How to Build Time-Lock Encryption

Tibor Jager

Horst Görtz Institute for IT Security, Ruhr-University Bochum
tibor.jager@rub.de

Abstract. *Time-lock encryption* is a method to encrypt a message such that it can only be decrypted after a certain deadline has passed. A computationally powerful adversary should not be able to learn the message before the deadline. However, even receivers with relatively weak computational resources should *immediately* be able to decrypt after the deadline, without any interaction with the sender, other receivers, or a trusted third party.

Our idea is to realize this strong notion of secure encryption is to make the additional, *very realistic* assumption that intermediate results of an iterative, public, large-scale computation are publicly available — like the computations performed by users of the popular cryptocurrency *Bitcoin*. We use these computations as a "computational reference clock", which mimics a physical clock in a computational setting, and show how the computations performed by the reference clock can be "reused" to build secure time-lock encryption. A nice feature of this approach is that it can be based on a public computation which is performed "anyway", independently of the time-lock encryption scheme.

We provide the first formal definitions of computational reference clocks and time-lock encryption, and give a proof-of-concept construction which combines a computational reference clock with *witness encryption* (Garg *et al.*, STOC 2013). We also explain how to construct a computational reference clock based on Bitcoins.

1 Introduction

Time-lock encryption. Alice has a document that she wants to make public in, say, a couple of days, but she is not willing to hand it out to anybody before this deadline. Therefore she puts the document into a box and attaches a *time-lock*. The lock keeps the box securely sealed, and thus the document confidential, for the determined period of time. It will unlock *automatically* when the deadline has passed, which makes it possible for everyone to access the document easily, without any further interaction with Alice. *Time-lock encryption* is a digital equivalent of such time-locked boxes. It allows to encrypt data for a period of time, up to a certain deadline, such that even a computationally powerful adversary is not able to learn any non-trivial information about the data before the deadline. However, when the time is over, even parties with relatively weak computational resources should immediately be able to decrypt *easily*.

Essentially, time lock encryption allows to send a message "into the future". The key novelty of time-lock encryption is that it achieves the following properties *simultaneously*:

1. Decryption is *non-interactive*. That is, the sender Alice is not required to be available for decryption.
2. Time lock encryption does not rely on trusted third parties. Thus, the sender is not required to trust any (set of) third parties to keep decryption keys (or shares of decryption keys) secret until the deadline has passed.
3. Parties interested in decrypting a ciphertext are not forced to perform expensive computations until decryption succeeds. This means that a party which simply waits till the decryption deadline has passed will be able to decrypt the ciphertext at about the same time as another party who attempts to decrypt the ciphertext earlier by performing a large (but reasonably bounded) number of computations. Thus, all reasonably bounded parties will be able to decrypt a ciphertext at essentially the same time, regardless of their computational resources.

These fact that these features are achieved simultaneously makes time-lock encryption a fascinating primitive, which enables applications that seem impossible to achieve with classical encryption schemes.

Efficient decryption and no trusted third parties. Time-lock encryption is related to *timed-release encryption*, investigated by Rivest, Shamir, and Wagner [33] and many follow-up works [17, 11, 10, 14, 15, 36]. It is a useful tool for applications like secure auctions, mortgage payment, key escrow [33], or fair multiparty computation [11, 1]. The main difference is that timed-release encryption suffers from the following shortcomings.

One line of research [33, 17, 11, 10, 15] realizes timed-release encryption by assuming a *trusted* third party (TTP), which reveals decryption keys at the right time. Therefore security relies crucially on the assumption that the TTP is trustworthy. In particular, it must not use its ability to allow decryption of ciphertexts earlier than desired by the sender in any malicious way, for instance by revealing a decryption key before the deadline. Some constructions share the decryption key among several (trusted) parties. However, a collusion of some (or in the worst case all) parties against the encrypter will always be able to decrypt before the deadline in the trusted third party approach. For some applications this may not be desirable for the encrypter.

The other line of research [33, 11, 36] considers constructions that require the receiver of a ciphertext to perform a feasible, but computationally *expensive* search for a decryption key. This puts a considerable computational overhead on the receiver. Suppose a sender wants that a ciphertext must only be decrypted in, say, ten days (a relatively short period of time). In order to make this work, the sender would have to be able to predict the computational resources available to the receiver relatively exactly, and the receiver would have to dedicate all these computational resources for ten days to the search, in order to be able to decrypt immediately after the deadline. This is not very practical. Particularly challenging in this setting are situations where the ciphertext is made public and there are many different receivers. For instance, suppose the ciphertext is posted on the Internet, and *everybody* should be able to decrypt immediately after the deadline. Then there are many receivers with different computational resources. The sender may not even know all of them. It seems impossible to encrypt with any known timed-release encryption scheme in a way, such that all receivers are able to decrypt *at*

the same time, unless one relies on trusted third parties. Moreover, even if we assume that all parties are able to solve the computational problem within (relatively exactly) the same time, note that this would still force that all parties to begin their computations at (relatively exactly) the same time, and that all parties are able to dedicate resources to the search for the decryption key. We think it an interesting theoretical question in its own to ask if it is possible to avoid this.

Contributions. We construct encryption schemes where a sender is able to encrypt a ciphertext, such that not even a computationally powerful (but reasonably bounded) adversary is able to learn any non-trivial information about the message before the deadline. Once the deadline has passed, even receivers with relatively limited computational resources are immediately able to decrypt. Decryption is *non-interactive*, in the sense that there is no communication between the sender and receivers except for the initial, unidirectional transmission of the ciphertext, or among receivers. We call encryption schemes with these properties *time-lock encryption* schemes.

Essentially, instead of assuming trusted third parties, we show how to "emulate" real-world time in a computational model, by basing security on a large-scale, iterative, public computation, which may be performed independently and for a completely different purpose than realizing time-lock encryption. We show how this effort can be "reused" to realize time-lock encryption.

Computational reference clocks. A first challenge in constructing time-lock encryption is to find a reasonable equivalent of *real-world time* in a computational model. Real-world time is usually determined by some *physical* reference, like the current state of atomic reference clocks. We do not see any reasonable way to mimic this notion of time in a computational model without trusted third parties.

Our main idea is to use the current state of an iterative, public computation as what we call a *computational reference clock*. The abstract notion of computational reference clocks stems from the concrete idea of using the popular digital cryptocurrency *Bitcoin* [32] as a reference clock, therefore let us explain the idea with this concrete example. The Bitcoin system performs an iterative, very large-scale, public computation, where so-called *miners* are contributing significant computational resources to the gradual extension of the *Bitcoin block chain*. Essentially, this block chain contains a sequence of hash values $B_1, \ldots, B_\tau$ that satisfy certain conditions.[1] These conditions determine the *difficulty* of finding new blocks in the chain. The Bitcoin system frequently adjusts the difficulty, depending on the computational resources currently contributing to the Bitcoin network, such that about every 10 minutes a new block $B_{\tau+1}$ is appended to the chain. Thus, the block chain can serve as a reference clock, where the current length τ of the chain tells the current "time", and there are about 10 minutes between each "clock tick".

Witness encryption. In order to be able to sketch our construction of time-lock encryption from computational reference clocks, let us briefly recap *witness encryption* [24]. A witness encryption scheme is associated with an NP-relation R (cf. Definition 1). For

[1] Section 3.1 contains a more detailed background on Bitcoins.

$(x, w) \in R$ we say that x is a "statement" and w is a "witness". A witness encryption scheme for relation R allows to encrypt a message m with respect to statement x as $c \xleftarrow{\$} \mathsf{Enc}_R(x, m)$. Any witness w which satisfies $(x, w) \in R$ can be used to decrypt this ciphertext c as $m = \mathsf{Dec}_R(c, w)$. Intuitively, one may think of a statement x as a "public key", such that any witness w with $(x, w) \in R$ can be used as a corresponding "secret key". A secure witness encryption scheme essentially guarantees that no adversary is able to learn any non-trivial information about a message encrypted for statement x, unless it already "knows" a witness w for $(x, w) \in R$. Witness encryption schemes with this property are called *extractable* [27, 7, 13].

Time-lock encryption. The key idea behind our construction is to combine a computational reference clock with witness encryption. For this introduction, let us consider time-lock encryption based on Bitcoins as one specific instantiation of a reference clock (we will consider more general constructions in the body of the paper). We define an NP-relation R such that

1. For $x \in \mathbb{N}$, statements have the form 1^x, that is, x in unary representation.
2. Any valid Bitcoin block chain $w = (B_1, \ldots, B_x)$ of length at least x is a witness for $(1^x, w) \in R$.

Let $(\mathsf{Enc}_R, \mathsf{Dec}_R)$ be a witness encryption scheme for this particular relation R. Suppose the current state of the Bitcoin block chain is $B_1, \ldots, B_\tau$. Then the block chain contains a witness w for $(1^x, w) \in R$ for all $x \leq \tau$. The Bitcoin block chain is *public*. Therefore everybody is immediately able to decrypt any ciphertext $c \xleftarrow{\$} \mathsf{Enc}_R(1^x, m)$ with $x \leq \tau$, by using the witness from the public block chain as the "decryption key".

Security of this construction. Let $c = \mathsf{Enc}_R(1^x, m)$ be a ciphertext with $x > \tau$. Under the assumption that the witness encryption scheme is secure, we will show that an adversary has only two possibilities to learn any non-trivial information about m.

1. The adversary waits until the public Bitcoin block chain has reached length x. Then the chain contains a witness w for $(1^x, w) \in R$, which immediately allows to decrypt. However, note that then not only the adversary, but also everybody else is able to decrypt, by reading w from the public Bitcoin block chain and computing $m = \mathsf{Dec}_R(c, w)$. Speaking figuratively, "the time-lock has opened".
2. The adversary tries to "put forward" the computational reference clock provided by the Bitcoin block chain, by computing the missing blocks $B_{\tau+1}, \ldots, B_x$ of the chain secretly on its own, *faster* than the public computation performed by the collection of all Bitcoin miners. Note that this means that the adversary would have to outperform the *huge* computational resources gathered in Bitcoin, which currently (May 2015) perform more than $350 \cdot 10^{15} \approx 2^{58}$ hash computations *per second*. Assuming that no adversary is able to perform this large amount of computation to learn the encrypted message earlier, the scheme is secure.
   A particularly interesting case is when the value of learning the message earlier than others is below the value of the computations an adversary would have to perform. For instance, in our Bitcoin-based instantiation, an adversary would earn Bitcoins for contributing its resources to the network. Then security is provided

simply by the fact that there is no incentive for the adversary to attack the time-lock encryption.

The above description is slightly simplified. The actual Bitcoin block chain (described in Section 3.1) and our construction (in Section 3.3) are more complex, but the underlying principle is the same. In particular, we will have to describe slightly more complex relations, because of the variable *difficulty* parameter in Bitcoin.

We stress that we do not have to put any form of *trust* in Bitcoin miners. The intermediate states of their computations can be completely public, and they do not have to store any secrets.

Further and future examples of computational reference clocks. It is possible to think of other instantiations of such computational reference clocks, like for instance other decentralized cryptocurrencies,[2] or possibly completely different iterative, public computations. We will therefore develop a more abstract view on computational reference clocks and time-lock encryption.

Our example construction of time lock encryption based on witness encryption for *all* NP-relations should rather be seen as a first proof-of-concept construction. We stress that we see the main contribution of this paper in the novel concept of extending the classical computational model with an iterative, public, large-scale computation, like the computations performed in Bitcoin, as a basis for realizing time-lock encryption.

