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

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

Aydin Abadi[1], Michele Ciampi[1], Aggelos Kiayias[2], and Vassilis Zikas[3]

[1] The University of Edinburgh. {aydin.abadi,mciampi}@ed.ac.uk
[2] The University of Edinburgh and IOHK. akiayias@inf.ed.ac.uk
[3] The University of Edinburgh and IOHK. vassilis.zikas@ed.ac.uk

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. Even though there are blockchain-based timestamping proposals, they are not formally defined and proved in a universally composable (UC) 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 propose timed versions of primitives commonly used for authenticating information, such as digital signatures, non-interactive zero-knowledge proofs, and signatures of knowledge. We show how they can be UC-securely constructed by a protocol that makes ideal (blackbox) access to a transaction ledger. 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 construct UC-securely. Given many potential use cases of such a beacon in cryptographic protocols, this result is of independent interest.

1 Introduction

Timestamping allows for a (digital) object—typically a document—to be associated with a creation time, such that anyone seeing the timestamp can verify that the document was not created before or after that 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 [25].

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 $T' > T$. 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.* [25] achieve timestamping using a *hash-chain of documents.* In the plain, centralized version of their scheme the parties have access to a semi-trusted 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 [18]. To circumvent such issues, [25] 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.* [5]. Later on, [13] formally models the timestamping mechanisms, previously proposed in [5,25], using the UC model. Moreover, it provides a construction very similar to [5,25] with the main difference that it utilises an additional trusted party, an auditor, who periodically verifies the TS. Also, [6] provides solutions for time stamping a specific data type, i.e., audiovisual, by using unpredictable information from a trusted public source. The authors also provide some interesting applications of the timestamping for the case of postdate and backdate security, (see [6] for more examples). More recently, [12] 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. First, it can only work if the servers are properly synchronized and their communication network is synchronous. Indeed, [5,12] 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. The above issues are implicit in the treatment of [25,5], and there is no known technique to mitigate them. These issues are similar to the core problem treated by blockchains and their associated cryptocurrencies [32,35,27]. 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[4], that use this idea to offer a blockchain-based timestampting system. Following this research line, very recently [29] presented a treatment of non-interactive timestamping schemes in the UC-model. The construction provided in [29] is based on proofs of sequential work such as VDF's [9]. However, as the authors stated in [29], 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[5]. In the concurrent work of Zhang et al. [36] it is also considered the use of a blockchain to time stamp digital files, by storing the file along with a hash of a series of blockchain blocks in the blockchain. However, [36] lacks an appropriate security definition and analysis and focuses only on the timestamping of digital documents.

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 [25,5,29], 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 [4] by a global clock functionality, it trivially extends to allow for parties having bounded-drift view of the clock [26]—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 [20,8], and signatures of knowledge [17] in Canetti's Universal Composition (UC) framework [14]. Our treatment explicitly captures security against *backdating* and *postdating* separately, and investigates the associated assumptions required to achieve each of

[4] https://guardtime.com

[5] In [29] 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.

these security notions. Finally, we devise UC secure constructions 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. [4,3] to implement timed versions of these primitives while making blackbox (hybrid) access to a 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[6], 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 [15] 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. We stress that all our constructions are proved to be UC-secure (as also the realization of the ledger functionality proposed in [4] is UC-secure). To the best of our knowledge this is the first result that provides a complete UC treatment of the notion of timed signature with respect to a global clock under a blockchain prospective. One of the main tool used in this paper is a *weak beacon*. In this work we provide a formalization of the weak beacon and show how it can be realized using an augmented version of the ledger provided in [4,3]. This augmented ledger captures the entropy contained in the blocks of a ledger. The formalization of such a ledger, and its instantiation (which we also provide) can be seen as result of independent interest.

Our Techniques. A standard idea for achieving security against postdating attacks 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 is: *How can we construct such a source of sufficiently unpredictable*

[6] In our result we make use of a ledger functionality that slightly extends the one proposed in [4] to capture the entropy of the blockchain.

