Transparent Batchable Time-lock Puzzles and Applications to Byzantine Consensus

Shravan Srinivasan[1], Julian Loss[2], Giulio Malavolta[3], Kartik Nayak[4], Charalampos Papamanthou[5], and Sri AravindaKrishnan Thyagarajan[6]

[1] University of Maryland sshravan@cs.umd.edu
[2] CISPA Helmoltz Center for Information Security lossjulian@gmail.com
[3] Max Planck Institute for Security and Privacy giulio.malavolta@hotmail.it
[4] Duke University kartik@cs.duke.edu
[5] Yale University charalampos.papamanthou@yale.edu
[6] NTT Research t.srikrishnan@gmail.com

Abstract. Time-lock puzzles (TLP) are a fascinating type of cryptographic problem that is easy to generate, but takes a certain time to solve, even when arbitrary parallel speedup is allowed. TLPs have wide-ranging applications including fairness, round efficient computation, and more. To reduce the effort needed to solve large numbers of TLPs, prior work has proposed batching techniques to reduce the cost of solving. However, these proposals either require: (1) a trusted setup or (2) the puzzle size be linear in the maximum batch size, which implies setting an a priori bound on the maximum size of the batch. Any of these limitations restrict the utility of TLPs in decentralized and dynamic settings like permissionless blockchains. In this work, we demonstrate the feasibility and usefulness of a TLP that overcomes all the above limitations using *indistinguishability obfuscation* to show that there are no fundamental barriers to achieving such a TLP construction.

As a main application of our TLP, we show how to improve the resilience of consensus protocols toward network-level adversaries in the following settings: (1) We show a generic compiler that boosts the resilience of a Byzantine broadcast protocol Π as follows: if Π is secure against $t < n$ weakly adaptive corruptions, then the compiled protocol is secure against $t < n$ strongly adaptive corruptions. Here, 'strong' refers to adaptively corrupting a party and deleting messages that it sent while still honest. Our compiler is round and communication preserving, and gives the *first* expected constant-round Byzantine broadcast protocol against a *strongly adaptive* adversary for the *dishonest majority* setting. (2) We adapt the Nakamoto consensus protocol to a *weak model* of synchrony where the adversary can adaptively create minority partitions in the network. Unlike prior works, we do not assume that *all* honest messages are delivered within a known upper bound on the message delay. This is the *first* work to show that it is possible to achieve consensus in the permissionless setting even after relaxing the standard synchrony assumption.

Keywords: Time-lock puzzles · Batch solving · Distributed consensus · Byzantine broadcast · Mobile-sluggish faults

1 Introduction

A *Time-Lock Puzzle* (TLP) is a cryptographic primitive that allows a sender to lock a message as a computational puzzle in a manner where the receiver will be able to solve the puzzle after a stipulated time $\mathbf{T}$. In terms of efficiency, a sender should be able to generate a puzzle substantially faster than the time required to solve it, and in terms of security, an adversary should not be able to solve the puzzle faster than the stipulated time, even with parallel computation. Rivest, Shamir, and Wagner (RSW) [39] proposed the first TLP construction based on the sequentiality of repeated modular squaring in the RSA group. Many other TLP constructions [8,33,44] have followed suit in different settings but require the same flavor of sequential operations during solving. TLPs have found a wide variety of applications including sealed-bid auctions [39,33], timed-commitments [28,44], e-voting [15,33], fair contract signing [9,33], zero-knowledge proofs [18], cryptocurrency payments [43], distributed consensus [45], blockchain front-running prevention [1], and more applications continue to emerge.

In many TLP applications involving multiple users, it is often the case that a user is required to solve the puzzles of all other users, and record all of the solutions. Say an auction house has to open all the time-locked bids and declare them publicly before announcing the winner. Batching and solving the puzzles is essential for scalability in such TLP applications that have large number of participating users. Intuitively, batch solving of TLPs allows a receiver to solve multiple puzzles simultaneously (at the price of solving one puzzle) without needing to solve each puzzle separately. Specifically, the total running time of the batch-solve operation is bounded by some $p(\lambda, \mathbf{T}) + \tilde{p}(\lambda, n)$ for some fixed polynomials p and $\tilde{p}$, where λ is the security parameter, n is the number of puzzles to be batched, and $\mathbf{T}$ is the timing hardness of a single puzzle.

Modern TLP constructions achieve the seemingly impossible batch-solve property under various settings and assumptions [33,43,44]. However, *all* existing batch solving TLP schemes suffer from the following limitations: (1) requires a trusted setup to generate structured reference string, and/or (2) individual puzzle size scales linearly in the maximum number of puzzles that can be batched, which further implies that there is an a priori upper bound on the number of puzzles that can be batched (which is set upfront during puzzle generation).

Yet emerging blockchain applications like Miner Extractable Value (MEV) prevention [1], cryptocurrency payments [43], distributed consensus [45], etc., either for security or performance, require the TLP scheme to have a *transparent setup*, the puzzle size *independent* of the number of puzzles batched and support batching an *unbounded* number of puzzles. These requirements arise in blockchain systems especially in the permissionless setting for the following reasons:

– First, it is often impossible or impractical to rely on a trusted party to generate the public parameters. Moreover, a compromised setup with trapdoors can violate the security of the system. Precisely for these reasons, such trusted parties are not assumed to exist and usage of cryptosystems requiring such

trusted setups are actively discouraged in permissionless blockchain systems.
- Second, in permissionless systems (like Bitcoin), nodes can join and leave the network at will, and there are no mechanisms to authenticate any participant. So the exact number of participants n is unknown at any point in time. However, existing TLP schemes require the bound on the number of users at the time of puzzle generation. Thus, restricting the ability to accommodate more participants on demand after the puzzle generation phase.
- Third, large puzzle size increases the total communication costs. For instance, in a setting where all participants have to exchange puzzles with each other, a linear-sized puzzle of the prior constructions blows up the total communication overhead to $O(n^3)$.

Motivated by these open problems and applications, we ask the following question: *Is it possible to build a TLP scheme with batch solving that has a transparent setup, puzzle size independent of the batch size, and therefore allows unbounded batching?* In this work, we affirmatively answer this question and use our new TLP scheme to solve two elusive problems in distributed consensus:

1. The problem of expected *constant round* Byzantine broadcast under *corrupt majority* and *strongly adaptive model*.
2. The problem of permissionless consensus in a *generalization of synchronous* model of communication called the *mobile sluggish model*.

Below, we motivate how our TLP with the above properties can enhance security and reduce communication costs in the two applications we consider.

At a high level, in both applications, TLPs help to defend against a powerful network adversary that has the ability to delay or delete messages from the network. For instance, a powerful network adversary can simply learn the contents of the message sent by any honest party before deciding to corrupt or deliver the message. However, by time-lock encrypting the message, the adversary cannot learn the contents of the message before time $\mathbf{T}$ without doing sequential work. In the meantime, all the honest messages would have been delivered. Thus, the adversary is forced to corrupt an honest node without inspecting the contents of the message. Prior works [45,16] showed the feasibility of the distributed consensus in the presence of a network level adversary using TLPs (without batching property) at the cost of polylogarithmic blowup in round complexity. However, batch solving and compact puzzles aid in improving the round complexity and communication costs, respectively, in these applications.

Round-efficient Byzantine broadcast. Byzantine broadcast (BB) is a well-studied problem in distributed consensus, and, in recent times, BB has emerged as a fundamental building block in blockchains [23]. Despite decades of study in improving the round efficiency, no prior BB protocol has expected constant round-complexity under strongly adaptive and dishonest majority setting. In the strongly adaptive model, an adversary can observe all the honest messages, corrupt honest nodes on the fly, and perform *after-the-fact* removal, that is, the adversary can delete any honest message *in-flight* before it reaches any other honest nodes.

Current success in constant round BB is in the weakly adaptive model where the adversary's power is severely limited [46]. Wan et al. proposed the first sub-linear BB protocol under strongly adaptive and dishonest majority setting [45]. Their work used TLPs to prevent the adversary from inspecting the contents of the message before corrupting a node. Since their TLP construction did not support batching, they proposed a sub-protocol with polylogarithmic round complexity to distribute the task of opening all puzzles to honest nodes, rather than solving the puzzles individually. We observe that by using a TLP with batching property and puzzle size independent of the batch size as a building block, it is possible to achieve expected constant round Byzantine broadcast under strongly adaptive and dishonest majority setting.

Permissionless protocol in the mobile sluggish model. Guo et al. introduced a relaxation of the synchronous model, which was subsequently called the *mobile sluggish model* [25,3]. In the mobile sluggish model, a fraction of honest nodes, called *sluggish* nodes, can arbitrarily lose synchrony, but they faithfully follow the rest of the protocol. The remaining honest nodes, called *prompt* nodes, are synchronous and faithfully follow the protocol. Additionally, sluggishness can be *mobile*, that is, any honest node can become sluggish over the protocol execution, and if a sluggish node becomes prompt by regaining synchrony, it will receive all the backlogged messages. This model is stronger than the partially synchronous and asynchronous model but weaker than the synchronous model. Pass and Shi showed that it is *impossible to achieve permissionless consensus* in a partially synchronous or asynchronous network [37]. Unfortunately, Nakamoto consensus is vulnerable to consistency violations even in the mobile sluggish model, as we show in this work. Specifically, even a *single* mobile sluggish fault can effectively reduce the collective mining rate of honest nodes by half!

One way to defend against a mobile sluggish adversary is to let an honest block winner simply time-lock encrypt the message before sending it, and other honest nodes time-lock encrypt a *decoy* to distract the adversary from spotting the block winner. Since the adversary cannot learn the contents of the puzzle without spending sufficient time, by setting the TLP duration slightly greater than the round duration, the adversary is now forced to corrupt or deliver the message randomly. At the end of the round, honest nodes can batch solve the TLPs they received and update their chains. Unfortunately, *no* prior TLP with batch solving works in this application, due to the requirements and challenges in the permissionless setting: we cannot rely on a trusted setup [39,33,43], we do not know the number of users in the network a priori, and we do not want to blowup the round [39] and communication complexity [33,43,44].

1.1 Our Contributions

We give the first TLP construction (§4) that *simultaneously* achieves:

- *Transparent Setup*: Requires a one-time *transparent (public-coin)* setup.
- *Batch Solving*: Supports batch solving of *any* polynomial number of puzzles even after puzzle generation, and the size of individual puzzles is *independent* of the number of puzzles to be batched.

Our construction is based on *Indistinguishability Obfuscation (IO)* [21] where users' puzzles are obfuscated programs. We employ new techniques to achieve compactness in the puzzle size while supporting unbounded batch sizes. Even though our construction is far from being practically efficient, our construction crucially shows that there are no fundamental barriers from achieving a TLP with the above properties and lays a blueprint for future work to instantiate our new techniques with more efficient tools. In §2, we briefly explain why existing techniques for TLP fail to achieve the desired properties, along with giving a brief overview of the key techniques used in our TLP scheme.