We stress also that our application actually does not require the full strength of witness encryption for *all* NP-relations, a scheme for a certain, specific, clock-dependent relation would suffice. Note that constructing witness encryption schemes for specific relations should be much simpler than constructing witness encryption schemes for *all* NP-relations. For example, very efficient witness encryption schemes are known for relations describing languages which are compatible with hash proof systems [16]. Therefore we envision future constructions of alternate cryptocurrencies (or other types of computational reference clocks), possibly based on more algebraic computational hardness assumption giving rise to hash proof systems, that allow for truly practical instantiations of time-lock encryption via practical witness encryption for a suitable relation.

Related work and further applications of time-lock encryption. Timed-release encryption was introduced by Rivest, Shamir, and Wagner [33] and considered in many follow-up works, including [17, 10, 14, 15, 36]. Our approach is fundamentally different from theirs. In particular, we neither need *trusted* third parties, nor have a considerable computational overhead for decryption. *Time-lock puzzles* and *proofs of (sequential) work* [18, 33, 19, 30, 31] are computational problems that can not be solved without running a computer continuously for a certain time. In a sense, our computational reference clocks can be seen as algorithms that continuously and publicly solve publicly verifiable [31] instances of an infinite sequence of time-lock puzzles. Essentially, we show how this computational effort, performed independently of any time-lock encryption scheme, can be "reused" to construct time-lock encryption with efficient decryption.

[2] See `http://altcoins.com/` for an overview of Bitcoin alternatives.

A clever idea to use Bitcoin deposits to facilitate fairness in multiparty computation was presented by Andrychowicz *et al.* [3, 2, 1]. Essentially, the idea is that a party involved in a multiparty computation publishes a commitment along with a Bitcoin "deposit". If the party plays fair, by opening the commitment before a certain deadline, then it is able to reclaim the deposit. If the party does not open the commitment, then it is not able to reclaim. Thus, the loss of the deposit serves as a penalty for unfairness. See also Bentov and Kumaresan [9]. Time-lock encryption can be seen as a different approach to enforce fairness such a setting. It can be used to construct the first *timed commitment* scheme in the sense of Boneh and Naor [11] that does not require an inefficient forced opening. The commitment would "open" efficiently after a certain time, regardless of whether the committing party likes to play fair or not.[3]

Very recently, Azar, Goldwasser, and Park [4] construct "timed-delay" multi-party computation (MPC) protocols, where participating parties obtain the result of the computation only after a certain time, possibly some parties earlier than others. The construction described in [4] is based on time-lock puzzles where, essentially, for each participating party the result of the computation is "encrypted" with a time-lock puzzle, such that each party has to solve the computationally expensive but feasible puzzle before it is able to learn the result. This is similar to the timed-release schemes of [33, 11, 36], in particular it inherits the drawback of inefficient decryption and the assumption that all parties are able to solve the puzzles in about the same time. Our notion of time-lock encryption gives rise to a different approach for constructing timed-delay MPC, with efficient decryption and based on a public computational reference clock. The idea is to simply encrypt the output of the computed function with a time-lock encryption scheme.

Garay *et al.* [21] analyze the Bitcoin "backbone" protocol, and show how to realize Byzantine agreement on top of this protocol. Another recent work, which shows how to construct useful cryptographic primitives under the assumption that no adversary is able to outperform the huge computational resources of the collection of all Bitcoin miners, is due to Katz *et al.* [29], who show how to obtain secure computation and so-called pseudonymous authenticated communication from time-lock puzzles.

Formal computational models capturing "real-world time" were described for instance by Cathalo *et al.* [14], who gave a security model for timed-release encryption with a "time oracle" that sequentially releases specific information, and recently by Schwenk [34], who described a security model for time-based key exchange protocols. Both works [14, 34] may assume that the party implementing the clock is honest. In contrast, we will have to deal with adversaries that may want to "put the clock forward", therefore we need to model the computational hardness of doing so in our setting.

[3] If the time-lock encryption scheme is used directly as a commitment scheme, then it follows immediately from the security of the encryption scheme that the commitment is "hiding". However, for the commitment to be "binding" we yet have to prove (or assume) that a sender is not able to generate a "non-binding" ciphertext, which can efficiently be opened to more than one unique message. Alternatively, one can combine any binding commitment scheme with time-lock encryption, by committing to a message with the commitment scheme, and encrypting the opening of the commitment with time-lock encryption.

Witness encryption for all NP-relations was introduced by Garg *et al.* [24]. Known constructions [24, 23, 25, 7, 13] are based on multilinear maps [12, 22], or obfuscation [5, 6, 23]. We will use *extractable* witness encryption. This notion was introduced by Goldwasser *et al.* [27], who justify the assumption that the scheme of [24] is extractable by a proof in the generic group model. Further constructions of extractable witness encryption schemes [7, 13] are based on variants of extractability obfuscation [5, 6]. We also note that extractability is only required to make our security proof go through, but not for functionality of the system. It is therefore conceivable that our constructions remain secure when instantiated with a witness encryption scheme which is not extractable, albeit we do not know yet how to prove it formally secure.

Other approaches for time-lock encryption. Bitcoin-*incentivized* timed-release encryption appears in a software published at Github [35]. Even though their scheme is called "time-lock encryption", the approach and the functionality are completely different to ours. First and foremost, the scheme in [35] requires expensive exhaustive-search computations for *both* encryption and decryption, where encryption can be parallelized, but decryption (most likely) not. Moreover, their scheme essentially encrypts the message along with a secret key, which allows to retrieve Bitcoins published in a public deposit. A successful decrypter is able to collect the deposited coins, which serves as an additional incentive decrypter to decrypt the message (at least for the first successful decrypter). Therefore, in our terminology, the scheme from [35] is a *timed-release* encryption scheme in the classical sense of Rivest, Shamir, and Wagner [33], where decryption is additionally incentivized by a Bitcoin deposit, but *not* a time-lock encryption scheme in our sense, where ciphertexts essentially decrypt "automatically" (given the public computational reference clock) and the plaintext becomes publicly available.

Among many examples of timed-release encryption schemes in the sense of Rivest, Shamir, and Wagner [33], the survey in [28] (referring to [26]) describes the idea to combine "public keys", which are generated from a public seed with a pseudorandom number generator, with a special-purpose cryptocurrency where "mining" of coins corresponds to computing the corresponding secret keys. As already mentioned in [26], we do not know how to make this idea work. The reason is that cryptocurrencies and time-lock encryption inherently require that progress in the underlying computations can be made *only sequentially* (in [26] this is called "progress-free"). For example, Bitcoin achieves sequentiality by making each block in the block chain dependent on all previous blocks. Thus, the approach of [26] requires the existence of "progress-free sequences of public keys". We are not aware of any proof-of-concept construction of such sequences, and it is unclear whether these objects exists. Moreover, in order to encrypt with the approach described in [28, 26] for "time τ", one would already need to know at least the τ-th public key in the sequence, which clearly contradicts sequentiality.

2 Time-Lock Encryption

In this section we will first formally define secure time-lock encryption, computational reference clocks, and their associated relations. Then we describe a generic construction of time-lock encryption from witness encryption.

On formally defining time-lock encryption. Defining security of time-lock encryption schemes will require a slightly more fine-grained notion of "computational hardness" than most other cryptographic primitives. We will consider two Turing machines $\mathcal{A}$ and $\mathcal{C}$, which both attempt to solve the same *polynomial-time solvable* computational problem. We will assume that $\mathcal{C}$ has access to *significantly* more computational resources than $\mathcal{A}$, such that it is infeasible for $\mathcal{A}$ to solve the problem faster than $\mathcal{C}$. Clearly, modeling both $\mathcal{A}$ and $\mathcal{C}$ simply as polynomial-time algorithms is not useful here, because then both will able to solve the computational problem easily. We will overcome this by making the *concrete* bounds on the running times of algorithms $\mathcal{A}$ and $\mathcal{C}$ explicit.

A remark on nomenclature. We will have to deal with two different notions of "time". First, the running time of algorithms, usually measured in the number of computational steps an algorithm performs. Second, our computational equivalent of physical time, measured in some abstract discrete time unit. To avoid ambiguity, we will use the word "time" only for our computational equivalent of physical time. Rather than specifying the "running time" of an algorithm, we will specify the "number of operations" performed by the algorithm, assuming that all algorithms are executed on universal Turing machines with identical instruction sets. For example, we will write "algorithm $\mathcal{A}$ performs t operations" instead of "algorithm $\mathcal{A}$ runs in time t".

2.1 Definitions

Definition 1. *Let R be a relation. We say that R is an* NP-relation*, if there exists a deterministic polynomial-time (in $|x|$) algorithm that, on input (x, w), outputs 1 if and only if $(x, w) \in R$.*

Computational reference clocks. The concepts of *computational reference clocks* and their *associated relations* will be necessary to define time-lock encryption.

Definition 2. *A* computational reference clock *is a stateful probabilistic machine $\mathcal{C}$ that outputs an infinite sequence $w_1, w_2, \ldots$ in the following way. The initial state of $\mathcal{C}$ is w_0. On input a symbol $\perp$, it runs a probabilistic algorithm $f_{\mathcal{C}}$ which computes $w_\tau = f_{\mathcal{C}}(w_{\tau-1})$ and outputs w_τ.*

We write $w_\tau \xleftarrow{\$} \mathcal{C}(\tau)$ for $\tau \in \mathbb{N}$ to abbreviate the process of executing the clock τ times in a row on input $\perp$, starting from initial state w_0, and outputting the state w_τ of $\mathcal{C}$ after τ executions.

Intuition for Definition 2. The intuition behind this definition is that the machine $\mathcal{C}$ performs an iterative, public computation, which iteratively computes $f_{\mathcal{C}}$. $\mathcal{C}$ outputs its complete internal state after each execution, therefore no secret keys or other secret values can be hidden inside $\mathcal{C}$. Algorithm $f_{\mathcal{C}}$ is public, too.

When executed for the τ-th time, the machine responds with the current state of the computation at "time" τ. Intuitively, w_τ serves as a "witness" that the current time is "at least τ".

Definition 2 will be useful for the construction of secure time-lock encryption, whenever it is computationally very hard, but not completely infeasible, to compute

w_τ from $w_{\tau-1}$. Think of $\mathcal{C}$ as a very fast machine that works on solving an infinite sequence of computational puzzles. Jumping slightly ahead, we will later instantiate $\mathcal{C}$ with the collection of all Bitcoin miners that contribute to expanding the publicly known Bitcoin block chain. When executed for the τ-th time, the machine returns the block chain of length τ.

Definition 3. *We say that relation R is* associated to *$\mathcal{C}$, if R is an NP-relation, and for all $x \leq \tau$ holds that*

$$\Pr\left[(1^x, w_\tau) \in R : w_\tau \xleftarrow{\$} \mathcal{C}(\tau)\right] = 1$$

Intuition for Definition 3. The purpose of the relation is to describe which values w_τ are acceptable as a "witness for time τ". Note that it makes sense to accept a witness w_τ with "for time τ" also as a witness for any "earlier time" x with $x \leq \tau$. Hence we require that $(1^x, w_\tau) \in R$ holds for all $x \leq \tau$.

Definition 4. *We say that a computational reference clock $\mathcal{C}$ is* (R, t, ϵ)-secure*, if for all $\tau \in \mathbb{N}$ and all adversaries $\mathcal{A}$ that have access to computational reference clock $\mathcal{C}$ and perform at most t operations holds that* $\Pr[\mathsf{Exp}^{\mathcal{C},R,\mathcal{A}}_{\mathsf{clk}}(1^\lambda) = 1] \leq \epsilon$*, where $\mathsf{Exp}^{\mathcal{C},R,\mathcal{A}}_{\mathsf{clk}}$ is the following experiment.*

$$\frac{\mathsf{Exp}^{\mathcal{C},R,\mathcal{A}}_{\mathsf{clk}}(1^\lambda):}{(1^\tau, w_\tau) \xleftarrow{\$} \mathcal{A}^{\mathcal{C}}(1^\lambda)}$$
Return $(1^\tau, w_\tau) \in R$

$\mathcal{A}$ may query $\mathcal{C}$ at most $\tau(\lambda) - 1$ times.

