcal{T} = |\rho_2 - \rho_1| \leq \texttt{MaxRound} \cdot \texttt{WindowSize}$. We now want to compute t, which represents the number all the possible candidate blocks that can be seen by the adversary in the interval $[\rho_1, \rho_2]$. Note that there could be other candidate blocks for ledger states that are shorter than $\texttt{state}$, but those blocks cannot be used by the adversary anymore.

From [GKL15, Remark 3] we know that the number of blocks generated by the honest parties in an interval of size $\mathcal{T}$ is $t = pq(n-m)\mathcal{T}$, where n is to the total number of parties, m is the number of parties controlled by the adversary, p is the probability that an honest party generates a block and q is the upper bound on the number of queries that each party can make to the random oracle. Moreover, from the proof of [GKL15, Lemma 6] we know that $pq(n-m) \leq \frac{1}{2}$, therefore $t \leq \mathcal{T}/2 \leq \texttt{MaxRound} \cdot \texttt{WindowSize}/2$. Given that $\texttt{WindowSize}$ and $\texttt{MaxRound}$ are polynomially related to the security parameter of the ledger κ, we have that $t = \mathsf{poly}(\kappa)$.

From the chain quality we also know that after *at most* $\ell - \mu$ blocks, an honestly generated block has to be added to the ledger state. This means that if $\delta = \texttt{MaxRound} \cdot (\texttt{WindowSize} + \ell - \mu)$, then by round $\rho + \delta$ there is one block included in the ledger state that has min-entropy $\lambda = \hat{\lambda} - \log(\kappa)$, conditional on the view of the adversary at round ρ. We are now ready to show that Π_{BB} implements $\mathcal{W}_{\mathrm{WBU}}[\mathcal{G}_{\texttt{ledger}}]$. We describe a corresponding polynomial-time simulator Sim. Let $\mathsf{Sim}_{\mathrm{BB}}$ be the simulator for Π_{BB}. Sim acts as the ideal-functionality $\mathcal{G}_{\texttt{ledger}}$ for $\mathsf{Sim}_{\mathrm{BB}}$ with the following difference. Whenever it is required to compute a nonce for the coinbase transaction, Sim queries $\mathcal{W}_{\mathrm{WBU}}[\mathcal{G}_{\texttt{ledger}}]$ with $\mathsf{new\_nonce}$ thus obtaining (N, ρ) and uses N a the nonce with $N \in \{0,1\}^{\hat{\lambda}}$.

[12] In this proof we call *candidate block* a block that could extend $\texttt{state}$.

F Functionalities with Dynamic Party Sets

UC provides support for functionalities in which the set of parties that might interact with the functionality is dynamic. We make this explicit by means of the following mechanism (that we describe almost verbatim from [BMTZ17, Sec. 3.1]): All the functionalities considered here include the following instructions that allow honest parties to join or leave the set $\mathcal{P}$ of players that the functionality interacts with, and inform the adversary about the current set of registered parties:

- Upon receiving $(\texttt{REGISTER}, sid)$ from some party p_i (or from $\mathcal{A}$ on behalf of a corrupted p_i), set $\mathcal{P} := \mathcal{P} \cup \{p_i\}$. Return $(\texttt{REGISTER}, sid, p_i)$ to the caller.
- Upon receiving $(\texttt{DE\_REGISTER}, sid)$ from some party $p_i \in \mathcal{P}$, the functionality updates $\mathcal{P} := \mathcal{P} \setminus \{p_i\}$ and returns $(\texttt{DE\_REGISTER}, sid, p_i)$ to p_i.
- Upon receiving $(\texttt{IS\_REGISTERED}, sid)$ from some party p_i, return $(\texttt{REGISTER}, sid, b)$ to the caller, where the bit b is 1 if and only if $p_i \in \mathcal{P}$.
- Upon receiving $(\texttt{GET\_REGISTERED}, sid)$ from $\mathcal{A}$, the functionality returns the response $(\texttt{GET\_REGISTERED}, sid, \mathcal{P})$ to $\mathcal{A}$.