We use our TLP construction as a fundamental building block and overcome other challenges to solve two longstanding open problems in consensus:

1. **Round-efficient Byzantine broadcast:** In the years of distributed consensus research, we present (§5) the first *expected constant round* Byzantine broadcast under strongly adaptive and corrupt majority setting. To realize our result, we develop a generic compiler that uses *any* batch solving TLP construction to *convert any* broadcast protocol secure against a *weakly* adaptive adversary [46] into a broadcast protocol secure against a *strongly* adaptive adversary in a *round preserving* way, which could also be of independent interest. With our TLP, this compiler is additionally *communication* preserving. We formally prove the security (Thm. 3) of our compiler in the *programmable random oracle* (RO) model.
2. **Permissionless protocol in the mobile sluggish model:** We first show an attack to illustrate that Nakamoto consensus is not secure *even* in the mobile sluggish model (§6.1). We then present a proof-of-work based permissionless protocol (§6.3) which does not assume that the network is synchronous or all honest messages arrive on time. To the best of our knowledge, this is the *first* work to show that it is possible to achieve consensus in the permissionless setting even after *relaxing the standard synchrony assumption!* To do this, we develop a novel proof-of-work based *decoy* mechanism that uses TLPs to defend against a mobile sluggish adversary that can arbitrarily delay a fraction of honest messages. We formally analyze our protocol to prove that it achieves *consistency* and *liveness* in the extended version of our paper [42]. Specifically, we show that our protocol realizes the standard properties namely, *chain growth*, *chain quality*, and *common prefix* [42].

1.2 Related Work

Time-lock puzzles. Bitanski et al. [8] proposed a different approach to construct TLPs, assuming the existence of succinct randomized encodings [7] and non-parallelizable languages. Similar to RSW puzzles, during solving each puzzle has to be *solved individually* to obtain their solutions. Liu et al. [32] combine (extractable) witness encryption [22] and a public reference clock like the Bitcoin blockchain. In their construction, one can batch open many puzzles as the blockchain reaches a certain height as the computational effort is shared by the entire blockchain network in mining new blocks. Their construction relies on

Succinct Non-Interactive Argument of Knowledge (SNARKs) [6] and thus non-falsifiable assumptions. Our construction on the other hand does not require such assumptions and does not rely on a global reference clock like a blockchain. Malavolta and Thyagarajan introduced Homomorphic TLPs [33]. Their constructions allowed homomorphic function evaluations to be performed on puzzles to obtain a single puzzle that embeds the function of all the original solutions. However, all constructions from [33] (including their fully homomorphic TLP) and [43] do not support unbounded batching of puzzles and require structured reference string generated using a trusted setup. Thyagarajan et al. [44] proposed a Class group based construction that gets rid of the trusted setup and only requires a transparent public-coin setup. In Table 1, we compare with prior constructions.

Table 1: Comparison with other batch TLP schemes. λ is the security parameter and $\mathbf{T}$ is time hardness parameter. Compactness of puzzles here refers to the size of puzzles being independent of the batch size.

Scheme	Transparent setup	Unbounded batching	Compact puzzles	One-time Setup time	Practical efficiency
RSA-based [43,33]	✗	✗	✗	$O(\log(\mathbf{T}), \lambda)$	✓
Class-groups based [44]	✓	✗	✗	$O(\mathbf{T}, \lambda)$	✓
IO-based (**This work**)	✓	✓	✓	$O(\mathbf{T}, \lambda)$	✗

Recently, Burdges and Feo [11] proposed a related but a new notion called *delay encryption*. On a high level, users encrypt their messages to some common previously unpredictable identity ID using an Identity-Based Encryption (IBE) scheme. The decryption key for the identity ID can be derived by anyone but the derivation is a delayed operation, meaning that it takes time $\mathbf{T}$ to derive the key. We can batch decrypt several encryptions provided they are w.r.t. to the same ID. The drawback of their construction is the requirement of a trusted setup which is considered a strong assumption in the applications of our interest. Encryption-to-the-future is a closely related primitive, however, unlike our construction prior works either use a public bulletin (like a blockchain) or a committee of users with an honest majority [12,19].

Strongly adaptive Byzantine broadcast. Wan et al. proposed the first expected sub-linear round protocol in the strongly adaptive setting using Public-Key Infrastructure (PKI) and TLPs [45]. Subsequently, Cohen et al. [16] explored the feasibility of *fair* broadcast in the strongly adaptive setting for both property-based and simulation-based definitions. However, our focus is on achieving expected constant-round BB under strongly adaptive and dishonest majority setting. We relate to other works in the extended version our paper [42].

Protocols in the mobile sluggish model. Guo et al. [25] first introduced the mobile sluggish model as "weakly synchronous" model and showed that it is impossible for a Byzantine broadcast protocol to tolerate majority faults (Byzantine or sluggish). Subsequently, Abraham et al. presented a Byzantine Fault Tolerant blockchain protocol that can tolerate minority corruptions in the mobile sluggish model [3]. Kim et al. [30] observed that many proof-of-stake protocols, such as Dfinity [26], Streamlet [14], OptSync [41], can support

mobile sluggish faults. These prior techniques heavily relied on using messages (votes) from a majority of the nodes (certificates) to establish communication with sluggish nodes and ensure safety of the protocol. Since Nakamoto consensus does not rely on such certificates, their techniques do not apply in our setting.

Nakamoto style protocols. Prior works can be categorized based on the flavor of synchrony used to analyze Nakamoto consensus. In the lock-step model of synchrony, Garay et al. formally analyzed Nakamoto consensus [20]. Subsequently, Pass et al. and Kiffer et al. showed that the Nakamoto consensus is secure even in the non-lock-step synchrony model where the message delay is bounded and the time proceeds in discrete rounds [35,20,29,48]. Ren discarded the notion of discrete rounds and proved the security of Nakamoto consensus in the continuous model [38]. Even parallelly composed Nakamoto protocols are also in the lock-step model of synchrony [4,47]. Unfortunately, all these analyses assume that *any* honest message reaches other honest nodes in Δ time units regardless of the flavor of synchrony. Our analysis is in the mobile sluggish model, which assumes that a fraction of honest nodes can violate the Δ-assumption. However, the prompt nodes in our setting are assumed to be in lock-step synchrony model.

Network-adversary lower bounds and impossibilities. Abraham et al. showed that a sub-quadratic protocol could not be resilient against a strongly adaptive adversary that can perform *after-the-fact* removal [2]. In Nakamoto consensus, delaying an honest block has the same effects as deleting the block. For example, if a newly mined block is delayed for a sufficiently long time, it could end up as an orphan block, which eventually gets pruned after the main chain stabilizes. Moreover, sluggishness can be mobile, thus making the sluggish adversary more powerful than the strongly adaptive adversary. Pass and Shi showed that it is impossible to achieve permissionless consensus in the partially synchronous/asynchronous network [37]. In these network models, the adversary can *arbitrarily* partition the honest nodes. However, in our setting, the adversary can create only *minority* partitions. Thus, this impossibility does not apply.

2 Technical Overview

We give an overview of our TLP construction that supports batch-solving an unbounded number of puzzles and protocols that use our TLP construction to tolerate network-level adversaries in the BB and Nakamoto consensus.

2.1 Time-Lock Puzzles with Batch Solving

Bounded batching of TLPs. Before delving into the specific of our construction, we show how standard techniques [33,44] readily give a construction of TLP with *bounded* batched solving, i.e., where the number n of batched solutions is fixed at puzzle generation time. Given a Linearly Homomorphic TLP scheme LHP with homomorphism over $\mathbb{Z}_q$ and a large enough q, we can homomorphically evaluate the packing algorithm. In more detail, we are given n puzzles $Z_1, \ldots, Z_n$

(of the LHP scheme) each encoding λ-bit values with timing hardness of each puzzle being $\mathbf{T}$. To batch solve these puzzles, we first homomorphically evaluate the linear function: $f(x_1, \ldots, x_n) = \sum_{i=1}^{n} 2^{(i-1)\cdot\lambda} \cdot x_i$.

The resultant evaluated puzzle Z^* is then solved in time $\mathbf{T}$ to obtain all the n values where each of these values were originally encoded as λ-bit values. Importantly, this means that the bit-representation of the plaintext space must be large enough to accommodate all n-secrets, i.e., $\log(q) \approx n \cdot \lambda$. Since the domain has to be fixed at the time of puzzle generation, this means that each puzzle scales linearly with n. In settings with n parties, where each party generates a puzzle and broadcasts it to the other parties, the total communication is $O(n^3)$, assuming a total of $O(n^2)$ communication for a broadcast of a single bit and ignoring factors that depend on λ.

Unbounded batching? The question that we set out to answer is whether it is possible to construct a TLP that supports *unbounded* batch-solving. One approach to do that is to "defer" the choice of the plaintext space at the solving time, so that the solver can select the appropriate domain, depending on how many puzzles need to be batched. A naive idea is to define a program $\mathbf{P}$ that, on input the batch size n, outputs a LHP puzzle Z embedding the user's message m and where the message space is sufficiently large to accommodate packing of n puzzles. This solution is clearly insecure as it reveals m in the plain, so to amend this we *obfuscate* the program $\widetilde{\mathbf{P}} := i\mathcal{O}(\mathbf{P})$, using indistinguishability obfuscation (IO) [21,27]. Setting a super-polynomial upper bound on $n \approx 2^{\omega(\log(\lambda))}$ allows one to batch any polynomial number of puzzles.

Unfortunately, this simple construction runs into issues when proving security. A natural strategy when proving security would be to hybrid over all possible n, hardwire the corresponding puzzle in the description of the circuit and the swap it with a puzzle encoding a fixed string (say 0) appealing to the security of the TLP. However it is not hard to see that this would quickly run into issues: As n grows to super-polynomial, the size of the corresponding puzzle, and consequently of the obfuscated circuit, would also be super-polynomial. This is not only an issue of the security proof, since the actual obfuscated circuit must be padded to the maximum size of the circuits that is defined in the analysis. To get this strategy to work, our construction would yield a super-polynomial size puzzle!