Intuition for Definition 4. Definition 4 essentially requires a lower bound on the number of operations that have to be performed in order to perform the same computations as $\mathcal{C}$. For certain choices of $\mathcal{C}$ and R it will be reasonable to assume that breaking the security of $(\mathcal{C}, R)$ with success probability at least ϵ requires to perform at least t' operations, where t' may be a (large) polynomial with $t < t'$. This will be useful when we consider a particular instantiation of $\mathcal{C}$ that gathers huge computational resources, such that it is is able to perform t' operations in reasonable time, and adversaries for which performing t operations *within the same time* is infeasible.

For concreteness, think of $\mathcal{C}$ as the collection of all Bitcoin miners, which are able to perform a very large number t' of operations within a certain period of time. We will consider adversaries which will only be able to perform $t \ll t'$ operations within the same time.

Remark 1. In our Bitcoin-based instantiation of a computational reference clock $\mathcal{C}$, the adversary will also be able to modify the state of $\mathcal{C}$ to a certain degree (for instance, by executing Bitcoin transactions). The above definition can easily be extended to capture this, see Appendix 2.3 for details.

Time-lock encryption. Based on the notion of computational reference clocks, we can now define time-lock encryption schemes and their security.

Definition 5. *A* time-lock encryption scheme *for computational reference clock $\mathcal{C}$ with message space $\mathcal{M}$ consists of two polynomial-time algorithms* $(\mathsf{Enc}_{\mathrm{TL}}, \mathsf{Dec}_{\mathrm{TL}})$.

Encryption. *The encryption algorithm $c \xleftarrow{\$} \mathsf{Enc}_{\mathrm{TL}}(1^\lambda, \tau, m)$ takes as input the security parameter λ, an integer $\tau \in \mathbb{N}$, and a message $m \in \mathcal{M}$. It computes and outputs a ciphertext c.*

Decryption. *The decryption algorithm $\mathsf{Dec}_{\mathrm{TL}}(w, c)$ takes as input $w \in \{0,1\}^*$ and ciphertext c, and outputs a message $m \in \mathcal{M}$ or a distinguished error symbol $\perp$.*

Correctness. *For correctness we require that*

$$\Pr\left[m = m' : \begin{array}{l} c \xleftarrow{\$} \mathsf{Enc}_{\mathrm{TL}}(1^\lambda, \tau_{\mathsf{dec}}, m) \\ w_\tau \xleftarrow{\$} \mathcal{C}(\tau) \\ m' := \mathsf{Dec}_{\mathrm{TL}}(w_\tau, c) \end{array}\right] = 1$$

for all $\lambda \in \mathbb{N}$, all $\tau \in \mathbb{N}$ with $\tau \geq \tau_{\mathsf{dec}}$, and all $m \in \mathcal{M}$.

Definition 6. *We say that a time-lock encryption scheme $\Pi = (\mathsf{Enc}_{\mathrm{TL}}, \mathsf{Dec}_{\mathrm{TL}})$ for $\mathcal{C}$ is (t, ϵ)-*secure*, if for all adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ performing at most t operations holds that $\Pr[\mathsf{Exp}_{\mathsf{tl}}^{\Pi,\mathcal{A}}(1^\lambda) = 1] \leq \epsilon$, where $\mathsf{Exp}_{\mathsf{tl}}^{\Pi,\mathcal{A}}(1^\lambda)$ is the following experiment.*

$$\begin{array}{l} \underline{\mathsf{Exp}_{\mathsf{tl}}^{\Pi,\mathcal{A}}(1^\lambda):} \\ (m_0, m_1, 1^\tau, st) \xleftarrow{\$} \mathcal{A}_0^{\mathcal{C}}(1^\lambda); \;\; b \xleftarrow{\$} \{0,1\} \\ c \xleftarrow{\$} \mathsf{Enc}_{\mathrm{TL}}(1^\lambda, \tau, m_b); \;\; b' \xleftarrow{\$} \mathcal{A}_1^{\mathcal{C}}(1^\lambda, c, st) \\ \textit{Return } b = b' \end{array}$$

We require that $|m_0| = |m_1|$. The adversary is allowed to make at most $\tau - 1$ queries to $\mathcal{C}$ in total.

Intuition for Definition 6. The intuition behind this security definition is essentially that no adversary should be able to distinguish an encryption of m_0 from an encryption of m_1 *before* $\mathcal{C}$ has output w with $(1^\tau, w) \in R$. At a first glance it might appear that security in this sense is impossible to achieve whenever $\mathcal{C}$ is a polynomial-time algorithm, because the adversary could simply perform the same computations as $\mathcal{C}$. However, recall that we put an explicit bound t on the number of operations that $\mathcal{A}$ may perform. This makes this definition useful when it is reasonable to assume that the number of operations t that can be performed within a certain time by the adversary is much smaller than the (also polynomially bounded, but much larger) number of operations t' required to compute w with $(1^\tau, w) \in R$.

Remark 2. Note that we require the adversary $\mathcal{A}_0$ in Definition 6 to output τ in unary. This implicitly forces $\mathcal{A}$ to output $\tau \in \mathbb{N}$ with is not too large (of exponential size, for instance), which would not be achievable by our construction described below. We do not see this as a restriction, because a time-lock encryption scheme that allows a

polynomially-bounded number of "time slots" appears sufficient for all conceivable applications.

Note also that $(m_0, m_1, 1^\tau)$ may depend on the responses of clock $\mathcal{C}$. One may define weaker security notions, however, we think that this form of adaptive security is the "right" one for time-lock encryption.

2.2 Constructing Time-Lock Encryption from Witness Encryption

Witness encryption. Witness encryption schemes were introduced by Garg *et al.* [24], and extended to *extractable* witness encryption in [27, 7, 13]. The following definition is based on [7].

Definition 7. *A* witness encryption *scheme for relation R with message space $\mathcal{M} \subseteq \{0,1\}^*$ consists of two polynomial-time algorithms $(\mathsf{Enc}_R, \mathsf{Dec}_R)$.*

Encryption. *The encryption algorithm $c \xleftarrow{\$} \mathsf{Enc}_R(1^\lambda, x, m)$ takes as input the security parameter λ, a string $x \in \{0,1\}^*$, and a message $m \in \mathcal{M}$. It outputs ciphertext c.*

Decryption. *The decryption algorithm $\mathsf{Dec}_R(w, c)$ takes as input $w \in \{0,1\}^*$ and ciphertext c, and outputs a message $m \in \mathcal{M}$ or a distinguished error symbol $\perp$.*

Correctness. *For correctness we require that*

$$\Pr\left[m = m' : c \xleftarrow{\$} \mathsf{Enc}_R(1^\lambda, x, m), m' = \mathsf{Dec}_R(w, c)\right] = 1$$

for all $\lambda \in \mathbb{N}$, all $(x, w) \in R$, and all $m \in \mathcal{M}$.

Following [27, 7, 13], we will say that a witness encryption scheme is secure, if the only way for an adversary to learn any non-trivial information about a message encrypted for statement x is to "know" a witness w for $(x, w) \in R$. This is formalized in the following definition.

Definition 8. *We say that a witness encryption scheme $\Pi = (\mathsf{Enc}_R, \mathsf{Dec}_R)$ for relation R is $(R, t_{\mathsf{we}}, t_{\mathsf{ext}}, \epsilon)$-*secure*, if for all adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ performing at most t_{we} operations there is an extractor algorithm $\mathcal{E}$ that performs at most t_{ext} operations such that $\Pr[\mathsf{Exp}^{\Pi,\mathcal{A}}_{\mathsf{we}}(1^\lambda) = 1] \leq \epsilon$, where $\mathsf{Exp}^{\Pi,\mathcal{A}}_{\mathsf{we}}$ is the following experiment.*

$$\begin{array}{l}\underline{\mathsf{Exp}^{\Pi,\mathcal{A}}_{\mathsf{we}}(1^\lambda):}\\ (x, m_0, m_1, st) \xleftarrow{\$} \mathcal{A}_0(1^\lambda);\ \ b \xleftarrow{\$} \{0,1\};\ \ c \xleftarrow{\$} \mathsf{Enc}_R(1^\lambda, x, m_b)\\ b' \xleftarrow{\$} \mathcal{A}_1(1^\lambda, st, c);\ \ w \xleftarrow{\$} \mathcal{E}(1^\lambda, x, m_0, m_1, st, c)\\ \textit{Return } (b = b' \wedge (x, w) \notin R)\end{array}$$

We require that $|m_0| = |m_1|$.

Time-lock encryption from witness encryption. Let $\mathcal{C}$ be a computational reference clock and let R be an NP-relation, such that R is associated to $\mathcal{C}$. Let $(\mathsf{Enc}_R, \mathsf{Dec}_R)$ be a witness encryption scheme for R. Define algorithms $(\mathsf{Enc}_{\mathrm{TL}}, \mathsf{Dec}_{\mathrm{TL}})$ of a time-lock encryption scheme as

$$\mathsf{Enc}_{\mathrm{TL}}(1^\lambda, \tau, m) := \mathsf{Enc}_R(1^\lambda, 1^\tau, m) \quad \text{and} \quad \mathsf{Dec}_{\mathrm{TL}}(w, c) := \mathsf{Dec}_R(w, c) \qquad (1)$$

Let us first prove correctness. We have to show that

$$\Pr\left[m = m' : \begin{array}{c} c \xleftarrow{\$} \mathsf{Enc}_R(1^\lambda, 1^{\tau_{\mathsf{dec}}}, m) \\ w_\tau \xleftarrow{\$} \mathcal{C}(\tau) \\ m' := \mathsf{Dec}_R(w_\tau, c) \end{array}\right] = 1$$

holds for all $\lambda \in \mathbb{N}$, all $\tau \geq \tau_{\mathsf{dec}}$, and all $m \in \{0,1\}$. The correctness of the witness encryption scheme guarantees that

$$\Pr\left[m = m' : c \xleftarrow{\$} \mathsf{Enc}_R(1^\lambda, 1^{\tau_{\mathsf{dec}}}, m), m' = \mathsf{Dec}_R(w, c)\right] = 1$$

for all $\lambda \in \mathbb{N}$, all w with $(1^{\tau_{\mathsf{dec}}}, w) \in R$, and all $m \in \mathcal{M}$. Thus, it remains only to show that

$$\Pr[(1^{\tau_{\mathsf{dec}}}, w_\tau) \in R : w_\tau \xleftarrow{\$} \mathcal{C}(\tau)] = 1$$

holds for all $\tau \geq \tau_{\mathsf{dec}}$. Since R is associated to $\mathcal{C}$, this follows by Definition 2.

