Astrolabous: A Universally Composable Time-Lock Encryption Scheme

Myrto Arapinis, Nikolaos Lamprou, and Thomas Zacharias

The University of Edinburgh, UK

Abstract. In this work, we study the *Time-Lock Encryption* (TLE) cryptographic primitive. The concept of TLE involves a party initiating the encryption of a message that one can only decrypt after a certain amount of time has elapsed. Following the *Universal Composability* (UC) paradigm introduced by Canetti [IEEE FOCS 2001], we formally abstract the concept of TLE into an ideal functionality. In addition, we provide a standalone definition for secure TLE schemes in a game-based style and we devise a hybrid protocol that relies on such a secure TLE scheme. We show that if the underlying TLE scheme satisfies the standalone game-based security definition, then our hybrid protocol UC realises the TLE functionality in the random oracle model. Finally, we present *Astrolabous*, a TLE construction that satisfies our security definition, leading to the first UC realization of the TLE functionality.
Interestingly, it is hard to prove UC secure any of the TLE construction proposed in the literature. The reason behind this difficulty relates to the UC framework itself. Intuitively, to capture semantic security, no information should be leaked regarding the plaintext in the ideal world, thus the ciphertext should not contain any information relating to the message. On the other hand, all ciphertexts will eventually open, resulting in a trivial distinction of the real from the ideal world in the standard model. We overcome this limitation by extending any secure TLE construction adopting the techniques of Nielsen [CRYPTO 2002] in the random oracle model. Specifically, the description of the extended TLE algorithms includes calls to the random oracle, allowing our simulator to equivocate. This extension can be applied to any TLE algorithm that satisfies our standalone game-based security definition, and in particular to Astrolabous.

Keywords: Time-lock encryption, Universal composability, Fairness.

1 Introduction

The concept of encryption involves a party, the encryptor, who encrypts a message, and a designated party, the decryptor, who can retrieve that message. The decryptor can retrieve the message because she holds a piece of secret information which is called the secret key. There are two well known and studied types of encryption schemes in the literature, namely *symmetric encryption* [36] and *public key encryption* [18].

Another special type of encryption is called *time-lock encryption* (TLE). The concept of TLE involves a party that initiates the encryption of a message that can be decrypted only after a certain amount of time has elapsed.

There are two main approaches to how TLE can be defined. In the first approach [13,40], a party, called the *manager*, releases the decryption keys at specific times in the future.

In the second approach [46,39,38] a *computational puzzle*, which is a mathematical problem, needs be solved so that the message can be revealed. We distinguish *relativistic time* constructions [46,39] designed so that a puzzle can be solved only after a certain amount of computations have been performed; and *absolute time* constructions [38] designed so that the solution of the puzzle can be delegated to external entities which try to solve the puzzle independently of the TLE protocol (e.g. Bitcoin miners in [38]), giving an essence of absolute time. In either case, the message can be decrypted only after a *puzzle* has been solved or its solution has been published. The solution of the puzzle is used as the secret key in the decryption algorithm so that the message can be revealed.

In contrast to "standard" encryption, TLE differs in one but major point. The message can be retrieved without the encryptor having to reveal any secret information; the decrypting parties can actually construct the secret information themselves after some time. In standard encryption this is computationally infeasible.

The number of applications TLE finds its own space are mostly related to a security requirement called *fairness* [29]. Informally, the fairness condition states that the initial decisions of a party are not affected by the way the protocol execution progresses. There are many cryptographic protocols where fairness is violated and TLE can find an application. For example, in e-voting and specifically in self-tallying election protocols (STE) [35,1], due to access to intermediate results some parties might change their mind and vote something different from their initial choice to favour another candidate (e.g., the winning one). Another example where fairness is important is in coin flipping protocols [15], where the party that initiates the coin flip decides to abort right after the other party reveals her coin, without revealing her share to the other party. Moreover, in secret sharing protocols [44], the party that reveals her share last holds a considerable advantage over the other parties. Similar is the case of *Distributed Key Generation* protocols (DKG) [24]. By utilizing TLE, we can tackle all of the aforementioned limitations.

Unfortunately, all of the mentioned limitations cannot be solved with standard encryption or a commitment scheme. For example, the self-tallying protocols in [43,35,47,30] do not satisfy the fairness condition as already mentioned by the authors. The limitation lies fundamentally in the way encryption works. Specifically, if we use encryption only the holder of the secret key can retrieve the hidden message. So either that key is a priori known, where fairness is violated trivially as every party can decrypt the message, or not known, where the protocol cannot terminate as the message cannot be retrieved. Similar is the case if we use a commitment scheme. TLE comes to fill the gap and keep the best of both of situations mentioned, which means, *semantic security* [26,36] until some time, and then the possibility of decryption without any a priori secret information neither further interaction with the encryptor.

The state-of-the-art of composable security framework in the literature is provided by the *Universal Composability* framework (UC) [11] introduced by Canetti, where security can be maintained even if many instances of the studied protocol are executed concurrently or the protocol is composed as a subroutine of a bigger protocol. Although there are formal treatments of TLE in the literature [38], these mainly provide standalone models of security while our work aims to provide a composable treatment of the TLE primitive. The only other such attempt to our knowledge is a recently published paper [4] that we discuss in details in Section 2.1.

In this work, we abstract the notion of TLE into an ideal functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$, that captures the concept of TLE naturally. Moreover, we introduce a security definition exploring the one-wayness of TLE algorithms. We show that the one-way property of a TLE scheme is enough so that we have a UC realization of $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ after extending the TLE algorithm in the random oracle model. Although UC is the state-of-the- for arguing about security, sometimes standalone definitions are more usable and intuitive. Furthermore, many UC functionalities can only be realized in idealized models such as the RO model, suggesting that UC definitions may be "too strong" (*e.g.*, unachievable). For this reason, we further provide a new TLE game based definition in IND-CPA security style. Last, we provide a novel TLE construction, named *Astrolabous*, and show that it satisfies both of our security definitions.

Contributions. Our contributions can be summarised as follows:

1. We present a UC definition of secure TLE via an ideal functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ that captures naturally the concept of TLE as it provides the necessary security guarantees a TLE scheme should provide. Specifically, it captures *semantic security* as the encryption of a message is not correlated with the message itself. Instead, it is correlated only with the length of the message similarly to the standard encryption functionality in [11]. In addition, it captures *correctness* [26,36], i.e., if $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ finds two different messages with the same ciphertext in its record, then it aborts. Finally, we note that in the literature, there are TLE constructions [38] where the adversary holds an advantage in comparison with the other parties and which might allow him to decrypt a message earlier than the intended time. To cater for such constructions, we parameterise $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ with a leakage function leak which specifies the exact advantage (in decryption time) of the adversary compared to the honest parties. Ideally, the leak function offers no advantage to the adversary. It is worth mentioning that TLE constructions in which the adversary holds an advantage in comparison with the honest parties in the decryption time, are still useful to study in the UC framework because the computational burden for solving the puzzle can be transferred to external entities of the protocol (e.g., Bitcoin miners), making the decryption more client friendly [38].

2. We define a hybrid TLE protocol and a standalone basic security definition in a game-based fashion. We show that if the pair of TLE algorithms that our protocol uses satisfies our basic security definition then we have a UC realization of $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$.
   Our TLE protocol does not use the vanilla version of a TLE algorithm (e.g. a TLE algorithm as defined in [38]). Instead, it relies on an extended one based on techniques introduced in [42,10] in the random oracle model. Our extension was necessary for the proof of UC realization. Specifically, in both real and ideal world, all the messages eventually can be decrypted by any party. To avoid trivial distinctions[1], the simulator must be able to equivocate so that the ciphertext opens to the correct message. As a result, the simulator programs the random oracle so that the ciphertext opens to the target message, something that is not feasible with the vanilla version of a TLE scheme without the equivocation feature which our extension provides.
   In our hybrid protocol, we defined both a functionality wrapper $\mathcal{W}_q$ and an evaluation functionality $\mathcal{F}_{\mathsf{eval}}$, to model the computation that is necessary for solving the time-lock puzzle. In our case, this computation is a random oracle query, thus $\mathcal{F}_{\mathsf{eval}}$ is the random oracle. Like in [2], the main function of a functionality wrapper is to restrict the access to $\mathcal{F}_{\mathsf{eval}}$ and thus to model the limited computational resources a party has at her disposal in each round. In our case, the limited amount of computation a party has in order to solve the time-lock puzzle through queries to $\mathcal{F}_{\mathsf{eval}}$.
   Our basic security definition of TLE schemes consists of two properties, named **Correctness** and **qSecurity**. The **Correctness** property states that the decryption of an encrypted message m leads to the message m again with high probability, similar to the definition of *correctness* in the standard encryption's case. We define the **qSecurity** property in a game-based style, between a challenger and an adversary where the latter tries to guess the *challenged message* with less than the required oracle queries. A TLE scheme satisfies the **qSecurity** property if the above happens with negligible probability, capturing the fact that a message can only be decrypted when "the time comes".

3. We provide a novel construction, named *Astrolabous*, and we show that it satisfies our basic security definition, thus it supports the UC realisation of $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ (in the random oracle model). Astrolabous combines ideas from both the constructions in [46] and in [39]. Nevertheless, we did not use either of them for the following reasons. A critical drawback of [39] is that parts of the plaintext are revealed through the process of solving the time-lock puzzle, which is based on a hash evaluation, as the message is hidden in the puzzle itself. On the other hand, the construction in [46] encrypts a message with a symmetric encryption scheme [36] and then hides the encryption key into the time-lock puzzle which is based on repeated squaring. The first problem with the latter construction was that the procedure for solving the puzzle is

[1] Recall that in the ideal world, to capture semantic security, ciphertexts do not contain any information about the actual message except its length

deterministic (repeated squaring) and thus a party can bypass the functionality wrapper and solve any time-lock puzzle in a single round, in contrast with the construction in [39] where the procedure for solving the puzzle is randomized (hash evaluation which is modeled as random oracle). The second problem with the construction in [46] was that even if a party provides the solution of the puzzle but the puzzle issuer does not provide the trapdoor information that is used by the time the time-lock puzzle was created (in this case, the factorization of a composite number N) then, in order to verify the validity of the provided solution, all the verifying parties must resolve the time-lock puzzle. Thus, the *optimal complexity* scenario is hard to achieve. In contrast, the time-lock puzzle in [39] is easily verifiable without the need of any trapdoor information from the puzzle issuer.
These were our motivations for defining Astrolabous that tackles all of the above-mentioned limitations. Specifically, Astrolabous uses a symmetric key encryption scheme to hide the message like in [46] and then "hides" the symmetric key in a time-lock puzzle similar to the one in [39].

4. We introduce an additional stronger game-based definition, named IND-CPA-TLE, to capture semantic security of TLE schemes in the spirit of IND-CPA security. Our stronger definition may serve as a standard for analysing TLE schemes in the standalone setting. To demonstrate the usefulness of our stronger definition and constructions, we prove that Astrolabous and an enhanced version of the construction in [39] achieve IND-CPA-TLE security.

2 Related work

TLE is a cryptographic primitive that allows a ciphertext to be decrypted only after a specific time period has elapsed. One way of achieving this is by "hiding" the decryption key in a puzzle [46] that can be solved after a set period of time. The reward for solving the puzzle is the decryption key. So the main purpose of the puzzle is to delay the party in opening the message before a specific amount of computation has been performed. In some proposals, decryption can further be performed without requiring knowledge of any secret information [38,46].

Previously proposed constructions are based either on *witness encryption* [23] or symmetric encryption [36]. The authors of these works provide game-based definitions to argue about the security of their constructions. Unfortunately, game-based definitions do not capture the variety of adversarial behavior the UC framework [11] does. For example, in the ideal world the capabilities of the adversary are defined explicitly. So, proving that our real protocol and the ideal one are indistinguishable (UC realization) from the environment's perspective, is like proving that whatever the adversary can do against the real protocol it can also do it in the ideal world. In contrast, in a game-based approach, we try to capture the capabilities of an adversary via an experiment without being certain if the experiment captures all the adversarial behaviours possible in the real protocol. Moreover, the task of transferring these definitions to the UC setting is quite challenging due to some incompatibilities between the two settings. More details can be found in Supplementary Material B.1

A particular TLE construction proposed in [46] is based on a block cipher, e.g., *Advanced Encryption System* (AES) [16], and repeated squaring. Specifically, first, a party encrypts a message m by using AES and a secret key sampled from a key space uniformly at random. Then the party chooses the time that finding the key should require and creates a "puzzle". The ciphertext is the encrypted message with AES under the solution to the puzzle that serves as the key. No formal proof of the security of this scheme is however provided in [46]. One drawback of this construction is that, to solve the puzzle, a party must be engaged in mathematical computations. The only way that these computations could be avoided for the puzzles to be solved is the issuer of the puzzle to announce the solution along with the trapdoor information (optimal case scenario), which is the factorization of a composite number N. Without the provision of the trapdoor information, even if a party announces the solution of the puzzle, the only way for verifying the solution is to solve the puzzle again.

A similar TLE construction is that in [39]. Here, the time-lock puzzle is based on hash evaluations. Specifically, the solver of the puzzle is engaged in serial hash evaluations until solving the puzzle. Unlike [46], if some party presents the solution of the puzzle any other party can verify it efficiently by doing all the hash evaluations in parallel. A drawback of this construction is that parts of the plaintext are revealed before the full solution of the puzzle. There are also TLE proposals [13,40,14] that instead of relying on computational puzzles, assume a *Trusted Third Party* (TTP) responsible for announcing the decryption keys. Most of these constructions are based on *Public Key Infrastructure* (PKI). An obvious drawback then is the fact that we ground a big part of the security of the scheme in the TTP, which in turn leads to weaker threat models.

There are other time-lock puzzle constructions [9,6] but none of them provide composable security guarantees. A generalization of time-lock puzzles are *Verifiable Delayed Functions* (VDF) [8,45,49] with the only addition that they require the solution of the puzzle to be publicly verifiable without having to solve the puzzle, something that is desirable but not obligatory with time-lock puzzles. Again, the constructions in [8,45,49] are not analyzed in the UC framework and thus security cannot be guaranteed either when composed as part of bigger protocols or in parallel execution (e.g. in on-line network conditions).

2.1 Comparison with [4] and [3]

A concurrent and independent work closely related to ours was very recently published at EUROCRYPT 2021 [4], with a subsequent work seemingly in preparation [3]. In particular, [4] proposes a composable treatment in the UC framework of time-lock puzzles whose security is captured by the ideal functionality $\mathcal{F}_{\mathsf{tlp}}$. It further proves how the scheme proposed by Rivest *et al.* in [46] can be used to UC realise $\mathcal{F}_{\mathsf{tlp}}$ in both the random oracle and generic group models. Their realisation, as ours, relies on techniques for equivocation borrowed from [42] and [10]. They further show that no time-lock puzzle is UC realizable outside the random oracle model. Finally, they show that time-lock puzzles can be used to ensure fairness in coin flipping protocols.

The time-lock scheme proposed in [4] is not verifiable. This is addressed in the subsequent pre-print [3] where they adapt the scheme to include the trapdoor information along the message to be time-lock encrypted, rendering it verifiable.

There are some key differences between these two works and ours, rendering the proposed treatments of time-lock primitives orthogonal. The premises and assumptions are intrinsically different and capture different concepts and security notions. We discuss these differences here and argue why our formal treatment of time-lock encryption, and our proposed TLE scheme, namely Astrolabous, are preferable in some respects and more suited to many scenarios.

Apprehending time with computational puzzles In [4] and [3], a resolutely different approach to ours is taken, when it comes to real time. In particular, they introduce the global $\mathcal{G}_{\mathsf{ticker}}$ functionality to capture delays without referring to a global *"wall clock"*, and thus without referring to real time.

We, on the other hand, insist on the importance of closely relating computational time and real time, and propose an alternative treatment in the global clock model ($\mathcal{G}_{\mathsf{clock}}$). Our approach is directly motivated by the seminal paper [46], in which R. L. Rivest, A. Shamir, and D. Wagner introduce the very concept of *time-release cryptography* to capture encryption schemes that ensure encrypted messages cannot be decrypted until a set amount of time has elapsed. The goal being to, as they put it, *"send information into the future [...] by making CPU time and real time agree as closely as possible"*.

This is key to explaining why and how time-release cryptography is used in an increasing number of distributed applications, and in particular schemes hinged on *computational puzzles*, *i.e.* puzzles that can only be solved if certain computations are performed continuously for at least a set amount of time. Indeed, the cryptographic protocols underlying these applications often rely on temporally disjoint phases. Time-release cryptographic primitives, as primitives apprehending real time through computations, allow thus these temporally disjoint stages of the protocol to be enforced yet in an asynchronous manner.

This is reflected in our protocol realising the proposed ideal TLE functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$. Parties only read the time from the global clock $\mathcal{G}_{\mathsf{clock}}$ to compute the amount of time the ciphertext needs to be protected for, and infer the corresponding puzzle difficulty. Decryption however requires continuous computations being performed until the set opening time is reached, and no read command being ever issued to $\mathcal{G}_{\mathsf{clock}}$. This protocol clearly demonstrates how time-lock puzzles apprehend real time through computations.

In contrast, the protocol π_{tlp} realising the ideal time lock-puzzle functionality $\mathcal{F}_{\mathsf{tlp}}$ proposed in [4] does not instruct parties to continuously work towards solving received puzzles (the scheduling of each step for solving a puzzle is left to the environment). So the treatment proposed in [4] and [3] leaves it to the protocol using π_{tlp} or $\mathcal{F}_{\mathsf{tlp}}$ as a subroutine to correctly takes care of appropriately enforcing relative delays between key events.

Ideal functionality and realisation $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ is more general than $\mathcal{F}_{\mathsf{tlp}}$. $\mathcal{F}_{\mathsf{tlp}}$ only captures constructions that rely on computational-puzzles for "hiding" a message and not the general concept of TLE. Specifically, the puzzle solution is provided not after some time has elapsed but after some computations have been performed (similar to Proof of Work (PoW)). In contrast, our time-lock encryption functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ does not. This is why our protocol instructs parties to continuously work towards solving received puzzles. For that reason the works in [4,3] fail to capture the connection between absolute time ($\mathcal{G}_{\mathsf{clock}}$ model) and puzzle solving, something naturally expected when studying time-lock primitives. As such our functionality can cater for TLE schemes that do not rely on time-lock puzzles at all, such as the centralized solutions proposed in [13,40] where a TTP releases the solution in specific time-slots.

Moreover, some constructions such as [38] allow the adversary an unavoidable advantage in solving TLE puzzles (*e.g.*, the adversary synchronizes faster than the honest parties in the Bitcoin network [22,2]). $\mathcal{F}_{\mathsf{tlp}}$ does not capture such constructions. Our $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ functionality is parameterized with a leakage function, which specifies exactly the advantage of the adversary in each case.

Turning now to the realisations of UC secure time-lock primitives, the realisation of $\mathcal{F}_{\mathsf{tlp}}$ proposed in [4] relies on stronger assumptions as it relies both on the random oracle model and the generic group model. In contrast, our realisation of $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ only relies on the random oracle model.

On public verifiability While the time-lock encryption scheme proposed in [3] is publicly verifiable in the sense that given a puzzle, the verifying party does not need to solve the puzzle for themselves to verify that an announced solution for that puzzle is valid. This is not enough in some scenarios. For instance, consider the scenario with a dedicated server to be the puzzle solver and all other parties to be "lite" verifiers. This is very realistic given the computational requirements for solving puzzles. For efficiency, one would let a server solve the puzzles and only check that the solutions it provided are valid ones. Now in such a scenario parties *i)* would not trust the server, *ii)* would not trust the issuer of the puzzle either, but *iii)* are also not willing to solve the puzzle themselves.

Now, in [3] public verifiability is achieved because the issuer of the puzzle concatenates the message and the trapdoor information, which is the factorization of N. Given the trapdoor, one can efficiently verify that the announced solution to the puzzle is valid. However, the trapdoor announced (dishonest server) or the trapdoor included (dishonestly generated ciphertext) might not be valid for the puzzle. The only way to identify the dishonest party is to solve the puzzle for oneself and check it against the solution to the puzzle announced by the server. If they match, then the ciphertext was dishonestly generated, otherwise the server is dishonest.