In addition to the above registration instructions, global setups, i.e., shared functionalities that are available both in the real and in the ideal world and allow parties connected to them to share state [CDPW07], allow also UC functionalities to register with them. Concretely, global setups include, in addition to the above party registration instructions, two registration/de-registration instructions for functionalities:

- Upon receiving $(\texttt{REGISTER}, sid_G)$ from a functionality F (with session-id sid), update $F := F \cup \{(F, sid)\}$.
- Upon receiving $(\texttt{DE\_REGISTER}, sid_G)$ from a functionality F (with session-id sid), update $F := F\{(F, sid)\}$.
- Upon receiving $(\texttt{GET\_REGISTERED}_F, sid_G)$ from $\mathcal{A}$, return $(\texttt{GET\_REGISTERED}_F, sid_G, F)$ to $\mathcal{A}$.

We use the expression sid_G to refer to the encoding of the session identifier of global setups. By default (and if not otherwise stated), the above four (or seven in case of global setups) instructions will be part of the code of all ideal functionalities considered in this work. However, to keep the description simpler we will omit these instructions from the formal descriptions unless deviations are defined.

G The Basic Transaction-Ledger Functionality of [BMTZ17]

Ledger Element	**Description**
$\mathcal{P}, \mathcal{H}, \mathcal{P}_{DS}$	The party sets and categories: Registered, honest, and honest-but-desynchronized, respectively.
$\vec{\mathcal{I}}_H^T$	The timed honest-input sequence.
predict-time	The function to predict the real-world time advancement.
`state`	The ledger state, i.e., a sequence of blocks containing the content.
`buffer`	The buffer of submitted input values.
$\texttt{pt}_i$, $\texttt{state}_i$	The pointer of party P_i into state `state`. This prefix is denoted $\texttt{state}_i$ for brevity.
$\vec{\tau}_{\texttt{state}}$	A vector containing for each state block the time when the block added to the ledger state.
τ_L	The current time as reported by the clock.
`NxtBC`	Stores the current adversarial suggestion for extending the ledger state.
Validate	Decides on the validity of a transaction with respect to the current state. Used to clean the buffer of transactions.
ExtendPolicy	The function that specifies the ledger's guarantees in extending the ledger state (e.g., speed, content etc.).
Blockify	The function to format the ledger state output.
`windowSize`	The window size (number of blocks) of the sliding window.
`Delay`	A general delay parameter for the time it takes for a newly joining (after the onset of the computation) miner to become synchronized.

Functionality $\mathcal{G}_{\text{LEDGER}}$

General: The functionality is parametrized by four algorithms $\mathsf{Validate}$, $\mathsf{ExtendPolicy}$, $\mathsf{Blockify}$, and $\mathsf{predict\text{-}time}$, along with two parameters $\mathtt{windowSize}, \mathtt{Delay} \in \mathbb{N}$. The functionality manages variables $\mathtt{state}$, $\mathtt{NxtBC}$, $\mathtt{buffer}$, τ_L, and $\vec{\tau}_{\mathtt{state}}$, as described above. Initially, $\mathtt{state} := \vec{\tau}_{\mathtt{state}} := \mathtt{NxtBC} := \varepsilon$, $\mathtt{buffer} := \emptyset$, $\tau_L = 0$.
For each party $P_i \in \mathcal{P}$ the functionality maintains a pointer $\mathtt{pt}_i$ (initially set to 1) and a current-state view $\mathtt{state}_i := \varepsilon$ (initially set to empty). The functionality keeps track of the timed honest-input sequence $\vec{\mathcal{I}}_H^T$ (initially $\vec{\mathcal{I}}_H^T := \varepsilon$).