Our solution. To understand our solution, we first discuss a way to circumvent the above issue. We change the output of the obfuscated circuit to output the n dimensional vector $(0, \ldots, m, \ldots, 0)$, where m is inserted in the i-th slot, masked by the output of a puncturable pseudorandom function (PRF) F and a LHP puzzle Z encoding the PRF key k.

$$\begin{bmatrix} 0 \\ \vdots \\ m \\ \vdots \\ 0 \end{bmatrix} + \begin{bmatrix} \mathsf{F}(k,1) \\ \vdots \\ \mathsf{F}(k,i) \\ \vdots \\ \mathsf{F}(k,n) \end{bmatrix} = \begin{bmatrix} \mathsf{F}(k,1) \\ \vdots \\ \mathsf{F}(k,i) + m \\ \vdots \\ \mathsf{F}(k,n) \end{bmatrix}$$

This structured solution allows us to solve the proof problem by puncturing each position individually, thereby avoiding an exponential blow-up in the size of the obfuscated circuit. To solve this puzzle, the solver can arrange the various masked plaintexts (from other obfuscated circuits of other users) diagonally, and sum up all the results, to obtain

$$\begin{bmatrix} \mathsf{F}(k_1,1)+m_1 \\ \mathsf{F}(k_1,2) \\ \vdots \\ \mathsf{F}(k_1,n) \end{bmatrix} + \begin{bmatrix} \mathsf{F}(k_2,1) \\ \mathsf{F}(k_2,2)+m_2 \\ \vdots \\ \mathsf{F}(k_2,n) \end{bmatrix} + \ldots + \begin{bmatrix} \mathsf{F}(k_n,1) \\ \mathsf{F}(k_n,2) \\ \vdots \\ \mathsf{F}(k_n,n)+m_n \end{bmatrix}$$

Here k_i and m_i are the PRF key and message, respectively, of the i-th obfuscated program. However at this point it is not clear how to batch solve the resulting puzzles, since each party will use an independent PRF key k_i. This means that we shifted the problem from recovering the n messages to recovering n PRF keys, bringing us back to square one. Our last idea is to use instead a *key-homomorphic* PRF $\overline{\mathsf{F}}$. Assuming a suitable instantiation [10], we have the above expression evaluate to

$$= \begin{bmatrix} \sum_i \overline{\mathsf{F}}(k_i,1)+m_1 \\ \sum_i \overline{\mathsf{F}}(k_i,2)+m_2 \\ \vdots \\ \sum_i \overline{\mathsf{F}}(k_i,n)+m_n \end{bmatrix} \approx \begin{bmatrix} \overline{\mathsf{F}}\left(\sum_i k_i,1\right)+m_1 \\ \overline{\mathsf{F}}\left(\sum_i k_i,2\right)+m_2 \\ \vdots \\ \overline{\mathsf{F}}\left(\sum_i k_i,n\right)+m_n \end{bmatrix}$$

We also have a LHP puzzle from each party encoding k_i. The solver can now add all keys homomorphically and solve the resulting LHP puzzle in time $\mathbf{T}$ to obtain $\sum_i k_i$. Once the key is known, the solver can simply unmask all the values in the above vector by evaluating the PRF at points $(1,\ldots,n)$ using the key $\sum_i k_i$. Subtracting the output yields the vector of n plaintexts. Note that in the full construction, the index i for the puzzle of a user is not chosen during puzzle generation, but is assigned during puzzle solving through some deterministic rule.

This gives an outline of our construction. In the actual scheme, extra care is needed to match the modulus of the TLPs with the key space of the PRF, to set the parameters to account for the imperfect homomorphism, and deal with the lack of imperfect correctness of punctured keys in the proof (see §4 for more details). Notice that the size of each puzzle is dominated by the obfuscated program $\widetilde{\mathbf{P}}$, which can be implemented to be of size logarithmic in n (the batch size). Therefore, in a multi-party setting, we get a total communication of $\tilde{O}(n^2)$ assuming $O(n^2)$ communication for single bit broadcast and ignoring factors that depend on λ.

2.2 Application 1: Efficient Byzantine Broadcast

Byzantine broadcast is a classical problem in distributed consensus, where a *designated sender* holds a bit b and wants to transmit b to all n nodes in the

presence of t faults. A BB protocol is secure if it can guarantee *consistency* (all honest nodes output the same bit) and *validity* (if the designated sender is honest, all honest nodes output the designated sender's input b).

With increasing applications for BB (cryptography, blockchains, etc.), we study the round-efficiency of BB under the dishonest majority setting. Prior BB protocols in the dishonest majority setting can broadly tolerate: (1) weakly adaptive or (2) strongly adaptive adversary. Both strongly and weakly adaptive adversary can corrupt honest nodes *on the fly*. But, a weakly adaptive adversary cannot perform *after-the-fact* removal.

Despite decades of study, the state-of-art round-efficient BB in the dishonest majority is in the weakly adaptive setting [46]. Thus, it raises the question:

Is it possible to achieve an expected constant-round Byzantine broadcast under strongly adaptive and corrupt majority?

We affirmatively answer this using PKI, RO, and *any* batch solvable TLP construction. Our solution is a *generic round preserving compiler* that can convert any weakly adaptive BB protocol into a strongly adaptive one. Thus, our compiler can be ***efficiently realized*** using the batch solvable TLP constructions based on RSA or Class-groups [44,43]. With our batch solvable TLP (§4.3), our compiler is additionally *communication* preserving as well!

The key ingredient in our compiler is that we use TLPs to hide the contents of the messages sent by the underlying protocol so that the strongly adaptive adversary cannot learn the contents of any message before honest nodes receives it. Prior works use the RSW puzzles to defend against a strongly adaptive adversary [45,16]. But, due to the inability to batch solve RSW proofs, opening all puzzles collectively adds an overhead of polylogarithmic rounds to any protocol [45]. An alternate approach is to use batch solvable TLPs defined in [44,43] to remove the polylogarithmic communication overhead incurred by RSW puzzles. But this increases communication complexity by a linear factor since prior batch solvable TLPs are not compact. Since our TLP is compact and batchable, we can solve all puzzles in one round without increasing the communication complexity.

Even though TLPs can prevent the adversary from inspecting the contents of the message, the primary challenge is in proving that the compiled protocol is secure against an adversary that can perform *after-the-fact* removal. This is because TLPs, apart from hiding the message contents for $\mathbf{T}$ time units, also serve as a commitment to the message inside the puzzle, which prevents the simulator from simulating the honest nodes without knowing the actual contents of the puzzle! Cohen et al. encountered a similar problem in the context of *fair* BB and proposed a non-committing TLP to overcome this challenge [16, Theorem 5].

Non-committing TLPs. Informally, it allows the simulator to equivocate a TLP. That is, the simulator first generates and sends a "fake" TLP to the network, which can be later "opened" be to *any* message. Thus, when the simulator is asked to explain the contents, it programs the RO to open the desired message. In §4.3, we show how to achieve this property with our TLP construction.

Compiler overview. Abstractly, in a weakly adaptive protocol $\Pi_{\mathsf{bb-wa}}$, a node performs three *basic steps*: In every round, (1) receives the messages sent by other nodes, (2) performs the state transition based on the messages received and computes the messages to send, and (3) sends the messages to other nodes. Our compiler interleaves each step of $\Pi_{\mathsf{bb-wa}}$ with TLP operations to obtain a strongly adaptive protocol, $\Pi_{\mathsf{bb-sa}}$.

In a bit more detail, before sending a message in the compiled $\Pi_{\mathsf{bb-sa}}$, a node uses non-committing TLPs to encrypt the message it wants to send and computes the puzzle with proof of well-formedness of the puzzle. Thus, instead of sending the plaintext in $\Pi_{\mathsf{bb-wa}}$, a node in $\Pi_{\mathsf{bb-sa}}$ sends the puzzle, ciphertext, and the proof of well-formedness to other nodes. When a node receives puzzles, ciphertexts, proofs of well-formedness from the network, instead of opening one puzzle at time, $\Pi_{\mathsf{bb-sa}}$ uses the batchable TLP proposed in this work to obtain all the solutions simultaneously without incurring additional round complexity or communication complexity to open all the puzzles. Thus, after opening all the puzzles, a node in $\Pi_{\mathsf{bb-sa}}$ invokes the state transition function just like a node in $\Pi_{\mathsf{bb-wa}}$. This process is repeated for every round. We defer the details to §5.2.

2.3 Application 2: Permissionless Consensus in the Mobile Sluggish Model

Nakamoto's protocol, used in Bitcoin, achieves consensus over the Internet in a *permissionless* setting, where: any node can join and leave the system at any time, the exact number of participating nodes is unknown, and the nodes have to communicate over unauthenticated channels. However, for security, the protocol assumes that the network is synchronous – all honest messages get delivered to one another within a known upper bound on time, Δ units.

Unfortunately, assuming that an Internet scale protocol is synchronous is excessively optimistic. Moreover, Pass and Shi [37] showed that it is *impossible* to achieve permissionless consensus in an asynchronous or even in a partially synchronous network [5], which are relaxations of the synchronous model. Thus, to deploy the protocol in the real-world, the protocol designers are compelled to choose a loose upper bound Δ as the network delay to accommodate nodes with slow network.

In this work, we relax the standard synchrony assumption and study Nakamoto consensus under the mobile sluggish model [3,25]. For Internet scale protocols, the sluggish model is a pragmatic trade-off between the synchronous model and partially synchronous/asynchronous model. Thus, we ask the following question:

Is it possible to achieve consensus in a permissionless setting in the presence of mobile sluggish faults?

We affirmatively answer this question by proposing a protocol that uses our TLP construction from §4 as a fundamental building block to show that it is possible to achieve *consistency* (any two prompt chains can differ only in the last few blocks) and *liveness* (every prompt node eventually commits all transactions)

even in the presence of mobile sluggish faults.

In this subsection, we show how to adapt the Nakamoto consensus to defend against a mobile sluggish adversary using the our TLP. In our protocol, we use the following ideas: (1) All honest nodes time-lock encrypt any message they transmit, (2) all honest nodes send *decoys* to protect the block winner from getting caught by the adversary, (3) restrict the adversary from flooding with decoys, and (4) ignore malformed puzzles sent by the adversary.

Formally, we define a round, a super-round, and duration of a round in §6.2 and §6.3. However, as a warm-up, we present strawman solutions to illustrate the inadequacies of the well-known approaches.

Strawman solutions. The first straightforward solution is to use RSW puzzles to time-lock encrypt any message with a duration equal to the network delay before transmitting across the network [39]. Unfortunately, this approach does not work for the following reasons:

- Recall that in a protocol like the Nakamoto consensus, only the block winner sends a message to the network. Thus, the adversary can easily stop the one message transmitted, whether or not the message is encrypted.
- Say the other honest nodes send out time-lock encrypted dummy messages, which act as a *decoys* to protect an honest block winner from getting caught. Unfortunately, the honest parties have to open all the puzzles to find the winning block. Thus, the honest parties either have to open all the puzzles individually or open them using the *distributed-solve* primitive proposed by Wan et al. [45, Section 4.2]. Both these approaches increase the round complexity of the protocol by linear and polylogarithmic rounds, respectively.

An alternate approach is to use TLPs with batch solving property defined in [44,43], but we would suffer from large communication costs and fixed batch size problem as explained before. Instead, we can now use our TLP that gets rid of the these issues. Below we give an overview of other challenges we encounter in designing our permissionless consensus protocol.

Decoys, spam prevention, and malformed puzzles. Since the Nakamoto consensus is in the permissionless setting, there are no identities to tackle Sybil attacks. This setting raises an important question: how to stop the adversary from spawning multiple identities to send decoys? We resort to proof-of-work to tackle the Sybil attack!

Say the difficulty threshold to mine a block is T, then we set the threshold to mine a decoy as T_c, such that $T < T_c$. Each RO query made by a node simultaneously tries to mine a block and a *decoy*. That is, say h is the output of the hash function. If $h < T$, then *a block* is mined, else if $T \leq h < T_c$, then *a decoy* is mined. This is the "2-for-1 POW" trick introduced by Garay et al. [20,36,4]. The parameter T_c presents an interesting trade-off: T_c should be sufficiently high so that honest nodes mine enough decoys whereas the adversary should not be able to overwhelm the honest nodes with many decoy puzzles.