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 [2,11,7]. But, 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 [7], 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 $\mathcal{B}^{\mathtt{w}}$, 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 `MaxR` rounds, where `MaxR` is a parameter that depends on the ledger's liveness parameter[7] (we discuss it in more details in Sec. 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 [24] (cf. Sec. 2): (1) the lengths of the prefixes seen by different parties do not differ by more than `WSize` again a parameter which depends on the ledger (which reflects the similarity of the blockchain to the dynamics of a so-called sliding window, where the window of size `WSize` contains the possible views of honest miners onto state and where the head of the window advances with the head of the state),(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 [4], the latter one is not. To address this, we introduce a simple *wrapper* functionality that upgrades the ledger functionality of [4] to possess this weak unpredictability property while we show that the main result of [4], namely that the Bitcoin backbone protocol of [24] implements the ledger, can be strengthened accordingly.

We provide a formal description of the above sketched weak beacon, and prove that it can be constructed by a protocol which makes ideal access to any of the ideal ledger functionalities from the literature [4,3] 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

[7] The ledger's liveness property from [4] corresponds to the *chain growth* property from [24].

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. 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 (a detailed treatment of timed non-interactive zero-knowledge and signature of knowledge is deferred to the full version of the paper [1]).

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. We include a more detailed survey of that literature in the full version [1] where we also discuss basic results in zero knowledge (including some recent attempts that use time [21,28,22]). To our knowledge none of the existing blockchain-based solutions obtains timestamping with only ideal (black-box) 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 [34,10,23,31,30]. Despite the similarity in the name, these works do not (aim to) achieve timestamping guarantees. As stated in [9], VDFs can be used for timestamping. However, as discussed in [9], this application of VDF requires precise bounds on the attacker's computation speed, otherwise would lead to a serious issue. Namely, if an attacker can speed up VDF evaluation by a factor of X using faster hardware, then once the fraudulent history is more than $1/X$ as old as the genuine history, the attacker can fool participants into believing the fraudulent history is actually older than the genuine one. We note that the output of our beacon can be used as input to a VDF as noted in [9].

Notation. We denote the security parameter by λ, and "||" as concatenation. 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 readers are familiar with standard notions such as *commitment* and UC-security (see the paper full version [1] for formal definitions).

Organization of Paper. The remainder of this paper is structured as follows. In Sec. 2 we put forth our execution modeling reviewing relevant aspects of the UC framework. In Sec. 3 we provide the description of wrapper for the ledger functionality to capture the entropy contained in the blockchains. In Sec. 4 we describe our (weak) beacon functionality describe how to realized it via the ledger functionality. In Sec. 5 we provide a technical overview of the results on timed signatures and deferred to the full version [1] the formal description of our timed signature UC-functionalities, their instantiations via the ledger functionality and the security proofs. For lack of space we defer the treatement of timed zero-knowledge and signature of knowledge to the full version as well.

2 The Model

Following the recent line of works proving composable security of blockchain ledgers [4,3] we provide our protocols and security proofs in Canetti's universal composition (UC) framework [14]. In this section we discuss the main components of our real-world model (including the associated hybrids). 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 [16]. 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 [16] 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 [4,26]: 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 [26,19]. 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. Note, 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 [4]. We adopt the *dynamic availability* model implicit in [4] which was fleshed out in [3]. 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 the full version [1]. 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. We conclude this section by elaborating on the hybrid functionalities and global setups used by our protocol. These are standard functionalities from literature; but, for self-containment we have included their descriptions here.

The Clock Functionality $\mathcal{G}_{\texttt{clock}}$. The *clock functionality* was initially proposed in [26] to enable synchronous execution of UC protocols. Here we adopt its global-setup version, denoted by $\mathcal{G}_{\texttt{clock}}$, proposed by [4] and was used in the UC proofs of the ledger's security.[8] $\mathcal{G}_{\texttt{clock}}$ 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 $\mathcal{F}_{RO}$. As in cryptographic proofs the queries to hash function are modeled by assuming access to a random oracle function-

[8] 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.

ality: 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 Ledger Functionality $\mathcal{G}_{\texttt{ledger}}$. The last functionality 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 [4,3] in order to describe a transaction ledger and its properties. As proved in [4,3] 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 $\texttt{tx}$—which is accessible from the parties under some restrictions discussed below. It enforces the following basic properties:

- *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*.[9]
- *Transaction liveness.* Old enough (and valid) transactions are included in the next block added to the ledger state.