Party management: The functionality maintains the set of registered parties $\mathcal{P}$, the (sub-)set of honest parties $\mathcal{H} \subseteq \mathcal{P}$, and the (sub-set) of de-synchronized honest parties $\mathcal{P}_{DS} \subset \mathcal{H}$ (following the definition in the previous paragraph). The sets $\mathcal{P}, \mathcal{H}, \mathcal{P}_{DS}$ are all initially set to $\emptyset$. When a new honest party is registered at the ledger, if it is registered with the clock already then it is added to the party sets $\mathcal{H}$ and $\mathcal{P}$ and the current time of registration is also recorded; if the current time is $\tau_L > 0$, it is also added to $\mathcal{P}_{DS}$. Similarly, when a party is deregistered, it is removed from both $\mathcal{P}$ (and therefore also from $\mathcal{P}_{DS}$ or $\mathcal{H}$). The ledger maintains the invariant that it is registered (as a functionality) to the clock whenever $\mathcal{H} \neq \emptyset$. A party is considered fully registered if it is registered with the ledger and the clock.

Upon receiving any input I from any party or from the adversary, send $(\text{CLOCK-READ}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$ and upon receiving response $(\text{CLOCK-READ}, \text{sid}_C, \tau)$ set $\tau_L := \tau$ and do the following:

1. Let $\widehat{\mathcal{P}} \subseteq \mathcal{P}_{DS}$ denote the set of desynchronized honest parties that have been registered (continuously) since time $\tau' < \tau_L - \mathtt{Delay}$ (to both ledger and clock). Set $\mathcal{P}_{DS} := \mathcal{P}_{DS} \setminus \widehat{\mathcal{P}}$. On the other hand, for any synchronized party $P \in \mathcal{H} \setminus \mathcal{P}_{DS}$, if P is not registered to the clock, then $\mathcal{P}_{DS} \cup \{P\}$.

2. If I was received from an honest party $P_i \in \mathcal{P}$:

   (a) Set $\vec{\mathcal{I}}_H^T := \vec{\mathcal{I}}_H^T || (I, P_i, \tau_L)$;

   (b) Compute $\vec{N} = (\vec{N}_1, \ldots, \vec{N}_\ell) := \mathsf{ExtendPolicy}(\vec{\mathcal{I}}_H^T, \mathtt{state}, \mathtt{NxtBC}, \mathtt{buffer}, \vec{\tau}_{\mathtt{state}})$ and if $\vec{N} \neq \varepsilon$ set $\mathtt{state} := \mathtt{state} || \mathsf{Blockify}(\vec{N}_1) || \ldots || \mathsf{Blockify}(\vec{N}_\ell)$ and $\vec{\tau}_{\mathtt{state}} := \vec{\tau}_{\mathtt{state}} || \tau_L^\ell$, where $\tau_L^\ell = \tau_L || \ldots || \tau_L$.

   (c) For each $\mathtt{BTX} \in \mathtt{buffer}$: if $\mathsf{Validate}(\mathtt{BTX}, \mathtt{state}, \mathtt{buffer}) = 0$ then delete $\mathtt{BTX}$ from $\mathtt{buffer}$. Also, reset $\mathtt{NxtBC} := \varepsilon$.

   (d) If there exists $P_j \in \mathcal{H} \setminus \mathcal{P}_{DS}$ such that $|\mathtt{state}| - \mathtt{pt}_j > \mathtt{windowSize}$ or $\mathtt{pt}_j < |\mathtt{state}_j|$, then set $\mathtt{pt}_k := |\mathtt{state}|$ for all $P_k \in \mathcal{H} \setminus \mathcal{P}_{DS}$.

3. Depending on the input I and the ID of the sender, execute the respective code:

   – *Submiting a transaction:*
   If $I = (\text{SUBMIT}, \text{sid}, \mathtt{tx})$ and is received from a party $P_i \in \mathcal{P}$ or from $\mathcal{A}$ (on behalf of a corrupted party P_i) do the following

   (a) Choose a unique transaction ID txid and set $\mathtt{BTX} := (\mathtt{tx}, \text{txid}, \tau_L, P_i)$

   (b) If $\mathsf{Validate}(\mathtt{BTX}, \mathtt{state}, \mathtt{buffer}) = 1$, then $\mathtt{buffer} := \mathtt{buffer} \cup \{\mathtt{BTX}\}$.