This is reflected in the public verifiability notion that $\mathcal{F}_{\mathsf{tlp}}$ captures that is one sided: if an announced solution to a puzzle is valid, then the verification is successful. But if the verification fails, then some party has deviated from the protocol but it could either be the server or the issuer of the ciphertext.

In contrast, the solution of our puzzle is publicly verifiable as it does not rely on any trapdoor information from the puzzle issuer being included in the ciphertext for fast verification. So dishonestly generated ciphertexts are not meaningful anymore, and only dishonest servers need to be considered. Now if the server announces an invalid solution to a given puzzle, it gets detected.

Standalone security Along with the composable definition of secure time lock encryption schemes provided by our ideal functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$, we further provide two game-based definitions of security. A weaker one, capturing one-way hardness of a TLE scheme; and a stronger one that captures semantic security of a TLE scheme, in the spirit of IND-CPA security. We show that a TLE scheme that satisfies the weaker definition suffices for UC realising the $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ functionality through our protocol π_{TLE}. The stronger game-based definition serves as a standard for the security analysis of TLE schemes in the standalone setting. To demonstrate the usefulness of our stronger definition, we show that Astrolabous and an enhanced version of Mahmoody *et al*'s construction [39] satisfy the said security standard. This result further validates our UC treatment and in particular our ideal functionality of time-lock encryption schemes.

2.2 Asymptotic vs concrete definitions

Time-lock puzzles have also previously been studied in the game-based framework [19,32,7]. These prior works provide standalone definitions of security of time-lock puzzles in a concrete setting. Specifically, they consider adversaries that are allowed to perform a fixed number of computational steps with each step being arbitrarily parallelizable. In this way, they capture both the privacy guarantees a time-lock puzzle should satisfy (no information leaked before a certain number of computations have been performed) and the resilience against parallel computation (the problem should not be parallelizable). They further introduce definitions that capture another important security property, namely *non-malleability.* Similar to public-key encryption, non-malleability for time-lock puzzles states that the adversary given a time-lock puzzle should not be able to generate another one in which the underlying solution is related without solving it first. While, our UC definition captures non-malleability, both our game-based definitions do not. This was out of the scope of this study and we leave it for future work.

In contrast, our approach in both game-based definitions we have introduced is an asymptotic one, similarly to the definitions of Proof of Work primitives provided in [21,22]. This was necessary to bridge UC security which is in the asymptotic setting, with the game-based approach [10]. This was not something necessary for prior standalone game-based definitions as these do not argue about security in a composable framework like UC.

3 Preliminaries

We use λ as the security parameter. We write $\mathsf{negl}(\lambda)$ to denote that a function is negligible in λ. When referring to a polynomial function we use the term p or p_x where x is an integer.

3.1 Universal Composability

The *Universal Composability (UC)* paradigm introduced by Canetti in [11] is the state-of-the-art cryptographic model for arguing about the security of protocols when run under concurrent sessions. More details about the UC framework can be found in Supporting Material A. *Setup functionalities.* In the UC literature, hybrid functionalities do not only play the role of abstracting some UC-secure real-world subroutine (e.g. a secure channel), but also formalize possible setup assumptions that are required to prove security when this is not done (and in many cases even impossible to achieve) in the "standard model". For example, this type of setup functionalities may capture the concept of a trusted source of randomness, a clock, or a Public Key Infrastructure (*PKI*). Moreover, these setup functionalities can be *global*, i.e. they act as shared states across multiple protocol instances and they can be accessed by other functionalities and even the environment that is external to the current session (recall that standard ideal functionalities do not directly interact with the environment). The extension of the UC framework in the presence of global setups has been introduced by Canetti *et al.* in [12]. In Supporting Material A.1 we present the setup functionalities that we consider across this work. Namely, the *Global clock* (GC) $\mathcal{G}_{\mathsf{clock}}$ [33,2], the *Random Oracle* (RO) $\mathcal{F}_{\mathsf{RO}}$ [42] and the *Broadcast* (BC) $\mathcal{F}_{\mathsf{BC}}$ [34,31] functionalities.

4 Definition of $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$

We provide our UC treatment of TLE in the $\mathcal{G}_{\mathsf{clock}}$ model by defining the functionality $\mathcal{F}_{\mathsf{TLE}}$, following the approach of [11]. The functionality is described in Figure 1, and at a high level operates as follows. The functionality is parameterized by a delay variable delay. This variable shows the time that a ciphertext needs to be created. There are settings where the ciphertetext generation needs some time, in some cases this time is very small or zero ($\mathsf{delay} = 0$) or noticeable ($\mathsf{delay} = 1$). The simulator $\mathcal{S}$ initially provides $\mathcal{F}_{\mathsf{TLE}}$ with the set of corrupted parties. Each time an encryption query issued by an honest party is handed to $\mathcal{F}_{\mathsf{TLE}}$, the functionality forwards the request to $\mathcal{S}$ without any information about the actual message except the size of the message and the party's identity. The simulator returns the token back to $\mathcal{F}_{\mathsf{TLE}}$ which replies with the message Encrypting to the dummy party. This illustrates both the fact that the ciphertext does not contain any information about the message and that encryption might require some time to be completed. The environment can access the ciphertexts that this party has generated so far by issuing the command

RETRIEVE, where $\mathcal{F}_{\mathsf{TLE}}$ returns all the ciphertexts that are created by that party back to it. It is worth mentioning, that the time labelling that is used in the encryption command refers to an absolute time rather than relative. On the other hand, the construction that we propose for realising $\mathcal{F}_{\mathsf{TLE}}$ is relativistic. That is why, as we see in detail in Section 5, the algorithm accepts the difference between the current time Cl and the time labelling τ as an input. In this way, the algorithm computes the difficulty for the puzzle such that the message can be decrypted when time τ has been reached. In addition, $\mathcal{F}_{\mathsf{TLE}}$ handles the decryption queries in the usual way, unless it finds two messages recorded along the same ciphertext, in which case it outputs $\perp$. This enforces that the encryption/decryption algorithms used by $\mathcal{S}$ should satisfy $\mathsf{Correctness}$. In addition, if $\mathcal{F}_{\mathsf{TLE}}$ finds the requested ciphertext in its database, the recorded time is smaller than the current one (which means that the ciphertext can be decrypted), but the party that requested the decryption of that ciphertext provided an invalid time labelling (labelling smaller than the one recorded in $\mathcal{F}_{\mathsf{TLE}}$'s database), it returns the message INVALID_TIME to that party. In the case where the encryption/decryption queries are issued by corrupted parties, $\mathcal{F}_{\mathsf{TLE}}$ responds according to the instructions of $\mathcal{S}$. When a party receives a decryption request from $\mathcal{Z}$, except from the ciphertext c, it receives as input a time labelling τ. Ideally, τ is the time when c can be decrypted. Of course, the labelling τ can also be different to the decryption time of c. Nevertheless, this does not affect the soundness of $\mathcal{F}_{\mathsf{TLE}}$. Without the labelling, $\mathcal{F}_{\mathsf{TLE}}$ or the engaging party in the real protocol would have to find the decryption time of c which is registered either in the functionality's database (ideal case) or in the party's list of received ciphertexts (real case) and then compare it with the current time Cl.

When a party P advances $\mathcal{G}_{\mathsf{clock}}$, the simulator $\mathcal{S}$ is informed. Then, $\mathcal{S}$ can generate ciphertexts for each tag[2] received from $\mathcal{F}_{\mathsf{TLE}}$ from P and send them to $\mathcal{F}_{\mathsf{TLE}}$ issuing the UPDATE command. Later, $\mathcal{F}_{\mathsf{TLE}}$ will return these to P. This illustrates the fact that after some time ciphertexts are created. In TLE constructions where the encryption and decryption time is equal, $\mathcal{S}$ will force a delay on the ciphertext generation equal to the number of rounds that the ciphertext needs to be decrypted. Thus, the way we model $\mathcal{F}_{\mathsf{TLE}}$ allows us to capture a broader spectrum of TLE constructions (not necessarily efficient) in the context of the Global Clock (GC) model.

Naturally, after some time, ciphertexts are eventually opened and every party, including $\mathcal{S}$, can retrieve the underlying plaintext. For that task, we include the command LEAKAGE. In the vanilla case, $\mathcal{S}$ can retrieve all the messages that can be opened by the current time Cl. However, there are cases where $\mathcal{S}$ can retrieve messages before their time comes. This advantage of $\mathcal{S}$ can be specified by the function leak. This function accepts as input an integer (e.g., the current time Cl) and outputs a progressive integer (e.g., the time that the adversary can

[2] The simulator gives back the ciphertext as this is the case in most encryption functionalities [10,11]. Now, because we allow the simulator to delay the delivery of messages, the simulator needs a handle for updating the functionality's database. Here the tag comes into play and works as a receipt for that call.

decrypt ciphertexts, which is the same or greater than Cl). For more details see Supporting Material C.1.

The time-lock encryption functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$.

It initializes the list of recorded messages/ciphertexts L_{rec} as empty and defines the tag space TAG.

■ Upon receiving $(\mathsf{sid}, \textsc{Corrupt}, \mathbf{P}_{\mathsf{corr}})$ from $\mathcal{S}$, it records the corrupted set $\mathbf{P}_{\mathsf{corr}}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Enc}, m, \tau)$ from $P \notin \mathbf{P}_{\mathsf{corr}}$, it reads the time Cl and does:

1. If $\tau < 0$, it returns $(\mathsf{sid}, \textsc{Enc}, m, \tau, \perp)$ to P.
2. It picks $\mathsf{tag} \xleftarrow{\$} \mathsf{TAG}$ and it inserts the tuple $(m, \mathsf{Null}, \tau, \mathsf{tag}, \mathsf{Cl}, P) \rightarrow L_{\mathsf{rec}}$.
3. It sends $(\mathsf{sid}, \textsc{Enc}, \tau, \mathsf{tag}, \mathsf{Cl}, 0^{|m|}, P)$ to $\mathcal{S}$. Upon receiving the token back from $\mathcal{S}$ it returns $(\mathsf{sid}, \textsc{Encrypting})$ to P.

■ Upon receiving $(\mathsf{sid}, \textsc{Update}, \{(c_j, \mathsf{tag}_j)\}_{j=1}^{p(\lambda)})$ from $\mathcal{S}$, for all $c_j \neq \mathsf{Null}$ it updates each tuple $(m_j, \mathsf{Null}, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, P)$ to $(m_j, c_j, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, P)$

■ Upon receiving $(\mathsf{sid}, \textsc{Retrieve})$ from P, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and it returns $(\mathsf{sid}, \textsc{Encrypted}, \{(m, c \neq \mathsf{Null}, \tau)\}_{\forall (m,c,\tau,\cdot,\mathsf{Cl}',P) \in L_{\mathsf{rec}} : \mathsf{Cl}-\mathsf{Cl}' \geq \mathsf{delay}})$ to P.

■ Upon receiving $(\mathsf{sid}, \textsc{Dec}, c, \tau)$ from $P \notin \mathbf{P}_{\mathsf{corr}}$:

1. If $\tau < 0$, it returns $(\mathsf{sid}, \textsc{Dec}, c, \tau, \perp)$ to P. Else, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and:
   (a) If $\mathsf{Cl} < \tau$, it sends $(\mathsf{sid}, \textsc{Dec}, c, \tau, \textsc{More\_Time})$ to P.
   (b) If $\mathsf{Cl} \geq \tau$, then
   – If there are two tuples $(m_1, c, \tau_1, \cdot, \cdot, \cdot), (m_2, c, \tau_2, \cdot, \cdot, \cdot)$ in L_{rec} such that $m_1 \neq m_2$ and $c \neq \mathsf{Null}$ where $\tau \geq \max\{\tau_1, \tau_2\}$, it returns to P $(\mathsf{sid}, \textsc{Dec}, c, \tau, \perp)$.
   – If no tuple $(\cdot, c, \cdot, \cdot, \cdot, \cdot)$ is recorded in L_{rec}, it sends $(\mathsf{sid}, \textsc{Dec}, c, \tau)$ to $\mathcal{S}$ and returns to P whatever it receives from $\mathcal{S}$.
   – If there is a unique tuple $(m, c, \tau_{\mathsf{dec}}, \cdot, \cdot, \cdot)$ in L_{rec}, then if $\tau \geq \tau_{\mathsf{dec}}$, it returns $(\mathsf{sid}$, $\textsc{Dec}$, c, τ, $m)$ to P. Else, if $\mathsf{Cl} < \tau_{\mathsf{dec}}$, it returns $(\mathsf{sid}$, $\textsc{Dec}$, c, τ, $\textsc{More\_Time})$ to P. Else, if $\mathsf{Cl} \geq \tau_{\mathsf{dec}} > \tau$, it returns $(\mathsf{sid}$, $\textsc{Dec}$, c, τ, $\textsc{Invalid\_Time})$ to P.

■ Upon receiving $(\mathsf{sid}, \textsc{Leakage})$ from $\mathcal{S}$, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and returns $(\mathsf{sid}, \textsc{Leakage}, \{(m, c, \tau)\}_{\forall (m,c,\tau \leq \mathsf{leak}(\mathsf{Cl}),\cdot,\cdot,\cdot) \in L_{\mathsf{rec}}})$ to $\mathcal{S}$.

■ Whatever message it receives from $P \in \mathbf{P}_{\mathsf{corr}}$, it forwards it to $\mathcal{S}$ and vice versa.

Fig. 1. Functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ parameterized by λ, a leakage function leak, a delay variable delay ,interacting with simulator $\mathcal{S}$, parties in $\mathbf{P}$, and global clock $\mathcal{G}_{\mathsf{clock}}$.

5 Realization of $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ via time-lock puzzles

In this section, we present the realization of $\mathcal{F}_{\mathsf{TLE}}$ via a protocol that uses a pair of encryption/decryption algorithms that satisfy a specific security notion that we formally define in Definition 1. We prove that our construction which is based on [39] and [46] is secure with respect to the required security notion.

The general idea of a time-lock puzzle scheme is that the parties have restricted access to a specific computation in any given period of time for solving a puzzle. In [46]'s case that computation is repeated squaring, and in [39] the computation is sequential hash evaluations. Of course, the underlying assumption here is that there is no "better" way to solve that puzzle except for sequentially applying the specific computation. Some of the most prominent proposed time-lock constructions are based on such assumption [46,38,2,39].

In the UC framework, to construct a time-lock protocol we need to abstract such computations through an oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$. The reasoning behind this modelling is simple. In the UC framework, all the parties are allowed to run polynomial time with respect to the protocol's parameter. As a result, it is impossible to impose on a party the restriction that in a specific period of time they can only execute a constant number of computations. This is why we abstract such computations as a functionality/oracle and wrap the oracle with a *functionality wrapper* that restricts the access to the oracle. The approach is similar to the one proposed in [2], for modelling Proof of Work in the Bitcoin protocol.

In the following paragraphs, we present the evaluation oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, the functionality wrapper $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ and the protocol Π_{TLE}. We provide a security definition that captures both *correctness* and *one-wayness* of TLE constructions. The latter is illustrated via an experiment in a game-based style described in Figure 5. We prove that Π_{TLE} UC realises $\mathcal{F}_{\mathsf{TLE}}$ given that the underlying TLE construction satisfies our security definition. Having at hand a UC realisation and given that our ideal functionality $\mathcal{F}_{\mathsf{TLE}}$ captures accurately the concept of what we expect from a TLE scheme, this validates the definition of security of TLE algorithms.

In the following section, we propose a new TLE construction and prove it satisfies our security definition, completing our construction argument. Finally, we provide a stand-alone security definition in the same spirit as *IND-CPA* security, named *IND-CPA-TLE*, which is captured via an experiment. We prove that Astrolabous satisfies this as well.

Our security definition that captures the one-wayness of a TLE construction was enough for having a UC realization. Although one-wayness as a property is very weak when arguing about the security of an encryption scheme, in our case was enough as we do not use the actual construction but we extend it in the random oracle model. On the other hand, such definition in the stand alone model is weak. That was the reason of why we introduced IND-CPA-TLE.

The evaluation functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ The evaluation functionality captures the computation that is needed for a time-lock puzzle to be solved by the designated parties. An explanatory example can be found bellow.

Initially, the functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, as described in Figure 2, creates the list L_{eval} for keeping a record of the queries received so far. Then, upon receiving a query from a party in $\mathbf{P}$, $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ checks if this query has been issued before. If this is the case, it returns the recorded pair. If not, then for the query x it samples the value y from the distribution $\mathbf{D}_x$ and returns to that party the pair (x, y).

The distribution $\mathbf{D}_x$ in cases such as in [38,2,39] is a random value over a specific domain. Thus, $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ is the random oracle in these cases. More precisely, $\mathbf{D}_x = \mathcal{U}\{0, 2^n - 1\}$ where $\mathcal{U}$ is the uniform distribution and $[0, 2^n - 1]$ is its domain, in our example the domain of the random oracle. In that case, the parametrization of $\mathbf{D}$ with x is unnecessary. On the other hand, if we study other time-lock puzzles such as the one in [46], where the computation to solve a puzzle is the repeated squaring, the parametrization of $\mathbf{D}$ with x becomes necessary. More intuition for $\mathbf{D}$ can be found in Supporting Material C.2.

Example 1. Adapting the relative time-lock puzzle of [39] to our modelling approach, the evaluation functionality is instantiated by the random oracle. Let us consider that the solution of the puzzle is the value r. The creator of the puzzle P chooses the desired difficulty of the puzzle, τ. Then, P splits the puzzle r into $q\tau$ equal pieces $r_0, \ldots, r_{q\tau}$ such that $r = r_0 || \ldots || r_{q\tau}$. Here, q is the maximum number of evaluation queries that the party can make to the oracle in one round. Remember that the essence of round can be defined with respect to the functionality $\mathcal{G}_{\mathsf{clock}}$. Next, P makes one call to the random oracle functionality with the values $(r_0, \ldots, r_{q\tau-1})$ and receives back $(y_{r_0}, \ldots, y_{r_{q\tau-1}})$. Note that this call is counted as one. Finally, P creates the puzzle $(r_0, y_{r_0} \oplus r_1, \ldots, y_{r_{q\tau-1}} \oplus r_{q\tau})$ for the secret r. Now, if some party P^* wants to solve the puzzle, it needs to send the query r_0 to the random oracle functionality. Upon receiving the value y_{r_0} back from the random oracle functionality, P^* computes $r_1 = y_{r_0} \oplus (y_{r_0} \oplus r_1)$. Next, it repeats the procedure with the value r_1. Note that, the maximum number of evaluation queries to the functionality oracle in one round is q and thus the puzzle to be solved needs τ rounds. It is worth mentioning that for capturing the limited access to the functionality in the UC framework, a functionality wrapper needs to be defined as it is described in a dedicated Paragraph below.

The evaluation functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}(\mathcal{D}, \mathbf{P})$.

Initializes an empty evaluation query list L_{eval}.

■ Upon receiving $(\mathsf{sid}, \textsc{Evaluate}, x)$ from a party $P \in \mathbf{P}$, it does:

1. It checks if $(x, y) \in L_{\mathsf{eval}}$ for some y. If no such entry exists, it samples y from the distribution $\mathbf{D}_x$ and inserts the pair (x, y) to L_{eval}. Then, it returns $(\mathsf{sid}, \textsc{Evaluated}, x, y)$ to P. Else, it returns the recorded pair.

Fig. 2. Functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ parameterized by λ, a family of distributions $\mathcal{D} = \{\mathbf{D}_x | x \in \mathbf{X}\}$ and a set of parties $\mathbf{P}$.

Functionality wrapper $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}, \mathcal{G}_{\mathsf{clock}}, \mathbf{P})$.

■ Upon receiving $(\mathsf{sid}, \textsc{Corrupt}, \mathbf{P}_{\mathsf{corr}})$ from $\mathcal{S}$, it records the corrupted set $\mathbf{P}_{\mathsf{corr}}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Evaluate}, (x_1, \ldots, x_j))$ from $P \in \mathbf{P} \setminus \mathbf{P}_{\mathsf{corr}}$ it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and does:

1. If there is not a list L^P it creates one, initially as empty. Then it does:
   (a) For every k in $\{1, \ldots, j\}$, it forwards the message $(\mathsf{sid}, \textsc{Evaluate}, x_k)$ to $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$.
   (b) When it receives back all oracle queries, it inserts the tuple-$(\mathsf{Cl}, 1) \in L^P$.
   (c) It returns $(\mathsf{sid}, \textsc{Evaluate}, ((x_1, y_1), \ldots, (x_j, y_j)))$ to P.
2. Else if there is a tuple-$(\mathsf{Cl}, j_{\mathsf{c}}) \in L^P$ with $j_{\mathsf{c}} < q$, then it changes the tuple to $(\mathsf{Cl}, j_{\mathsf{c}} + 1)$, and repeats the above steps 1a,1c.
3. Else if there is a tuple-$(\mathsf{Cl}^*, j_{\mathsf{c}}) \in L^P$ such that $\mathsf{Cl}^* < \mathsf{Cl}$, it updates the tuple as $(\mathsf{Cl}, 1)$, and repeats the above steps 1a,1b,1c.