One of the challenges is that nodes do not know the exact number of decoys mined at a given time. However, since our TLP construction can batch a variable number of puzzles, nodes can flexibly batch puzzles on demand. Observe that

T_c restricts the number of decoys that the adversary (and the honest nodes) can mine. But, this does not stop the adversary from flooding the honest nodes with malformed puzzles. Batching malformed puzzles along with honest puzzles prevents a node from obtaining the solutions to honest puzzles. To circumvent this problem we equip our TLP with a verifiability property that allows an honest node to reject a puzzle that is not *well-formed* according to the puzzle generation algorithm. Thus, a valid proof guarantees that the plaintext can be obtained by solving the puzzle.

Mine phase and solve phase. Since the mining process is stochastic, the arrival times of a decoy and a block are random. Say if an honest node sends the puzzle as soon as it finds the block, it is unlikely that the rest of the honest nodes will also be sending the decoy puzzles at the same time. If enough honest nodes do not provide "cover" to the block winner, then the probability of the adversary guessing the block winner is high. However, if all honest nodes wait until a pre-determined time to send the respective puzzles, then block winner will have the best chance of not being detected by the adversary.

In order to capture this intuition, we have two phases in our protocol:

- *Mine phase*: All nodes spend a sequence of m rounds mining a block or decoy without sending or receiving any messages.
- *Solve phase*: This phase begins as soon as the mine phase ends and consists of *two* rounds. In the first round, nodes send and receive the puzzles they have mined in the *mine phase*, and check the well-formedness of the received puzzles. In the second round, nodes will batch solve the TLPs to find the block, if any, and update the longest chain.
  We generically denote the duration of the solve phase as D rounds. If one employs RSW puzzles and the *distributed-solve* primitive from Wan et al. [45] instead of our TLP construction, then D can be thought of as the number of rounds required to perform *distributed-solve* procedure. However, when our protocol is instantiated with our TLP construction we have $D = 2$.

Thus, the duration of a super-round is $(m + D)$ rounds.

Putting it all together. In summary, by using our TLP, the decoy mechanism, and super-rounds, our protocol works as follows: Every honest node performs the following steps in every super-round: (1) Receive transactions from the environment, (2) choose the longest chain it has seen so far and break ties arbitrarily, (3) mine for m rounds (mine phase), and (4) solve for D rounds (solve phase) and update the longest chain. We defer the details of the protocol to §6.3.

3 Cryptographic Background

We denote by $\lambda \in \mathbb{N}$ the security parameter. We say that a function μ is negligible if it vanishes faster than any polynomial. The notation $[n]$ denotes a set $\{1, \ldots, n\}$. Background and notations relevant to the two applications are deferred to §6.2 and §5.1, respectively.

3.1 Time-Lock Puzzles

In the following we give a definition for the main object of interest of this work, namely time-lock puzzles (TLPs) [39]. The syntax follows the standard notation for TLPs except that we consider an additional setup phase that depends on the hardness parameter but not on the secret.

Definition 1 (Time-Lock Puzzles). *Let $\mathcal{S}$ be a finite domain. A time-lock puzzle (TLP) with solution space $\mathcal{S}$ is tuple of four algorithms* $(\mathsf{PSetup}, \mathsf{PGen}, \mathsf{PSol})$ *defined as follows.*

- $\mathsf{pp} \leftarrow \mathsf{PSetup}(1^\lambda, \mathbf{T})$ *a probabilistic algorithm that takes as input a security parameter 1^λ and a time hardness parameter $\mathbf{T}$, and outputs public parameters* pp.
- $Z \leftarrow \mathsf{PGen}(\mathsf{pp}, s)$ *a probabilistic algorithm that takes as input public parameters* pp*, and a solution $s \in \mathcal{S}$, and outputs a puzzle Z.*
- $s \leftarrow \mathsf{PSol}(\mathsf{pp}, Z)$ *a deterministic algorithm that takes as input public parameters* pp *and a puzzle Z and outputs a solution s.*

Definition 2 (Correctness). *A TLP scheme* $(\mathsf{PSetup}, \mathsf{PGen}, \mathsf{PSol})$ *is correct if for all $\lambda \in \mathbb{N}$, all polynomials $\mathbf{T}$ in λ, all secrets $s \in \mathcal{S}$, and all* pp *in the support of* $\mathsf{PSetup}(1^\lambda, \mathbf{T})$*, it holds that:* $\Pr\left[\mathsf{PSol}(\mathsf{pp}, \mathsf{PGen}(\mathsf{pp}, s)) = s\right] = 1$.

Security requires that the solution of the puzzles is hidden for all adversaries that run in (parallel) time less than $\mathbf{T}$.

Definition 3 (Security). *A TLP scheme* $(\mathsf{PSetup}, \mathsf{PGen}, \mathsf{PSol})$ *is secure with gap $\varepsilon < 1$ if there exists a polynomial $\tilde{\mathbf{T}}(\cdot)$ such that for all polynomials $\mathbf{T}(\cdot) \geq \tilde{\mathbf{T}}(\cdot)$ and every polynomial-size adversary $(\mathcal{A}_1, \mathcal{A}_2) = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda \in \mathbb{N}}$ where the depth of $\mathcal{A}_2$ is bounded from above by $\mathbf{T}^\varepsilon(\lambda)$, there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$ it holds that*

$$\Pr\left[\begin{array}{c} b \leftarrow \mathcal{A}_2(\mathsf{pp}, Z, \mathsf{st}) \\ \wedge\ (s_0, s_1) \in \mathcal{S}^2 \end{array} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{PSetup}(1^\lambda, \mathbf{T}(\lambda)) \\ (\mathsf{st}, s_0, s_1) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{pp}) \\ b \leftarrow \{0,1\}, Z \leftarrow \mathsf{PGen}(\mathsf{pp}, s_b) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda)$$

Homomorphic Time-Lock Puzzles. We also recall the definition of homomorphic TLPs [33], which allows one to compute functions on secrets homomorphically, without solving the puzzles first.

Definition 4 (Homomorphic TLPs). *Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of circuits (together with their respective representations). A TLP scheme* $(\mathsf{PSetup}, \mathsf{PGen}, \mathsf{PSol})$ *is homomorphic if the syntax is augmented with the following interface:*

- $Z' \leftarrow \mathsf{PEval}(C, \mathsf{pp}, Z_1, \ldots, Z_n)$ *a probabilistic algorithm that takes as input a circuit $C \in \mathcal{C}_\lambda$, public parameters* pp *and a set of n puzzles $(Z_1, \ldots, Z_n)$ and outputs a puzzle Z'.*

Homomorphic TLPs must satisfy the following notion of evaluation correctness.

Definition 5 (Evaluation Correctness). *Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of circuits (together with their respective representations). An homomorphic TLP scheme* $(\mathsf{PSetup}, \mathsf{PGen}, \mathsf{PSol}, \mathsf{PEval})$ *is correct (for the class $\mathcal{C}$) if for all $\lambda \in \mathbb{N}$, all polynomials $\mathbf{T}$ in λ, all circuits $C \in \mathcal{C}_\lambda$ and respective inputs $(s_1, \ldots, s_n) \in \mathcal{S}^n$, all* pp *in the support of* $\mathsf{PSetup}(1^\lambda, \mathbf{T})$*, and all Z_i in the support of* $\mathsf{PGen}(\mathsf{pp}, s_i)$*, the following conditions are satisfied:*

- *It holds that*

$$\Pr\left[\mathsf{PSol}(\mathsf{pp}, \mathsf{PEval}(C, \mathsf{pp}, Z_1, \ldots, Z_n)) = C(s_1, \ldots, s_n)\right] = 1.$$

- *There exists a fixed polynomial $p(\cdot)$ such that the runtime of* PSol *is bounded by $p(\lambda, \mathbf{T})$ and the runtime of* PEval *is bounded by $p(\lambda)$.*

We require homomorphic TLPs specifically that support homomorphic evaluations of linear functions over the puzzles, that are secure against depth bounded but sub-exponential size adversaries. We have such constructions from RSA groups [33] and Class groups with imaginary quadratic order [44]. These constructions are proven secure against such adversaries by conjecturing the hardness of the *sequential squaring assumption* [33,31] against depth bounded but sub-exponential size adversaries.

3.2 Puncturable Pseudorandom Functions

A puncturable pseudorandom function (PRF) is an augmented PRF that has an additional puncturing algorithm. Such an algorithm produces a punctured version of the key that can evaluate the PRF at all points except for the punctured one. It is required that the PRF value at that specific point is pseudorandom even given the punctured key. A puncturable PRF can be constructed from any one-way function [24].

Definition 6 (Puncturable PRFs). *A puncturable family of PRFs is a tuple of polynomial-time algorithms* $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Punc}, \mathsf{F})$ *defined as follows.*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ *a probabilistic algorithm that takes as input the security parameter 1^λ and outputs public parameters* pp*. Public parameters* pp *are taken as input in all other algorithms.*
- $K \leftarrow \mathsf{KGen}(\mathsf{pp})$ *a probabilistic algorithm that takes as input the public parameters* pp *and outputs a key K.*
- $K_i \leftarrow \mathsf{Punc}(K, i)$ *a deterministic algorithm that takes as input a key $K \in \mathcal{K}$ and a position $i \in \mathcal{X}$ and returns a punctured key K_i.*
- $y \leftarrow \mathsf{F}(K, i)$ *a deterministic algorithm that takes as input a key K and a string $i \in \mathcal{X}$ and returns a string $y \in \mathcal{Y}$.*

Definition 7 (Correctness). *For all $\lambda \in \mathbb{N}$, for all outputs $K \leftarrow \mathsf{KGen}(1^\lambda)$, for all points $i \in \mathcal{X}$ and $x \in \mathcal{X} \setminus i$, and for all $K_{-i} \leftarrow \mathsf{Punc}(K, i)$, we have that $\mathsf{F}(K_{-i}, x) = \mathsf{F}(K, x)$.*

We require that punctured points are pseudorandom to the eyes of any efficient distinguisher.

Definition 8 (Pseudorandomness at Punctured Points). *For all $\lambda \in \mathbb{N}$ and for every PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ there is a negligible function $\mu(\cdot)$, such that*

$$\Pr\left[b \leftarrow \mathcal{A}_2(\tau, K_i, i, y) : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), (i, \tau) \leftarrow \mathcal{A}_1(\mathsf{pp}) \\ K \leftarrow \mathsf{KGen}(\mathsf{pp}), K_i \leftarrow \mathsf{Punc}(K, i), b \leftarrow \{0,1\} \\ \text{if } b = 0 \text{ then } y \leftarrow \mathcal{Y}, \text{else } y \leftarrow \mathsf{F}(K, i) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda).$$

Key Homomorphism. We also assume the existence of constructions of puncturable PRFs that satisfy key homomorphism [10].

Definition 9 (γ-Almost Key-Homomorphic PRF). *Let function $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathbb{Z}_p^m$ be such that $(\mathcal{K}, +)$ is a group. Then the tuple $(\mathsf{F}, +)$ is said to be γ-almost key-homomorphic PRF if the following two conditions hold:*
- *F is a (puncturable) pseudorandom function.*
- *For all $k_1, k_2 \in \mathcal{K}$ and all $x \in \mathcal{X}$, there exists a vector $\mathbf{e} \in [0, \gamma]^m$ such that*

$$\mathsf{F}(k_1, x) + \mathsf{F}(k_2, x) = \mathsf{F}(k_1 + k_2, x) + \mathbf{e} \ (mod\ p).$$

The scheme presented in [10] satisfies (additive) key-homomorphism over $\mathbb{Z}_q^n$, which we also use in this work. Their scheme satisfies a weaker notion of correctness, which we state below.