Theorem 1. *Algorithms* $(\mathsf{Enc}_{\mathrm{TL}}, \mathsf{Dec}_{\mathrm{TL}})$ *form a* $(t_{\mathsf{tl}}, \epsilon_{\mathsf{tl}})$*-secure time-lock encryption scheme for computational reference clock* $\mathcal{C}$ *with*

$$\epsilon_{\mathsf{tl}} \leq \epsilon_{\mathsf{clk}} + \epsilon_{\mathsf{we}}$$

provided that the witness encryption scheme is $(R, t_{\mathsf{tl}} + t', t_{\mathsf{ext}}, \epsilon_{\mathsf{we}})$*-secure, and reference clock* $\mathcal{C}$ *is* $(R, 2 \cdot t_{\mathsf{tl}} + t_{\mathsf{ext}} + t'', \epsilon_{\mathsf{clk}})$*-secure with respect to Definition 4, where* t' *and* t'' *are small, the concrete values will become clear in the proof.*

Intuition for the proof of Theorem 1. We reduce security of the time-lock encryption scheme to the security of the underlying witness encryption scheme and the security of the computational reference clock. Intuitively, we want to use the existence of an extractor $\mathcal{E}$, which is guaranteed by the security of the witness encryption scheme, to turn any adversary $\mathcal{A}_{\mathsf{tl}}$ on the time-lock encryption scheme into an adversary $\mathcal{A}_{\mathsf{clk}}$ against the security of the computational reference clock. This contradicts the assumption about the security of $\mathcal{C}$. Adversary $\mathcal{A}_{\mathsf{clk}}$ essentially works as follows.

1. First, $\mathcal{A}_{\mathsf{clk}}$ constructs a witness encryption adversary $\mathcal{A}_{\mathsf{we}}$ from time-lock encryption adversary $\mathcal{A}_{\mathsf{tl}}$.
2. Then it runs the witness encryption extractor $\mathcal{E}$ for $\mathcal{A}_{\mathsf{we}}$, to extract a witness w_τ from $\mathcal{A}_{\mathsf{tl}}$.
3. Finally, it uses the witness w_τ extracted from $\mathcal{A}_{\mathsf{we}}$ to break the security of $\mathcal{C}$.

We will then bound the success probability of $\mathcal{A}_{\mathsf{clk}}$, and use this to derive a bound on the success probability of $\mathcal{A}_{\mathsf{tl}}$.

Note that in Step 1 we have to turn the algorithm $\mathcal{A}_{\mathsf{tl}}$ into an algorithm $\mathcal{A}_{\mathsf{we}}$, such that we can run extractor $\mathcal{E}$ on $\mathcal{A}_{\mathsf{we}}$. However, $\mathcal{A}_{\mathsf{tl}}$ is an oracle machine, which expects access to an oracle $\mathcal{C}$ and the witness encryption extractor $\mathcal{E}$ is not guaranteed to work for oracle machines. We show that it is possible to overcome this issue by letting $\mathcal{E}^{\mathcal{C}}$ query its oracle on all required values τ in advance, and then hard-coding the responses into adversary $\mathcal{A}_{\mathsf{we}}$.

Proof. Assume towards contradiction that there exists an adversary $\mathcal{A}_{\mathsf{tl}}$ which breaks the security of the time-lock encryption scheme by performing at most t_{tl} operations, but with success probability

$$\epsilon_{\mathsf{tl}} > \epsilon_{\mathsf{clk}} + \epsilon_{\mathsf{we}}$$

Consider the following adversary $\mathcal{A}_{\mathsf{clk}}$ against $\mathcal{C}$, which runs $\mathcal{A}_{\mathsf{tl}}$ as a subroutine.

1. $\mathcal{A}_{\mathsf{clk}}^{\mathcal{C}}$ runs $(m_0, m_1, \tau, st) \xleftarrow{\$} \mathcal{A}_{\mathsf{tl}0}^{\mathcal{C}}(1^\lambda)$ by relaying all oracle queries and the responses between $\mathcal{A}_{\mathsf{tl}0}^{\mathcal{C}}$ and $\mathcal{C}$. Let cnt denote the number of oracle queries issued by $\mathcal{A}_{\mathsf{tl}0}$. Then $\mathcal{A}_{\mathsf{clk}}^{\mathcal{C}}$ defines $\mathcal{A}_{\mathsf{we}0}$ as the algorithm which outputs $(1^\tau, m_0, m_1, st)$ on input 1^λ. Clearly, $\mathcal{A}_{\mathsf{we}0}$ does not issue any queries to $\mathcal{C}$.
2. Then $\mathcal{A}_{\mathsf{clk}}^{\mathcal{C}}$ continues to query $\mathcal{C}$ further $\tau - 1 - \mathsf{cnt}$ times, until $\mathcal{C}$ was queried exactly $\tau - 1$ times in total (including Step 1). For $i \in \{\mathsf{cnt}+1, \ldots, \tau-1\}$ let w_i denote the i-th response of $\mathcal{C}$.
3. $\mathcal{A}_{\mathsf{clk}}^{\mathcal{C}}$ creates a procedure $\mathcal{C}_{\mathsf{sim}}$ which takes as input a symbol $\perp$ and outputs $w_{\mathsf{cnt}+i}$ when queried for the i-th time. Then $\mathcal{A}_{\mathsf{clk}}^{\mathcal{C}}$ defines $\mathcal{A}_{\mathsf{we}1} := \mathcal{A}_{\mathsf{tl}1}^{\mathcal{C}_{\mathsf{sim}}}$ and $\mathcal{A}_{\mathsf{we}} := (\mathcal{A}_{\mathsf{we}0}, \mathcal{A}_{\mathsf{we}1})$. Note that $\mathcal{A}_{\mathsf{we}1}$ does not issue any oracle queries, because it is provided with procedure $\mathcal{C}_{\mathsf{sim}}$ instead of oracle $\mathcal{C}$. Note also that procedure $\mathcal{C}_{\mathsf{sim}}$ is able to simulate $\mathcal{C}$ perfectly for all at most $\tau - 1 - \mathsf{cnt}$ queries issued by $\mathcal{A}_{\mathsf{tl}1}$.
4. Then $\mathcal{A}_{\mathsf{clk}}$ computes $c \xleftarrow{\$} \mathsf{Enc}_R(1^\lambda, 1^\tau, m_b)$ for uniformly random $b \xleftarrow{\$} \{0,1\}$.
5. Finally, $\mathcal{A}_{\mathsf{clk}}^{\mathcal{C}}$ runs the extractor $w \xleftarrow{\$} \mathcal{E}(1^\tau, m_0, m_1, st, c)$ for $\mathcal{A}_{\mathsf{we}}$, and returns whatever $\mathcal{E}$ returns.
6. (Only for the analysis: $\mathcal{A}_{\mathsf{clk}}^{\mathcal{C}}$ executes $b' \xleftarrow{\$} \mathcal{A}_{\mathsf{we}1}(1^\lambda, st, c)$.)

The rigorous analysis of the success probability of this adversary $\mathcal{A}_{\mathsf{clk}}$ requires to add the last step to $\mathcal{A}_{\mathsf{clk}}$. We stress that this last step is never executed by $\mathcal{A}_{\mathsf{clk}}$, we include it only for the analysis of its success probability.

Overview. Note that $\Pr[\mathsf{Exp}_{\mathsf{clk}}^{\mathcal{C},R,\mathcal{A}_{\mathsf{clk}}}(1^\lambda) = 1]$ is the probability that $\mathcal{A}_{\mathsf{clk}}^{\mathcal{C}}$ outputs w with $(1^\tau, w) \in R$ in the experiment from Definition 4. Therefore we have

$$\begin{aligned}\Pr[\mathsf{Exp}_{\mathsf{clk}}^{\mathcal{C},R,\mathcal{A}_{\mathsf{clk}}}(1^\lambda) = 1] &= \Pr\left[(1^\tau, w) \in R\right] \\ &\geq \Pr\left[(1^\tau, w) \in R \wedge b = b'\right] \\ &= \Pr\left[b = b'\right] - \Pr\left[(1^\tau, w) \notin R \wedge b = b'\right] \qquad (2)\end{aligned}$$

From here, we will proceed in three steps.

1. We will first show that $\Pr\left[(1^\tau, w) \in R \wedge b = b'\right]$ in (2) is at most ϵ_{we}.
2. Then we will show that the term $\Pr\left[b = b'\right]$ in (2) is at least ϵ_{tl}.
3. Finally, we show that these bounds contradict our initial assumption that the success probability of $\mathcal{A}_{\mathsf{tl}}$ satisfies $\epsilon_{\mathsf{tl}} > \epsilon_{\mathsf{clk}} + \epsilon_{\mathsf{we}}$.

Step 1: an upper bound on $\Pr\left[(1^\tau, w) \in R \wedge b = b'\right]$. Note that $\mathcal{A}_{\mathsf{we}}$ is a witness encryption adversary, which performs at most $t_{\mathsf{tl}} + t'$ operations. Here t' is the number of operations required to execute procedure $\mathcal{A}_{\mathsf{we}0}$ once and procedure $\mathcal{C}_{\mathsf{sim}}$ at most $\tau - 1$ times. Note that both procedures simply output hard-coded values, therefore t' is relatively small.

By the $(R, t_{\mathsf{tl}} + t', t_{\mathsf{ext}}, \epsilon_{\mathsf{we}})$-security of the witness encryption scheme, there exists an extractor $\mathcal{E}$ that performs at most t_{ext} operations and outputs a witness w such that

$$\epsilon_{\mathsf{we}} \geq \Pr\left[b = b' \wedge (1^\tau, w) \notin R\right] \tag{3}$$

Step 2: a lower bound on $\Pr[b = b']$. Note that $\mathcal{A}_{\mathsf{clk}}$ simulates the time-lock encryption security experiment for $\mathcal{A}_{\mathsf{tl}}$ perfectly. In particular, it relays all oracle queries between $\mathcal{A}_{\mathsf{tl}0}$ and $\mathcal{C}$. The only difference is that instead of executing $\mathcal{A}_{\mathsf{tl}1}^{\mathcal{C}}$, it executes $\mathcal{A}_{\mathsf{we}1} = \mathcal{A}_{\mathsf{tl}1}^{\mathcal{C}_{\mathsf{sim}}}$. But, by construction of $\mathcal{C}_{\mathsf{sim}}$, the responses of $\mathcal{C}_{\mathsf{sim}}$ are distributed identically to the responses of $\mathcal{C}$. Thus, the probability that $\mathcal{A}_{\mathsf{we}1}$ outputs b' with $b = b'$ is equal to the success probability of the time-lock encryption adversary, which yields

$$\Pr[b = b'] = \epsilon_{\mathsf{tl}} \tag{4}$$

Step 3: a contradiction. Note that $\mathcal{A}_{\mathsf{clk}}$ runs $\mathcal{A}_{\mathsf{tl}0}$ once, which costs at most t_{tl} operations. Then it issues exactly $\tau - 1 < t_{\mathsf{tl}}$ queries to $\mathcal{C}$, performs a minor amount of t'' operations to create procedure $\mathcal{C}_{\mathsf{sim}}$, and finally performs at most t_{ext} operations to execute $\mathcal{E}$. Thus, $\mathcal{A}_{\mathsf{clk}}$ performs at most $2 \cdot t_{\mathsf{tl}} + t_{\mathsf{ext}} + t''$ operations in total. By the assumed $(R, 2 \cdot t_{\mathsf{tl}} + t_{\mathsf{ext}} + t'', \epsilon_{\mathsf{clk}})$-security of the computational reference clock $\mathcal{C}$, it must hold that $\Pr[\mathsf{Exp}_{\mathsf{clk}}^{\mathcal{C},R,\mathcal{A}_{\mathsf{clk}}}(1^\lambda) = 1] \leq \epsilon_{\mathsf{clk}}$.