■ Upon receiving $(\mathsf{sid}, \textsc{Evaluate}, (x_1, \ldots, x_j))$ from $P \in \mathbf{P}_{\mathsf{corr}}$ it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and repeats steps 1,3 except that it maintains the same list, named L^{corr}, for all the corrupted parties.

Fig. 3. The Functionality wrapper $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ parameterized by λ, a number of queries q, functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, $\mathcal{G}_{\mathsf{clock}}$ and parties in $\mathbf{P}$.

The functionality wrapper $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ Our wrapper is defined along the lines of [2]. The functionality wrapper is an ideal functionality parameterized by another ideal functionality, mediating the access to the latter functionality only possible through the wrapper. Moreover, the wrapper restricts the access to the parameter functionality allowing parties to access it only a certain number of times per round. Here, the notion of round is defined with respect to the $\mathcal{G}_{\mathsf{clock}}$ functionality defined in Figure 7. In a nutshell, the wrapper models in the UC setting the limited resources a party has at their disposal for solving the underlying puzzle. Because in UC every party is a PPT ITM, the same holds for the adversary. So, the adversary can interact with any functionality polynomially many times in each round. There are several protocols that hinge their security on the limited computational capabilities of the participants [22,2]. Next, follows the description of $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$. The description of $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ and insightful comments behind its design can be found in Supporting Material C.4. In the rest of this work we use the abbreviation $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ instead of $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}, \mathcal{G}_{\mathsf{clock}}, \mathbf{P})$ when it is obvious from the context.

The protocol Π_{TLE} We are now ready to present the protocol Π_{TLE} which is proved in later Sections that it UC realises the $\mathcal{F}_{\mathsf{TLE}}$ functionality. The protocol consists of the functionality wrapper $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ as described in Figure 3, the global clock $\mathcal{G}_{\mathsf{clock}}$, the random oracle $\mathcal{F}_{\mathsf{RO}}$, the broadcast functionality $\mathcal{F}_{\mathsf{BC}}$ and a set of parties $\mathbf{P}$ (the descriptions can be found in Supporting Material A.1).

Example 2. Recall Example 1 and assume the time-lock puzzle $c = (r_0, y_0 \oplus r_1, \ldots, y_{r_{q\tau_{\mathsf{dec}}-1}} \oplus r_{q\tau_{\mathsf{dec}}})$. If the function $\mathsf{wit\_con}$ is given less than $q\tau_{\mathsf{dec}}$ oracle responses (e.g. $(y_0, \ldots, y_{q\tau_{\mathsf{dec}}-3})$) for the puzzle c, it returns $\perp$ else it returns $w_{\mathsf{dec}} = (r_0, y_0, \ldots, y_{r_{q\tau_{\mathsf{dec}}-1}}, c)$. Note that here, the ciphertext and the puzzle coincide as there is no actual encryption of a message. Thus, $\mathsf{f_{puzzle}}$ is simply the identity function.

Necessity of extending the TLE algorithms: In order to realise $\mathcal{F}_{\mathsf{TLE}}$ with some TLE construction we need to extend a given TLE algorithm in the random oracle model ($\mathcal{F}_{\mathsf{RO}}$). Recall that in $\mathcal{F}_{\mathsf{TLE}}$ all the ciphertexts eventually open. To capture semantic security, the ciphertext contains no information about the actual message, in contrast to the real protocol that contains the encryption of the actual message. So, for the simulator to simulate this difference when the messages are opened, $\mathcal{S}$ must be able to *equivocate* the opening of the ciphertext, else the environment $\mathcal{Z}$ can trivially distinguish the real from the ideal execution of the protocol. When we say that $\mathcal{S}$ equivocates the opening of the ciphertext, it means that $\mathcal{S}$ can open a ciphertext to whatever plaintext message needs to be opened. Equivocation has also been used for other cryptographic primitives, such as bit commitments, where the simulator can equivocate because it knows the trapdoor information related to the *common reference string* (CRS) [37]. This is actually fundamental, unless we restrict the environment's running time. But then we lose the composition theorem.

Our extension, that can be applied to any TLE construction, offers the feature of equivocation but at the expense of assuming the random oracle model. More information and insightful comments can be found in Supporting Material C.5.

Description of protocol Π_{TLE}: Each party P maintains the list of recorded messages/ciphertexts L^P_{rec}, in which the requested messages for encryption by $\mathcal{Z}$ are stored along with the ciphertext of that message (initially stored as Null), a random identifier of the message tag, the time τ that the message should open, the time Cl that it is recorded for the first time and a flag which shows if that message has been broadcast or not to the other parties. When a party receives the broadcast ciphertext, she extracts the underlying puzzle with the function $\mathsf{f_{puzzle}}$ from that ciphertext and stores it along with its difficulty τ_{dec}, the set of oracle queries/responses issued to the oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ so that puzzle to be solved with the help of the preparation function state, the time Cl that this tuple was last time updated, a counter j that shows how many queries are issued for that puzzle this turn and a counter j_t that shows the total number of queries issued for that puzzle.

If party accepts encryption requests by $\mathcal{Z}$, she returns the message ENCRYPTING, delaying the encryption for one round. When a party either receives a clock advancement command or decryption, she performs the procedure *PuzzleEncryption*, in which the party issues all her q oracle queries both for solving and encrypting the pending messages for that round. More details on the description of Π_{TLE} can be found in Supporting Material C.6.

Table 1. Functions and list each party holds in Π_{TLE}.

Functions/Lists	**Description**
$\mathbf{P}, \mathbb{N}, \mathbf{Q}, \mathbf{R}, \mathbf{C}, \mathbf{M}, \mathbf{W}$	The space of time-lock puzzles, integers, oracle queries and responses to/from $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, ciphertexts, plaintexts and witnesses.
$e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}} : \mathbf{M} \times \mathbb{N} \times \mathbf{Q}/\mathbf{R} \to \mathbf{C}$	The encryption algorithm takes as input the plaintext, the puzzle difficulty and the pair of oracle queries/responses so that the puzzle can be created.
$d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}} : \mathbf{C} \times \mathbf{W} \to \mathbf{M}$	The decryption algorithm takes as input the ciphertext and the secret key.
$\mathsf{f}_{\mathsf{state}} : \mathbf{P} \times \mathbb{N} \times \mathbf{Q}/\mathbf{R} \to \mathbf{Q}$	It prepares the next oracle query to $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$. Specifically, it accepts a puzzle, the number of queries that need to be prepared and all the previous queries and responses from the oracle.
$\mathsf{f}_{\mathsf{puzzle}} : \mathbf{C} \to \mathbf{P}$	It extracts the time-puzzle from a ciphertext.
$\mathsf{puz\_cr} : \mathbf{M} \times \mathbb{N} \to \mathbf{Q}$	The puzzle creation function takes as input the plaintext and the desired difficulty and creates the oracle queries so that a puzzle for that plaintext of that difficulty can be created.
$\mathsf{wit\_con} : \mathbf{Q}/\mathbf{R} \times \mathbb{N} \times \mathbf{P} \to \mathbf{W}$	The witness construction function that returns the solution of the puzzle or the witness if that is possible.
L^P_{rec}	The list of the generated ciphertexts.
L^P_{puzzle}	The list of the recorded oracle queries for puzzle solving.
$(z, \tau, \{(\mathsf{state}^z_k, y_k)\}^{j_t}_{k=0}, j_c, j_t)$	The tuple contains a puzzle z, the difficulty of the puzzle τ, the pairs of oracle queries/responses to solve puzzle z, the current number j_c of oracle queries in that round and the total number of oracle queries j_t.

$\Pi_{\mathsf{TLE}}(\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}), e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, \mathsf{f}_{\mathsf{state}}, \mathsf{wit\_con}, \mathsf{f}_{\mathsf{puzzle}}, \mathsf{puz\_cr}, \mathcal{G}_{\mathsf{clock}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{BC}}, \mathbf{P})$.

Each party maintains the list of recorded messages/ciphertexts L^P_{rec} and the list of the recorded oracle queries for puzzle solving L^P_{puzzle}, initially as empty, a tag space TAG and the algorithms $(e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}})$. Moreover, she follows the procedure described below:

Puzzle:

1. *Preparing queries for puzzle creation:* She collects all tuples $\{(m_j, \mathsf{Null}, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, 0) \in L^P_{\mathsf{rec}}\}^{p_1(\lambda)}_{j=1}$ for $\mathsf{Cl}_j = \mathsf{Cl}$. She picks $\{r^j_1 \overset{\$}{\leftarrow} \{0,1\}^{p^*(\lambda)}\}^{p_1(\lambda)}_{j=1}$. For each j she computes $\mathsf{puz\_cr}(r^j_1, \tau_j - (\mathsf{Cl}+1)) \to \{x_k\}^{p_2(\lambda)}_{k=1}$.
2. *Puzzle solving:* For $(j_l = 0,\ j_l < q,\ j_l{+}{+})$ she collects all $\{\mathsf{state}^{z_n}_{j_t}\}^{p_3(\lambda)}_{n=1}$, such that $(z_n, \tau_{\mathsf{dec}}, \{(\mathsf{state}^{z_n}_k, y_k)\}^{j_t}_{k=0}, \mathsf{Cl}, 0, j_t) \in L^P_{\mathsf{puzzle}}$ (see command 5 for initialization).

(a) *Parallelize puzzle creation queries and puzzle solve:* If $j_l = 0$, she sends $(\mathsf{sid}, \textsc{Evaluate}, \{\mathsf{state}_{j_t}^{z_n}\}_{n=1}^{p_3(\lambda)} \cup \{x_k\}_{k=1}^{p_2(\lambda)})$ to $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ and receives back $(\mathsf{sid}, \textsc{Evaluate}, \{(\mathsf{state}_{j_t}^{z_n}, y_{j_t}^*)\}_{n=1}^{p_3(\lambda)} \cup \{(x_k, y_k)\}_{k=1}^{p_2(\lambda)})$. Else she sends $(\mathsf{sid}, \textsc{Evaluate}, \{\mathsf{state}_{j_t}^{z_n}\}_{n=1}^{p_3(\lambda)})$ to $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$.

(b) *Parallelize puzzle creation queries and puzzle solve:* If $j_l = 0$, she sends $(\mathsf{sid}, \textsc{Evaluate}, \{\mathsf{state}_{j_t}^{z_n}\}_{n=1}^{p_3(\lambda)} \cup \{x_k\}_{k=1}^{p_2(\lambda)})$ to $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ and receives back $(\mathsf{sid}, \textsc{Evaluate}, \{(\mathsf{state}_{j_t}^{z_n}, y_{j_t}^*)\}_{n=1}^{p_3(\lambda)} \cup \{(x_k, y_k)\}_{k=1}^{p_2(\lambda)})$. Else she sends $(\mathsf{sid}, \textsc{Evaluate}, \{\mathsf{state}_{j_t}^{z_n}\}_{n=1}^{p_3(\lambda)})$ to $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$.

(c) *Update the record:* In each case, she updates each tuple as $(z_n, \tau_{\mathsf{dec}}, \{(\mathsf{state}_k^{z_n}, y_k)\}_{k=0}^{j_t+1}, \mathsf{Cl}, j_l++, j_t++)$ where $\mathsf{state}_{j_t+1}^{z_n} = \mathsf{f}_{\mathsf{state}}(z_n, j_t, \{(\mathsf{state}_k^{z_n}, y_k)\}_{k=0}^{j_t})$, $y_{j_t+1} = \mathsf{Null}$ and $y_{j_t} \leftarrow y_{j_t}^*$. In case that $j_l = q$, she changes the Cl in the tuple to $\mathsf{Cl}+1$ and $j_l = 0$.

Encryption:

1. *Time-lock encryption:* She computes $\{c_1^j \leftarrow e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}(r_1^j, \{(x_k, y_k)\}_{k=1}^{p_2(\lambda)}, \tau_j - (\mathsf{Cl}+1))\}_{j=1}^{p_1(\lambda)}$.
2. *Extended encryption:* For each r_1^j, she sends $(\mathsf{sid}, \textsc{Query}, r_1^j)$ to $\mathcal{F}_{\mathsf{RO}}$. Upon receiving $(\mathsf{sid}, \textsc{Random\_Oracle}, r_1^j, h^j)$ from $\mathcal{F}_{\mathsf{RO}}$, P sends $(\mathsf{sid}, \textsc{Query}, r_1^j||m_j)$ to $\mathcal{F}_{\mathsf{RO}}$. Upon receiving $(\mathsf{sid}, \textsc{Random\_Oracle}, r_1^j||m_j, c_3^j)$ from $\mathcal{F}_{\mathsf{RO}}$, she computes $c_j \leftarrow (c_1^j, h \oplus m, c_3^j)$ and updates the tuple $(m_j, c_j, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, 0) \rightarrow L_{\mathsf{rec}}^P$.

■ Upon receiving $(\mathsf{sid}, \textsc{Enc}, m, \tau)$ from $\mathcal{Z}$, P reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and if $\tau < 0$ she returns $(\mathsf{sid}, \textsc{Enc}, m, \tau, \perp)$ to $\mathcal{Z}$. Else, it does:

1. She picks $\mathsf{tag} \xleftarrow{\$} \mathsf{TAG}$ and she inserts the tuple $(m, \mathsf{Null}, \tau, \mathsf{tag}, \mathsf{Cl}, 0) \rightarrow L_{\mathsf{rec}}^P$.
2. She returns $(\mathsf{sid}, \textsc{Encrypting})$ to $\mathcal{Z}$.

■ Upon receiving $(\mathsf{sid}, \textsc{Advance\_Clock})$ from $\mathcal{Z}$, P reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. She executes both *Puzzle* and *Encryption* procedure. Then, she sends $(\mathsf{sid}, \textsc{Broadcast}, \{(c_j, \tau_{\mathrm{j}})\}_{j=1}^{p_1(\lambda)})$ to $\mathcal{F}_{\mathsf{BC}}$. Upon receiving $(\mathsf{sid}, \textsc{Broadcasted}, \{(c_j, \tau_{\mathrm{j}})\}_{j=1}^{p_1(\lambda)})$ from $\mathcal{F}_{\mathsf{BC}}$, for each j she updates each tuple $(m_j, \mathsf{Null}, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, 0)$ to $(m_j, c_j, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, 1)$ and sends $(\mathsf{sid}, \textsc{Advance\_Clock})$ to $\mathcal{G}_{\mathsf{clock}}$. ■ Upon receiving $(\mathsf{sid}, \textsc{Retrieve})$ from $\mathcal{Z}$, P reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and returns $(\mathsf{sid}, \textsc{Encrypted}, \{(m_j, c_j, \tau_j) : (m_j, c_j, \tau_j, \cdot, \mathsf{Cl}_j, 1) \in L_{\mathsf{rec}}^P : \mathsf{Cl} - \mathsf{Cl}_j \geq 1\})$ to $\mathcal{Z}$. ■ Upon receiving $(\mathsf{sid}, \textsc{Broadcast}, \{(c_j, \tau_j)\}_{j=1}^{p_1(\lambda)})$ from $\mathcal{F}_{\mathsf{BC}}$ where $c_j = (c_1^j, c_2^j, c_3^j)$, P reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and does for every j:

1. She computes $\mathsf{state}_0^{\mathsf{f}_{\mathsf{puzzle}}(c_1^j)} \leftarrow \mathsf{f}_{\mathsf{state}}(\mathsf{f}_{\mathsf{puzzle}}(c_1^j), 0, \mathsf{Null})$.
2. She creates the tuple-$(\mathsf{f}_{\mathsf{puzzle}}(c_1^j), \tau_{\mathsf{dec}}, \{(\mathsf{state}_0^{\mathsf{f}_{\mathsf{puzzle}}(c_1^j)}, \mathsf{Null})\}, \mathsf{Cl}, 0, 0)$ and stores it in L_{puzzle}^P.

■ Upon receiving $(\mathsf{sid}, \textsc{Dec}, c := (c_1, c_2, c_3), \tau_{\mathsf{dec}})$ from $\mathcal{Z}$, P reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. Then she does:

1. If $\tau_{\mathsf{dec}} < 0$, she returns $(\mathsf{sid}, \textsc{Dec}, c, \tau_{\mathsf{dec}}, \perp)$ to $\mathcal{Z}$.
2. If $\mathsf{Cl} < \tau_{\mathsf{dec}}$, she returns $(\mathsf{sid}, \textsc{Dec}, c, \tau_{\mathsf{dec}}, \textsc{More\_Time})$.
3. She searches for a tuple $(\mathsf{f}_{\mathsf{puzzle}}(c_1), \tau, \{(\mathsf{state}_k^{\mathsf{f}_{\mathsf{puzzle}}(c_1)}, y_k)\}_{k=0}^{j_t}, \mathsf{Cl}, q, j_t)$ in L^P_{puzzle}. If $\tau_{\mathsf{dec}} < \tau \leq \mathsf{Cl}$ then she returns $(\mathsf{sid}, \textsc{Dec}, c, \tau_{\mathsf{dec}}, \textsc{Invalid\_time})$ to $\mathcal{Z}$.
4. She computes $w_{\tau_{\mathsf{dec}}} \leftarrow \mathsf{wit\_con}(\{(\mathsf{state}_k^{\mathsf{f}_{\mathsf{puzzle}}(c_1)}, y_k)\}_{k=0}^{j_t}, \tau_{\mathsf{dec}}, \mathsf{f}_{\mathsf{puzzle}}(c_1))$.
5. She runs $x \leftarrow d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}(c_1, w_{\tau_{\mathsf{dec}}})$ and she sends $(\mathsf{sid}, \textsc{Query}, x)$ to $\mathcal{F}_{\mathsf{RO}}$. Upon receiving $(\mathsf{sid}, \textsc{Random\_Oracle}, x, h)$ from $\mathcal{F}_{\mathsf{RO}}$, she computes $m \leftarrow h \oplus c_2$. She sends $(\mathsf{sid}, \textsc{Query}, x||m)$ to $\mathcal{F}_{\mathsf{RO}}$. Upon receiving $(\mathsf{sid}, \textsc{Random\_Oracle}, x||m, c_3^*)$ from $\mathcal{F}_{\mathsf{RO}}$: If $c_3 \neq c_3^*$, she returns to $\mathcal{Z}$ $(\mathsf{sid}, \textsc{Dec}, c, \tau_{\mathsf{dec}}, \perp)$. Else, she returns to $\mathcal{Z}$ $(\mathsf{sid}, \textsc{Dec}, c, \tau_{\mathsf{dec}}, m)$.
6. If such tuple does not exist then she returns $(\mathsf{sid}, \textsc{Dec}, c, \tau_{\mathsf{dec}}, \perp)$ to $\mathcal{Z}$.

Fig. 4. The Protocol Π_{TLE} in the presence of a functionality wrapper $\mathcal{W}_q$, an evaluation functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, a random oracle $\mathcal{F}_{\mathsf{RO}}$, a broadcast functionality $\mathcal{F}_{\mathsf{BC}}$, a global clock $\mathcal{G}_{\mathsf{clock}}$, where $e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}$, $\mathsf{f}_{\mathsf{state}}$, $\mathsf{wit\_con}$ and $\mathsf{f}_{\mathsf{puzzle}}$ are hard-coded in each party in $\mathbf{P}$.

5.1 Security definitions of (computational-puzzle-based) time-lock encryption

In this Subsection we turn to standalone security and focus only on TLE schemes based on computational puzzles. We identify minimal standalone requirements sufficient for a computational-puzzles-based construction to provide a UC realization of our $\mathcal{F}_{\mathsf{TLE}}$ functionality. We specify these minimal requirements, namely Correctness and qSecurity, in the game-base style. In Section 6, we define IND-CPA security for (computational-puzzle-based) TLE schemes.