Definition 10 (Computational Functionality Preservation). *For all $\lambda \in \mathbb{N}$ and all PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mu(\cdot)$, such that*

$$\Pr\left[\begin{array}{c} x^* \leftarrow \mathcal{A}_2^{\mathsf{F}(K,\cdot)}(1^\lambda, K_{i^*}, \tau) \wedge \\ x^* \neq i^* \wedge \\ \mathsf{F}(K, x^*) \neq \mathsf{F}(K_{i^*}, x^*) \end{array} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), K \leftarrow \mathsf{KGen}(\mathsf{pp}) \\ (i^*, \tau) \leftarrow \mathcal{A}_1(\mathsf{pp}) \\ K_{i^*} \leftarrow \mathsf{Punc}(K, i^*) \end{array}\right] \leq \mu(\lambda).$$

For our purposes, we require the above property of the key-homomorphic puncturable PRF from [10] to hold against super-polynomial adversaries, which is possible assuming the hardness of LWE against super-polynomial adversaries.

3.3 Indistinguishability Obfuscation

We recall the definition of indistinguishability obfuscation (iO) for circuits from [21].

Definition 11 (iO for Circuits [21]). *A uniform PPT machine $i\mathcal{O}$ is an indistinguishable obfuscator for circuit class $\{\mathcal{C}_\lambda\}$, if the following are satisfied:*
- *For all $\lambda \in \mathbb{N}$, or all $C \in \mathcal{C}_\lambda$, for all inputs x, we have*

$$\Pr\left[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(\lambda, C)\right] = 1$$

- *For all $\lambda \in \mathbb{N}$, all pairs of circuit $(C_0, C_1) \in \mathcal{C}_\lambda$ such that $|C_0| = |C_1|$ and $C_0(x) = C_1(x)$ on all inputs x, it holds that the distributions $\{i\mathcal{O}(\lambda, C_0)\}$ and $\{i\mathcal{O}(\lambda, C_1)\}$ are computationally indistinguishable.*

4 Time-Lock Puzzles with Batch Solving

In this section we formally present the notion and constructions for time-lock puzzles with batched solving.

4.1 Definition

We define the notion of TLPs with batched solving. We borrow the standard interfaces of a TLP from §3.1 and append it with an interface to allow for batched solving of n puzzles.

Definition 12 (Batch Solving). *A TLP scheme* $(\mathsf{PSetup}, \mathsf{PGen}, \mathsf{PSol})$ *supports* batch solving *with the aid of an additional interface defined below*

- $(s_1, \ldots, s_n) \leftarrow \mathsf{BatchPSol}(\mathsf{pp}, Z_1, \ldots, Z_n)$ *a deterministic algorithm that takes as input public parameters* pp *and puzzles* $Z_1, \ldots, Z_n$*, and outputs solutions* $s_1, \ldots, s_n$*.*

Definition 13 (Batch Solving Correctness). *An TLP scheme* $(\mathsf{PSetup}, \mathsf{PGen}, \mathsf{PSol})$ *with batch solving interface* $\mathsf{BatchPSol}$ *is correct if for all* $\lambda \in \mathbb{N}$*, all polynomials* $\mathbf{T}$ *in* λ*, all polynomials* n *in* λ*, all solutions* $(s_1, \ldots, s_n) \in \mathcal{S}^n$*, all* pp *in the support of* $\mathsf{PSetup}(1^\lambda, \mathbf{T})$*, and all* Z_i *in the support of* $\mathsf{PGen}(\mathsf{pp}, s_i)$*, the following conditions are satisfied:*

- *There exists a negligible function* $\mu(\cdot)$ *such that*

$$\Pr\left[\mathsf{BatchPSol}(\mathsf{pp}, Z_1, \ldots, Z_n) \neq (s_1, \ldots, s_n)\right] \leq \mu(\lambda).$$

- *There exist fixed polynomials* $p(\cdot), \tilde{p}(\cdot)$ *such that the size complexity of the circuit evaluating* $\mathsf{BatchPSol}(\mathsf{pp}, Z_1, \ldots, Z_n)$ *is bounded by* $p(\lambda, \mathbf{T}) + \tilde{p}(\lambda, n)$*.*

Notice that the above definition rules out trivial solutions, where you solve the n puzzles individually and output the solutions. This is because, in this solution the size scales with $n \cdot \mathbf{T}$, while the definition above only permits the scale to be $n + \mathbf{T}$. One can view $\tilde{p}(\lambda, n)$ as capturing the time taken to read and process the n puzzles, and returning the n solutions. The factor $p(\lambda, \mathbf{T})$ captures the solving of a single puzzle and itself is independent of n.

4.2 Bounded Batching of TLPs

As hinted to in §2.1, given a linearly homomorphic TLP with homomorphism over $\mathbb{Z}_q$, that has a large enough message space, it was shown in [43,44] that we can homomorphically pack several puzzles into a single puzzle using standard techniques. Solving the single puzzle reveals the solutions to all the n puzzles that we started out with. A crucial requirement for the above batch solving to work is that the message space of the homomorphic time-lock puzzle must be large enough to accommodate all the n λ-bit values. It was shown [43] that this is indeed possible by instantiating the Paillier-based linearly homomorphic time-lock puzzle construction from [33] in the same way Damgård-Jurik [17]

extended the Paillier cryptosystem [34]. That is, instantiate the Paillier-based linearly homomorphic TLP from [33] with a modulus N^s instead of modulus N^2, for a large enough value s. A similar domain extension was also shown in the settings of class groups of imaginary quadratic orders [44] that only require a public-coin (transparent) setup.

More formally, the LHP scheme from [33] has the $\mathsf{LHP.PSetup}$ algorithm output $\mathsf{pp}_{\mathsf{LHP}} = (\mathbf{T}, N, g, h)$, where $N = pq$ for some λ-bit primes p and q, g is the generator of $\mathbb{J}_N^*$, and $h := g^{2^{\mathbf{T}}} \bmod N$. Here $\mathbb{J}_N^*$ denotes the elements in $\mathbb{Z}_N^*$ with Jacobi symbol $+1$. The puzzle Z embedding message m is of the form (u, v) where

$$u := g^r \bmod N \text{ and } v := h^{r \cdot N}(1+N)^m \bmod N^2,$$

with randomness $r \leftarrow [N^2]$. The security of the scheme follows from the *sequential squaring assumption* [39,33]. The Damgård-Jurik extension from [33,43] lets the puzzle generation algorithm additionally choose $s \in \mathbb{Z}$, and set the puzzle $Z := (u, v)$ where

$$u := g^r \bmod N \text{ and } v := h^{r \cdot N^{s-1}}(1+N)^m \bmod N^s.$$

Here, the message space is $\mathbb{Z}_{N^{s-1}}$ while the puzzle component v is in $\mathbb{Z}_{N^s}$.

Consider n puzzles $Z_1, \ldots, Z_n$ each encoding λ-bit values with timing hardness $\mathbf{T}$, and each of these puzzles are of the Damgård-Jurik extended form. The $\mathsf{LHP.BatchPSol}$ algorithm internally evaluates the following linear function

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n} 2^{(i-1)\cdot\lambda} \cdot x_i$$

homomorphically over the puzzles using the $\mathsf{LHP.PEval}$ algorithm. The effect of this evaluation is that the resultant puzzle Z^* embeds the λ-bit values of $(x_1, x_2, \ldots, x_n)$. The $\mathsf{LHP.BatchPSol}$ algorithm proceeds to solve the resultant puzzle Z^* in time $\mathbf{T}$ to obtain the n values encoded as λ-bit values.

However, both of the above constructions only support bounded batching as they require the size of each puzzle Z_i (the v component) to scale linearly with the maximum batch size. Also, since the domain extension factor s has to be fixed at puzzle generation time, it determines an upper bound on the input size of the function f and therefore the number of puzzles we can batch solve later.

4.3 Unbounded Batching of TLPs

In this section, we present a new TLP scheme with batched solving which overcome both drawbacks of the scheme above. Namely, our new construction allows for batching where the size of the TLPs output by puzzle generation algorithm is independent of the number of puzzles to be batched. As a consequence, we have unbounded batching (bounded above by a super polynomial $2^{\omega(\log \lambda)}$) meaning that any polynomial number of puzzles can be batched with an one-time setup.

Our construction uses the following ingredients:

- $\underline{\mathsf{PSetup}(1^\lambda, \mathbf{T})}$:
  - Run $\mathsf{pp}_{\mathsf{LHP}} \leftarrow \mathsf{LHP.PSetup}(1^\lambda, \mathbf{T})$.
  - Run $\mathsf{pp}_{\mathsf{F}} \leftarrow \mathsf{Setup}(1^\lambda)$ and $\mathsf{pp}_{\overline{\mathsf{F}}} \leftarrow \overline{\mathsf{Setup}}(1^\lambda)$.
  - Return $\mathsf{pp} := (\mathsf{pp}_{\mathsf{LHP}}, \mathsf{pp}_{\mathsf{F}}, \mathsf{pp}_{\overline{\mathsf{F}}})$.
- $\underline{\mathsf{PGen}(\mathsf{pp}, m)}$:
  - Generate $k \leftarrow \mathsf{KGen}(\mathsf{pp}_{\mathsf{F}})$.
  - Define $\mathbf{P}_{m,k,\mathsf{pp}}(n, i, j)$ as the following circuit:[a]
    - Ensure $i, j \in [n]$.
    - Compute $(\overline{r}, r) \leftarrow \mathsf{F}(k, (n, i))$.
    - Compute $\overline{k} \leftarrow \overline{\mathsf{KGen}}(\mathsf{pp}_{\overline{\mathsf{F}}}; \overline{r})$.
    - Compute $Z \leftarrow \mathsf{LHP.PGen}(\mathsf{pp}_{\mathsf{LHP}}, \overline{k}; r)$.
    - If $j = i$ set $c = \overline{\mathsf{F}}(\overline{k}, j) + m \cdot \lceil p/2 \rfloor \pmod p$.
    - Else if $j \leq n \leq N$ set $c = \overline{\mathsf{F}}(\overline{k}, j)$.
    - Return (Z, c).
  - Return $\widetilde{\mathbf{P}} := i\mathcal{O}(1^\lambda, \mathbf{P}_{m,k,\mathsf{pp}})$.
- $\underline{\mathsf{BatchPSol}(\mathsf{pp}, Z_1, \ldots, Z_n)}$:
  - For each $i \in [n]$,
    - Parse $Z_i := \widetilde{\mathbf{P}}_i$.
    - For each $j \in [n]$, compute $(Z_i^*, c_{i,j}) \leftarrow \widetilde{\mathbf{P}}_i(n, i, j)$.
    - Define $\mathbf{c}_i = (c_{i,1}, \ldots, c_{i,n}) \in \mathbb{Z}_p^n$.
  - Set $Z^* \leftarrow \mathsf{LHP.PEval}(+, \mathsf{pp}, Z_1^*, \ldots, Z_n^*)$.
  - Compute $k^* \leftarrow \mathsf{LHP.PSol}(\mathsf{pp}, Z^*)$.
  - Compute $\mathbf{f}^* = \left(\overline{\mathsf{F}}(k^*, 1), \ldots, \overline{\mathsf{F}}(k^*, n)\right)$
  - Compute $\mathbf{c}^* = \sum_{i=1}^n \mathbf{c}_i$.
  - Return $\mathbf{c}^* - \mathbf{f}^*$ rounded component-wise.