Plugging this bound and the bounds from (5) and (4) into Inequality (2), we obtain

$$\begin{aligned}\epsilon_{\mathsf{clk}} \geq \Pr[\mathsf{Exp}_{\mathsf{clk}}^{\mathcal{C},R,\mathcal{A}_{\mathsf{clk}}}(1^\lambda) = 1] &\geq \Pr\left[b = b'\right] - \Pr\left[(1^\tau, w) \notin R \wedge b = b'\right] \\ &\geq \epsilon_{\mathsf{tl}} - \epsilon_{\mathsf{we}}\end{aligned}$$

which yields

$$\epsilon_{\mathsf{tl}} \leq \epsilon_{\mathsf{clk}} + \epsilon_{\mathsf{we}}$$

However, this contradicts the assumption that $\mathcal{A}_{\mathsf{tl}}$ breaks the security of the time-lock encryption scheme with success probability $\epsilon_{\mathsf{tl}} > \epsilon_{\mathsf{clk}} + \epsilon_{\mathsf{we}}$. □

2.3 Extension to Adaptively-Secure Computational Reference Clocks

In our Bitcoin-based instantiation of a computational reference clock $\mathcal{C}$ described below, the adversary will also be able to modify the state of $\mathcal{C}$ to a certain degree (for instance, by executing Bitcoin transactions). This is not yet captured by Definitions 2 and 4 and the proof of Theorem 1. In this section, we extend the definitions and the security proof from the Section 2 to this case.

Definition 9. *A* computational reference clock with auxiliary input *is a stateful probabilistic machine $\mathcal{C}$ that outputs an infinite sequence $w_1, w_2, \ldots$ in the following way. The initial state of $\mathcal{C}$ is w_0. On input a string $\mathsf{aux} \in \{0,1\}^*$, it runs a probabilistic algorithm $f_{\mathcal{C}}$ which computes $w_\tau = f_{\mathcal{C}}(w_{\tau-1}, \mathsf{aux})$ and outputs w_τ.*

Intuition for Definition 9. The main difference to Definition 2 is that now we allow the output w_τ to depend on some auxiliary input aux, which is motivated by the specific computational reference clock given by the Bitcoin block chain. In Bitcoin the auxiliary input will consist of a list of Bitcoin transactions broadcasted by Bitcoin users in the network. An adversary may influence this list of transactions to some degree, by performing and broadcasting transactions. We will reflect this in the security definition given below, by letting the adversary choose the *entire* auxiliary input for each iteration of the $f_{\mathcal{C}}$-function.

Definition 10. *We say that a computational reference clock $\mathcal{C}$ is* (R, t, ϵ)-adaptively secure*, if for all $\tau \in \mathbb{N}$ and all adversaries $\mathcal{A}$ that have access to computational reference clock $\mathcal{C}$ and perform at most t operations holds that $\Pr[\mathsf{Exp}_{\mathsf{clk}}^{\mathcal{C},R,\mathcal{A}}(1^\lambda) = 1] \leq \epsilon$, where $\mathsf{Exp}_{\mathsf{clk}}^{\mathcal{C},R,\mathcal{A}}$ is the following experiment.*

$$\begin{array}{ll} \underline{\mathsf{Exp}_{\mathsf{clk}}^{\mathcal{C},R,\mathcal{A}}(1^\lambda):} & \underline{\mathcal{O}(\mathsf{aux}):} \\ w := w_0; (1^\tau, w_\tau) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}(\cdot)}(1^\lambda) & w := f_{\mathcal{C}}(w, \mathsf{aux}) \\ \textit{Return } (1^\tau, w_\tau) \in R & \textit{Return } w \end{array}$$

$\mathcal{A}$ may query $\mathcal{O}$ at most $\tau(\lambda) - 1$ times.

Intuition for Definition 10. The main difference to Definition 4 is that now the adversary has access to a *stateful* oracle $\mathcal{O}$, which takes as input aux chosen by the adversary (possibly adaptively and depending on previous oracle responses). The initial state of the oracle is equal to the initial state w_0 of reference clock $\mathcal{C}$. When queried on input aux, the oracle computes $f_{\mathcal{C}}(w, \mathsf{aux})$, using its internal state w and the adversarially-provided input aux. It updates its internal state by assigning $w = f_{\mathcal{C}}(w, \mathsf{aux})$, and returns w.

Remark 3. Note that we do not have to adapt the security definition for time-lock encryption (Definition 6) to *adaptive* computational reference clocks, because Definition 6 already fits to the adaptive setting. Technically, we would have to adopt the *correctness* requirement in Definition 5 by adding the additional auxiliary inputs $\mathsf{aux}_1, \ldots, \mathsf{aux}_\tau$ of each clock iteration. However, this is straightforward and therefore omitted. We only note that correctness should hold for *all* possible auxiliary inputs, of course.

Theorem 2. *Algorithms $(\mathsf{Enc}_{\mathrm{TL}}, \mathsf{Dec}_{\mathrm{TL}})$ from Section 2.2 form a $(t_{\mathsf{tl}}, \epsilon_{\mathsf{tl}})$-secure time-lock encryption scheme for computational reference clock $\mathcal{C}$ with*

$$\epsilon_{\mathsf{tl}} \leq \epsilon_{\mathsf{clk}} + \epsilon_{\mathsf{we}}$$

provided that the witness encryption scheme is $(R, t_{\mathsf{tl}} + t', t_{\mathsf{ext}}, \epsilon_{\mathsf{we}})$-secure, and reference clock $\mathcal{C}$ is adaptively *$(R, 2 \cdot t_{\mathsf{tl}} + t_{\mathsf{ext}}, \epsilon_{\mathsf{clk}})$-secure with respect to Definition 4, where t' is the time required to evaluate $f_{\mathcal{C}}$ at most $\tau - 1$ times.*

Proof outline for Theorem 2. The proof of Theorem 2 is nearly identical to the proof of Theorem 1. The only difference is that now we have to treat the fact that the adversary $\mathcal{A}_{\mathsf{clk}}$ in the adaptive clock security experiment is allowed to submit (possibly

adaptively-chosen) auxiliary input values to the clock. Therefore the step from the proof of Theorem 1, where $\mathcal{A}_{\mathsf{clk}}$ constructs the simulated clock $\mathcal{C}_{\mathsf{sim}}$ by querying $\mathcal{C}$ on all possible inputs in advance, to turn oracle-machine $\mathcal{A}_{\mathsf{tl}} = (\mathcal{A}_{\mathsf{tl}0}, \mathcal{A}_{\mathsf{tl}1})$ into a machine $\mathcal{A}_{\mathsf{we}}$ without oracle access, fails. To overcome this difficulty, we prove Theorem 2 by providing $\mathcal{A}_{\mathsf{tl}1}$ with an algorithm $\mathcal{C}^f_{\mathsf{sim}}$, which computes function $f_{\mathcal{C}}$ (rather than just returning hard-coded constants).

Proof. Again we assume towards contradiction that there exists an adversary $\mathcal{A}_{\mathsf{tl}}$ which breaks the security of the time-lock encryption scheme by performing at most t_{tl} operations, but with success probability

$$\epsilon_{\mathsf{tl}} > \epsilon_{\mathsf{clk}} + \epsilon_{\mathsf{we}}$$

Consider the following adversary $\mathcal{A}_{\mathsf{clk}}$ against $\mathcal{C}$, which runs $\mathcal{A}_{\mathsf{tl}}$ as a subroutine.

1. $\mathcal{A}^{\mathcal{C}}_{\mathsf{clk}}$ runs $(m_0, m_1, \tau, st) \xleftarrow{\$} \mathcal{A}_{\mathsf{tl}0}{}^{\mathcal{C}}(1^\lambda)$ by relaying all oracle queries and the responses between $\mathcal{A}_{\mathsf{tl}0}{}^{\mathcal{C}}$ and $\mathcal{C}$. Let w denote the state returned by $\mathcal{C}$ in response to the last query issued by $\mathcal{A}_{\mathsf{tl}0}$. Then $\mathcal{A}^{\mathcal{C}}_{\mathsf{clk}}$ defines $\mathcal{A}_{\mathsf{we}0}$ as the algorithm which outputs $(1^\tau, m_0, m_1, st)$ on input 1^λ. Clearly, $\mathcal{A}_{\mathsf{we}0}$ does not issue any queries to $\mathcal{C}$.
2. Next, $\mathcal{A}^{\mathcal{C}}_{\mathsf{clk}}$ creates a procedure $\mathcal{C}^f_{\mathsf{sim}}$. On input $\mathsf{aux} \in \{0,1\}^*$, $\mathcal{C}^f_{\mathsf{sim}}$ computes an updated state $w := f_{\mathcal{C}}(w, \mathsf{aux})$ and outputs w. Then $\mathcal{A}^{\mathcal{C}}_{\mathsf{clk}}$ defines $\mathcal{A}_{\mathsf{we}1} := \mathcal{A}_{\mathsf{tl}1}{}^{\mathcal{C}_{\mathsf{sim}}}$ and $\mathcal{A}_{\mathsf{we}} := (\mathcal{A}_{\mathsf{we}0}, \mathcal{A}_{\mathsf{we}1})$. Note that $\mathcal{A}_{\mathsf{we}1}$ does not issue any oracle queries, because function $f_{\mathcal{C}}$ is computed by procedure $\mathcal{C}_{\mathsf{sim}}$ instead of oracle $\mathcal{C}$. Note also that procedure $\mathcal{C}_{\mathsf{sim}}$ is able to simulate $\mathcal{C}$ perfectly for all queries issued by $\mathcal{A}_{\mathsf{tl}1}$.
3. Then $\mathcal{A}_{\mathsf{clk}}$ computes $c \xleftarrow{\$} \mathsf{Enc}_R(1^\lambda, 1^\tau, m_b)$ for uniformly random $b \xleftarrow{\$} \{0,1\}$.
4. Finally, $\mathcal{A}^{\mathcal{C}}_{\mathsf{clk}}$ runs the extractor $w \xleftarrow{\$} \mathcal{E}(1^\tau, m_0, m_1, st, c)$ for $\mathcal{A}_{\mathsf{we}}$, and returns whatever $\mathcal{E}$ returns.
5. (Only for the analysis: $\mathcal{A}^{\mathcal{C}}_{\mathsf{clk}}$ executes $b' \xleftarrow{\$} \mathcal{A}_{\mathsf{we}1}(1^\lambda, st, c)$.)

The rigorous analysis of the success probability of this adversary $\mathcal{A}_{\mathsf{clk}}$ is nearly identical to the proof of Theorem 1. The only difference is in Step 1 of the proof. As in Equation (2), we have

$$\Pr[\mathsf{Exp}^{\mathcal{C},R,\mathcal{A}_{\mathsf{clk}}}_{\mathsf{clk}}(1^\lambda) = 1] \geq \Pr[b = b'] - \Pr[(1^\tau, w) \notin R \wedge b = b']$$

Step 1: an upper bound on $\Pr[(1^\tau, w) \in R \wedge b = b']$. Note that $\mathcal{A}_{\mathsf{we}}$ is a witness encryption adversary, which performs at most $t_{\mathsf{tl}} + t'$ operations. Here t' is the number of operations required to execute procedure $\mathcal{A}_{\mathsf{we}0}$ once and procedure $\mathcal{C}^f_{\mathsf{sim}}$ at most $\tau - 1$ times. Note that t' may become relatively large, compared to the value of t' from the proof of Theorem 1, but it is still polynomially-bounded.

By the $(R, t_{\mathsf{tl}} + t', t_{\mathsf{ext}}, \epsilon_{\mathsf{we}})$-security of the witness encryption scheme, there exists an extractor $\mathcal{E}$ that performs at most t_{ext} operations and outputs a witness w such that

$$\epsilon_{\mathsf{we}} \geq \Pr[b = b' \wedge (1^\tau, w) \notin R] \tag{5}$$

Step 2: a lower bound on $\Pr[b = b']$. With exactly the argument from Step 2 of the proof of Theorem 1, we obtain $\Pr[b = b'] = \epsilon_{\mathsf{tl}}$ (cf. Equation 4).