We next give a brief overview of the ledger functionality $\mathcal{G}_{\texttt{ledger}}$. Along the way we also introduce some useful notation and terminology. Note, 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 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. 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 [24,33] 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 [4], 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.

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

For convenience, we denote by $\texttt{state}_{|\texttt{pt}_i}$ 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 `WSize` state blocks. The parameter $\texttt{WSize} \in \mathbb{N}$ reflects the similarity of the blockchain to the dynamics of *sliding window*, where the window of size `WSize` contains the possible views of honest miners onto `state` and where the head of the window advances with the head of `state`.

3 Weak Block Unpredictability (WBU)

A delicate point about the ledger from [4,3] is the way it enforces the chain quality property from [24]. 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 [4,3] and the backbone abstraction of the protocol [24,33]—uses the assumption that the coinbase transaction of such *honest* blocks includes at least $\hat{\lambda}$ bits randomly chosen by an honest party[10]. 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 [24], 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 [4]. 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

[10] Formally, in [4,3] 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.

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, also proposed in [4] for explicitly capturing assumptions—in the case of [4] 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 $\mathcal{W}_{\text{WBU}}$. 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 interfaces). If it observes that the simulator does not ask for a nonce for more than $(\ell - \mu) \cdot \mathtt{MaxR}$ 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 $\delta = \mathtt{MaxR} \cdot (\ell - \mu)$, 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 $\mathcal{G}_{\mathtt{ledger}}$) 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.*
- *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 $\rho > \rho' + \delta = \rho' + \mathtt{MaxR} \cdot (\ell - \mu)$ then halt. In any other case relay messages between the wrapped functionality and the entities it is connected to (i.e., the simulator, the environment, and the global setups it registered with.)*

The above definition provides a lot of freedom to the adversary for the dishonestly generated blocks. Indeed, the adversary could potentially decide entirely the content of a malicious block. We note that this might not be the case for some existent blockchains. However, since we would like our definitions to be as generic as possible we consider such a powerful adversary. We also prove that the (UC abstraction of the) Bitcoin backbone protocol from [4] emulates the wrapped ledger $\mathcal{W}_{\text{WBU}}[\mathcal{G}_{\mathtt{ledger}}]$, where, $\mathcal{G}_{\mathtt{ledger}}$ is the ledger from [4]. The lemma follows directly by observing that the simulator of [4] internally generates the coinbase for honest blocks by emulating the honest protocol. Our detailed proof can be found in the full version [1].

4 The (Weak) Beacon Functionality and Construction

Here, 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, due to the adversarial influence on the contents of the ledger, we cannot obtain such a perfect beacon from the ledgers implemented by common cryptocurrencies (cf. also [7] for an impossibility). Nonetheless, as it turns out, even under a worst-case analysis as in [24,4], the contents of the ledger are periodically updated 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 functionality $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\texttt{ledger}})$. We refer to this beacon as a *weak beacon*. As we show, this beacon will be sufficient for our timestamping schemes.

Our Beacon Functionality. In this section we provide a definition of our weak beacon by means of UC-functionality. Then we show how to realize this functionality assuming the existence of a (wrapped) ledger. 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 $\mathcal{G}_{\texttt{clock}}$-hybrid model. Note, 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 $\mathtt{w} = ((\mu, \ell), \mathtt{MaxR}, \mathtt{WSize}, \mathtt{MaxSize})$ 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 having 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 $\mathtt{WSize}$. More precisely, each party p_i can see only the first $\mathtt{pt}_i$ elements of $\mathcal{H}$, where $\mathtt{pt}_i$ is adversarially chosen in each round, with the restriction $|\mathcal{H}| - \mathtt{pt}_i \leq \mathtt{WSize}$ 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 $\mathsf{check\_t\_table}$, which will be executed whenever the adversary attempts to rewrite indexes; if the check fails

then another procedure, force_t_table, is invoked which overwrites the adversary's choices with values of $\mathtt{pt}_i$ 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 MaxR rounds.