Intuitively, the Correctness property states that the decryption of the ciphertext with underlying plaintext m results in the message m itself with high probability provided that the underlying time-lock puzzle has been solved. The qSecurity property is described in a game-based style via the experiment in Figure 5 and states that an adversary can win the experiment only with a very small probability. Specifically, the experiment captures the one-way security of a TLE scheme as in the concept of *one-way functions* security [27,26]. Although indistinguishability, like in *IND-CPA* security [26,36], is stronger than the hardness to reverse a function, for our purpose of achieving UC realization (Theorem 1) it is enough. This is possible because we extend our TLE construction into a bigger one in the random oracle model and we rely on the hardness of inverting the underlying TLE construction. Because of that, in Subsection 6.3, we provide an indistinguishability game-based definition, similar to IND-CPA but in the context of TLE so that we can argue about the security of a TLE construction even in the standalone model.

In Figure 5, we present the experiment $\mathbf{EXP}_{\mathsf{TLE}}$ in the presence of a challenger Ch and an adversary $\mathcal{B}$. More details on the description of $\mathbf{EXP}_{\mathsf{TLE}}$ can be found in Supporting Material C.7.

The experiment $\mathbf{EXP}_{\mathsf{TLE}}(\mathcal{B}, \mathcal{O}_{\mathsf{eval}}, e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}}, \mathsf{f}_{\mathsf{state}}, \mathsf{f}_{\mathsf{puzzle}}, q)$

Initialization Phase.

■ Ch is initialized with $e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}}$ and sends them to $\mathcal{B}$. In addition it creates a local time counter $\mathsf{Cl}_{\mathsf{exp}}$.

Learning Phase.

■ When $\mathcal{B}$ issues the query $(\textsc{Evaluate}, x)$ to $\mathcal{O}_{\mathsf{eval}}$ through the Ch, he gets back $(\textsc{Evaluate}, x, y)$.

■ $\mathcal{B}$ can request the encryption of a message $m \in \mathbf{M}_\lambda$ with time label τ_{dec} by sending $(\textsc{Enc}, m, \tau_{\mathsf{dec}})$ to Ch.

■ When Ch receives a $(\textsc{Enc}, m, \tau_{\mathsf{dec}})$ request from $\mathcal{B}$, it runs the algorithm $e_{\mathcal{O}_{\mathsf{eval}}}(m, \tau_{\mathsf{dec}}) \rightarrow c$ and returns c to $\mathcal{B}$.

■ Ch increases $\mathsf{Cl}_{\mathsf{exp}}$ by 1 for every q queries $\mathcal{B}$ issues to $\mathcal{O}_{\mathsf{eval}}$.

■ $\mathcal{B}$ can request the decryption of a ciphertext c by sending $(\textsc{Dec}, c, w_\tau)$ to Ch. Then, Ch just runs the algorithm $d_{\mathcal{O}_{\mathsf{eval}}}(c, w_\tau) \rightarrow y \in \{m, \perp\}$ and returns to $\mathcal{B}$ $(\textsc{Dec}, c, w_\tau, y)$.

Challenge Phase.

■ $\mathcal{B}$ can request for a single time a challenge from Ch by sending $(\textsc{Challenge}, \tau)$. Then, Ch picks a value $r \overset{\$}{\leftarrow} \mathbf{M}_\lambda$ and sends $(\textsc{Challenge}, \tau, c_r \leftarrow e_{\mathcal{O}_{\mathsf{eval}}}(r, \tau - \mathsf{Cl}_{\mathsf{exp}}))$ to $\mathcal{B}$. Then, $\mathcal{B}$ is free to repeat the *Learning Phase*.

■ $\mathcal{B}$ sends as the answer of the challenge the message $(\textsc{Challenge}, \tau, c_r, r^*)$ to Ch.

■ If $(r^* = r) \wedge (\tau > \mathsf{Cl}_{\mathsf{exp}})$ (i.e. $\mathcal{B}$ manages to decrypt c_r before the decryption time comes) then $\mathbf{EXP}_{\mathsf{TLE}}$ outputs 1. Else, $\mathbf{EXP}_{\mathsf{TLE}}$ outputs 0.

Fig. 5. Experiment $\mathbf{EXP}_{\mathsf{TLE}}$ for a number of queries q, function $\mathsf{f}_{\mathsf{state}}$, message domain $\mathbf{M}_\lambda$, algorithms $e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}}$ in the presence of an adversary $\mathcal{B}$, oracle $\mathcal{O}_{\mathsf{eval}}$ and a challenger Ch all parameterized by 1^λ.

Definition 1. *A* one-way secure time-lock encryption scheme *with respect to an evaluation oracle* $\mathcal{O}_{\mathsf{eval}}$*, a relation* $\mathsf{R}_{\mathcal{O}_{\mathsf{eval}}}$*, a state function* $\mathsf{f}_{\mathsf{state}}$*, puzzle function* $\mathsf{f}_{\mathsf{puzzle}}$ *and a witness construction function* $\mathsf{wit\_con}$ *for message space* $\mathbf{M}$ *and a security parameter* λ *is a pair of PPT algorithms* $(e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}})$ *such that:*

– $e_{\mathcal{O}_{\mathsf{eval}}}(m, \tau_{\mathsf{dec}})$*: The encryption algorithm takes as input message a* $m \in \mathbf{M}$*, an integer* $\tau_{\mathsf{dec}} \in \mathbb{N}$ *and outputs a ciphertext* c*.*
– $d_{\mathcal{O}_{\mathsf{eval}}}(c, w_{\tau_{\mathsf{dec}}})$*: The decryption algorithm takes as input* $w_{\tau_{\mathsf{dec}}} \in \{0,1\}^*$ *and a ciphertext* c*, and outputs a message* $m \in \mathbf{M}$ *or* $\perp$*.*

The pair $(e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}})$ *satisfies the following properties:*

1. Correctness*: For every* $\lambda, \tau_{\mathsf{dec}} \in \mathbb{N}, m \in \mathbf{M}$ *and* $w_{\tau_{\mathsf{dec}}}$*, it holds that*

$$\Pr\left[\begin{array}{l} m' \leftarrow d_{\mathcal{O}_{\mathsf{eval}}}(e_{\mathcal{O}_{\mathsf{eval}}}(m, \tau_{\mathsf{dec}}), w_{\tau_{\mathsf{dec}}}) \\ \mathsf{R}_{\mathcal{O}_{\mathsf{eval}}}(w_{\tau_{\mathsf{dec}}}, (\mathsf{f}_{\mathsf{puzzle}}(c), \tau_{\mathsf{dec}})) \end{array} : m' = m\right] > 1 - \mathsf{negl}(\lambda)$$

where $w_{\tau_{\mathsf{dec}}}$ *can be constructed from the received responses of* $\mathcal{O}_{\mathsf{eval}}$ *and function* wit_con *as it is described in both Table 1 and Figure 4.*
2. qSecurity*: For every PPT adversary* $\mathcal{B}$ *with access to oracle* $\mathcal{O}_{\mathsf{eval}}$*, the probability to win the experiment* $\mathbf{EXP}_{\mathsf{TLE}}$ *and thus output* 1 *in Figure 5 is* $\mathsf{negl}(\lambda)$*.*

5.2 Proof of UC realizing $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$

In this Subsection we show that if the TLE scheme used in protocol Π_{TLE} in Figure 4 is a secure time-lock encryption scheme according to Definition 1 then the protocol Π_{TLE} UC realizes $\mathcal{F}_{\mathsf{TLE}}$. We provide the proof of the theorem below in Supporting Material C.8.

Theorem 1. *Let* $(e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}})$ *be a pair of* `encryption/decryption` *algorithms that satisfies Definition 1. Then, the protocol* Π_{TLE} *in Figure 4 UC-realizes functionality* $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ *in the* $(\mathcal{W}_q(\mathcal{F}^*_{\mathsf{RO}}), \mathcal{G}_{\mathsf{clock}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{BC}})$*-hybrid model with leakage function* $\mathsf{leak}(x) = x+1$, $\mathsf{delay} = 1$*, where* $\mathcal{F}_{\mathsf{RO}}$ *and* $\mathcal{F}^*_{\mathsf{RO}}$ *are two distinct random oracles.*

On the importance of instantiating $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ *with* $\mathcal{F}^*_{\mathsf{RO}}$*:* In our proof, we instantiate the functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ with $\mathcal{F}^*_{\mathsf{RO}}$, so that $\mathcal{Z}$ cannot bypass the interaction with the functionality wrapper and thus breach the security argument of our proof. For more information and insightful comments see Supporting Material C.9.

Efficiency cost for UC security: As we have seen, to provide UC security we need to extend a secure TLE construction (according to Definition 1) in the RO model. Specifically, in step 2 of the **Encryption** procedure of Figure 4, the party makes two extra calls to the $\mathcal{F}_{\mathsf{RO}}$ functionality. In reality, this means two extra hash function evaluations, which is very efficient [25]. Thus, the performance is not affected by providing UC security.

Relation between absolute and relativistic time: Our $\mathcal{F}_{\mathsf{TLE}}$ captures naturally the concept of TLE in absolute time (messages are encrypted at time t_0 so that they open at time t_m). On the other hand, the construction that UC realizes our functionality is a relativistic one in the sense that a message is protected for a period of time $(t_m - t_0)$.

6 Astrolabous: A UC-secure TLE construction

We present and prove that our relative TLE construction is a secure time-lock encryption scheme according to Definition 1. Our scheme combines the construction of [39] and [46].

First, we present our TLE construction, namely *Astrolabous*, and the proof of security, i.e. Astrolabous satisfies Definition 1. Finally, for the sake of completeness, we present the equivocable Astrolabous algorithm, which is the algorithm that is used in the hybrid protocol in Figure 4.

We did not adopt any of the TLE constructions provided in [46] and [39] because they can not provide us with the necessary security properties we are seeking in our theoretical framework so that we can UC realise $\mathcal{F}_{\mathsf{TLE}}$. More details and insightful comments can be found in Supporting Material C.9.

Description of the* Astrolabous *scheme Initially, we provide the necessary glossary in Table 2. We name our construction Astrolabous from the ancient Greek clock device Astrolabe, which was used by the astronomers of that era to perform different types of calculations including the measurement of the altitude above the horizon of a celestial body, identification of stars and the determination of the local time.

We refer to the encryption/decryption algorithms of the Astrolabous scheme in Subsection 6.1 as $\mathsf{AST.enc}, \mathsf{AST.dec}$ where AST is the abbreviation of $\mathsf{Astrola-bous}$. In Subsection 6.2, we refer to the equivocable encryption/decryption algorithms as $\mathsf{EAST.enc}, \mathsf{EAST.dec}$ where the letter E indicates the extended algorithms of the Astrolabous scheme.

Table 2. The glossary of Astrolabous scheme.

Notation	Description
$\mathsf{E} = (\mathsf{enc}, \mathsf{dec})$	A symmetric key encryption scheme.
$\mathcal{H}, \mathcal{G}$	Two hash functions (modelled as random oracles.)
$b \xleftarrow{\$} \mathsf{D}$	b is sampled uniformly at random from D.
$\mathsf{X.enc}, \mathsf{X.dec}$	Encryption and decryption algorithm respectively of scheme X.
$\oplus$	The XOR bit operation, e.g. $0 \oplus 1 = 1, 1 \oplus 1 = 0$.
$x\|\|y$	The concatenation of two bit strings x and y.

6.1 AST scheme description ($\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}, \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}$)

$\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}(m, \tau_{\mathsf{dec}})$**:** The algorithm accepts as input the message m and the time-lock's puzzle difficulty τ_{dec} [3] and does:

- Picks $k_{\mathsf{E}} \xleftarrow{\$} \mathsf{K}_{\mathsf{E}}$, where K_{E} is the key space of the symmetric encryption scheme E and the size of the key is equal to the domain of the hash function $\mathcal{H}$ equal to $p_1(\lambda)$. Then compute $c_{m,k_{\mathsf{E}}} \leftarrow \mathsf{enc}(m, k_{\mathsf{E}})$.

[3] Note that this time difficulty is relative, that means that it specifies the duration for solving the puzzle rather than the specific date at which the puzzle should be solved.

– It picks $r_0||r_1||\dots||r_{q\tau_{\mathsf{dec}}-1} \xleftarrow{\$} \{0,1\}^{\mathsf{p}_2(\lambda)}$ and computes $c_{k_\mathsf{E},\tau_{\mathsf{dec}}} \leftarrow (r_0, r_1 \oplus \mathcal{H}(r_0), r_2 \oplus \mathcal{H}(r_1), \dots, k_\mathsf{E} \oplus \mathcal{H}(r_{q\tau_{\mathsf{dec}}-1})$[4].
– It outputs $c = (\tau_{\mathsf{dec}}, c_{m,k_\mathsf{E}}, c_{k_\mathsf{E},\tau_{\mathsf{dec}}})$ as the ciphertext.

$\mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}(c, w_{\tau_{\mathsf{dec}}})$**:** The algorithm accepts as input the ciphertext c of the form $(\tau_{\mathsf{dec}}, c_{m,k_\mathsf{E}}, c_{k_\mathsf{E},\tau_{\mathsf{dec}}})$ and the witness $w_{\tau_{\mathsf{dec}}} = (r_0, \mathcal{H}(r_0), \mathcal{H}(r_1), \dots, \mathcal{H}(r_{q\tau_{\mathsf{dec}}-1}), c)$ that can be computed by issuing $q\tau_{\mathsf{dec}}$ random oracle queries. Specifically, to solve the puzzle the first oracle query is r_0 and the response $\mathcal{H}(r_0)$. Then, the decryptor computes the value r_1 from c_{k_E} by using the XOR operation such as $r_1 \leftarrow c_{k_\mathsf{E},\tau_{\mathsf{dec}}}[1] \oplus \mathcal{H}(r_0)$. Similarly, it computes the pair of values $(r_2, \mathcal{H}(r_2)), \dots, (r_{q\tau_{\mathsf{dec}}-1}, \mathcal{H}(r_{q\tau_{\mathsf{dec}}-1}))$. Then it does:

– It computes $k_\mathsf{E} = \mathcal{H}(r_{q\tau_{\mathsf{dec}}-1}) \oplus c_{k_\mathsf{E},\tau_{\mathsf{dec}}}[q\tau_{\mathsf{dec}}]$, where $c_{k_\mathsf{E},\tau_{\mathsf{dec}}}[j]$ indicates the jth element in vector $c_{k_\mathsf{E},\tau_{\mathsf{dec}}}$.
– It computes and outputs $m \leftarrow \mathsf{dec}(c_{m,k_\mathsf{E}}, k_\mathsf{E})$.

In Table 3, we summarize the oracle, algorithms, functions and relation that define a TLE scheme as in Definition 1. We instantiate these to specify our TLE construction.

Table 3. Oracle, algorithms, functions and relation that define a TLE construction.

TLE items	**Description**
$\mathcal{O}_{\mathsf{eval}}$	The oracle to which the parties issue queries for solving/creating time-lock puzzles.
$(e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}})$	The pair of encryption/decryption algorithms with respect to the oracle $\mathcal{O}_{\mathsf{eval}}$.
$\mathsf{f}_{\mathsf{state}}$	The state function that prepares the next oracle query to $\mathcal{O}_{\mathsf{eval}}$.
$\mathsf{f}_{\mathsf{puzzle}}$	The puzzle function that extracts the time-lock puzzle from a given ciphertext.
$\mathsf{wit\_con}$	The witness construction function that returns the solution of the puzzle or the witness if that is possible.
$\mathsf{R}_{\mathcal{O}_{\mathsf{eval}}}$	The relation that specifies when a witness w is a solution to a puzzle c with difficulty τ.

We instantiate the items from Table 3 based on our construction as shown below.

1. The oracle $\mathcal{O}_{\mathsf{eval}}$ is the random oracle RO.
2. The encryption and decryption algorithms $(e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}})$ are described as $\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}, \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}$. Our algorithm is relative, meaning that we define the difficulty of the time-lock puzzle rather than the specific time that the

[4] To do this efficiently all the hash queries can be performed simultaneously as k_E and $r_0||r_1||\dots||r_{q\tau_{\mathsf{dec}}-1}$ are known. In the UC setting, the party sends $(\mathsf{sid}, \text{EVALUATE}, \tau_{\mathsf{dec}})$ to $\mathcal{W}_q$ and receives back $(\mathsf{sid}, \text{EVALUATE}, \tau_{\mathsf{dec}}, \{(r_j, y_j)\}_{j=0}^{q\tau_{\mathsf{dec}}-1})$.

message will eventually open. For our algorithms to be compatible with the UC setting, for a given time τ_{dec} we must define the difficulty of the puzzle. In that case, given the current time is Cl, the puzzle complexity is $\tau_{\mathsf{dec}} - \mathsf{Cl}$. The time τ_{dec} gives us the essence of absolute time that a ciphertext should be opened. On the other hand, both constructions in [39,46] function in relative time. To compute relative time, both values Cl and τ_{dec} are provided to $e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}$.

3. The state function $\mathsf{f_{state}}$ for a ciphertext $c = (\tau_{\mathsf{dec}}, c_{m,k_{\mathsf{E}}}, c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}})$ as described previously, is defined as:

$$\mathsf{f_{state}}(c, 0, \mathsf{Null}) = c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}}[0] \tag{1}$$

and $\forall j \in \{1, \ldots, q(\tau_{\mathsf{dec}} - \mathsf{Cl}) - 1\}$ it holds that:

$$\mathsf{f_{state}}(c, j, y = \mathcal{H}(r_{j-1})) = y \oplus c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}}[j] \tag{2}$$

4. The puzzle function $\mathsf{f_{puzzle}}$ for a ciphertext $c = (\tau_{\mathsf{dec}}, c_{m,k_{\mathsf{E}}}, c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}})$ is defined as:

$$\mathsf{f_{puzzle}}(c) = c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}} \tag{3}$$

5. The witness construction function $\mathsf{wit\_con}$ accepts the input described in Figure 4 and outputs the witness described in the same figure. More details can be found in Supporting Material C.6.
6. A pair $(w_{\tau_{\mathsf{dec}}} = (r_0, \mathcal{H}(r_0), \mathcal{H}(r_1), \ldots, \mathcal{H}(r_{q(\tau_{\mathsf{dec}} - \mathsf{Cl})-1}))$, $(\mathsf{f_{puzzle}}(c), \tau))$ is in $\mathsf{R}_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}$, where $w_{\tau_{\mathsf{dec}}}$ and c as appeared in the description of $\mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}$, if $|w_{\tau_{\mathsf{dec}}}| = |\mathsf{f_{puzzle}}(c)|$ and $w[j] = c_{k_{\mathsf{E}},\tau_{\mathsf{dec}}}[j] \oplus \mathcal{H}(w[j-1])$ for all $j \in [0, q(\tau_{\mathsf{dec}} - \mathsf{Cl}) - 2]$, where $w[-1] = 1$.

The following theorem states that our TLE construction satisfies Definition 1. The proof is provided in Supporting Material D.1.

Theorem 2. *Let* $\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}, \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}$ *be the pair of* encryption/decryption *algorithms just described. If the underlying symmetric encryption scheme* E *satisfies* $\mathsf{IND-CPA}$ *security and* correctness*, then the pair* $(\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}, \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}})$ *is a secure TLE scheme according to Definition 1 in the random oracle model.*

Verifiability and efficiency of Astrolabous: Astrolabous is efficiently verifiable. Solvers can present all the hash evaluations and, similar to puzzle creation, the verifier can in parallel evaluate them and recreate the puzzle. If both puzzles match it accepts the solution. So for a puzzle set for computation time τ, *i.e.* requiring $q\tau$ sequential hash evaluations to be solved, the verification can be parallelized, *i.e.* $q\tau$ parallel hash evaluations.