Step 3: a contradiction. With exactly the argument from Step 3 of the proof of Theorem 1, we obtain $\epsilon_{\mathsf{tl}} \leq \epsilon_{\mathsf{clk}} + \epsilon_{\mathsf{we}}$, which contradicts the assumption that $\mathcal{A}_{\mathsf{tl}}$ breaks the security of the time-lock encryption scheme with success probability $\epsilon_{\mathsf{tl}} > \epsilon_{\mathsf{clk}} + \epsilon_{\mathsf{we}}$. □

3 Time-Lock Encryption based on Bitcoins

In this section, we will first describe the necessary background on Bitcoins. Then we explain how the scheme from Section 2.2 can be instantiated based on Bitcoins. Finally, we discuss some engineering tasks that arise in the context of Bitcoin-based time-lock encryption.

3.1 The Bitcoin Block Chain

Cryptocurrencies are a cryptographic equivalent of regular currencies. The concept of *decentralized* cryptocurrencies has recently received a lot of attention, mostly motivated by the tremendous success of the most prominent decentralized cryptocurrency *Bitcoin* [32] and the emerge of a large number of alternative decentralized cryptocurrencies.[4]

Using Bitcoin as an example, we will show how decentralized cryptocurrencies can be used as a concrete instantiation of the abstract concept of computational reference clocks. We stress, however, that Bitcoin serves merely as one concrete example. For instance, other decentralized cryptocurrencies also provide mechanisms that may be used to instantiate such reference clocks.

A complete description of the full Bitcoin system is out of scope of this paper. In particular, we omit all details about Bitcoin *transactions*, and give only a simplified description that captures the relevant features of Bitcoins. In the sequel we focus on one central building block of Bitcoin, the so-called *Bitcoin block chain*. We refer to [32] for a description of the full system.

The Bitcoin block chain. The *block chain* is used in Bitcoin to prevent *double-spending* of Bitcoins.[5] It is a sequence of tuples

$$(T_1, r_1, D_1, B_1), \ldots, (T_s, r_s, D_s, B_s)$$

that satisfies

$$B_i := H(T_i, r_i, D_i, B_{i-1})$$

where H is a cryptographic hash function based on SHA-256 and $B_1, \ldots, B_s$ are called *blocks*. B_0 is a distinguished value, called the *genesis block*, which is a hard-coded constant in the Bitcoin software. The values T_i, r_i, D_i are described below.

Bitcoin users may attempt to find the next block B_{s+1} in the chain, which is a computationally expensive (but feasible) task, because B_{s+1} must meet certain conditions

[4] See `http://altcoins.com/` for an overview of Bitcoin alternatives.

[5] For readers not familiar with digital currencies, we give additional background information in Appendix A.

that we will describe below. Users contributing to this search are called *miners*. The main incentive to contribute significant computational resources to the progress of the block chain is that for each new block the respective miner is rewarded with a certain amount of Bitcoins.[6]

Each miner keeps a full local copy of the block chain, and collects all recent transactions broadcasted by other Bitcoin peers. For each transaction, the miner first checks if it is "malicious", that is, if it contains any coins that, according to the transaction ledger, are not in possession of the spending party. These transactions are discarded. T_{s+1} denotes the list of new transactions which are not discarded. The miner now attempts to approve these transactions, by finding that a new block B_{s+1} in the block chain which includes these transactions. To this end, the miner increments a counter value r_{s+1}, until the hash

$$B_{s+1} := H(T_{s+1}, r_{s+1}, D_{s+1}, B_s)$$

satisfies $B_{s+1} \leq D_{s+1}$, where the binary string B_{s+1} is interpreted canonically as an integer, and D_{s+1} is the current value of a variable public system parameter called the *target*. The size of the target determines the computational hardness of finding new blocks. It is related to the *Bitcoin difficulty* by the definition

$$\mathit{difficulty} := \frac{\sigma}{\mathit{target}}$$

where $\sigma = (2^{16} - 1) \cdot 2^{208}$ is a constant, called the *Bitcoin maximum target*. Each new block B_{s+1} serves as a *proof of work* for the computational resources contributed by the miner that found B_{s+1}. New blocks are broadcasted to all other Bitcoin peers, along with their associated data $(T_{s+1}, r_{s+1}, D_{s+1}, B_s)$. All miners receiving the new block B_{s+1} will then turn to searching for the next block B_{s+2}.

It may happen that at some point the block chain forks (for instance, if it happens that two miners simultaneously find a new block B_{s+1}), such that different miners continue their work on different branches of the fork. This problem is resolved in Bitcoin by considering only these transactions as valid, which correspond to the *longest* branch of the fork. Only newly mined blocks that correspond to the longest branch are rewarded. This provides an incentive for miners to contribute only to the longest branch.

Remark 4. Actually, the "length" of a chain in Bitcoin is not determined by the number of blocks, but by sum of the difficulty of all blocks in the chain. This is done to prevent that an adversary forks the chain by appending some low-difficulty blocks.[7] This distinction is not relevant for our paper. As in [3, 1], we will therefore assume that the longest chain also corresponds to the chain with the largest sum of difficulties. When referring to "the Bitcoin block chain" in the sequel, we will mean the longest chain.

Note that the complexity of the problem of finding a new block B_{s+1} with $B_{s+1} \leq D_{s+1}$ grows with decreasing D_{s+1}.[8] This allows to dynamically modify the complexity of finding a new block by modifying the difficulty. The Bitcoin system frequently

[6] 25 Bitcoins per block, at a value of approximately 235 US-$ per Bitcoin, in May 2015.

[7] See `https://en.bitcoin.it/wiki/Block_chain`.

[8] Under the assumption that H is a sufficiently secure hash function.

adjusts the difficulty, depending on the computational power contributed by miners to the progress of the Bitcoin block chain,[9] such that about *every 10 minutes* a new block is appended to the block chain.

Important properties of the Bitcoin block chain. The Bitcoin block chain has the following two properties, which are particularly relevant to our work.

- Bitcoin miners have an incentive to contribute *significant computational resources* to the progress of the block chain, and to *publish their solutions* in order to get rewarded for their effort.
  The total computing power contributed to the Bitcoin block chain is *huge*, as of May 2015 the network computes more than $350 \cdot 10^{15} \approx 2^{58}$ hashes *per second*.
- The Bitcoin block chain grows *constantly* and with *predictable progress*. The difficulty, and thus the size of the target, is *frequently adjusted* (about every two weeks) to the computational power currently available in the Bitcoin network, such that about *every 10 minutes* a new block is appended to the chain.

3.2 NP-Relations Based on Hash Block Chains

Let $H : \{0,1\}^* \to \{0,1\}^d$ be a hash function for some constant d. Let $\beta \in \{0,1\}^d$, and let $\delta : \mathbb{N} \to [0, 2^d - 1]$ be a function with polynomially-bounded description. We will call β the *starting block* and δ the *target bound function*.

Definition 11. *Let $R_{\beta,\delta}$ be the relation where $(1^x, w) \in R_{\beta,\delta}$ if and only if*

$$w = ((T_1, r_1, D_1, B_1), \ldots, (T_\tau, r_\tau, D_\tau, B_\tau))$$

and w satisfies all the following properties:

- *$|T_i|$ and $|r_i|$ are polynomially bounded, and $D_i \in [0, 2^d - 1]$*
- *w contains at least x tuples (T_i, r_i, D_i, B_i)*
- *$B_1 = H(T_1, r_1, D_1, \beta)$*
- *$B_i = H(T_i, r_i, D_i, B_{i-1})$ for all $i \in [2, x]$*
- *$\delta(i) \geq B_i$ for all $i \in [1, x]$, where we interpret bit strings $B_1, \ldots, B_x$ canonically as integers.*

Note that $R_{\beta,\delta}$ is an NP-relation, because there is an efficient deterministic algorithm that, given $(1^x, w)$, β, and δ, verifies that $(1^x, w) \in R_{\beta,\delta}$ by checking the conditions from Definition 11. Note also that the problem of finding a witness w for a given statement 1^x corresponds to the problem of finding a valid block chain w of length x with respect to (β, δ), which gets increasingly difficult with decreasing δ.

3.3 Time-Lock Encryption from Bitcoins

We will now describe the Bitcoin-based computational reference clock $\mathcal{C}_{\mathsf{btc}}$ along with a suitable associated relation $R_{\beta,\delta}$, such that we may assume that $\mathcal{C}_{\mathsf{btc}}$ is $(R_{\beta,\delta}, t)$-secure for reasonable t. As described in Section 2.2, we can then combine these building blocks with witness encryption, to obtain a Bitcoin-based time-lock encryption scheme.

[9] See `https://bitcoinwisdom.com/bitcoin/difficulty` for a chart depicting the recent development of the difficulty in Bitcoin.

NP-relations based on the Bitcoin block chain. Let $H : \{0,1\}^* \to \{0,1\}^{256}$ be the hash function used in Bitcoin. Let $R_{\beta,\delta}$ be the relation from Definition 11, instantiated with H and the following parameters β and δ.

- Set $\beta := B_0$, where B_0 is the Bitcoin genesis block.
- Fix a target bound function $\delta : \mathbb{N} \to [0, 2^{256} - 1]$.

Bitcoin-based computational reference clock. Let f_{btc} be the function that expands the Bitcoin block chain. That is, on input

$$w_{\tau-1} = (T_1, r_1, D_1, B_1), \ldots, (T_{\tau-1}, r_{\tau-1}, D_{\tau-1}, B_{\tau-1})$$

and auxiliary input $\mathsf{aux} = T_\tau$, it computes and outputs the new state w_τ such that $w_\tau = (T_1, r_1, D_1, B_1), \ldots, (T_\tau, r_\tau, D_\tau, B_\tau)$ with $H(T_\tau, r_\tau, D_\tau, B_{\tau-1}) = B_\tau \leq D_\tau$, where D_τ is the current Bitcoin target.

Let $\mathcal{C}_{\mathsf{btc}}$ denote the computational reference clock that computes f_{btc}. Then the state of $\mathcal{C}_{\mathsf{btc}}$ at "time" τ consists of the first τ tuples of the Bitcoin block chain. Recall that the progress of the chain is relatively predictable. Assuming that the current length of the chain is τ tuples, and that new blocks will continuously be found at a rate of approximately one block every 10 minutes, then we know that in approximately $10 \cdot x$ minutes the block chain will contain $\tau + x$ tuples. x Note that the relations $R_{\beta,\delta}$ from Definition 11 are associated to $\mathcal{C}_{\mathsf{btc}}$ in the sense of Definition 3, provided that $\beta = B_0$ and δ satisfies $\delta(i) \geq B_i$ for all $i \in \mathbb{N}$. Unfortunately, the latter is not guaranteed, as it depends on the choice of the target bound function δ and the future development of the size of the Bitcoin target. Therefore δ must be chosen carefully.

Choosing δ carefully. Let x such that $x > \tau$, that is, no witness w for $(1^x, w) \in R_{\beta,\delta}$ is yet contained in the Bitcoin block chain. We note that any sequence

$$w = ((T_1, r_1, D_1, B_1), \ldots, (T_x, r_x, D_x, B_x)$$

computed by the Bitcoin network in the future is only a *potential* witness for $(1^x, w) \in R_{\beta,\delta}$. This is because relation $R_{\beta,\delta}$ depends on the target bound function δ. By definition, w will only be a witness for $(1^x, w) \in R_{\beta,\delta}$, if $B_i \leq \delta(i)$ holds for all $i \in [1, x]$. It might happen that at some point $\gamma \in [\tau + 1, x]$ in the future the Bitcoin target increases to a value D_γ such that $D_\gamma \geq \delta(\gamma)$. In this case we will have $(1^x, w) \notin R_{\beta,\delta}$.