Weak Unpredictability: An adversary has the following influence on the beacon output: 1) 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 $|\mathcal{S}_i| \leq \mathtt{MaxSize} = \mathsf{poly}(\lambda)$) 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; 2) The adversary can predict, in the worst case, the next $\ell - \mu$ outputs of the beacon. Specifically, 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 at any point the adversary might *predict* the output of a beacon for up to the next $\delta = (\ell - \mu + \mathtt{WSize}) \cdot \mathtt{MaxR}$ 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 tsl that we call *time-slot index*. This value 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 tsl outputs of $\mathcal{B}^{\mathsf{w}}$, denoted by $\mathcal{H}[1], \ldots, \mathcal{H}[\mathtt{tsl}]$. 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 MaxR 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 WSize far apart (this together will enforce the eventual agreement property). These properties are formally enforced by two procedures, called force_t_table and check_t_table that check if the adversary complies with the above policy as follows: The procedure check_t_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 force_t_table is invoked to enforce the policy mandated by check_t_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_t_table is invoked in the following two cases: 1) If $\mathcal{H}$ has not been extended in the last MaxR rounds. In this case $\mathcal{B}^{\mathsf{w}}$ generates a random output, appends it

to $\mathcal{H}$ and extends $\mathcal{T}$ using force_t_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_t_table. 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 $\{0,1\}^\lambda$ Looking ahead in our beacon protocol, λ corresponds to the bits of entropy guaranteed to be included in an honestly generated ledger block. $\mathcal{S}_{i+\ell-\mu+1}$ is 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 MaxR 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 $\mathsf{force\_liveness}(\mathsf{max}_{\mathsf{tsl}}, \mathcal{T}, \mathcal{H})$; if it fails, i.e., the adversary tries to delay the beacon's update by more than MaxR rounds, then procedure $\mathsf{force\_liveness}(\mathsf{max}_{\mathsf{tsl}}, \mathcal{T}, \mathcal{H})$ is invoked which forces the above policy in a default manner. The formal description of the helper procedures and of our weak beacon functionality are referred to the full version [1].

Our weak beacon protocol. At a high level, our beacon protocol works as follows. A party that wants to compute the beacon's output reads state from $\mathcal{W}_{\mathrm{WBU}}(\mathcal{G}_{\mathtt{ledger}})$ 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 $\mathtt{WSize}$ blocks. Therefore, in the worst case, the adversary sees $\mathtt{WSize}$ 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 $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\mathtt{ledger}})$ with chain quality parameters (μ, ℓ) and window size $\mathtt{WSize}$, 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 $\Delta = \ell - \mu + \mathtt{WSize}$ outputs. To see why the output of the beacon is unpredictable, we recall that $\mathcal{W}_{\text{WBU}}(\mathcal{G}_{\mathtt{ledger}})$ guarantees that the blocks generated by the honest party contains some entropy. In practise, this entropy comes from a random value inserted by an honest miner into the block it mines. Similarly to [3,4,24,33] this random value is based on the assumption that the coinbase transaction of honest blocks includes some random bits chosen the honest parties. We refer the reader to the full version [1] for the formal description of our protocol.

5 Timed Signatures (TSign)

In this section, we extend the standard notion of the digital signature (described in [15]) 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. 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 $\mathcal{F}_{\sigma}^{\mathsf{w},\mathtt{t}}$. $\mathcal{F}_{\sigma}^{\mathsf{w},\mathtt{t}}$ is parameterized by a flag $\mathtt{t} \in \{+, -, \pm\}$ where $\mathtt{t} =$ "$-$" indicates that the functionality guarantees backdate security, $\mathtt{t} =$ "$+$" indicates postdate security, and $\mathtt{t} =$ "$\pm$" 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 $\mathsf{w} = (\Delta, \mathtt{MaxR}, \mathtt{WSize}, \mathtt{waitingTime})$, 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 $\mathtt{MaxR}$ 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 $\mathtt{WSize}$-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 $\mathtt{tsl} - \mathtt{WSize}$ 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 $\mathtt{WSize}$ 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 $\mathsf{check\_t\_table}$ and $\mathsf{force\_t\_table}$ 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 $\mathtt{tsl} > \mathsf{max}_{\mathsf{tsl}}$ via the command $(\mathtt{DELETE}, sid, \cdot)$. We also introduce a parameter $\mathtt{waitingTime}$, 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 $\mathtt{waitingTime}$ 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, $\mathtt{waitingTime} \cdot \mathtt{MaxR}$ rounds in order to see σ time-marked. To guarantee that a new time-slot is available every $\mathtt{MaxR}$ (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. 4 for more details on how the liveness is enforced). The formalization of our functions follows the signature functionality $\mathcal{F}_{\mathsf{SIGN}}$ proposed in [15]. Roughly, $\mathcal{F}_{\mathsf{SIGN}}$ stores all the signatures that are issued, and when a verification request for a message m occurs then $\mathcal{F}_{\mathsf{SIGN}}$ 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}. 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 creates a signature using a standard signature scheme (formally we invoke the ideal signature functionality that we denote with $\mathcal{F}_{\mathsf{SIGN}}$) and inserts its signature, via a transaction to the blockchain ($\mathcal{G}_{\mathtt{ledger}}$). 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 prove in the full version, combining the above two ideas yields a signature with both backdate and postdate security. For the formal constructions and definitions we refer the reader to the full version [1].