6.2 Equivocable Astrolabous scheme description ($\mathsf{EAST.enc}_{\mathsf{E},\mathcal{H},\mathcal{G}}$, $\mathsf{EAST.dec}_{\mathsf{E},\mathcal{H},\mathcal{G}}$)

For our purposes, it is not enough to directly adopt a TLE construction such as Astrolabous and make security claims in the UC framework because we cannot equivocate, which is essential. For that reason, in our hybrid protocol in Figure 4 we extend the input TLE construction in order for our security claims to be compatible with the UC framework and in particular with the UC treatment of time. Specifically, in Figure 4, in procedure **Puzzle**, the party prepares the time-lock puzzles for encrypting a random string in step 1 of the **Encryption** procedure. Then, in step 2 the party makes two calls to the random oracle and extends the previously resulting ciphertext by two arguments for equivocation and non-malleability as explained above. This procedure is initiated by the party every time she advances the clock. So the execution of the Extended Astrolabous, as it appeared in Figure 4 in both procedures, needs to be interleaved with the interaction of parties with the global clock. Because of this interaction, extended Astrolabous cannot be given black-box (*e.g.* as protocol input), it rather needs to be described as part of the Π_{TLE} protocol itself. However, outside of the UC framework, parties will actually use the extended algorithms as they do not actually interact with any global clock. This is merely the result of the way time is treated in the UC $\mathcal{G}_{\mathsf{clock}}$ model. So morally, the Equivocable Astrolabous TLE scheme satisfies the UC security notion captured by our $\mathcal{F}_{\mathsf{TLE}}$. The full specification of the corresponding extended algorithms is given in the Supporting Material D.2.

6.3 IND-CPA-TLE security

Game-based definitions are often natural and easy to use. Unfortunately, the experiment $\mathbf{EXP}_{\mathsf{TLE}}$ presented in Figure: 5 is not enough to argue about the security of a TLE scheme on its own, and is only useful in the context of the Theorem: 1. The reason is that $\mathbf{EXP}_{\mathsf{TLE}}$ argues about only the onewayness of a TLE scheme, leaving aside any semantic security. On the other hand, it is enough for the proof of Theorem: 1 as we use an extension of the TLE scheme in the random oracle model and not the scheme as it is.

Below, we present the analogous experiment of the IND-CPA security notion in the time-lock setting. In a nutshell, this experiment is the same as the one in Figure: 5 except that the adversary in the Challenge command specifies two messages (m_0, m_1) as in the classical IND-CPA game. Again, in order to win the game, the adversary $\mathcal{B}$ must guess correctly which of the two messages is encrypted by the challenger Ch without engaging with the oracle more than the desired amount of times. In case he wins, that would mean that he managed to "break" the TLE scheme in the sense that he decrypted the message before its decryption time.

The experiment $\mathbf{EXP}_{\mathsf{IND-CPA-TLE}}(\mathcal{B}, \mathcal{O}_{\mathsf{eval}}, e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}}, \mathsf{f}_{\mathsf{state}}, \mathsf{f}_{\mathsf{puzzle}}, q)$

Initialization Phase.

■ Ch is initialized with $e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}}$ and sends them to $\mathcal{B}$. In addition, it creates a local time counter $\mathsf{Cl}_{\mathsf{exp}}$.

Learning Phase.

■ When $\mathcal{B}$ issues the query $(\textsc{Evaluate}, x)$ to $\mathcal{O}_{\mathsf{eval}}$ through the Ch, he gets back $(\textsc{Evaluate}, x, y)$.

■ $\mathcal{B}$ can request the encryption of a message $m \in \mathbf{M}_\lambda$ with time label τ_{dec} by sending $(\textsc{Enc}, m, \tau_{\mathsf{dec}})$ to Ch.

■ When Ch receives a $(\textsc{Enc}, m, \tau_{\mathsf{dec}})$ request from $\mathcal{B}$, it runs the algorithm $e_{\mathcal{O}_{\mathsf{eval}}}(m, \tau_{\mathsf{dec}}) \to c$ and returns c to $\mathcal{B}$.

■ Ch increases $\mathsf{Cl}_{\mathsf{exp}}$ by 1 every time $\mathcal{B}$ queries $\mathcal{O}_{\mathsf{eval}}$ q times.

■ $\mathcal{B}$ can request the decryption of a ciphertext c by sending $(\textsc{Dec}, c, w)$ to Ch. Then, Ch just runs the algorithm $d_{\mathcal{O}_{\mathsf{eval}}}(c, w) \to y \in \{m, \bot\}$ and returns to $\mathcal{B}$ $(\textsc{Dec}, c, w, y)$.

Challenge Phase.

■ $\mathcal{B}$ can request for a single time a challenge from Ch by sending $(\textsc{Challenge}, (m_0, m_1), \tau)$. Then, Ch picks a value $b \overset{\$}{\leftarrow} \{0, 1\}$ and sends $(\textsc{Challenge}, \tau, c \leftarrow e_{\mathcal{O}_{\mathsf{eval}}}(m_b, \tau - \mathsf{Cl}_{\mathsf{exp}}))$ to $\mathcal{B}$. Then, $\mathcal{B}$ is free to repeat the *Learning Phase.*

■ $\mathcal{B}$ sends as the answer of the challenge the message $(\textsc{Challenge}, \tau, c, m_{b^*})$ to Ch.

■ If $(m_{b^*} = m_b) \wedge (\tau > \mathsf{Cl}_{\mathsf{exp}})$ (i.e. $\mathcal{B}$ manages to decrypt c_r before the decryption time comes) then $\mathbf{EXP}_{\mathsf{TLE}}$ outputs 1. Else, $\mathbf{EXP}_{\mathsf{TLE}}$ outputs 0.

Fig. 6. Experiment $\mathbf{EXP}_{\mathsf{IND-CPA-TLE}}$ for a number of queries q, function $\mathsf{f}_{\mathsf{state}}$, message domain $\mathbf{M}_\lambda$, algorithms $e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}}$ in the presence of an adversary $\mathcal{B}$, oracle $\mathcal{O}_{\mathsf{eval}}$ and a challenger Ch all parameterized by 1^λ.

Definition 2. *A pair of TLE algorithms* $(e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}})$ *as described in Definition 1 is* IND-CPA-TLE*, if for every PPT adversary* $\mathcal{B}$ *the probability to win the experiment described in Figure 5 is* $1/2 + negl(\lambda)$.

Mahmoody et al. construction is not IND-CPA-TLE: Recall the construction in [39] for encrypting a message m or a secret in general. It can be easily seen that it does not satisfy Definition 2, as the secret is spread across the puzzle, and thus part of it is leaked as the puzzle is solved (see Supporting Material D.3). In contrast, next we show both Astrolabous and an enhanced version of the construction in [39], we called it MMV 2.0 from the first letter of each author, are IND-CPA-TLE.

MMV 2.0*:* As we explained above, the construction in [39] does not satisfy IND-CPA-TLE security because it spreads the message all over the puzzle. A natural question is if it satisfies our game based definition when the message is not spread across all over the puzzle, but instead, it is XORed in the last hash evaluation. Specifically, $e_{\mathsf{MM0.1}}(m, \tau) \to (r_0, r_1 \oplus \mathcal{H}(r_0), \ldots, m \oplus \mathcal{H}(r_{\tau q-1})$, where $r = r_0 || \ldots || r_{\tau q-1}$ is a random string. In that case, as we see next, the MMV 2.0 satisfies IND-CPA-TLE. The proof can be found in Supporting Material D.4.

Theorem 3. *The construction MMV* 2.0 *as described above is IND-CPA-TLE secure in the random oracle model.*

Next we show that Astrolabous is also IND-CPA-TLE secure. The reasoning again is exactly the same as the one in theorem 2 except that the IND-CPA adversary sends the messages m_0, m_1 received from the IND-CPA-TLE adversary to the challenger instead of choosing his own. The rest are exactly the same and thus we omit the proof.

Theorem 4. *Astrolabous,* i.e. *the pair* $(\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}, \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}})$*, is IND-CPA-TLE secure given that the underlying symmetric encryption scheme* E *satisfies* $\mathsf{IND-CPA}$ *security.*

Even if both Astrolabous and MMV 2.0 are IND-CPA-TLE secure, Astrolabous has a potential advantage in terms of efficiency. Namely, Astrolabous hides the key of the symmetric cryptosystem that it uses into the puzzle, instead of the message itself as in MMV 2.0. As a result, many messages can be encrypted under the same key and be opened at the same time solving just one puzzle. In contrast, with MMV 2.0, for every message, a new puzzle must be generated, making the encryption more time-consuming. For example, for a puzzle with difficulty that should last 24-hours, an 8-core CPU can generate it in 3 hours (24/8). The total time for encrypting two messages with MMV 2.0 with the above difficulty is 3 hours for the first message and 2.625 hours (24-1.5/8) for the second, in total 5.625 hours. With Astrolabous one puzzle can be used for both messages, making the total encryption time just 3 hours. The gap becomes even bigger if we consider several encryptions instead of just two. In both examples with did not consider the time to perform AES, as in practice is very efficient.

Asymmetry of puzzle generation and puzzle solving time with Astrolabous: A natural question is if the puzzle generation time is significantly smaller than the time that is required for solving the puzzle. The answer is positive. Specifically, there are hash functions that are not meant to have an efficient evaluation, such as *Argon2* [5]. Equipped with such function we can create puzzles that are small (in terms of space) and fast, but at the same time difficult enough. For example, Argon2 can be parameterized in such a way that a single hash evaluation can take roughly 60 seconds [48], meaning that an 8-core processor can generate a puzzle that meant to be solved in 4 hours (equably 14.400 seconds or 14.400/60 = 240 hash evaluations) in just 30 minutes (puzzle generation is parallelizable so an 8-core processor can do 8 hash evaluation simultaneously which each one of them takes 60 seconds. So 240 hash evaluations can be done in 30 minutes.). As the number of CPU cores increases the puzzle generation can become even smaller but at the same time, the time for solving the puzzle remains unchanged (no parallelization for puzzle solving).

References

1. Myrto Arapinis, Nikolaos Lamprou, Lenka Marekov, and Thomas Zacharias. E-cclesia: Universally composable self-tallying elections. Cryptology ePrint Archive, Report 2020/513, 2020.
2. Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *CRYPTO*, 2017.
3. Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. Craft: Composable randomness and almost fairness from time. Cryptology ePrint Archive, Report 2020/784, 2020.
4. Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. Tardis: A foundation of time-lock puzzles in uc. Advances in Cryptology - EUROCRYPT, 2021.
5. Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: New generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, 2016.
6. Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In *ITCS*, 2016.
7. D. Boneh and M. Naor. Timed commitments. In *CRYPTO*, 2000.
8. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *CRYPTO 2018*, 2018.
9. Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*. Springer Berlin Heidelberg, 2000.
10. Jan Camenisch, Anja Lehmann, Gregory Neven, and Kai Samelin. Uc-secure non-interactive public-key encryption. In *CSF 2017*, 2017.
11. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.
12. Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *TCC*, 2007.
13. Jung Hee Cheon, Nicholas Hopper, Yongdae Kim, and Ivan Osipkov. Timed-release and key-insulated public key encryption. In Giovanni Di Crescenzo and Avi Rubin, editors, *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2006.
14. 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.
15. Dana Dachman-Soled, Mohammad Mahmoody, and Tal Malkin. Can optimally-fair coin tossing be based on one-way functions? In Yehuda Lindell, editor, *Theory of Cryptography*. Springer Berlin Heidelberg, 2014.
16. Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag, 2002.
17. Cynthia Dwork. *Non-Malleability*. Springer US, 2011.
18. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George Robert Blakley and David Chaum, editors, *Advances in Cryptology*, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.
19. Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Non-malleable time-lock puzzles and applications. Cryptology ePrint Archive, Report 2020/779, 2020.
20. Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2014.

21. J. Garay, A. Kiayias, and Giorgos Panagiotakos. Proofs of work for blockchain protocols. *IACR Cryptol. ePrint Arch.*, 2017, 2017.
22. Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, 2015.
23. Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, 2013.
24. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20, 2007.
25. Henri Gilbert and Helena Handschuh. Security analysis of sha-256 and sisters. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*. Springer Berlin Heidelberg, 2004.
26. Oded Goldreich. *The Foundations of Modern Cryptography*. Springer Berlin Heidelberg, 1999.
27. Oded Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, USA, 2006.
28. Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *J. Cryptol.*, 18(3), 2005.
29. Dov Gordon, Yuval Ishai, Tal Moran, Rafail Ostrovsky, and Amit Sahai. On complete primitives for fairness. In Daniele Micciancio, editor, *Theory of Cryptography*. Springer Berlin Heidelberg, 2010.
30. Jens Groth. Evaluating security of voting schemes in the universal composability framework. In *Applied Cryptography and Network Security*. Springer Berlin Heidelberg, 2004.
31. Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*. Springer Berlin Heidelberg, 2010.
32. Jonathan Katz, Julian Loss, and Jiayu Xu. On the security of time-lock puzzles and timed commitments. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography*. Springer International Publishing, 2020.
33. Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *TCC*, 2013.
34. A. Khisti, A. Tchamkerten, and G. W. Wornell. Secure broadcasting over fading channels. *IEEE Transactions on Information Theory*, 54(6), 2008.
35. Aggelos Kiayias and Moti Yung. Self-tallying elections and perfect ballot secrecy. In David Naccache and Pascal Paillier, editors, *Public Key Cryptography*, volume 2274. Springer Berlin Heidelberg, 2002.
36. Czesław Kościelny, Mirosław Kurkowski, and Marian Srebrny. *Foundations of Symmetric Cryptography*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
37. Yehuda Lindell. Highly-efficient universally-composable commitments based on the DDH assumption. In *EUROCRYPT 2011*, 2011.
38. Jia Liu, Tibor Jager, Saqib A. Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, 2018.
39. Mohammad Mahmoody, Tal Moran, and Salil Vadhan. Time-lock puzzles in the random oracle model. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*. Springer Berlin Heidelberg, 2011.
40. Timothy C. May. Timed-release crypto, 1993.
41. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf, 2008.
42. Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *CRYPTO*, 2002.

43. Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Security Protocols*, 1998.
44. Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.
45. Krzysztof Pietrzak. Simple Verifiable Delay Functions. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, volume 124 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.
46. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, 1996.
47. Alan Szepieniec and Bart Preneel. New techniques for electronic voting. USENIX Association, 2015.
48. Aaron Toponce. Further investigation into scrypt and argon2 password hashing, 2016.
49. Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*. Springer International Publishing, 2019.

A Supporting Material for Preliminary section

Universal Composable (UC) framework In the UC framework, the parties engage in a protocol session (labeled by a unique session ID, sid) modeled as Interactive Turing Machines (ITMs) that communicate in the presence of an *adversary* ITM $\mathcal{A}$ that may control some of the parties. The protocol execution is scheduled by an *environment* ITM $\mathcal{Z}$ that provides parties with inputs and may interact arbitrarily with $\mathcal{A}$. The intuition here is that (i) $\mathcal{Z}$ captures the external "observer" that aims to break security by interacting with the protocol interface during session sid, while (ii) $\mathcal{A}$ plays the role of the "insider" that helps $\mathcal{Z}$ via any possible information it can obtain through engaging in the session in the back-end of the current execution. The UC security of a protocol Π follows the *real-world/ideal-world indistinguishability* approach. Namely, security is captured via a special *ideal protocol* that has the same interface as Π that $\mathcal{Z}$ interacts with, but now the parties are "dummy", in the sense that they only forward their inputs provided by $\mathcal{Z}$ to an *ideal functionality* $\mathcal{F}$. The functionality $\mathcal{F}$ is in the center of the back-end (i.e., the ideal protocol has a star topology) and *does not interact* with $\mathcal{Z}$ directly. The ideal functionality $\mathcal{F}$ formalizes a trusted party carrying out the task that Π intends to realize (e.g., secure communication, key agreement, authentication, etc.). The functionality $\mathcal{F}$ interacts with the adversary present in the ideal protocol, usually called a *simulator* $\mathcal{S}$, and this interaction results in a "minimum leakage of information" that determines the ideal level of security that *any protocol* realizing the said task should satisfy (not only Π). For instance, if $\mathcal{F}$ formalizes an ideal secure channel, then the minimum leakage could be the ciphertext length. In case that $\mathcal{Z}$ gives an input to a corrupted party P in the ideal world, the functionality $\mathcal{F}$ passes that message to $\mathcal{S}$ and returns back to P whatever it receives from $\mathcal{S}$. In both executions, if a party has the token and halts, then by convention the token is passed to the environment. We say that the real-world protocol is UC-secure if no environment $\mathcal{Z}$ can distinguish its execution from the one of the ideal protocol managed by $\mathcal{F}$. More formally, let $\mathrm{EXEC}^{\Pi}_{\mathcal{Z},\mathcal{A}}$ denote an execution of a real-world protocol Π in the presence of the adversary $\mathcal{A}$ scheduled by an environment $\mathcal{Z}$, and $\mathrm{EXEC}^{\mathcal{F}}_{\mathcal{Z},\mathcal{S}}$ denote an execution of the ideal protocol managed by $\mathcal{F}$ in the presence of a simulator $\mathcal{S}$, again scheduled by $\mathcal{Z}$. The UC security of Π is defined as follows.

Definition 3 (UC realization [11]). *The protocol Π is said to* UC-realize *the ideal functionality $\mathcal{F}$ if for any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that for any PPT environment $\mathcal{Z}$, the random variables* $\mathrm{EXEC}^{\Pi}_{\mathcal{Z},\mathcal{A}}$ *and* $\mathrm{EXEC}^{\mathcal{F}}_{\mathcal{Z},\mathcal{S}}$ *are computationally indistinguishable.*

Composition and modularity. Perhaps the most prominent feature of the UC paradigm is the preservation of security of a protocol that runs concurrently with other protocol instances, or as a subroutine of another (often more complex) execution. In particular, assume a protocol Π that UC-realizes an ideal functionality $\mathcal{F}$ according to Definition 3, and is used as a subroutine of a "larger" protocol $\tilde{\Pi}$. Then, UC guarantees that if we replace any instance of Π with $\mathcal{F}$, we obtain a "hybrid" protocol, denoted by $\tilde{\Pi}^{\Pi\to\mathcal{F}}$, that enjoys the same security as $\tilde{\Pi}$. Namely, if $\tilde{\Pi}$ UC-realizes some ideal functionality $\tilde{\mathcal{F}}$, then so does $\tilde{\Pi}^{\Pi\to\mathcal{F}}$.

The power of composition facilitates the design and analysis of complex cryptographic schemes with a *high-degree of modularity.* Namely, the scheme's formal description can be over the composition of ideal modules that are concurrently executed as subroutines. When a protocol Π using the functionalities $\mathcal{F}_1,\ldots,\mathcal{F}_k$ UC-realizes a functionality $\mathcal{F}$, we say that it does so in the $\{\mathcal{F}_1,\ldots,\mathcal{F}_k\}$*-hybrid model* and we write $\Pi^{\mathcal{F}_1,\ldots,\mathcal{F}_k}$ to clearly denote the hybrid functionalities. For instance, an e-voting system Π_{vote} can be described using the ideal functionalities $\mathcal{F}_{\mathsf{sc}}$, $\mathcal{F}_{\mathsf{auth}}$ and $\mathcal{F}_{\mathsf{BB}}$ that formalize the notions of a secure channel, an authenticated channel, and a Bulletin Board, respectively. In this case, we say that Π_{vote} is UC-secure in the $\{\mathcal{F}_{\mathsf{sc}},\mathcal{F}_{\mathsf{auth}},\mathcal{F}_{\mathsf{BB}}\}$*-hybrid model* and we write $\Pi_{\mathsf{vote}}^{\mathcal{F}_{\mathsf{sc}},\mathcal{F}_{\mathsf{auth}},\mathcal{F}_{\mathsf{BB}}}$ to clearly denote the hybrid functionalities. Furthermore, composition allows us to extend secure modular design into multiple ($\mathsf{poly}(\lambda)$ many) layers, since a protocol that uses a hybrid functionality as a subroutine may itself be the subroutine of another protocol of an "upper layer" until we reach the level of the root ideal protocol (in our example, an ideal e-voting functionality $\mathcal{F}_{\mathsf{vote}}$).

A.1 Setup functionalities

The global clock functionality $\mathcal{G}_{\mathsf{clock}}$: In Figure 7, we provide the definition of a *global clock* functionality $\mathcal{G}_{\mathsf{clock}}$ similar to [2]. Time advances only when the environment has allowed all involved parties to advance [33,2].

The Global Clock functionality $\mathcal{G}_{\mathsf{clock}}(\mathbf{P},\mathbf{F})$.

For each session sid, the functionality initializes the global clock variable $\mathsf{Cl} \leftarrow 0$ and the set of advanced parties per round as $L_{\mathsf{adv}} \leftarrow \emptyset$.