It is possible to overcome this by choosing δ carefully. To this end, consider the following observations.

- First, note that we must not choose δ *too small*. More precisely, we must not choose δ such that there exists $i \in \mathbb{N}$ with $\delta(i) < B_i$. This is because then our reference clock $\mathcal{C}_{\mathsf{btc}}$ would not provide suitable witness w_i for $(1^i, w_i) \in R_{\beta,\delta}$. Thus, the time-lock encryption scheme would not be *correct*.
- Observe also that δ must not be *too large*. For instance, recall that $B_i \in \{0,1\}^d$. Thus, we could try to simply set δ such that $\delta(i) = 2^d$ for all $i \in \mathbb{N}$. Then $\delta(i) > B_i$ holds trivially for all $i \in \mathbb{N}$.
  However, then it becomes very easy to compute witnesses w_i for $(i, w_i) \in R_{\beta,\delta}$, by simply evaluating the hash function H i-times to build a chain of length i. Essentially, we have eliminated the size restriction on the B_i values, such that clock

$\mathcal{C}_{\mathsf{btc}}$ is clearly not $(R_{\beta,\delta}, t)$-secure for any such relation $R_{\beta,\delta}$ and any reasonable t. The resulting time-lock encryption scheme would be trivially insecure.

Ideally, δ is chosen such that $\delta(i) = D_i$ matches the Bitcoin target parameter D_i for *all future* $i > \tau$. However, since D_i depends on the computational resources that currently contribute to the Bitcoin network, this would require to predict the amount of these resources. Therefore it seems impossible to predict D_i *exactly*. But we can try to approximate D_i with δ as closely as possible, such that it always holds that $\delta(i) = D_i - \epsilon_i$ for *small* values ϵ_i.

Dependence on the sender's preferences. Note that the "right" choice of δ depends on the preference of the sender, which one of the following options is more desirable for the encrypted message.

- If δ is chosen too small, then the witness required to decrypt the message may never be contained in the public Bitcoin block chain. This may happens if the Bitcoin difficulty decreases faster than expected by the encrypting party choosing δ. Thus, it may be infeasible to decrypt the ciphertext in reasonable time, such that nobody will ever learn the message (at least not within close time distance to the desired deadline).
- If δ is chosen too large, then an adversary that is able to perform a very large number of computations within a short time may be able to decrypt the message before the deadline, even though its computational resources are significantly below the resources of all Bitcoin miners.

Since it is not clear which of the above options is preferable *in general*, we think it makes most sense to let the sender choose δ *application-dependent*, or possibly even *individually* for each encrypted message. If it is more desirable that an encrypted message remains secret (possibly for a *much* longer time than originally desired by the sender), rather than being decrypted before the deadline, then one would choose a very small target bound function δ. If it is preferred that the message is rather decrypted earlier than possibly never, then one would choose δ larger. The substantial computational resources currently contributed to the Bitcoin network provide a generous margin of error for the choice of δ.

We consider the "right" choice of δ as an application-dependent engineering problem, which we will discuss in Section 3.4, along with other engineering questions arising from the Bitcoin-based instantiation of time-lock encryption.

Security of the Bitcoin-based time-lock encryption scheme. The following theorem follows from Theorem 2 (the adaptive variant of Theorem 1, see Appendix 2.3).

Theorem 3. *Let* $(\mathsf{Enc}_{\mathrm{TL}}, \mathsf{Dec}_{\mathrm{TL}})$ *be the time-lock encryption scheme obtained from combining computational reference clock* $\mathcal{C}_{\mathsf{btc}}$ *with a witness encryption scheme for relations* $R_{\beta,\delta}$ *by applying the construction from Section 2.2. Then* $(\mathsf{Enc}_{\mathrm{TL}}, \mathsf{Dec}_{\mathrm{TL}})$ *is a* $(t_{\mathsf{tl}}, \epsilon_{\mathsf{tl}})$*-secure time-lock encryption scheme for computational reference clock* $\mathcal{C}$ *with*

$$\epsilon_{\mathsf{tl}} \leq \epsilon_{\mathsf{clk}} + \epsilon_{\mathsf{we}}$$

provided that the witness encryption scheme is $(R_{\beta,\delta}, t_{\mathsf{tl}} + t', t_{\mathsf{ext}}, \epsilon_{\mathsf{we}})$*-secure, and reference clock* $\mathcal{C}$ *is* $(R_{\beta,\delta}, 2 \cdot t_{\mathsf{tl}} + t_{\mathsf{ext}}, \epsilon_{\mathsf{clk}})$*-secure, where* t' *is as in Theorem 2.*

The assumption that $\mathcal{C}_{\mathsf{btc}}$ is $(R_{\beta,\delta}, t, \epsilon)$-secure for the relations $R_{\beta,\delta}$ defined above and reasonable t and ϵ, can be analyzed in the Random Oracle Model [8]. To this end, consider the following lemma.

Lemma 1. *Let* $R_{\beta,\delta}$ *be a relation according to Definition 11, instantiated with a random oracle* $H : \{0,1\}^* \to \{0,1\}^d$*. Let* $\mathcal{A}^H$ *be an adversary, which receives as input* (β, δ) *and* $(1^\tau, w_\tau)$ *with* $(1^\tau, w_\tau) \in R_{\beta,\delta}$*, and issues at most* t *random oracle queries. Then for all* $x \in \mathbb{N}$ *holds that*

$$\Pr\left[(1^{\tau+x}, w) \in R : w \xleftarrow{\$} \mathcal{A}^H(1^\lambda, \delta, \beta, 1^\tau, w_\tau)\right] \leq \left(\frac{e \cdot (t+x) \cdot \delta_{\mathsf{max}}}{x \cdot 2^d}\right)^x$$

where e *is Euler's number and* $\delta_{\mathsf{max}} := \max_{i \in \{1,\ldots,x\}}\{\delta(\tau + i)\}$.

Proof. We have to bound the probability that $\mathcal{A}^H$ outputs

$$w = ((T_1, r_1, D_1, B_1), \ldots, (T_{\tau+x}, r_{\tau+x}, D_{\tau+x}, B_{\tau+x})$$

such that w satisfies all conditions of Definition 11. Note that $\mathcal{A}$ receives as input $(1^\tau, w_\tau)$ with $(1^\tau, w_\tau) \in R_{\beta,\delta}$, which corresponds to a chain

$$w_\tau = ((T_1, r_1, D_1, B_1), \ldots, (T_\tau, r_\tau, D_\tau, B_\tau)$$

$\mathcal{A}$ has to find at least the remaining x tuples $(T_{\tau+i}, r_{\tau+i}, D_{\tau+i}, B_{\tau+i})$, such that

$$B_{\tau+i} = H(T_{\tau+i}, r_{\tau+i}, D_{\tau+i}, B_{\tau+i-1}) \leq \delta(\tau + i)$$

holds for all $i \in \{1, \ldots, x\}$. A necessary condition for this is that $\mathcal{A}$ outputs at least x values whose random oracle hash is smaller than or equal to δ_{max}, where $\delta_{\mathsf{max}} = \max_{i \in \{1,\ldots,x\}}\{\delta(\tau + i)\}$. Note also that $\mathcal{A}$ is able to output at most $t + x$, namely the at most t queries $h_1, \ldots, h_t$ to the random oracle, plus the values $h_{t+1}, \ldots, h_{t+x}$ returned by $\mathcal{A}$, where

$$h_{t+i} := (T_{\tau+i}, r_{\tau+i}, D_{\tau+i}, B_{\tau-1+i})$$

We will bound the probability that at least x of these $t + x$ values satisfy $H(h_j) \leq \delta_{\mathsf{max}}$, $j \in \{1, \ldots, t + x\}$.

Each output h_j corresponds to an independent Bernoulli experiment, which evaluates to 1 if and only if $H(h_j) \leq \delta_{\mathsf{max}}$. Let

$$X_j := \begin{cases} 1, \text{ if } H(h_j) \leq \delta_{\mathsf{max}} \\ 0, \text{otherwise} \end{cases}$$

be random variables, and let X denote the random variable $X := \sum_{j=1}^{t+x} X_j$. In order to bound the probability that $X \geq x$, we will apply the Chernoff bound. Note that the the probability that a single Bernoulli experiment outputs 1 is at most $\delta_{\mathsf{max}}/2^d$,

and the expected number X of experiments that output 1 after $(t + x)$ trials is $\mu := (t + x) \cdot \delta_{\mathsf{max}}/2^d$. Then we have

$$\Pr[X \geq x] \leq \frac{e^x \mu^x}{e^\mu x^x} = \frac{1}{e^\mu} \cdot \left(\frac{e\mu}{x}\right)^x \leq \left(\frac{e \cdot (t + x) \cdot \delta_{\mathsf{max}}}{x \cdot 2^d}\right)^x$$

where e is Euler's number, the first inequality is the Chernoff bound,[10] and the second inequality uses that $\mu > 0$. □

Example applications of Lemma 1. To consider some applications of Lemma 1, let $d := 256$ to match the output length of the hash function used in Bitcoin, and assume the current length of the block chain is τ. Let $R_{\beta,\delta}$ be a relation with β equal to the Bitcoin genesis block and constant target bound function $\delta := 2^{190}$.[11]

As a first example, suppose a time-lock encrypted ciphertext

$$c = \mathsf{Enc}_{\mathrm{TL}}(1^\lambda, \tau + 6, m)$$

is given. Note that c can be decrypted when the Bitcoin block chain has reached length $\tau + 6$, which will most likely happen in about 60 minutes. According to Lemma 1, an adversary which is capable of evaluating the hash function $t = 2^{50} - 1$ times is able to find witness w with $(1^{\tau+6}, w) \in R_{\beta,\delta}$ with probability at most

$$\left(\frac{e(t + 1)\delta}{x2^d}\right)^x \approx \left(\frac{2^{50+190}}{2^{256}}\right)^6 = \frac{1}{2^{96}}$$

Thus, even an adversary which is able to evaluate the hash function about 2^{50} times within minutes would only have a very small probability of being able to decrypt the ciphertext before the deadline, provided that the witness encryption scheme is secure.

As another example, suppose a time-lock encrypted ciphertext

$$c = \mathsf{Enc}_{\mathrm{TL}}(1^\lambda, \tau + 144, m)$$

is given. Note that c can be decrypted when the block chain has reached length $\tau + 144$, which will most likely happen in about 24 hours. By Lemma 1, an adversary which is able to evaluate the hash function $t = 2^{60}-1$ times finds a witness w with $(1^{\tau+144}, w) \in R_{\beta,\delta}$ with probability at most

$$\left(\frac{e(t + 1)\delta}{x2^d}\right)^x \approx \left(\frac{2^{60+190}}{2^{256}}\right)^{144} = \frac{1}{2^{864}}$$

Thus, even an adversary which is able to evaluate the hash function about 2^{60} times within a few hours would only have a very small probability of being able to decrypt the ciphertext before the deadline, unless it finds an attack on the witness encryption scheme.

[10] Obtained from the classical formulation $\Pr[X \geq (1+\ell)\mu] \leq e^{\ell\mu}/(1+\ell)^{\mu(1+\ell)}$ by substituting $x := (1 + \ell)\mu$ and $\ell := x/\mu - 1$.