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

References

1. Abadi, A., Ciampi, M., Kiayias, A., Zikas, V.: Timed signatures and zero-knowledge proofs -timestamping in the blockchain era-. Cryptology ePrint Archive, Report 2019/644 (2019), https://eprint.iacr.org/2019/644
2. Andrychowicz, M., Dziembowski, S.: PoW-based distributed cryptography with no trusted setup. In: Gennaro, R., Robshaw, M.J.B. (eds.) Advances in Cryptology – CRYPTO 2015, Part II. Lecture Notes in Computer Science, vol. 9216, pp. 379–399. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015). https://doi.org/10.1007/978-3-662-48000-7_19
3. Badertscher, C., Gazi, P., Kiayias, A., Russell, A., Zikas, V.: Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018: 25th Conference on Computer and Communications Security. pp. 913–930. ACM Press, Toronto, ON, Canada (Oct 15–19, 2018). https://doi.org/10.1145/3243734.3243848
4. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: A composable treatment. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017, Part I. Lecture Notes in Computer Science, vol. 10401, pp. 324–356. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017). https://doi.org/10.1007/978-3-319-63688-7_11
5. Benaloh, J., de Mare, M.: Efficient broadcast time-stamping. Tech. rep. (1991)
6. Bennett, C.H.: Improvements to time bracketed authentication. CoRR **cs.CR/0308026** (2003)

7. Bentov, I., Gabizon, A., Zuckerman, D.: Bitcoin beacon. CoRR (2016)
8. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications (extended abstract). In: 20th Annual ACM Symposium on Theory of Computing. pp. 103–112. ACM Press, Chicago, IL, USA (May 2–4, 1988). https://doi.org/10.1145/62212.62222
9. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: CRYPTO 2018 (2018)
10. Boneh, D., Naor, M.: Timed commitments. In: Advances in Cryptology - CRYPTO 2000 (2000)
11. Bonneau, J., Clark, J., Goldfeder, S.: On bitcoin as a public randomness source. Cryptology ePrint Archive, Report 2015/1015 (2015), http://eprint.iacr.org/2015/1015
12. Buldas, A., Laanoja, R., Truu, A.: Efficient quantum-immune keyless signatures with identity. Cryptology ePrint Archive, Report 2014/321 (2014), http://eprint.iacr.org/2014/321
13. Buldas, A., Laud, P., Saarepera, M., Willemson, J.: Universally composable time-stamping schemes with audit. In: Zhou, J., Lopez, J., Deng, R.H., Bao, F. (eds.) ISC 2005: 8th International Conference on Information Security. Lecture Notes in Computer Science, vol. 3650, pp. 359–373. Springer, Heidelberg, Germany, Singapore (Sep 20–23, 2005)
14. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science. pp. 136–145. IEEE Computer Society Press, Las Vegas, NV, USA (Oct 14–17, 2001). https://doi.org/10.1109/SFCS.2001.959888
15. Canetti, R.: Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239 (2003), http://eprint.iacr.org/2003/239
16. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: Vadhan, S.P. (ed.) TCC 2007: 4th Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 4392, pp. 61–85. Springer, Heidelberg, Germany, Amsterdam, The Netherlands (Feb 21–24, 2007). https://doi.org/10.1007/978-3-540-70936-7_4
17. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: Dwork, C. (ed.) Advances in Cryptology – CRYPTO 2006. Lecture Notes in Computer Science, vol. 4117, pp. 78–96. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2006). https://doi.org/10.1007/11818175_5
18. Chaum, D.: Blind signature systems. U.S. Patent #4,759,063 (Jul 1988)
19. Coretti, S., Garay, J.A., Hirt, M., Zikas, V.: Constant-round asynchronous multi-party computation based on one-way functions. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016, Part II. Lecture Notes in Computer Science, vol. 10032, pp. 998–1021. Springer, Heidelberg, Germany, Hanoi, Vietnam (Dec 4–8, 2016). https://doi.org/10.1007/978-3-662-53890-6_33
20. De Santis, A., Micali, S., Persiano, G.: Non-interactive zero-knowledge proof systems. In: Pomerance, C. (ed.) Advances in Cryptology – CRYPTO'87. Lecture Notes in Computer Science, vol. 293, pp. 52–72. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 1988). https://doi.org/10.1007/3-540-48184-2_5
21. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. In: 30th Annual ACM Symposium on Theory of Computing. pp. 409–418. ACM Press, Dallas, TX, USA (May 23–26, 1998). https://doi.org/10.1145/276698.276853