[a] The circuit is padded to the maximum size of the circuits among those defined in the security proof. We refer the reader to the end of this Section for a discussion on the size of this circuit.

Fig. 1: Our construction for TLP with unbounded batch solving.

- A linearly homomorphic TLP scheme $\mathsf{LHP} := (\mathsf{LHP.PSetup}, \mathsf{LHP.PGen}, \mathsf{LHP.PSol}, \mathsf{LHP.PEval})$ where the homomorphism is over $\mathbb{Z}_q$.
- A puncturable PRF $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Punc}, \mathsf{F})$ denoted in short by F.
- An indistinguishable obfuscator $i\mathcal{O}$ for circuits.
- A γ-almost key-homomorphic puncturable PRF $(\overline{\mathsf{Setup}}, \overline{\mathsf{KGen}}, \overline{\mathsf{Punc}}, \overline{\mathsf{F}})$ (denoted in short by $\overline{\mathsf{F}}i$) with key space $\mathbb{Z}_q^n$ and where the noise bound is γ such that $p = 2^{\omega(\log \lambda)} \cdot \gamma$.

Let $N = 2^{\omega(\log \lambda)}$ denote an upper bound on the number of participants. Our construction $(\mathsf{PSetup}, \mathsf{PGen}, \mathsf{BatchPSol})$ is shown in Fig. 1. For simplicity we consider the messages encoded to be in $\{0, 1\}$, and argue that its straightforward to extend the construction for multiple bits.

A puzzle in our case is an obfuscation of the program $\mathbf{P}$ which has the message m, a PRF F key k, and the public parameters pp hardwired in it. The program $\mathbf{P}$ takes as input three values: n indicating number of puzzles to be batched, i "index" of the current puzzle and j "index" of other puzzles. It is important to note that the exact indices for each puzzle are only set later during batch solving. Let i be the symbolic index of the puzzle being generated now (whose concrete value will be set during batch solving). The program internally generates the PRF key $\overline{k}$ for the key-homomorphic puncturable PRF which then is embedded

inside the LHP puzzle Z. In case the indices i and j are the same, a ciphertext c is set to encrypt the message m using the value $\overline{\mathsf{F}}(\overline{k}, j)$ as the masking factor. In any other case, c encrypts 0 with $\overline{\mathsf{F}}(\overline{k}, j)$ as the masking factor. The program returns the puzzle Z and the ciphertext c.

The batch solving algorithm in the beginning, locally indexes the n puzzles in some order based on some rule (e.g., lexicographic ordering). We have them now ordered $(Z_1, \ldots, Z_n)$ where the i-th puzzle is an obfuscated program denoted by $Z_i := \widetilde{\mathbf{P}}_i$. Then, for all $i \in [n]$, we execute the program $\widetilde{\mathbf{P}}_i$ on values (n, i, j) for all $j \in [n]$. In the end we obtain a LHP time-lock puzzle Z_i^* and ciphertexts $c_{i,j}$ for each $j \in [n]$. Recall that when $i = j$ the program $\widetilde{\mathbf{P}}_i$ sets $c_{i,j}$ to encrypt the message m_i (where m_i is the message inside puzzle Z_i), and for all $i \neq j$, the ciphertext $c_{i,j}$ encrypts 0. We then obtain a LHP puzzle Z^* by homomorphically adding the puzzles Z_i^* for all $i \in [n]$ and solving Z^* returns a PRF key k^* of the key-homomorphic puncturable PRF. We retrieve the message m_j (for all $j \in [n]$) by doing the following: (1) compute $c_j^* = \sum_{i=1}^{n} c_{i,j}$, (2) evaluate $\overline{\mathsf{F}}(k^*, j)$, (3) set m_j as the rounding of $\left(c_j^* - \overline{\mathsf{F}}(k^*, j)\right)$. The correctness and security of our construction is formalized in the theorems below and the formal proofs are deferred to the extended version our paper [42].

Theorem 1. *Let LHP be a linearly homomorphic TLP scheme where the homomorphism is over $\mathbb{Z}_q$, let F be a puncturable PRF, let $i\mathcal{O}$ be an indistinguishable obfuscator for circuits and let $\overline{\mathsf{F}}$ be a γ-almost key-homomorphic puncturable PRF with key space $\mathbb{Z}_q^n$ and where the noise bound is γ such that $p = 2^{\omega(\log \lambda)} \cdot \gamma$. If all the above primitives are perfectly correct, then the TLP scheme with batch solving from Fig. 1 is perfectly correct.*

Theorem 2. *Let LHP be secure against depth $\mathbf{T}^{\varepsilon}(\lambda)$-bounded adversaries with sub-exponential advantage, F be a sub-exponentially secure puncturable PRF, $\overline{\mathsf{F}}$ be a sub-exponentially secure γ-almost key-homomorphic puncturable PRF and $i\mathcal{O}$ be a sub-exponentially secure indistinguishable obfuscator. Then, the construction from Fig. 1 is a secure time-lock puzzle with batch solving against all depth $\mathbf{T}^{\varepsilon}(\lambda)$-bounded adversaries.*

Size of the obfuscated circuit. Observe that at any point in the proof, we only hardwire information of size bounded by a fixed polynomial in λ, and in particular independent of the number of parties. Since the size of the obfuscated circuit must be padded to the maximum size of the circuit at any point in the security proof, the size overhead is also independent of the number of parties.

Instantiations and Setup assumptions. We can instantiate: the linearly homomorphic TLP, LHP, with the Class group based scheme from [44]. We can instantiate the puncturable PRF F with the GGM based PRF [40,24], the $i\mathcal{O}$ scheme with the scheme from [27], and the γ-almost key-homomorphic puncturable PRF with the scheme from [10]. Notice that the above instantiations do not require trusted setups, thus our TLP scheme does not require a trusted setup. However, requires a one-time transparent public-coin setup (for LHP from [44]).

TLP runtime. The runtime of PGen is dominated by the obfuscation of the circuit $\mathbf{P}$ which is polynomial in λ and size of $\mathbf{P}$. Moreover, the size of $\mathbf{P}$ is independent of the batch size n or the number of users. Thus, the total runtime is polynomial in λ and size of the time-locked message. The $\mathsf{BatchPSol}$ involves executing n obfuscated circuits, combining their outputs homomorphically using $\mathsf{LHP.PEval}$, and solving the resulting TLP using $\mathsf{LHP.PSol}$. The runtime of the first two operations is $poly(\lambda, n)$, whereas the last operation is $poly(\lambda, T)$.

Verifiable TLPs. We can support verifiability for our puzzles where the puzzle generator along with the puzzle also outputs a proof, that convinces a verifier that the puzzle is well-formed. In applications of our TLP scheme (including the ones in later sections), verifying whether a puzzle is in the support of the PGen is paramount for the correctness (Def. 13) of batched solving to hold. To provide such verifiability, we can add two new interfaces: $\mathsf{PProve}(Z, m, r)$ run by the puzzle generator that outputs a proof π to ascertain a puzzle Z is well-formed (with message m and randomness r), and $\mathsf{PVer}(Z, \pi)$ run by a verifier that validates the proof w.r.t. the puzzle. In terms properties we want that a verifier shouldn't be convinced of a malformed puzzle and that the proof does not help in solving the puzzle any faster. For a formal definition and a discussion on concrete instantiations, see the extended version of our paper [42].

Non-committing TLPs. A non-committing TLP lets a simulator generate a puzzle first and later "explain" the puzzle as committing to a message m by opening it to reveal m. Note that a TLP is committing to the message once the puzzle is generated. Cohen et al. [16] showed a generic approach to build such non-committing TLPs in the programmable random oracle model (PROM) and we can transform our TLP scheme into one that is non-committing in the same way. The idea is to run $Z' \leftarrow \mathsf{PGen}(\mathsf{pp}, x)$ for some random r, and the final puzzle Z is set as $Z := (Z', c)$ where $c := H(r) \oplus m$. The simulator when required to equivocate Z as a puzzle embedding the message m, sets the value $H(r) := c \oplus m$ on the fly, as $H()$ is modeled as a PROM. We can modify the construction from §4.3 by having letting PGen run as before, but output $(\widetilde{\mathbf{P}}, H(r) \oplus m)$ as the final puzzle, where $\widetilde{\mathbf{P}} := i\mathcal{O}(\lambda, \mathbf{P}_{r,k,\mathsf{pp}})$. Specifically this means that the PROM computation is outside the $i\mathcal{O}$.

5 Application 1: Byzantine Broadcast

In this section, we present our generic compiler to transform any BB protocol secure against weakly adaptive adversaries to one that is secure against strongly adaptive adversaries.

5.1 Model and Definitions

In our setting, there are n nodes, numbered 1 to n, running a distributed protocol where the identity of each node is known to one another through a PKI.

Communication model. We assume that each node has access to a shared global clock and all parties are connected by a pairwise reliable channel. We

consider the standard synchronous model of communication where there is a known upper bound on the message delay (Δ). The protocols are executed in a round-based fashion, where the duration of each round is Δ time units. Any message sent by an honest node in a round reaches all other honest nodes by the beginning of the next round. Also, each node has access to the functionalities: RECEIVE and SEND. When a node u invokes SEND$(m, \mathsf{recipients})$ in round $r-1$, then m is delivered to recipients using the pairwise reliable channels from u by round r. When a node u invokes RECEIVE in round r, then all messages that were sent to u using the pairwise reliable channels by round $r-1$ are returned. The adversary can read, rearrange, insert, and drop messages between any two nodes (if strongly adaptive). But, cannot forge signatures. Moreover, we also assume that each round is sufficiently long to perform standard cryptographic operations except BatchPSol and PSol.

Let $\mathcal{P}$ be the set of possible internal states of a node and $\mathcal{M}$ be the set of possible messages that can be sent and received by a node.

Definition 14 (Δ-secure Synchronous protocol). *Let $\mathcal{F}_n$ denote the family of transition functions such that:*

$$\mathcal{F}_n = \{f_{r,u} : \mathcal{P} \times \mathcal{M}^n \to \mathcal{P} \times \mathcal{M}^n : u \in [n], r \in \mathbb{Z}\}$$

A synchronous *protocol Π_{sync} is executed by n nodes and proceeds in rounds. In every round r, every node $u \in [n]$, reads the messages addressed to it using the* RECEIVE *functionality, updates its state and computes the messages to be sent using $f_{r,u}$, and sends the messages to intended recipients using the* SEND *functionality.*

Protocol $\Pi_{\mathsf{sync}}(\lambda, \Delta)$

Setup.
- Let $S_{0,u}$ be the initial state of node $u \in [n]$
- Generate and publish public parameters

Protocol. A node $u \in [n]$, for each round r:
- Fetch messages from each sender: $m := (m_1, \ldots, m_n) \leftarrow$ RECEIVE()
- Compute next state and messages: $(S_{r+1,u}, m' := (m'_1, \ldots, m'_n)) \leftarrow f_{r,u}(S_{r,u}, m)$
- Send messages: SEND$(m', \mathsf{recipients})$

Adversary model. The adversary can make at most t out of n nodes to arbitrarily deviate from the protocol execution, where $t < n$. Moreover, we assume that the adversary controls the delivery of all the messages in the network.