■ Upon receiving $(\mathsf{sid}, \textsc{Advance\_Clock})$ from $P \in \mathbf{P}$, if $P \notin L_{\mathsf{adv}}$, then it adds P to L_{adv}, sends the message $(\mathsf{sid}, \textsc{Advance\_Clock})$ to P and notifies $\mathcal{A}$ by forwarding $(\mathsf{sid}, \textsc{Advance\_Clock}, P)$. If $L_{\mathsf{adv}} = \mathbf{P} \cup \mathbf{F}$, then it updates as $\mathsf{Cl} \leftarrow \mathsf{Cl} + 1$ and resets $L_{\mathsf{adv}} \leftarrow \emptyset$.

■ Upon receiving $(\mathsf{sid}, \textsc{Advance\_Clock})$ from $\mathcal{F} \in \mathbf{F}$, if $\mathcal{F} \notin L_{\mathsf{adv}}$, then it adds $\mathcal{F}$ to L_{adv} and sends the message $(\mathsf{sid}, \textsc{Advance\_Clock})$ to $\mathcal{F}$. If $L_{\mathsf{adv}} = \mathbf{P} \cup \mathbf{F}$, then it updates as $\mathsf{Cl} \leftarrow \mathsf{Cl} + 1$ and resets $L_{\mathsf{adv}} \leftarrow \emptyset$.

■ Upon receiving $(\mathsf{sid}, \textsc{Read\_Clock})$ from $X \in \mathbf{P} \cup \mathbf{F} \cup \{\mathcal{Z}, \mathcal{A}\}$, then it sends $(\mathsf{sid}, \textsc{Read\_Clock}, \mathsf{Cl})$ to X.

Fig. 7. The global clock functionality $\mathcal{G}_{\mathsf{clock}}(\mathbf{P},\mathbf{F})$ interacting with the parties in $\mathbf{P}$, the functionalities in $\mathbf{F}$, the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$.

That is the standard way of capturing synchronicity in the UC model. Namely, $\mathcal{G}_{\mathsf{clock}}$ is publicly accessible by all entities, and time advances only when the environment has allowed all involved parties to advance. Intuitively, UC synchronicity suggests that the environment must respect the synchronization reference points, yet between consecutive points the protocol flow may be adversarially scheduled.

The random oracle functionality $\mathcal{F}_{\mathsf{RO}}$: In Figure 8, we define a UC *random oracle* (RO) as in [42], a setup assumption widely used in the security analysis of efficient protocols. Like an RO, $\mathcal{F}_{\mathsf{RO}}$ behaves as a truly random function, by providing random yet consistent responses to evaluation queries (i.e., multiple queries for the same preimage x from domain set A result in the same response h from range set B).

The Random Oracle functionality $\mathcal{F}_{\mathsf{RO}}(\mathbf{P}, A, B)$.

The functionality initializes a list $L_{\mathcal{H}} \leftarrow \emptyset$.

■ Upon receiving $(\mathsf{sid}, \textsc{Query}, x)$ from $P \in \mathbf{P}$, if $x \in A$, then

1. If there exists a pair $(x, h) \in L_{\mathcal{H}}$, it returns $(\mathsf{sid}, \textsc{Random\_Oracle}, x, h)$ to P.
2. Else it picks $h \overset{\$}{\leftarrow} B$, and it inserts the pair to the list $L_{\mathcal{H}} \leftarrow (x, h)$. Then it returns $(\mathsf{sid}, \textsc{Random\_Oracle}, x, h)$ to P.

Fig. 8. The random oracle functionality $\mathcal{F}_{\mathsf{RO}}$ w.r.t. domain A and range B interacting with the parties in $\mathbf{P}$.

The broadcast functionality $\mathcal{F}_{\mathsf{BC}}$: We make use of a broadcast channel in order to broadcast to the other parties the resulting ciphertext that includes the time-lock puzzle. The reason behind this design decision was that the parties need to start solving immediately the puzzle by the time of its creation, for more see Supporting Material C.6. The communication interface is formalized via the functionality $\mathcal{F}_{\mathsf{BC}}$ described in Figure 9. We stress that our formalization captures adversaries that can block communication at will via *public delayed output*, i.e., the simulator $\mathcal{S}$ learns the identities of the parties who send the messages. Providing a provably UC-secure realization of $\mathcal{F}_{\mathsf{BC}}$ is out of the scope of this work. However, there are works that present constructions [34,31] and provide simulation security [31] in the secure channel model.

The broadcast functionality is parameterized by a set of parties $\mathbf{P}$ and it is along the lines of [28].

The Broadcast functionality $\mathcal{F}_{\mathsf{BC}}(\mathbf{P})$.

■ Upon receiving $(\mathsf{sid}, \text{BROADCAST}, M)$ from $P \in \mathbf{P}$, it sends $(\mathsf{sid}, \text{BROADCAST}, P, M)$ to $\mathcal{S}$.

■ Upon receiving $(\mathsf{sid}, \text{ALLOW\_BROADCAST}, P, M)$ from $\mathcal{S}$, it sends $(\mathsf{sid}, \text{BROADCAST}, M)$ to all $P^* \in \mathbf{P} \setminus P$ and $\mathcal{S}$ and $(\mathsf{sid}, \text{BROADCASTED}, M)$ to P.

Fig. 9. The broadcast functionality $\mathcal{F}_{\mathsf{BC}}$ interacting with the parties in $\mathbf{P} = \{P_1, \ldots, P_n\}$.

B Supporting Material for Literature review section

B.1 Comparison with [38]

In [38] the authors define a *Computational Reference Clock* (CRC) which after specific time period produces the secret key so that a message with the corresponding time labelling can be decrypted. In this construction, the authors instantiate the CRC with the Bitcoin ledger [2,22,41]. Specifically, they use a witness encryption scheme [38] with the witness being a part of Bitcoin's blockchain. So, every message that was encrypted with label τ can be decrypted with that part of the chain, which is proportional to τ. So the CRC, in that case, is the Bitcoin ecosystem that is maintained by Bitcoin miners [41]. Moreover, they provide security arguments of their construction in a game-based style in the sense that an adversary that executes t-steps cannot win the game except with small probability. If we want to argue in UC about the security of this scheme we have to consider adversaries that execute not a concrete number of steps (in this case t-steps) but instead asymptotically polynomially many steps for arbitrary polynomials, which leads us to a new definition.

C Supporting Material for Time-Lock Encryption section

C.1 The leakage function leak

For example, if $\mathsf{leak}(x) = x + 1$, this means that at current time Cl the adversary $\mathcal{S}$ can retrieve messages that are supposed to be opened at time $\mathsf{Cl} + 1$, meaning that the honest parties will gain access to these messages at the next clock advancement. Specifically, on demand, $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ gives the record of all messages with encryption up to $\mathsf{leak}(\mathsf{Cl})$ to $\mathcal{S}$, where Cl is the current time provided by $\mathcal{G}_{\mathsf{clock}}$, and leak a leakage function that takes the current time as input and returns a later time. This function leak captures the fact that in some cases the adversary can decrypt messages before their opening time has come. The ideal leak function with respect to security is the identity one, the one that gives no real advantage to the adversary in comparison to all other parties. There are however some time-lock encryption schemes which allow the adversary to decrypt a little bit earlier than the honest parties. For example, the Bitcoin based time-lock encryption scheme proposed in [38]. In this scheme, the adversary can locally compute some witness (e.g *selfish mining* [20]) without announcing them to the rest of the parties, providing him with an advantage with respect to decryption.

C.2 The distribution D

Example 3. In our modelling approach, for a random value $b \in \mathbb{Z}_n$, where n is a composite number, the k-repeated squaring of b is the value b^{2^k}. In that case, the oracle queries are of the form $x = b^{2^k}$ and the oracle response is $y = b^{2^{k+1}}$. Thus, the distribution $\mathbf{D}_{b^{2^k}}$ is equal to the constant distribution $\mathcal{C}\{b^{2^{k+1}}\}$ where the probability to sample the value $b^{2^{k+1}}$ is equal to 1.

If $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ is instantiated by the random oracle, then the distribution $\mathbf{D}_x$ is the uniform distribution for every x over the domain $(0, 2^n - 1)$. Similarly in [46], the distribution is constant as argued above (accepts as input x and returns b^{2^x}).

C.3 The function $\mathsf{wit\_con}$

*Example 4 (**The function** $\mathsf{wit\_con}$).* Recall Example 1 and assume time-lock puzzle $c = (r_0, y_0 \oplus r_1, \ldots, y_{r_{q\tau_{\mathsf{dec}}-1}} \oplus r_{q\tau_{\mathsf{dec}}})$. If the function $\mathsf{wit\_con}$ is given less than $q\tau_{\mathsf{dec}}$ oracle responses (e.g. $(y_0, \ldots, y_{q\tau_{\mathsf{dec}}-3})$) for the puzzle c, it returns $\perp$ else it returns $w_{\mathsf{dec}} = (r_0, y_0, \ldots, y_{r_{q\tau_{\mathsf{dec}}-1}}, c)$. Note that here, the ciphertext and the puzzle coincide as there is no actual encryption of a message. So $\mathsf{f}_{\mathsf{puzzle}}$ is simply the identity function here.

C.4 Description of $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$

Our wrapper is defined along the lines of [2]. The functionality wrapper is an ideal functionality parameterized by another ideal functionality, mediating the access to the latter functionality only possible through the wrapper. Moreover, the wrapper restricts the access to the parameter functionality allowing parties to access it only a certain number of times per round. Here, the notion of round is defined with respect to the $\mathcal{G}_{\mathsf{clock}}$ functionality defined in Figure 7. In a nutshell, the wrapper models in the UC setting the limited resources a party has at their disposal for solving the underlying puzzle. Because in UC every party is a PPT ITM, the same holds for the adversary. So, the adversary can interact with any functionality polynomially many times in each round. There are several protocols that hinge their security on the limited computational capabilities of the participants. For example, the whole security argument for the Bitcoin protocol [41] goes as follows: if the adversary does not maintain more than 50% of the network's hashing power, then some desired properties hold. Modelling this in the UC framework would mean that the parties try to extend the ledger by engaging in a series of hash evaluations [22]. If the parties and the adversary have unlimited access to the random oracle functionality (the modelling of the hash function in UC) that would mean that an adversary with less than 50% of hashing power can violate the *common prefix* property in [22]. For that reason, we need to restrict the access to the random oracle functionality, as in [2]. The same holds for our case. We need to restrict the access each party has to $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, else the time-lock puzzle can be solved in just one round, making the whole modelling of TLE in UC defective. Next, follows the description of $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$.

The functionality wrapper as described in Figure 3 is parameterized by the evaluation oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ as described in Figure 2, the global clock $\mathcal{G}_{\mathsf{clock}}$, a set of parties $\mathbf{P}$, and the function $\mathsf{f}_{\mathsf{state}}$.

When $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ receives an evaluation query from a party P, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If this is the first time that this party issued a query, then it creates the list L^P to keep track of how many queries that party does in one round. Else, $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ checks if the number of queries the party issued that round does not exceed q, modelling in this way the limited computational resources a party has in every round. Last, if the party was activated in previous rounds, then the counter of issued oracle queries resets to 1, modelling that unused queries in previous rounds are lost if not made.

When $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ receives the answer from the functionality oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, it returns the oracle's answer to party P.

Adversary can issue q queries in total: The wrapper handles independently queries issued by corrupted parties. Specifically, it allows q queries in total for all corrupted parties instead of q for every corrupted party. With that, we model that the adversary does not possess any advantage for sequential computation despite the fact how many parties are corrupt, in comparison with each party individually. Specifically, in reality, a computation can be either parallelized or not. Each computational task is carried away from a single CPU core at a time. For example, a 10-core CPU can parallelize a 10-step computation at once. On the other hand, in the UC framework, all parties can parallelize any arbitrary polynomial-step computation, assuming that the number of CPU cores they possess is arbitrary polynomial many. When the computation cannot be parallelized, despite how many cores a party has at their disposal, they can process one computation at a time, leaving aside any advantage of the number of CPU cores they possess. For example, if the adversary corrupts two parties or ten with 10-CPU each (twenty and one hundred in each case respectively) it is the same for sequential computation. This is exactly what we illustrate in our functionality wrapper, and thus giving to the adversary q queries in total despite the number of the corrupted parties.

In some settings, the interaction with the oracle is necessary even for the creation of the time-lock puzzle and not just for solving it. In reality, the creator of the puzzle can parallelize this computation, and this is what happens here. In a single oracle query, the party can both ask the oracle queries for puzzle creation and puzzle solving. Recall the example: 1. In order to create a puzzle for time labelling τ_{dec}, the party needs to engage with the oracle just a single time with $q\tau_{\mathsf{dec}}$ values in total to create the puzzle $c = (r_0, y_0 \oplus r_1, \ldots, y_{r_{q\tau_{\mathsf{dec}}-1}} \oplus r_{q\tau_{\mathsf{dec}}})$, where the secret is the value $r_0 || \ldots || r_{q\tau_{\mathsf{dec}}}$. Note that the solver of the puzzle cannot parallelize the computation for solving the puzzle, because she does not know the secret.

The functionality wrapper is parameterized by q the number of oracle queries per round that are allowed from each party, the oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, which evaluates these queries, the global clock $\mathcal{G}_{\mathsf{clock}}$ and a set of parties $\mathbf{P}$ that are allowed to engage with the oracle. The total number of queries q per round captures the fact that the parties have limited resources per round. In addition, we allow multiple value evaluation in a single query. With that we illustrate the fact that the computation can be parallelized (e.g. hash evaluation is parallelizable if the queries are stateless). The number of evaluations in a single oracle query is upper bounded by an arbitrary polynomial (like a UC execution). This in turn means that we assume that the parties have access to an arbitrary polynomial number of CPU cores that can handle independent computations.

C.5 Necessity of extending the TLE algorithms

Our extension, that can be applied in any TLE construction, offers the feature of equivocation but at the expense of assuming the random oracle model. For example, consider a TLE scheme $(e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}})$ with respect to oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$. In the ideal world, when a party wants to encrypt a message m with time labelling τ, the functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ informs $\mathcal{S}$ about this request without revealing the identity of the party and the message m. The simulator creates a ciphertext c without knowing the message m and returns it back to $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$. After the current time Cl exceeds τ, $\mathcal{Z}$ can compute the underlying message to the ciphertext c, which in the ideal world does not contain any information about the message m. This allows $\mathcal{Z}$ to distinguish the real from the ideal execution of the protocol. For that reason, we extend the $(e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}})$ to $(e^*_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d^*_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}})$ by borrowing techniques from [42,10] as follows: the ciphertext for a message m and time τ is the tuple $e^*_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}(m, \tau) = (c_1, c_2, c_3)$, where c_1 results from the encryption of a random string r, i.e., $c_1 = e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}(r, \tau)$; c_2 is the XOR between the message m and the random oracle call $\mathcal{H}(r)$ on r, i.e., $c_2 = m \oplus \mathcal{H}(r)$; and c_3 is a random oracle call on the concatenation $r||m$, i.e., $c_3 = \mathcal{H}(r||m)$. The c_3 makes the encryption scheme non-malleable [42]. This extension allows $\mathcal{S}$ to equivocate when needed. We informally explain why this holds and formalize this in the proof of Theorem 1.

When $\mathcal{S}$ receives an encryption request from $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ for time labelling τ, he returns the ciphertext $c = (e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}(r_1, \tau), r_2, r_3)$ where r_1, r_2, r_3 are random values. Observe that, in the ideal world, neither the evaluation functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ nor the random oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{RO}}}$ exist. Instead, both of them are emulated by $\mathcal{S}$. As a result, when the time Cl exceeds τ, $\mathcal{Z}$ can retrieve r_1 but in order to retrieve the message m he must issue a random oracle query on r_1 through a corrupted party. In that case, $\mathcal{S}$ can retrieve the message m from $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$, because the time Cl exceeded τ, programme the $\mathcal{F}_{\mathsf{RO}}$ so that $\mathcal{H}(r_1) = m \oplus r_2$ (equivocate) and return the answer back to $\mathcal{Z}$.

C.6 Description of protocol Π_{TLE}

Each party $P \in \mathbf{P}$ is parameterized/maintains the following:

- She maintains the list of recorded messages/ciphertexts L^P_{rec}, in which the requested messages for encryption by $\mathcal{Z}$ are stored along with the ciphertext of that message (initially stored as Null), a random identifier of the message tag, the time τ that the message should open, the time Cl that is recorded for the first time and a flag which shows if that message has been broadcast or not to the other parties. Broadcast is necessary, as the construction that UC realises our $\mathcal{F}_{\mathsf{TLE}}$ is relativistic. More precisely, it is based on a time-lock puzzle. So, for that message to be opened by any honest party when the time comes, that party should start to solve the puzzle as soon as it can. So the transmission of the ciphertext (and thus the puzzle) is necessary.
- She maintains the list of the recorded oracle queries for puzzle solving L^P_{puzzle}, in which the time-lock puzzle z_n is stored along with its difficulty τ_{dec}, a set that contains the pairs of queries and responses from the functionality oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ that are necessary for solving the puzzle, the current time Cl, the number of oracle queries that have been made in the current round for solving the puzzle and the total number of oracle queries.
- She parameterized by The tag space TAG and a pair of TLE algorithms $(e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}})$ are hard-coded in each party.
- A function $\mathsf{f}_{\mathsf{state}}$ which prepares the next oracle query to $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ for a puzzle to be solved.
- A function $\mathsf{f}_{\mathsf{puzzle}}$ which extracts from a TLE ciphertext c_* the underlying time-lock puzzle.
- A function $\mathsf{puz\_cr}$ which generates the oracle queries to $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ so that a puzzle of the desired difficulty can be created.
- A function $\mathsf{wit\_con}$ which computes witnesses by performing the necessary sequential computations. More precisely, given oracle queries/responses to/from the functionality $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, time labelling τ and the time-lock puzzle, it returns a witness w_τ or $\perp$ if the computation fails.

When a party receives ENCRYPTION from $\mathcal{Z}$ for a message m with difficulty τ, it picks a random tag for future reference of that message, reads the current time Cl from $\mathcal{G}_{\mathsf{clock}}$, and stores the tuple $(m, \mathsf{Null}, \tau, \mathsf{tag}_m, \mathsf{Cl}, 0)$ to L^P_{rec}. Then, it returns the message $\mathsf{Encrypting}$ to $\mathcal{Z}$. That means that the encryption is going to take some time, in our case one turn. When the party receives from $\mathcal{Z}$ ADVANCE_CLOCK, she reads time Cl from $\mathcal{G}_{\mathsf{clock}}$ and checks if a decryption command has been issued in this turn. If this is the case, that means that the party, before attempting to decrypt, she depletes all her oracle queries for both solving puzzles and creating puzzles for encrypting a message in this turn by executing the procedure *Puzzle*. This is necessary, as the party attempts to decrypt after her witness is updated for that turn and this is possible only by querying the oracle $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$. If no decryption command has been issued in this turn, the party executes both the procedures *Puzzle*, for puzzle solving and puzzle creation, and *Encrypt*, for encrypting the messages issiued by $\mathcal{Z}$ in the current round. Then she broadcasts the ciphertexts that correspond to messages received by $\mathcal{Z}$ in this round (after the end of the turn encryption ends). Finally, the party changes the flags from 0 to 1 in the tuples that the broadcast ciphertexts are stored in L^P_{rec} and informs the global clock that she was activated in that round by sending a clock advancement command.

Broadcast: The broadcast is necessary because the TLE constructions we study are *relativistic* [46,39], and thus the message can be opened only when a certain amount of computations has been spent by the parties to solve the puzzle. In contrast, with *absolute* time-lock constructions such as in [38], the broadcast of the ciphertext is unnecessary because the message will be opened once the current time reaches the decryption time of the time puzzle. That is why we require that the ciphertext must be sent to the designated parties upon its creation. In this work, we realise $\mathcal{F}^{\mathsf{leak},\mathsf{delay}}_{\mathsf{TLE}}$ only with relativistic based constructions. When the party receives the broadcast ciphertexts, she creates a tuple that contains the time-lock puzzle of the ciphertext, the difficulty, the queries for solving the puzzle and the responses and two counters that show how many oracle queries she has issued both this round and in total. The time-lock puzzle can be extracted from a ciphertext with the help of the function f_{puzzle}.