[11] The current Bitcoin target value on February 11, 2015 is about $2^{188.63}$.

Remark 5. Note that we silently assume in this analysis that the adversary performs its computations "secretly", without publishing found blocks in the block chain. Alternatively, an adversary may contribute its resources to the block chain. The analysis of such adversaries is more involved. We explain this in Section 3.4.

3.4 Variants and Further Analysis

In this section, we will discuss several ideas for solving the engineering tasks and further questions related to the Bitcoin-based instantiation of time-lock encryption.

Choosing δ for short time periods. Let $R_{\beta,\delta}$ for some particular choice of δ, and let

$$c = \mathsf{Enc}_R(1^\lambda, 1^{\tau_{\mathsf{dec}}}, m)$$

be a time-lock encryption with a witness encryption scheme for $R = R_{\beta,\delta}$. Observe that for this particular ciphertext we do not need $\delta(i) \geq B_i$ for *all* $i \in \mathbb{N}$ in order to be able to use a witnesses provided by $\mathcal{C}_{\mathsf{btc}}$ to decrypt c. It suffices if $\delta(i) \geq B_i$ holds for all $i \leq \tau_{\mathsf{dec}}$.

Given that the computational resources available in the Bitcoin network were relatively predictable in the past, and that the huge computational power gathered in the network provides a generous margin of error, we think that it is relatively easy to determine a suitable target bound function δ for all ciphertexts which should be decrypted within a *short* period of time. By "short" we mean hours, days, or a few weeks.

More robust relations and time-lock encryption for long time periods. The longer the time between encryption and decryption of a ciphertext is, the more difficult it becomes to find a suitable target bound function δ. One could, however, instead use relations which are more "robust" than the ones described in Definition 11. For example, a sender may choose a relation $R_{\beta,\delta,\omega}$, which is defined almost identical to the relations $R_{\beta,\delta}$ defined above, with the exception that $R_{\beta,\delta,\omega}$ accepts

$$w = (T_1, r_1, D_1, B_1), \ldots, (T_\tau, r_\tau, D_\tau, B_\tau)$$

as a witness for $(1^\tau, w) \in R_{\beta,\delta,\omega}$, even if $\delta(i) \leq B_i$ holds *at most ω times* for $i \in \{1, \ldots, \tau\}$. More generally, if a witness encryption scheme for all NP-relations is used as a building block, then a sender could in principle choose any NP-relation it considers reasonable here.

On the difficulty of advancing the Bitcoin block chain faster. Note that an adversary $\mathcal{A}$ has two options to advance the Bitcoin block chain faster than one block every ten minutes.

The first option is that $\mathcal{A}$ performs all its computations *secretly*. This means that it does not contribute any blocks to the public Bitcoin block chain, but instead keeps all newly found block secret. This is the approach of an adversary which wants to be *exclusively* able to decrypt the ciphertext before the deadline. Note that in this case the adversary would have to compute all missing blocks to decrypt a ciphertext on its own. If the target bound function δ is chosen well, such that it closely approximates

the actual Bitcoin target, then this essentially means that the adversary would have to perform about the same amount of computations as all Bitcoin miners together (unless it finds a better algorithm for solving the computational problem processed by the miners). Assuming that no single adversary is capable of performing this large amount of computations, this is infeasible.

Alternatively, an adversary might not be interested in being *exclusively* able to decrypt the ciphertext before the deadline. Instead, it might simply want that the ciphertext can be publicly decrypted earlier than desired by the sender. In this case, the adversary could contribute its computational resources to the public Bitcoin block chain. Even though the Bitcoin system adjusts the difficulty of advancing the block chain frequently by modifying the size of the *target*, note that this happens only relatively slowly, every 2016 blocks. Therefore the difficulty adjustment in Bitcoin is not able to completely prevent such attacks, it may only cushion its effectiveness. However, an adversary that aims at a speed-up by, say, 10% would have to contribute additional resources in the order of 10% of the resources of all other Bitcoin miners together. Thus, the larger the speedup desired by the adversary, the larger are the computational resources it would have to contribute. Moreover, the incentive of other Bitcoin miners to contribute their resources depends on the current difficulty. If this difficulty is too high, then the cost of contributing resources to Bitcoin mining exceeds the revenue obtainable from Bitcoin mining. This would deter other Bitcoin miners from further contributing to the Bitcoin block chain, which cushions the effectiveness of the resources contributed by the adversary further.

A detailed analysis of adversarial strategies to advance the Bitcoin block chain significantly faster than one block every ten minutes therefore needs to take such game-theoretic aspects into account, it is therefore out of scope of this paper. See [20] for a related work.

Further game-theoretic aspects. Ideally, the target bound function δ is chosen such that decrypting the ciphertext earlier would require so many computational resources, that from a game-theoretic perspective it is more reasonable to use these resources to perform a different computation. This is always the case when the value of learning the encrypted message before the deadline is below the revenue obtainable from the different computation.

In particular, an adversary might gain more revenue from mining Bitcoins directly than from trying to learn the time-lock encrypted message earlier than others. The revenue obtainable from Bitcoin mining is easily quantifiable, we think this is a very nice aspect of the Bitcoin-based instantiation of time-lock encryption.

Time-lock encryption beyond Bitcoins. In order to obtain a stable and robust time-lock encryption scheme from Bitcoin, it is required that Bitcoin miners will continue to contribute significant computational resources to the progress of the Bitcoin block chain. It is conceivable that Bitcoin will be discontinued at some point in the future. This may happen, for instance, due to a market crash making Bitcoins worthless, or simply because its popularity decreases over time, possibly replaced by a different cryptocurrency. This is of course unpredictable.

We stress that the approach for constructing time-lock encryption described in Section 2 is *general*. That is, it is motivated by, but not reliant on Bitcoins. In principle, our approach can be used with completely different types of iterative, public, large-scale computations. For instance, time-lock encryption could be based on other decentralized cryptocurrencies, or even on completely different types of public computations. The construction from Section 3.3 is only one concrete application of the techniques developed in Section 2, motivated by the fact that Bitcoins currently seem to be the most interesting candidate instantiation, in particular due to their wide adoption and the significant computational resources contributed to the Bitcoin network.

Acknowledgements. We thank Christoph Bader for many helpful comments on an early version of this manuscript, and the anonymous reviewers of CRYPTO 2015 for pointing us to the schemes of [35, 28, 26].

References

1. Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 443–458, Berkeley, California, USA, May 18–21, 2014. IEEE Computer Society Press.
2. Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Łukasz Mazurek. Fair two-party computations via bitcoin deposits. Cryptology ePrint Archive, Report 2013/837, 2013. `http://eprint.iacr.org/`.
3. Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Łukasz Mazurek. Secure multiparty computations on bitcoin. Cryptology ePrint Archive, Report 2013/784, 2013. `http://eprint.iacr.org/`.
4. Pablo Azar, Shafi Goldwasser, and Sunoo Park. On time and order in multiparty computation. Cryptology ePrint Archive, Report 2015/178, 2015. `http://eprint.iacr.org/`.
5. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Berlin, Germany.
6. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.
7. Mihir Bellare and Viet Tung Hoang. Adaptive witness encryption and asymmetric password-based cryptography. Cryptology ePrint Archive, Report 2013/704, 2013. `http://eprint.iacr.org/`.
8. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.
9. Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 421–439, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Berlin, Germany.
10. Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440–456, Aarhus, Denmark, May 22–26, 2005. Springer, Berlin, Germany.
11. Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 236–254, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Berlin, Germany.

12. Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324:71–90, 2003.
13. Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 52–73, San Diego, CA, USA, February 24–26, 2014. Springer, Berlin, Germany.
14. Julien Cathalo, Benoît Libert, and Jean-Jacques Quisquater. Efficient and non-interactive timed-release encryption. In Sihan Qing, Wenbo Mao, Javier López, and Guilin Wang, editors, *ICICS 05*, volume 3783 of *LNCS*, pages 291–303, Beijing, China, December 10–13, 2005. Springer, Berlin, Germany.
15. Jung Hee Cheon, Nicholas Hopper, Yongdae Kim, and Ivan Osipkov. Provably secure timed-release public key encryption. *ACM Trans. Inf. Syst. Secur.*, 11(2), 2008.
16. Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Berlin, Germany.
17. Giovanni Di Crescenzo, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan. Conditional oblivious transfer and timed-release encryption. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 74–89, Prague, Czech Republic, May 2–6, 1999. Springer, Berlin, Germany.
18. Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 139–147, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Berlin, Germany.
19. Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 37–54, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Berlin, Germany.
20. Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 436–454, Christ Church, Barbados, March 3–7, 2014. Springer, Berlin, Germany.
21. Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol: Analysis and applications. Cryptology ePrint Archive, Report 2014/765, 2014. `http://eprint.iacr.org/`, accepted to *EUROCRYPT 2015*.
22. Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17, Athens, Greece, May 26–30, 2013. Springer, Berlin, Germany.
23. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press.
24. Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.
25. Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 426–443, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Berlin, Germany.
26. Gmaxwell. Gmaxwell/alt ideas. 2014. `https://en.bitcoin.it/wiki/User:Gmaxwell/alt_ideas`.
27. Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 536–553, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany.

28. gwern.net. Time-lock encryption. 2015. http://www.gwern.net/Self-decrypting%20files.
29. Jonathan Katz, Andrew Miller, and Elaine Shi. Pseudonymous secure computation from time-lock puzzles. Cryptology ePrint Archive, Report 2014/857, 2014. http://eprint.iacr.org/.
30. Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 39–50, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Berlin, Germany.
31. Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In Robert D. Kleinberg, editor, *ITCS 2013*, pages 373–388, Berkeley, CA, USA, January 9–12, 2013. ACM.
32. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009. http://www.bitcoin.org/bitcoin.pdf.
33. Ronald L. Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, 1996.
34. Jörg Schwenk. Modelling time for authenticated key exchange protocols. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014, Part II*, volume 8713 of *LNCS*, pages 277–294, Wroclaw, Poland, September 7–11, 2014. Springer, Berlin, Germany.
35. Peter Todd. Timelock encryption incentivised by Bitcoin. 2014. https://github.com/petertodd/timelock.
36. Dominique Unruh. Revocable quantum timed-release encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 129–146, Copenhagen, Denmark, May 11–15, 2014. Springer, Berlin, Germany.

A The double-spending problem in cryptocurrencies

A central problem that a secure digital currency has to solve is to prevent *double-spending* of digital coins. It seems that this is only achievable by somehow keeping track of all previous transactions, in order to be able to determine whether a malicious party tries to spend the same digital coin more than once. There are essentially two approaches to prohibit double-spending:

1. A central trusted third party ("the bank") keeps a ledger, which contains all transactions executed in the past. This allows the trusted third party to keep track of which party is in possession of which coin. Coins can only be transferred from one party to another if the trusted third party approves the transaction. Note that this approach is inherently *centralized*, as it requires a central trustworthy instance that keeps track of all transactions.
2. The approach used by *decentralized* cryptocurrencies, like Bitcoin, is somewhat similar, however, the trusted third party is implemented by all (or most) parties simultaneously. Instead of a centralized ledger, the transaction ledger is distributed among all (or many) parties. Essentially, all transactions are broadcasted to all other parties and stored in the ledger, such that each party is able to check whether a given coin already appeared in a previous transaction, and if the claimed owner of a coin is still in possession of this coin. Thus, all parties jointly implement "the bank". All the recently successful cryptocurrencies, in particular Bitcoin, are decentralized.

The decentralized approach requires a mechanism that allows to find a mutual agreement on the sequence of approved transactions. In Bitcoin, this mechanism is implemented by the *Bitcoin block chain*.