- We consider a *strongly adaptive adversary* that can corrupt nodes on the fly and perform *after-the-fact* removal.
- Whereas, a *weakly adaptive adversary* can *only* corrupt nodes on the fly, but cannot prevent the delivery of any message that was already sent.

Additionally, we consider a *rushing* adversary that can inspect the messages sent by any honest node before delivering it to other nodes. Moreover, we assume that honest nodes can irrecoverably erase (part of) its state and memory at any time.

Computational model. All honest nodes are sequential, random access PPT, but the adversary is a non-uniform probabilistic parallel machine with polynomially bounded parallelism running in polynomially bounded parallel steps.

5.2 Protocol

For an n node protocol Π, we define a deterministic function called *output derivation function* for each node $u \in [n]$. This function allows a node to compute its output bit for Π based on the transcript of public messages exchanged by the participants and public parameters.

Definition 15 (Output derivation function). *Let Π be an n node protocol and $\mathcal{Y}$ denote the **public** transcript space of the protocol Π, then $\mathcal{G}_n$ denote the family of **output derivation** functions such that:*

$$\mathcal{G}_n = \{g_u : \mathcal{Y} \to \{0,1\} : u \in [n]\}$$

Functions in $\mathcal{G}_n$, despite being deterministic, may not be efficiently computable without a party's keys.

We recall the definition of a secure Byzantine broadcast protocol below.

Definition 16 ((Δ, t)-secure Byzantine broadcast). *Let λ be the security parameter, Δ be the known upper-bound on the network delay, and node $d \in [n]$ be the* designated sender*. A protocol Π executed by n nodes with specified family of functions $\mathcal{G}_n$, where the* designated sender *holds an input bit $b \in \{0,1\}$, is a (Δ, t)-secure broadcast protocol tolerating at most t corruptions if it satisfies the following properties with probability $1 - negl(\lambda)$:*

- *Consistency: If two honest nodes output bit b_i and b_j respectively, then $b_i = b_j$.*
- *Validity: If the designated sender is honest, then every honest node outputs the designated sender's input bit b.*
- *Termination: Every honest node u outputs a bit from $g_u(\mathsf{transcript})$, where* transcript *is the transcript from running Π.*

*If the protocol can tolerate corruptions by a strongly adaptive and a weakly adaptive adversary, then it is **strongly adaptive** (Δ, t)-secure and **weakly adaptive** (Δ, t)-secure, respectively.*

Let $\Pi_{\mathsf{bb-wa}}$ be a weakly adaptive protocol, we formally describe $\Pi_{\mathsf{bb-sa}}$ below:

Protocol $\Pi_{\mathsf{bb-sa}}(\lambda, \Delta, \Pi_{\mathsf{bb-wa}}, \mathcal{G}_n)$

Text in gray indicates the instructions from $\Pi_{\mathsf{bb-wa}}$.

Setup.

- Let $S_{0,u}$ be the initial state of node $u \in [n]$
- For each round r, $\mathsf{pp} \leftarrow \mathsf{PSetup}(1^\lambda, \Delta)$
- Generate and publish public parameters

Input.

- Let $b \in \{0,1\}$
- If designated sender, d, then $S_{0,d} := S_{0,d} \cup b$

Protocol. A node $u \in [n]$, for each round r:

- Fetch messages from each sender: $m := (m_1, \ldots, m_n) \leftarrow \mathsf{RECEIVE}()$
- Parse message m_v as puzzle Z_v, ciphertext C_v, proof of well-formed π_v for all $v \in [n]$
- Check π_v's to verify if Z_v's are well-formed by $\mathsf{PVer}(\mathsf{pp}, Z_v, \pi_v)$
- Extract the individual solutions $(s_1, \ldots, s_n) \leftarrow \mathsf{BatchPSol}(\mathsf{pp}, Z_1, \ldots, Z_n)$
- Decrypt C_v's, set $m_v := C_v \oplus H(s_v)$ for all $v \in [n]$, and $m := (m_1, \ldots, m_n)$
- Set internal state for round r as $S_{r,u} := m_u$
- Compute next state and messages: $(S_{r+1,u}, m' := (m'_1, \ldots, m'_n)) \leftarrow f_{r,u}(S_{r,u}, m)$
- Pick $s \in \mathcal{S}$, $Z \leftarrow \mathsf{PGen}(\mathsf{pp}, s)$, and compute π to prove that Z is well-formed.
- Reassign $m'_u := (Z, S_{r+1,u} \oplus H(s), \pi)$ and $m'_v := (Z, m'_v \oplus H(s), \pi)$ for all $v \in [n] \setminus \{u\}$

– Set output messages as $m' := (m'_1, \ldots, m'_n)$ and erase $S_{r+1,u}$, s, π
– Send messages: SEND(m', recipients)

Output.
– Let transcript be the public transcript of the protocol execution
– Return $b \leftarrow g_u$(transcript)

Theorem 3. *Let $\Pi_{\mathsf{bb-wa}}$ be a weakly adaptive (Δ, t)-secure Byzantine broadcast protocol with output derivation functions $\mathcal{G}_n$ and $\Pi_{\mathsf{bb-sa}}$ be the compiled strongly adaptive (δ, t)-secure protocol with output derivation functions $\mathcal{G}_n$, such that $\Delta = 2\delta$. If an $\mathcal{A}$ violates $\Pi_{\mathsf{bb-sa}}$ with probability at least p, then there exists an adversary $\mathcal{B}$ that violates $\Pi_{\mathsf{bb-wa}}$ with probability at least p.*

Analysis. Suppose $\exists$ an $\mathcal{A}$ that can break $\Pi_{\mathsf{bb-sa}}$, then we build another adversary $\mathcal{B}$ that breaks $\Pi_{\mathsf{bb-wa}}$. At a high level, we show that every attack by $\mathcal{A}$ on $\Pi_{\mathsf{bb-sa}}$ can be translated to an attack on $\Pi_{\mathsf{bb-wa}}$. Observe that $\mathcal{B}$ is as powerful as $\mathcal{A}$, except $\mathcal{B}$ cannot perform *after-the-fact* removal. Thus, to translate the *after-the-fact* removal, $\mathcal{B}$ must know whether $\mathcal{A}$ delivers or removes messages in $\Pi_{\mathsf{bb-sa}}$. $\mathcal{B}$ can know this only by waiting for δ steps to see $\mathcal{A}$'s actions! Hence, $\mathcal{B}$ starts the simulation δ steps ahead of $\Pi_{\mathsf{bb-wa}}$. But, when the simulation begins, $\mathcal{B}$ doesn't yet have the real-world messages from $\Pi_{\mathsf{bb-wa}}$ that *can be copied* to $\Pi_{\mathsf{bb-sa}}$. So $\mathcal{B}$ sends non-committing TLPs to equivocate the contents of the puzzle (possible because of PROM). When $\mathcal{A}$ solves the TLP and queries the RO, actual messages from $\Pi_{\mathsf{bb-wa}}$ will be available, and $\mathcal{B}$ programs the RO to open the corresponding message from $\Pi_{\mathsf{bb-wa}}$. Since the duration between when the messages are sent and the contents learned by the honest nodes should be the same in the simulation and the real-world, we set $\Delta = 2\delta$. Thus, asymptotically, $\Pi_{\mathsf{bb-sa}}$ is round preserving (as $\Delta = 2\delta$) and communication preserving (due to compactness of our TLP). We present the detailed analysis in the extended version of our paper [42].

Expected constant-round Byzantine broadcast. Wan et al. [46] proposed an expected constant round BB protocol under a weakly adaptive and dishonest majority setting. Thus, using the compiler (§5.2), we can obtain resilience in the strongly adaptive setting!

6 Application 2: Nakamoto Consensus Secure Against a Mobile Sluggish Adversary

In this section, we show an attack against the Nakamoto consensus in the mobile sluggish model and how to secure the Nakamoto consensus using our TLP.

6.1 Attack on Nakamoto Consensus in the Mobile Sluggish Model.

In Nakamoto consensus, a chain forks when two distinct blocks extend the same parent block. Forks are inherently bad for security as it splits the honest mining efforts across the two branches of the tree. A benign example is when two blocks are mined less than Δ time units apart. Since the messages take Δ to reach

others, the winner of the second block would not have been aware of the previous block. Nakamoto consensus is parameterized in a way that the inter-arrival time between two blocks is much longer than the time to transmit between any two farthest nodes in the system. The security threat posed by forks is the exact reason the Nakamoto consensus is secure only in the synchronous model.

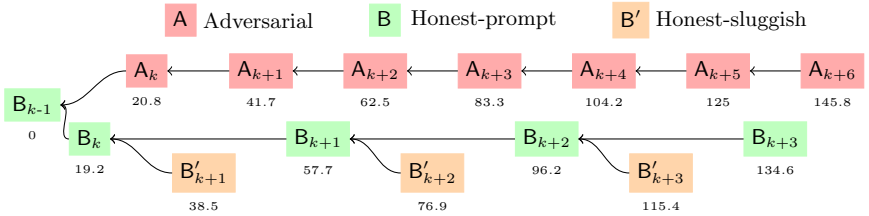

Fig. 2: Double spend attack: This plot depicts average block arrival times. Assuming 52 honest nodes (51 prompt + 1 sluggish) and 48 adversarial nodes, the average inter-arrival time of honest blocks and adversarial blocks in Bitcoin is 19.2 and 20.8 minutes, respectively. Observe that over 19.2×2 minutes, even though the honest nodes have mined two blocks, due to sluggishness, the honest chain has grown only by one block.

The mobile sluggish adversary, whenever an honest node mines a block, can simply delay the block propagation until another block extends the same header (see Fig. 2). At this point, the adversary can release both the blocks simultaneously to split the honest mining efforts. The adversary can sustain the forks as long as it has sufficient sluggish budget. Since the adversary is responsible for message delivery and sluggishness can be mobile, it could perform this attack repeatedly. In the meantime, adversarial nodes will continue to extend their chain in private. Using this strategy, even a single mobile sluggish fault has the ability to reduce the honest mining rate by *half*! Thus, an honest majority assumption may not be sufficient to guarantee security in this model. We elaborate this attack in the extended version of our paper [42].

6.2 Model

Let n be the total number of nodes, d be the maximum number of sluggish nodes, and t be the maximum number of adversarial nodes. Thus, there are at least $n-t$ honest nodes and at least $n-d-t$ prompt nodes. We adopt the formal framework from Garay et al. [20], a model inspired by the prior formulations of secure multiparty computation [13].

Sluggish network model. We assume that the time proceeds in rounds. Moreover, we assume that the adaptivity of the adversary is static. That is before the protocol execution, the adversary picks the set of nodes to corrupt. Moreover, we also assume that every node has access to a *shared global clock* and a pairwise reliable channel between any two parties.