Flag that distinguishes broadcast from non-broadcast messages: When a message is created but is not allowed to be broadcast by $\mathcal{A}$, it means that the other parties will not receive it. Thus, they cannot solve the underlying puzzle so it can be opened when the time comes. In a nutshell, it is like that message did not exist. So, when the environment issues a RETRIEVE command to retrieve the ciphertexts created in this turn, the non-broadcast ciphertexts are not returned. The only ciphertexts returned to $\mathcal{Z}$ are the ones that will eventually be opened by all parties, which is the ones that are broadcast. If we allow the non-broadcast ciphertexts to be returned to $\mathcal{Z}$ then we will have a trivial distinction between the ideal and the real setting for the reasons explained above.

When a party receives a decryption command from $\mathcal{Z}$ for a ciphertext c it uses the function $\mathsf{wit\_con}$ to construct the decryption key. The input of $\mathsf{wit\_con}$

is the collection of states that the party received so far from the $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ through the functionality wrapper $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$. Next, the party returns to $\mathcal{Z}$ either the message m, if the decryption was successful, or $\perp$, otherwise. Note that, as in the construction of [42,10], the third argument in the ciphertext renders the scheme non-malleable [17]. In trivial cases where the difference $\tau_{\mathsf{dec}} - \mathsf{Cl}$ is negative or zero, decryption can occur instantly.

As mentioned, there are two procedures, named *Puzzle* and *Encrypt* that each party executes one or both either when she receives a clock advancement or decryption command from $\mathcal{Z}$. Specifically, in *Puzzle* the party issues q oracle queries in total both for puzzle creation and for puzzle solving. This is achievable since the computation can be parallelized. Specifically, the first oracle query to $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$ contains both the queries that are needed for the creations of the ciphertexts and the queries for solving the puzzles. The next procedure called *Encrypt*, and uses the puzzles created from *Puzzle* to create the new ciphertexts. Specifically, it uses the puzzles to encrypt a random string by using the TLE scheme (*Time-lock encryption*). Then, encrypts the actual message by XORed the message with the random oracle response of the random string (*Extended encryption*). Thus, the first argument of the ciphertext is the TLE encryption of the random string, the second argument is the XORed message with the random oracle responce on the random string, and the third and final argument is the random oracle response on the concatenation of the message and the random string.

C.7 Description of $\mathbf{EXP}_{\mathsf{TLE}}$

We present the experiment $\mathbf{EXP}_{\mathsf{TLE}}$ in the presence of a challenger Ch and an adversary $\mathcal{B}$. This experiment illustrates the security of a TLE scheme in the sense that no adversary can open a message before a certain number of computations has been performed. Specifically, we allow access to the adversary to the evaluation oracle $\mathcal{O}_{\mathsf{eval}}$. It is worth mentioning that the time τ in encryption requests refers to a *relativistic* notion of time (the time that the puzzle needs to be solved) rather than an *absolute* one (the time that the puzzle will eventually be decrypted). If the adversary queries the oracle q times for a ciphertext c, the challenger, which maintains a counter for that ciphertext, increases that counter by one, allowing him to keep track of the number of queries the adversary made for that particular ciphertext. With this, we model the essence of the round and the limited resources the adversary has at his disposal but in a game-based style (without $\mathcal{G}_{\mathsf{clock}}$ and $\mathcal{W}_q(\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}})$). The oracle queries are formed with the help of the state function $\mathsf{f}_{\mathsf{state}}$ and puzzle function $\mathsf{f}_{\mathsf{puzzle}}$, as described in Table 1 and in a dedicated paragraph on page 16, with the initial query for ciphertext c being $\mathsf{f}_{\mathsf{state}}(\mathsf{f}_{\mathsf{puzzle}}(c), 0, \mathsf{Null})$. Again, the state function $\mathsf{f}_{\mathsf{state}}$ takes as an input the time puzzle of the ciphertext c, the number oracle query issued so far in the current round and the previous response of the oracle (e.g., for the initial query it is Null). The state function $\mathsf{f}_{\mathsf{state}}$ illustrates the sequential oracle queries a party does in order to solve the time-lock puzzle. Moreover, $\mathsf{f}_{\mathsf{state}}$ gives a precise description (and enforcement) of how each oracle query must be formed before being issued to the oracle. In that way, we "enforce" the property that the time-lock puzzle cannot be parallelized. Although the adversary can issue encryption and decryption queries on his own because he knows the description of the encryption and decryption algorithms, the challenger only records the encryption and decryption requests that are issued through him. The reason behind this modelling choice is that we only care only to keep track of legitimate encryption and decryption queries, similar to 1. In other words, we cannot guarantee that the adversary uses the correct algorithm to encrypt a message and thus we cannot argue about the security of these ciphertexts. Moreover, a valid witness w_{t} for time label τ with $\tau \leq t$, can be constructed from the responses of the oracle $\mathcal{O}_{\mathsf{eval}}$ with the help of the function $\mathsf{wit\_con}$ as described in Table 1 and in a dedicated paragraph on page 16. Again, the function $\mathsf{wit\_con}$ takes as input oracle queries, a time labelling and a time puzzle and outputs either a witness, if it can be constructed from the provided oracle queries, or $\perp$ otherwise. Upon request, the adversary receives a challenge ciphertext from the challenger. If the adversary can guess correctly the underlying plaintext with less than the expected computations, then he wins the game. For example, if the challenge queried by the adversary is formed with time label τ (e.g. following experiment's glossary, he sends $(\textsc{Challenge}, \tau)$ to the challenger) but the adversary manages to retrieve the message with less then $q\tau$ oracle queries, then he wins the game. In this game the description of the oracle $\mathcal{O}_{\mathsf{eval}}$ in Figure 5, is exactly that of the ideal functionality in Figure 2 without the UC interface.

C.8 Proof of Theorem 1

***Theorem** 1:* Let $(e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}})$ be a pair of `encryption/decryption` algorithms that satisfies Definition 1. Then, the protocol Π_{TLE} in Figure 4 UC-realizes functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ in the $(\mathcal{W}_q(\mathcal{F}^*_{\mathsf{RO}}), \mathcal{G}_{\mathsf{clock}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{BC}})$-hybrid model with leakage function $\mathsf{leak}(x) = x + 1$, where $\mathcal{F}_{\mathsf{RO}}$ and $\mathcal{F}^*_{\mathsf{RO}}$ are two distinct random oracles.

Proof. Let us suppose that protocol Π_{TLE} does not UC-realize $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$. Then, by Definition 3, there is an adversary $\mathcal{A}$ s.t. for every simulator $\mathcal{S}$ there is an environment $\mathcal{Z}$ s.t.:

$$|\Pr[\mathrm{EXEC}_{\mathcal{Z},\mathcal{A}}^{\Pi_{\mathsf{TLE}}} = 0] - \Pr[\mathrm{EXEC}_{\mathcal{Z},\mathcal{S}}^{\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}} = 0]| > \alpha(\lambda) \quad (4)$$

where $\alpha()$ is a non negligible function.

Now consider the specific simulator $\mathcal{S}$ below: At the beginning, $\mathcal{S}$ receives the corruption vector from $\mathcal{Z}$ and informs $\mathcal{A}$ as if it was $\mathcal{Z}$. When $\mathcal{S}$ gets the token back from $\mathcal{A}$, he sends the corruption vector to $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$. Moreover, $\mathcal{S}$ registers the encryption/decryption algorithms $(e_{\mathcal{S}}, d_{\mathcal{S}})$, which are the same as in protocol Π_{TLE}, namely $(e_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}}, d_{\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}})$. However, the *Extended encryption* is not the same, specifically the created cipher texts c_2, c_3 are equal to a random value. Observe that still the distribution of both (c_2, c_3) in both executions are still the same as both c_2, c_3 in the real protocol are random. If $\mathcal{S}$ receives an encryption request $(\mathsf{sid}, \textsc{Enc}, \tau, \mathsf{tag}, \mathsf{Cl}, 0^{|m|}, P)$ from $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ on behalf of an honest party P, he stores the tuple $(\tau_{\mathsf{dec}}, \mathsf{tag}_m, \mathsf{Cl}, 0^{|m^*|}, c, \mathsf{nobroadcast}, P)$, where c is the encryption of $0^{|m^*|}$ by using the algorithm $e_{\mathcal{S}}$, he updates his list, named $L_{\mathsf{RO}^*}^{\mathcal{S}}$ (initially empty), for the generation of that ciphertext. Moreover, he updates his list for the second and the third argument of the encryption as if it was $\mathcal{F}_{\mathsf{RO}}$ (e.g c_2 and c_3). Then, he returns the token back to $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$.

Upon receiving $(\mathsf{sid}, \textsc{Advance\_Clock}, P)$ from $\mathcal{G}_{\mathsf{clock}}$ from an honest party P, $\mathcal{S}$ reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. Then, for every stored tuple $(\tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, 0^{|m_j|}, c_j, \mathsf{broadcast}, \cdot)$, he updates his list, named $L_{\mathsf{RO}^*}^{\mathcal{S}}$, with q evaluation queries for solving the ciphertexts issued by honest parties on previous rounds, as if it was $\mathcal{F}_{\mathsf{RO}}^*$ in the real protocol. Then, he seeks the permission for broadcasting the ciphertetext created for P in this round from $\mathcal{A}$ as if it was $\mathcal{F}_{\mathsf{BC}}$. If $\mathcal{A}$ allows the broadcast, he updates the tuples $(\tau_{\mathsf{dec}}, \mathsf{tag}_m, \mathsf{Cl}, 0^{|m^*|}, c, \mathsf{nobroadcast}, P)$ to $(\tau_{\mathsf{dec}}, \mathsf{tag}_m, \mathsf{Cl}, 0^{|m^*|}, c, \mathsf{broadcast}, P)$ and returns back to $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ the resulting ciphertexts along with their difficulty issued by P in this round. When $\mathcal{S}$ receives an encryption request from $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ on behalf of a corrupted party he reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$, he forwards the message to $\mathcal{A}$ as if it was from that party and keeps record both corrupted party's identity, message and the current time Cl (e.g. (P, m, Cl)). Then, $\mathcal{S}$ returns whatever he receives from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ after updating his record with that response. In any of these cases, $\mathcal{S}$ keeps the randomness that he used for that task. In case $\mathcal{S}$ receives a decryption request from $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ with ciphertext c and time label τ on behalf of an honest party, he does: If c was recorded as a ciphertext of a corrupted party as above, then $\mathcal{S}$ generates the witness $w_{\tau_{\mathsf{dec}}}$ similar to protocol Π_{TLE} as if it was an honest party and updates his list $L_{\mathsf{RO}^*}^{\mathcal{S}}$ exactly as $\mathcal{F}_{\mathsf{RO}}^*$ in protocol Π_{TLE} for consistency between the witness and the oracle queries. Specifically, $\mathcal{S}$ reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and records to $L_{\mathsf{RO}^*}^{\mathcal{S}}$ as many queries as the honest party in Π_{TLE} should do between the time that c was recorded from $\mathcal{S}$ and the current time Cl. Next, $\mathcal{S}$ generates the witness based on these queries exactly as in the real protocol. Then, $\mathcal{S}$ returns to $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ the message $\{m, \perp\} \leftarrow d_{\mathcal{S}}(c, w_{\tau_{\mathsf{dec}}})$. The only way for $\mathcal{S}$ to be asked the opening of such a ciphertext is that the ciphertext is not legitimate (e.g. not issued through $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$). This can be easily observed by the $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$'s command interface. The $\perp$ occurs in the case that the algorithm detects no knowledge over the plaintext (recall the check $c_3 = \mathcal{H}(r_1||m)$ in Figure 4). If $\mathcal{S}$ receives a decryption request for a ciphertext c with time label τ from $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ on behalf of a corrupted party, he forwards the message to $\mathcal{A}$ as if it was from that party. $\mathcal{S}$ returns whatever he receives from $\mathcal{A}$ as if it was the corrupted party back to $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$. In case $\mathcal{S}$ receives a random oracle query request ($\mathcal{F}_{\mathsf{RO}}$) from $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ on behalf of a corrupted party, he forwards the message to $\mathcal{A}$ as if it was from that party. When $\mathcal{S}$ receives this request from $\mathcal{A}$ playing the role of $\mathcal{F}_{\mathsf{RO}}$, he sends the command $\textsc{Leakage}$ to $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$. Then $\mathcal{S}$ checks if the received record from $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ contains any relation between a message m and the random oracle query that $\mathcal{S}$ received initially from the corrupted party. If $\mathcal{S}$ finds such relation, he programs the oracle so that ciphertext can be opened to message m. Then, he responds to $\mathcal{A}$ as if it was the $\mathcal{F}_{\mathsf{RO}}$. For example, let us suppose that the oracle query is the value r_1. Remember that $\mathcal{S}$ issues all the ciphertexts, so he knows the randomness that it was used in each one of them. As a result, he can check if r_1 used for the production of a ciphertext. In case that he founds that r_1 was used for the production ciphertext c, he sends the command LEAKAGE to $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$. In the fortunate scenario where he finds in the received list a tuple that contains a message m and the ciphertext $c = (c_1, c_2, c_3)$, he registers and returns as if it was $\mathcal{F}_{\mathsf{RO}}$ the response $\mathcal{H}(r_1) = c_2 \oplus m$ to $\mathcal{A}$ (equivocation).

In the case $\mathcal{S}$ founds the oracle query but the list does not contain the message, he outputs "$\perp$" (meaning that the adversary was lucky enough to guess a plaintext before the time comes, or the adversary "broke" the security of the encryption scheme). Specifically, when $\mathcal{S}$ receives $(\mathsf{sid}, \text{QUERY}, x)$ from $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ on behalf of a corrupted party he forwards the message to $\mathcal{A}$ as if it was from that party. When $\mathcal{S}$ receives the same message from $\mathcal{A}$ as if it was $\mathcal{F}_{\mathsf{RO}}$, he sends $(\mathsf{sid}, \text{LEAKAGE})$ to $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$. Upon receiving $(\mathsf{sid}, \text{LEAKAGE}, \{(m, c, \tau_{\mathsf{dec}}) \in L_{\mathsf{rec}}\}_{\tau_{\mathsf{dec}}:\tau_{\mathsf{dec}} \leq \mathsf{leak}(\mathsf{Cl})})$ from $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$, $\mathcal{S}$ searches into his database ($\mathcal{S}$ generates all the ciphertexts so he knows the randomness of each) for a ciphertext c_1 on message x. If such ciphertext does not exist, he behaves exactly like the $\mathcal{F}_{\mathsf{RO}}$. If it does, he searches the set $\{(m, c, \tau_{\mathsf{dec}}) \in L_{\mathsf{rec}}\}_{\tau_{\mathsf{dec}}:\tau_{\mathsf{dec}} \leq \mathsf{leak}(\mathsf{Cl})}$ to find a c such that $c[1] = c_1$. If $\mathcal{S}$ does not find such ciphertext, he outputs $\perp$, else he retrieves the corresponding message m and returns as the answer to the random oracle query the message $(\mathsf{sid}, \text{QUERY}, x, y = c[2] \oplus m)$ to $\mathcal{A}$ as if it was from $\mathcal{F}_{\mathsf{RO}}$. In any other case he behaves just like a random oracle. Finally, when $\mathcal{S}$ receives the command EVALUATE from $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ on behalf of a corrupted party, he forwards the message to $\mathcal{A}$ as if it was that party. When $\mathcal{S}$ receives the EVALUATE command from $\mathcal{A}$ on behalf of the corrupted party as if it was $\mathcal{W}_q(\mathcal{F}_{\mathsf{RO}}^*)$, he behaves exactly as $\mathcal{W}_q(\mathcal{F}_{\mathsf{RO}}^*)$ in protocol Π_{TLE}.

By the assumption of $\mathcal{A}$ for $\mathcal{S}$ defined above there is an $\mathcal{Z}_{\mathcal{S}}$ such that Equation 4 holds. There are two possible ways for $\mathcal{Z}$ to distinguish the real from the ideal execution of the protocol based on the syntax of $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$.

Distinction when $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ outputs $\perp$: The first way for $\mathcal{Z}$ to distinguish the two executions is when $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ outputs the special $\perp$ symbol. This happens when $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ detects the same ciphertext for two different messages, meaning that the $\mathsf{Correctness}$ property has been violated. In all other cases when $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ returns $\perp$ the same occurs in the real execution, thus the $\mathcal{Z}$ can not distinct the two execution in such cases.

Distinction when leak *is not "enough":* Last, $\mathcal{Z}$ can distinct the two executions when $\mathcal{S}$ cannot retrieve the message m via the command LEAKAGE and $\mathcal{Z}$ managed to solve the puzzle that correspond to that message. Note that the puzzle is created by $\mathcal{S}$. As a result, $\mathcal{S}$ cannot equivocate the message correctly and $\mathcal{Z}$ can distinguish the real from the ideal execution. For example, if we have a protocol that uses a TLE scheme such that it is not necessary for a party to ask all the oracle queries so that she can solve the puzzle at the desired time, instead she can solve it much faster (broken by design). In such cases, $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ is not realizable.

Lets us suppose that the pair $(e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}})$ satisfies the $\mathsf{Correctness}$ property. We construct an adversary $\mathcal{B}$ that can break the $\mathsf{qSecurity}$ with probability at least $\tilde{\alpha}(\lambda)$, where $\tilde{\alpha}()$ a non negligible function.

The only way for $\mathcal{Z}_{\mathcal{S}}$ to distinguish the real from the ideal execution with non-negligible probability based on the argumentation of Paragraphs C.8 and C.8 is to decrypt/solve the first argument of a ciphertext/puzzle, namely c_1, generated by an honest party before the time comes and issues a random oracle query on it so that $\mathcal{Z}$ retrieves the message. This is possible if $\mathcal{Z}_{\mathcal{S}}$ is able to construct a witness $w_{\tau_{\mathsf{dec}}}$ for an honest generated ciphertext c_1 via the queries issued by a corrupted party to $\mathcal{W}_{\mathcal{F}_{\mathsf{RO}}^*}$ in the real execution of the protocol or in $\mathcal{S}$ in the ideal execution given that the global time Cl provided by $\mathcal{G}_{\mathsf{clock}}$ is strictly smaller than τ_{dec}. Next, $\mathcal{Z}_{\mathcal{S}}$ will request a random oracle query from a corrupted party with the query value to be the plaintext of the ciphertext c_1. Next, $\mathcal{S}$ in order to equivocate correctly, he needs the corresponding message. But if the time of that message has not come yet (e.g. $\mathsf{Cl} < \tau_{\mathsf{dec}}$), the recorded table that $\mathcal{S}$ will request from $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ via the LEAKAGE command, it will not contain that message. As a result, $\mathcal{S}$ will fail to equivocate correctly and $\mathcal{Z}_{\mathcal{S}}$ can distinguish the two executions. Now $\mathcal{B}$ takes advantage of that environment, and uses it in order to win the experiment $\mathbf{EXP}_{\mathsf{TLE}}$ with non negligible probability in the following way: $\mathcal{B}$ simulates the interface to the environment as in the ideal execution of the protocol in the presence of the global clock. Specifically, $\mathcal{B}$ runs every procedure locally simulating every role in the ideal execution, without engaging Ch at all. Every time $\mathcal{B}$ receives q queries $(\mathsf{sid}, \mathsf{Evaluate}, \{x_j\}_{j=0}^{p_l(\lambda)})$ where p_l a polynomial function, from $\mathcal{Z}$ as if it was a corrupted party, he increases by 1 the local counter Cl, (similar to the one Ch has) and forwards $(\mathsf{sid}, \mathsf{Evaluate}, \{x_j\}_{j=0}^{p_l(\lambda)})$ to the oracle $\mathcal{O}_{\mathsf{eval}}$ through the challenger in $\mathbf{EXP}_{\mathsf{TLE}}$. Then returns to $\mathcal{Z}$ whatever it receives. After that point if $\mathcal{Z}$ does not send a clock advancement command, $\mathcal{B}$ does not allow $\mathcal{Z}$ to issue more queries. Now, $\mathcal{B}$ knows that the environment will make at most $p_{\mathcal{H}}(\lambda), p_{\mathsf{enc}}(\lambda)$ random oracle and encryption queries respectively, where $p_{\mathcal{H}}(), p_{\mathsf{enc}}()$ are polynomial functions. At least one of these random oracle queries made by $\mathcal{Z}_{\mathcal{S}}$, from the observation at the beginning of the Paragraph, will contain the plaintext (namely the value r_1 as described in Figure 4) of one of the $p_{\mathsf{enc}}(\lambda)$ ciphertexts that has been decrypted by $\mathcal{Z}_{\mathcal{S}}$ before its decryption time with non negligible probability $\alpha(\lambda)$. Therefore, $\mathcal{B}$ picks $j_1 \overset{\$}{\leftarrow} \{1, \ldots, p_{\mathsf{enc}}(\lambda)\}$. When $\mathcal{Z}_{\mathcal{S}}$ issues the j_1-th encryption query $(\mathsf{sid}, \text{ENC}, m, \tau_{\mathsf{dec}})$ to an honest party simulated by $\mathcal{B}$, $\mathcal{B}$ proceeds as follows: If $\tau_{\mathsf{dec}} > \mathsf{Cl}$ ($\mathcal{B}$ simulates $\mathcal{G}_{\mathsf{clock}}$), then he sends $(\text{CHALLENGE}, \tau_{\mathsf{dec}} - \mathsf{Cl})$ to Ch. When $\mathcal{B}$ receives $(\text{CHALLENGE}, \tau_{\mathsf{dec}} - \mathsf{Cl}, c_1)$ from Ch, $\mathcal{B}$ picks c_2, c_3 exactly as $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak,delay}}$ and returns $(\mathsf{sid}, \text{ENC}, m, \tau, c \leftarrow (c_1, c_2, c_3))$ to $\mathcal{Z}_{\mathcal{S}}$. Then, $\mathcal{B}$ picks $j_2 \overset{\$}{\leftarrow} \{1, \ldots, p_{\mathcal{H}}(\lambda)\}$. When $\mathcal{Z}_{\mathcal{S}}$ issues the j_2-th random oracle query $(\mathsf{sid}, \text{QUERY}, x)$ to a corrupted party, $\mathcal{B}$ sends x to Ch as the answer to the challenge. It can be seen that the probability x to be the answer of the challenge is at least $1/(p_{\mathsf{enc}}(\lambda)p_{\mathcal{H}}(\lambda)) \cdot \tilde{\alpha}(\lambda)$. Note that, although that the ciphertexts of the honest parties simulated by $\mathcal{B}$ are created based on the $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ simulated by $\mathcal{B}$ as well in contrast with the challenged one that is created from Ch trough $\mathcal{O}_{\mathsf{eval}}$ the distribution are exactly the same and the probability for collision on inputs in negligible.