   (c) Send $(\text{SUBMIT}, \mathtt{BTX})$ to $\mathcal{A}$.

   – *Reading the state:*
   If $I = (\text{READ}, \text{sid})$ is received from a fully registered party $P_i \in \mathcal{P}$ then set $\mathtt{state}_i := \mathtt{state}|_{\min\{\mathtt{pt}_i, |\mathtt{state}|\}}$ and return $(\text{READ}, \text{sid}, \mathtt{state}_i)$ to the requestor. If the requestor is $\mathcal{A}$ then send $(\mathtt{state}, \mathtt{buffer}, \vec{\mathcal{I}}_H^T)$ to $\mathcal{A}$.

   – *Maintaining the ledger state:*
   If $I = (\text{MAINTAIN-LEDGER}, \text{sid}, \text{minerID})$ is received by an honest party $P_i \in \mathcal{P}$ and (after updating $\vec{\mathcal{I}}_H^T$ as above) $\mathsf{predict\text{-}time}(\vec{\mathcal{I}}_H^T) = \hat{\tau} > \tau_L$ then send $(\text{CLOCK-UPDATE}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$. Else send I to $\mathcal{A}$.

   – *The adversary proposing the next block:*
   If $I = (\text{NEXT-BLOCK}, \text{hFlag}, (\text{txid}_1, \ldots, \text{txid}_\ell))$ is sent from the adversary, update $\mathtt{NxtBC}$ as follows:

   (a) Set listOfTxid $\leftarrow \epsilon$

   (b) For $i = 1, \ldots, \ell$ do: if there exists $\mathtt{BTX} := (x, \text{txid}, \text{minerID}, \tau_L, P_i) \in \mathtt{buffer}$ with ID $\text{txid} = \text{txid}_i$ then set listOfTxid := listOfTxid$||\text{txid}_i$.

   (c) Finally, set $\mathtt{NxtBC} := \mathtt{NxtBC} || (\text{hFlag}, \text{listOfTxid})$ and output $(\text{NEXT-BLOCK}, ok)$ to $\mathcal{A}$.

   – *The adversary setting state-slackness:*
   If $I = (\text{SET-SLACK}, (P_{i_1}, \widehat{\mathtt{pt}}_{i_1}), \ldots, (P_{i_\ell}, \widehat{\mathtt{pt}}_{i_\ell}))$, with $\{P_{i_1}, \ldots, P_{i_\ell}\} \subseteq \mathcal{H} \setminus \mathcal{P}_{DS}$ is received from the adversary $\mathcal{A}$ do the following:

   (a) If for all $j \in [\ell]$: $|\mathtt{state}| - \widehat{\mathtt{pt}}_{i_j} \leq \mathtt{windowSize}$ and $\widehat{\mathtt{pt}}_{i_j} \geq |\mathtt{state}_{i_j}|$, set $\mathtt{pt}_{i_1} := \widehat{\mathtt{pt}}_{i_1}$ for every $j \in [\ell]$ and return $(\text{SET-SLACK}, ok)$ to $\mathcal{A}$.

   (b) Otherwise set $\mathtt{pt}_{i_j} := |\mathtt{state}|$ for all $j \in [\ell]$.

   – *The adversary setting the state for desychronized parties:*
   If $I = (\text{DESYNC-STATE}, (P_{i_1}, \mathtt{state}'_{i_1}), \ldots, (P_{i_\ell}, \mathtt{state}'_{i_\ell}))$, with $\{P_{i_1}, \ldots, P_{i_\ell}\} \subseteq \mathcal{P}_{DS}$ is received from the adversary $\mathcal{A}$, set $\mathtt{state}_{i_j} := \mathtt{state}'_{i_j}$ for each $j \in [\ell]$ and return $(\text{DESYNC-STATE}, ok)$ to $\mathcal{A}$.