22. Eng, T., Okamoto, T.: Single-term divisible electronic coins. In: Santis, A.D. (ed.) Advances in Cryptology – EUROCRYPT'94. Lecture Notes in Computer Science, vol. 950, pp. 306–319. Springer, Heidelberg, Germany, Perugia, Italy (May 9–12, 1995). https://doi.org/10.1007/BFb0053446
23. Garay, J.A., Jakobsson, M.: Timed release of standard digital signatures. In: Blaze, M. (ed.) FC 2002: 6th International Conference on Financial Cryptography. Lecture Notes in Computer Science, vol. 2357, pp. 168–182. Springer, Heidelberg, Germany, Southampton, Bermuda (Mar 11–14, 2003)
24. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part II. Lecture Notes in Computer Science, vol. 9057, pp. 281–310. Springer, Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015). https://doi.org/10.1007/978-3-662-46803-6_10
25. Haber, S., Stornetta, W.S.: How to time-stamp a digital document. Journal of Cryptology **3**(2), 99–111 (Jan 1991). https://doi.org/10.1007/BF00196791
26. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Sahai, A. (ed.) TCC 2013: 10th Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 7785, pp. 477–498. Springer, Heidelberg, Germany, Tokyo, Japan (Mar 3–6, 2013). https://doi.org/10.1007/978-3-642-36594-2_27
27. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017, Part I. Lecture Notes in Computer Science, vol. 10401, pp. 357–388. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017). https://doi.org/10.1007/978-3-319-63688-7_12
28. Lam, T., Tan, C.C., Chang, Y.J., Liu, J.C.: Timed zero-knowledge proof (tzkp) protocol. In: IEEE Real-Time and Embedded Technology and Application Symposium (2007)
29. Landerreche, E., Stevens, M., Schaffner, C.: Non-interactive cryptographic timestamping based on verifiable delay functions. Cryptology ePrint Archive, Report 2019/197 (2019), https://eprint.iacr.org/2019/197
30. Liu, J., Garcia, F., Ryan, M.: Time-release protocol from bitcoin and witness encryption for sat. IACR Cryptology ePrint Archive (2015)
31. Liu, J., Jager, T., Kakvi, S.A., Warinschi, B.: How to build time-lock encryption. Designs, Codes and Cryptography (2018)
32. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
33. Pass, R., Seeman, L., shelat, a.: Analysis of the blockchain protocol in asynchronous networks. In: Coron, J., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017, Part II. Lecture Notes in Computer Science, vol. 10211, pp. 643–673. Springer, Heidelberg, Germany, Paris, France (Apr 30 – May 4, 2017). https://doi.org/10.1007/978-3-319-56614-6_22
34. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
35. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**, 1–32 (2014)
36. Zhang, Y., Xu, C., Li, H., Yang, H., Shen, X.S.: Chronos: Secure and accurate time-stamping scheme for digital files via blockchain. In: 2019 IEEE International Conference on Communications, ICC 2019, Shanghai, China, May 20-24, 2019. pp. 1–6. IEEE (2019). https://doi.org/10.1109/ICC.2019.8762071, https://doi.org/10.1109/ICC.2019.8762071