C.9 On the importance of instantiating $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ with $\mathcal{F}^*_{\mathsf{RO}}$

Let us suppose for instance that $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ was not instantiated with $\mathcal{F}^*_{\mathsf{RO}}$, instead it was instantiated by any other functionality parameterized by a constant distribution $\mathbf{D}_x$. In that case, $\mathcal{Z}$ could simply sample values from that distribution locally, solve the puzzle, and encrypt/decrypt any messages in a single round. Specifically, in [46], the procedure for solving a time-lock puzzle consists in repeatedly squaring a base a specific polynomial number of times. However, this computation is deterministic. So any PPT Turing machine, including $\mathcal{Z}$, can produce identical results if they engaged in the same computation without necessarily interacting with the functionality wrapper at all, breaching the security argument of our proof.

A promising way to tackle such deterministic $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ could be to allow the encryption/decryption algorithms to interact with the oracle through the functionality wrapper, verifying that the provided solution for the puzzle was constructed through the evaluation oracle. Of course, this would require more modelling assumptions such as the definition of the encryption/decryption algorithms as ITMs so that they could interact with the oracle. On the other hand, if we instantiate $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ with $\mathcal{F}^*_{\mathsf{RO}}$ then the modelling is more natural. We address the limitations of [46] by defining a new construction, namely *Astolabous*, defined in the Section 6.

Neccesity for defining Astrolabous: The construction in [46] is very simple and easily implementable, which is not the case in our theoretical framework (e.g. UC framework). The security of the construction is based on the repeated squaring problem, which states that: "Given a composite number n and an element $b \in \mathbb{Z}_n$ it is hard to compute b^{2^τ} with less than τ repeated squaring". To define this construction in UC, we have to introduce this new hardness assumption and we have to correlate it with the pair of encryption/decryption algorithms. Specifically, we would have to define an oracle, like the $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$, that is responsible for that computation. The algorithms must communicate with the oracle to ensure that a provided witness is created only from queries through the oracle rather than local computations, where we can not restrict the access via a functionality wrapper and thus can not capture the whole concept of TLE in UC framework. If we want to formulate the communication of the encryption/decryption algorithms with the functionality oracle, we have to define them as ITMs rather than just plain algorithms. This approach is rather new to UC and out of the scope of this work. Instead, we searched solutions where the functionality oracle is the random oracle, such as in [39]. With that approach, the algorithm need not communicate with the oracle because the computations to solve the time-lock puzzle are not deterministic (e.g. like in [46]), in fact, they are probabilistic. So, even if the adversary knows the distribution where the oracle responses to the queries, he can not predict the actual outcome. As a result, when the adversary tries to decrypt the message that is created based on the random oracle functionality, it is impossible to do so without interacting with the oracle first. However, we can not adapt directly the construction from [39] because the adversary can learn parts of the plaintext before the desired decryption time, leading to a weak encryption scheme concerning cryptographic standards [36]. That is why we use the construction from [39] to encrypt not the actual message but the key that is used to encrypt our message with some symmetric encryption scheme, like AES, in the same spirit as [46]. Although that this construction without the extension presented in Supporting Material C.5 is enough if we want to stress the security of Astrolabous against a standalone definition, like the one in Subsection 6.3, for a UC realization is not enough as we have already discussed.

D Supporting Material for Astrolabous section

D.1 Proof of Theorem 2

Theorem 2: *Let* $\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}, \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}$ *be the pair of* encryption/decryption *algorithms described in Subsection 6.1. If the underlying symmetric encryption scheme* E *satisfies* $\mathsf{IND-CPA}$ *security and* correctness *then the pair* $(\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}, \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}})$ *is a secure TLE scheme according to Definition 1 in the random oracle model.*

Proof. In order to prove that the pair $\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}, \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}$ satisfies Definition 1 we need to prove that it satisfies both Correctness and qSecurity.

Proving Correctness*:* We know that the decryption algorithm of the symmetric scheme E returns the correct plaintext with probability 1 [16]. Specifically it holds $\forall m \in \mathsf{M}$:

$$\Pr[k_{\mathsf{E}} \xleftarrow{\$} \mathsf{K}_{\mathsf{E}}; m' \leftarrow \mathsf{dec}(\mathsf{enc}(m, k_{\mathsf{E}}), k_{\mathsf{E}}) : m = m'] = 1$$

where K_{E} and M is the key space and message space of the E respectively.

Let $\mathsf{R}_{\mathcal{H}}$ be the relation as defined in Subsection 6.1 with $\mathcal{F}_{\mathcal{O}_{\mathsf{eval}}}$ instantiated by the random oracle, abbreviating here as $\mathcal{H}$, that correlates the time τ_{dec} and the puzzle c with the correct witness for decryption $w_{\tau_{\mathsf{dec}}}$. Because the correct decryption of $\mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}$ is solely based on the correct decryption of the underlying symmetric scheme E, $\forall m \in \mathsf{M}$ and $\tau_{\mathsf{dec}} \leftarrow \mathbb{N}$ it holds that:

$$\Pr\begin{bmatrix} m' \leftarrow \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}(\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}(m, \tau_{\mathsf{dec}}), w_{\tau_{\mathsf{dec}}}) & \\ \mathsf{R}_{\mathcal{H}}(w_{\tau_{\mathsf{dec}}}, \mathsf{f}_{\mathsf{puzzle}}((\mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}(m, \tau_{\mathsf{dec}})), \tau_{\mathsf{dec}})) & : m' = m \end{bmatrix} = 1$$

Proving $\mathsf{qSecurity}$*:* We argue about $\mathsf{qSecurity}$ by defining a new experiment, similar to the one in Figure 5, where the decryption key used in the symmetric encryption scheme E does not appear at all but still the distribution of messages the adversary sees in both experiments are statistically close based on the security parameter λ. Thus, there is no way for the adversary to learn the real key with less queries than the maximum allowed number and as long as E is secure, the adversary can retrieve the plaintext only with negligible probability.

First, let us define the event that the adversary $\mathcal{B}$ wins in the experiment $\mathbf{EXP}_{\mathsf{TLE}}$ as $\mathsf{Win}_{\mathbf{EXP}_{\mathsf{TLE}}}$ and the event to make less oracle queries than the expected ones for the challenged ciphertext (e.g. $\tau > \mathsf{Cl}_{\mathsf{exp}}$, see Figure 5) as Bad. Note that it holds that $\mathsf{Win}_{\mathbf{EXP}_{\mathsf{TLE}}} \subseteq \mathsf{Bad}$ because the necessary requirements for the adversary to win the $\mathbf{EXP}_{\mathsf{TLE}}$ is by making less oracle queries than the expected ones for the challenged ciphertext. Thus, it holds that

$$\Pr[\mathsf{Win}_{\mathbf{EXP}_{\mathsf{TLE}}}] = \Pr[\mathsf{Win}_{\mathbf{EXP}_{\mathsf{TLE}}} \wedge \mathsf{Bad}] \tag{5}$$

Thus, we need to show that $\Pr[\mathsf{Win}_{\mathbf{EXP}_{\mathsf{TLE}}} \wedge \mathsf{Bad}]$ is negligible with respect to λ. Let us define the experiment $\mathbf{EXP}^*_{\mathsf{TLE}}$ which is the same as $\mathbf{EXP}_{\mathsf{TLE}}$ except that the challenged ciphertext does not contain the key that is used to encrypt the message with the symmetric encryption scheme. Specifically, the last part of the time-lock puzzle in the challenged ciphertext in $\mathbf{EXP}_{\mathsf{TLE}}$ is $k_{\mathsf{E}} \oplus \mathcal{H}(r_{q\tau_{\mathsf{dec}}-1})$, whereas in $\mathbf{EXP}^*_{\mathsf{TLE}}$ it is $\mathcal{H}(r_{q\tau_{\mathsf{dec}}-1})$ instead. Observe that the distribution of messages that $\mathcal{B}$ receives in the two experiments are exactly the same, in the case the adversary did less oracle queries for the challenged ciphertext (the event Bad), because we are in the random oracle model. So we have:

$$\Pr[\mathsf{Win}_{\mathbf{EXP}_{\mathsf{TLE}}} \wedge \mathsf{Bad}] = \Pr[\mathsf{Win}_{\mathbf{EXP}^*_{\mathsf{TLE}}} \wedge \mathsf{Bad}] \tag{6}$$

In the case event Bad does not happen, $\mathcal{B}$ can retrieve the key of the challenged ciphertext from the puzzle. As a result, the distributions of messages in the two experiments are no longer the same because the key that the challenged ciphertext was created with and the key that $\mathcal{B}$ retrieved from the puzzle in $\mathbf{EXP}^*_{\mathsf{TLE}}$ do not match.

We argue that the event $\mathsf{Win}_{\mathbf{EXP}^*_{\mathsf{TLE}}} \wedge \mathsf{Bad}$ happens with negligible probability. Let us assume that:

$$\Pr[\mathsf{Win}_{\mathbf{EXP}^*_{\mathsf{TLE}}} \wedge \mathsf{Bad}] > \alpha(\lambda) \tag{7}$$

where α is a non-negligible function. We construct an adversary $\mathcal{B}_{\mathsf{IND-CPA}}$ that uses the adversary $\mathcal{B}$ to win in the $\mathsf{IND-CPA}$ game of the symmetric scheme E with non-negligible probability. Specifically, $\mathcal{B}_{\mathsf{IND-CPA}}$ works as follows:

He initializes the algorithms $e_{\mathcal{O}_{\mathsf{eval}}}, d_{\mathcal{O}_{\mathsf{eval}}}$, responds to $\mathcal{B}$ and keeps the same counters/database as if it was Ch and $\mathcal{O}_{\mathsf{eval}}$ in the experiment $\mathbf{EXP}^*_{\mathsf{TLE}}$ except when he receives the challenged query from $\mathcal{B}$ for a labelling τ. When the latter happens, $\mathcal{B}_{\mathsf{IND-CPA}}$ chooses two random messages $m_0, m_1 \overset{\$}{\leftarrow} \mathbf{M}_\lambda$ and sends them to the challenger of the $\mathsf{IND-CPA}$ game. Upon receiving the ciphertext c back from the challenger, $\mathcal{B}_{\mathsf{IND-CPA}}$ picks $(r_0||r_1||\ldots||r_{q\tau-1}) \overset{\$}{\leftarrow} \{0,1\}^{\mathsf{p}_2(\lambda)}$(see description: 6.1) and computes $c_\tau \leftarrow (r_0, r_1 \oplus \mathcal{H}(r_0), r_2 \oplus \mathcal{H}(r_1), \ldots, \mathcal{H}(r_{q\tau-1}))$

where the random oracle calls $\mathcal{H}(\cdot)$ are simulated by $\mathcal{B}_{\mathsf{IND-CPA}}$. Then, he returns (τ, c, c_τ) to $\mathcal{B}$ as if it was Ch. Observe that, $\mathcal{B}_{\mathsf{IND-CPA}}$ does not know the key that it is used for the production of the ciphertext c and thus the probability to create a time puzzle c_τ where the actual key appears in the last *XOR* operation would be negligible. For that reason it was necessary to define the intermediate experiment $\mathbf{EXP}^{*}_{\mathsf{TLE}}$.

At some point, $\mathcal{B}_{\mathsf{IND-CPA}}$ receives the answer for the challenged ciphertext, namely $\tilde{m}$, from $\mathcal{B}$. If $\tilde{m} = m_0 \vee \tilde{m} = m_1$, $\mathcal{B}_{\mathsf{IND-CPA}}$ returns $\tilde{m}$ to the challenger of $\mathsf{IND-CPA}$ as the answer to the challenged ciphertext, else it returns m_b where $b \xleftarrow{\$} \{0,1\}$.

Let us define the event $\mathcal{B}_{\mathsf{IND-CPA}}$ to win the experiment $\mathsf{IND-CPA}$ as $\mathsf{Win}_{\mathsf{IND-CPA}}$. Observe that, if $\mathcal{B}$ correctly finds the message in experiment $\mathbf{EXP}^{*}_{\mathsf{TLE}}$ and the event Bad holds then $\mathcal{B}_{\mathsf{IND-CPA}}$ wins as well in the experiment $\mathsf{IND-CPA}$. Specifically:

$$\Pr[\mathsf{Win}_{\mathsf{IND-CPA}}] = \Pr[\mathsf{Win}_{\mathsf{IND-CPA}}|\mathsf{ABad}]\Pr[\mathsf{ABad}] + \Pr[\mathsf{Win}_{\mathsf{IND-CPA}}|\overline{\mathsf{ABad}}]\Pr[\overline{\mathsf{ABad}}] \tag{8}$$

where ABad is the abbreviation for the event $\mathsf{Win}_{\mathbf{EXP}^{*}_{\mathsf{TLE}}} \wedge \mathsf{Bad}$.

By the description of the adversary $\mathcal{B}_{\mathsf{IND-CPA}}$, we have that $\Pr[\mathsf{Win}_{\mathsf{IND-CPA}}|\mathsf{ABad}] = 1$ and $\Pr[\mathsf{Win}_{\mathsf{IND-CPA}}|\overline{\mathsf{ABad}}] \geq 1/2$. Therefore, by Equation (8), it holds that:

$$\Pr[\mathsf{Win}_{\mathsf{IND-CPA}}] \geq 1/2 + 1/2\Pr[\mathsf{ABad}] \tag{9}$$

By Equations (7),(9) it holds that:

$$\Pr[\mathsf{Win}_{\mathsf{IND-CPA}}] > 1/2 + \alpha(\lambda)/2 \tag{10}$$

which is a contradiction. As a result it holds that:

$$\Pr[\mathsf{Win}_{\mathbf{EXP}^{*}_{\mathsf{TLE}}} \wedge \mathsf{Bad}] = \mathsf{negl}(\lambda) \tag{11}$$

Finally, by Equations (5),(6),(11) we have:

$$\Pr[\mathsf{Win}_{\mathbf{EXP}_{\mathsf{TLE}}}] = \mathsf{negl}(\lambda) \tag{12}$$

which completes the proof.

D.2 Equivocable Astrolabous scheme description ($\mathsf{EAST.enc}_{\mathsf{E},\mathcal{H},\mathcal{G}}$, $\mathsf{EAST.dec}_{\mathsf{E},\mathcal{H},\mathcal{G}}$)

$\mathsf{EAST.enc}_{\mathsf{E},\mathcal{H},\mathcal{G}}$**:** The algorithm accepts as input the message m and the time-lock puzzle difficulty τ_{dec} and does the following:

- It picks $r_1 \xleftarrow{\$} \{0,1\}^{\mathsf{p}_3(\lambda)}$ and computes $c_1 \leftarrow \mathsf{AST.enc}_{\mathsf{E},\mathcal{H}}(r_1, \tau_{\mathsf{dec}})$.
- It computes $c_2 \leftarrow \mathcal{G}(r_1) \oplus m$ and $c_3 \leftarrow \mathcal{G}(r_1||m)$.
- It outputs $c = (c_1, c_2, c_3)$.

$\mathsf{EAST.dec}_{\mathsf{E},\mathcal{H},\mathcal{G}}(c, w_{\tau_{\mathsf{dec}}})$**:** The algorithm accepts as input the ciphertext c and the witness $w_{\tau_{\mathsf{dec}}}$:

- It computes $r_1 \leftarrow \mathsf{AST.dec}_{\mathsf{E},\mathcal{H}}(c_1, w_{\tau_{\mathsf{dec}}})$ and $m \leftarrow \mathcal{G}(r_1) \oplus c_2$.
- If $c_3 \neq \mathcal{G}(r_1||m)$ it outputs $\perp$, else it outputs m.

D.3 Mahmoody *et al.*'s construction is not IND-CPA-TLE

The encryption $e_{\mathsf{MM0.1}}(m, \tau)$ for a message m and difficulty τ works as follows:

1. It uses an encoding function $\mathsf{F_e}$ to divide m into τq+1 bit-blocks, $\mathsf{F_e}(m, \tau, q) \rightarrow (m_0, \dots, m_{\tau q})$.
2. Then it computes $c = (m_0, m_1 \oplus \mathcal{H}(m_0), \dots, m_{\tau q} \oplus \mathcal{H}(m_{\tau q-1})$ as the ciphertext of the plaintext m.

The decryption algorithm $d_{\mathsf{MM0.1}}(c, (m_0, \mathcal{H}(m_0), \dots, \mathcal{H}(m_{\tau q-1}))$ for a ciphertext c and witness $(m_0, \mathcal{H}(m_0), \dots, \mathcal{H}(m_{\tau q-1})$ which acts as the secret key, works as follows:

1. It computes the $\tau q + 1$ blocks of message m as $m_j = c_j \oplus \mathcal{H}_{j-1}$.
2. It computes the message m with the decoding function $\mathsf{F_d}((m_0, \dots, m_{\tau q})) \rightarrow m$

It is worth mentioning that this algorithm as presented in [39] , it was not intended to be used as an encryption algorithm rather than a puzzle creation one. Observe that the message is spread all over the puzzle. As a result, the adversary $\mathcal{B}$ can easily win the IND-CPA-TLE game with probability 1. Specifically, he chooses the messages m_0 and m_1 such that the leading bit is different. Next he starts to solve the puzzle. As the message is revealed in a progressive way, when he finds either the bit 0 or 1 first he will know with probability 1 which of the two messages is without depleting all the available oracle queries and thus wins the game.

D.4 Proof of Theorem 3

Theorem 3: *The construction MMV* 2.0 *is IND-CPA-TLE secure according to Definition 2.*

Proof. The reasoning of the proof is very similar with the one in Theorem 2. Specifically, we define a second experiment where the last *XOR* instead of containing the message it is just a random hash evaluation. Again, with exactly the same reasoning we argue that:

$$\Pr[\mathsf{Win}_{\mathbf{EXP}_{\mathsf{IND-CPA-TLE}}} \wedge \mathsf{Bad}] = \Pr[\mathsf{Win}_{\mathbf{EXP}^*_{\mathsf{IND-CPA-TLE}}} \wedge \mathsf{Bad}] \qquad (13)$$

In $\mathbf{EXP}^*_{\mathsf{IND-CPA-TLE}}$ the challenged message it does not appear at all (in contrast with Astrolabous where it appears in the symmetric encryption scheme), so the probability to win there is 1/2 exactly. This completes the proof.