The standard (lock-step) model of synchrony assumes that any message sent in round r reaches other nodes by $r+1$. We consider a generalization of this model called the *mobile sluggish* model. In this model, if a node is prompt at round r, then any message sent by the node in round $\leq r$ reaches all the nodes that are

prompt in round $r+1$ by round $\leq r+1$. Due to mobility of the sluggishness, set of prompt nodes in any two adjacent rounds need not be the same.

The adversary is responsible for message delivery. Thus, an adversary can reorder or delay messages (according to prompt and sluggish delay requirement), but *cannot* delete messages. Moreover, any message sent to a prompt node *by a prompt or an adversarial node* reaches all prompt nodes. We can relax this assumption by assuming that nodes gossip/echo any message they receive [35, Footnote 4]. The adversary inspects all messages (including puzzles and blocks) first before delivering to any node.

Round duration. We assume that the duration of a round is $O(\Delta)$. Specifically, we assume that a round is sufficiently long to send/receive messages and perform cryptographic operations (such as verifying a hash of a message, generating and verifying a zero-knowledge proof of well-formedness of a TLP, computing $\mathsf{PGen}/\mathsf{PEval}$, and signing and verifying a signature), *except* $\mathsf{PSol}, \mathsf{BatchPSol}$, and RO invocations to mine a block or a decoy.

Computational model. We adopt the *flat* model of computation introduced by Garay et al. [20]. In this model, all nodes are assumed to have the same computational power. Moreover, any node can make at most q proofs-of-work invocations to the RO in a round. Thus, the adversary can perform $t \cdot q$ RO queries in each round. We remark that each node has an unlimited number of proof-of-work verification queries to the RO [20].

Additionally, we assume that all honest nodes are sequential, random access PPT, but the adversary is a non-uniform probabilistic parallel machine with polynomially bounded parallelism running in polynomially bounded parallel steps.

Environment. The entity *environment* handles the external aspects of the protocol execution such as spawning the nodes and the adversary, injecting transactions, writing inputs and reading outputs of each node, etc. However, the environment *cannot* make queries to RO. This is to prevent the adversary from outsourcing the RO queries to an external entity.

6.3 Protocol

Super-round. Since our protocol proceeds in two phases: (1) Mine phase (m rounds) and (2) Solve phase (D rounds), a super-round consists of a mine phase followed by a solve phase. Thus, the duration is $(m+D)$ rounds.

Mobile Sluggish Nakamoto Protocol

Input.
- pp, TLP public parameters with $\mathbf{T}$ as one round
- m, duration of mine phase
- D, duration of solve phase
- q, maximum number of RO queries per round
- T, difficulty threshold to mine a block
- T_c, difficulty threshold to mine a decoy where $T < T_c$

Initialize. Chain C containing agreed-upon genesis block $\mathsf{C}[0]$

Protocol. Every super-round R (which consists of $(m+D)$ rounds)
- Get the $\mathsf{payload}$ from the environment

- Let $h_{-1} := \mathsf{H}(\mathsf{C}[-1])$ be the hash of the last block on the longest chain C
- Let $B = \perp$ be an empty block
- For m rounds of **mine phase**:
  - For q RO queries:
    - Pick random $\eta \in \{0,1\}^\lambda$ and compute $h := \mathsf{H}(h_{-1}, \mathsf{payload}, \eta)$
    - If $h < T_c$ (mined a decoy)
      - Overwrite $B := (h_{-1}, \mathsf{payload}, \eta)$
    - If $h < T$ (mined a block)
      - Overwrite $B := (h_{-1}, \mathsf{payload}, \eta)$
      - Set $\mathsf{C} := \mathsf{C}||B$
      - Break out of the q and m loop
- If $B \neq \perp$
  - Compute the TLP $Z := \mathsf{PGen}(\mathsf{pp}, B)$
  - Compute proof of well-formed $\pi := \mathsf{PProve}(\mathsf{pp}, Z, B)$
- **Solve phase** for $D = 2$ rounds:
  - First round, multicast (Z, π) (if one exists), receive all the w puzzles from the network $Z_1, \ldots, Z_w$, and check their well-formedness.
  - Second round, batch solve $(s_1, \ldots, s_w) := \mathsf{BatchPSol}(Z_1, \ldots, Z_w)$.
- Update the chain C based on output from the solve phase

Assumptions. Let a block mined in a super-round R be a *prompt block*, if mined by an honest node and the node was prompt *at the beginning* of solving phase of *both* $R-1$ and R. Moreover, let f be the probability of one or more prompt blocks were mined in a super-round, c be the probability of every honest node mining at least one decoy in a super-round, $\varepsilon, \delta \in (0,1)$ be parameters, and p be the probability of a RO query mining a block. Our analysis assumes that:

$$\frac{(m+D)t + md}{cm(n-2d-t)} \leq (1-\delta) \quad (1) \qquad pqm(n-2d-t) < 1/2 \quad (3)$$

$$\varepsilon + f < \delta/3 \quad (2) \qquad \frac{2\varepsilon}{1-\varepsilon} < \delta^2 \quad (4)$$

Analysis. At a high level, our analysis extends the formal tools proposed by Garay et al. [20]. But there are several differences due to mobile sluggish faults and the use of TLPs:

1. The adversary can deviate from the protocol and invoke RO queries even during the solve phase. Intuitively, Eq. 1 quantifies the required advantage of the prompt nodes over sluggish and adversarial nodes for our protocol to be secure. Specifically, the numerator captures the computational advantage enjoyed by the adversarial nodes due to additional RO queries during the solve phase (the term $(m+D)t$) and the loss in honest mining efforts due to sluggish nodes (the term md). Large values of D decreases t (assuming other values can remain the same). But, due to the batch solving property of our TLP, $D = 2$ in our protocol. Thus, the impact of D is minimal.
2. The mobility of the sluggishness provides the adversary timing based opportunities to reduce the contributions to the "prompt" chain. The adversary with d sluggish budget can toggle the sluggishness of $2d$ nodes. If the adversary toggles the sluggishness when the honest nodes release TLPs at the end of the mining phase, it can reduce the number of nodes contributing to the prompt chain to $(n-2d-t)$. This is because the d nodes that are sluggish through the mining phase of a super-round may not be mining on the longest chain, and at the end of the mining phase, the adversary can use its mobility

to make d prompt node sluggish (See [42, Remark 1]).

3. The sluggish nodes can inadvertently contribute to the adversarial chain. This is because the sluggish nodes may only have access to the view provided to them by the adversary.
4. Coordinated release of TLPs: Observe that from Eq. 3, large values of m decreases p, thus reducing the block arrival frequency. But, a bounded p ensures that the honest nodes do not fork one another and there are sufficient "convergence opportunities" to resolve forks [20,35]. Moreover, no prior permissionless protocol is secure under mobile-sluggish faults even under reduced performance.
5. Impact of decoys: In Eq. 1, the security impact of mining decoys by honest nodes is captured by c. We set the probability of mining a decoy such that honest nodes can mine sufficiently many decoys while simultaneously bounding the total number of decoys mined. Recall that our batch solvable TLP allows simultaneously opening a polynomial number of puzzles.

Notice that our analysis is a generalization of [20], thus by substituting $m = 1, c = 1, d = 0$, and $D = 0$, our analysis, in principle, collapses to [20]'s analysis. We prove liveness and consistency by assuming that the mining-hardness parameter is appropriately set in Eqs. 1 to 4. We present the complete analysis of the protocol in the extended version of the paper [42].

Acknowledgments This research was partially funded by the German Federal Ministry of Education and Research (BMBF) in the course of the 6GEM research hub under grant number 16KISK038 and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972. This work was also supported in part by Novi and VMware gift research grant. Charalampos Papamanthou was supported in part by the National Science Foundation, the Algorand Foundation through the ACE program, VMware, and Protocol Labs.

References

1. Time-lock: Block producer extractable value - tezos (2022), https://tezos.gitlab.io/alpha/timelock.html, [Online; accessed 01-Sept-2022]
2. Abraham, I., Chan, T.H.H., Dolev, D., Nayak, K., Pass, R., Ren, L., Shi, E.: Communication Complexity of Byzantine Agreement, Revisited. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (2019)
3. Abraham, I., Malkhi, D., Nayak, K., Ren, L., Yin, M.: Sync HotStuff: Simple and Practical Synchronous State Machine Replication. In: 2020 IEEE Symposium on Security and Privacy (SP) (2020)
4. Bagaria, V., Kannan, S., Tse, D., Fanti, G., Viswanath, P.: Prism: Deconstructing the blockchain to approach physical limits. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (2019)
5. Barak, B., Canetti, R., Lindell, Y., Pass, R., Rabin, T.: Secure Computation Without Authentication. In: Advances in Cryptology – CRYPTO 2005 (2005)
6. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: Snarks for c: Verifying program executions succinctly and in zero knowledge. In: Annual cryptology conference — CRYPTO 2013. pp. 90–108 (2013)

7. Bitansky, N., Garg, S., Lin, H., Pass, R., Telang, S.: Succinct randomized encodings and their applications. In: Proceedings of the forty-seventh annual ACM symposium on Theory of Computing. pp. 439–448 (2015)
8. Bitansky, N., Goldwasser, S., Jain, A., Paneth, O., Vaikuntanathan, V., Waters, B.: Time-lock puzzles from randomized encodings. In: Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science. pp. 345–356 (2016)
9. Boneh, D., Naor, M.: Timed Commitments. In: Advances in Cryptology — CRYPTO 2000 (2000)
10. Brakerski, Z., Vaikuntanathan, V.: Constrained key-homomorphic prfs from standard lattice assumptions. In: Theory of Cryptography Conference. pp. 1–30 (2015)
11. Burdges, J., Feo, L.D.: Delay encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 302–326 (2021)
12. Campanelli, M., David, B., Khoshakhlagh, H., Konring, A., Nielsen, J.B.: Encryption to the Future. In: Advances in Cryptology – ASIACRYPT 2022 (2022)
13. Canetti, R.: Security and Composition of Multiparty Cryptographic Protocols. Journal of Cryptology (2000)
14. Chan, B.Y., Shi, E.: Streamlet: Textbook Streamlined Blockchains. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies (2020)
15. Chen, H.C., Deviani, R.: A Secure E-Voting System Based on RSA Time-Lock Puzzle Mechanism. In: 2012 Seventh International Conference on Broadband, Wireless Computing, Communication and Applications (2012)
16. Cohen, R., Garay, J., Zikas, V.: Adaptively Secure Broadcast in Resource-Restricted Cryptography. Cryptology ePrint Archive, Report 2021/775 (2021)
17. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In: International workshop on public key cryptography. pp. 119–136 (2001)
18. Dwork, C., Naor, M.: Zaps and their applications. In: Proceedings 41st Annual Symposium on Foundations of Computer Science (2000)
19. Döttling, N., Hanzlik, L., Magri, B., Wohnig, S.: McFly: Verifiable Encryption to the Future Made Practical. Cryptology ePrint Archive, Paper 2022/433 (2022)
20. Garay, J., Kiayias, A., Leonardos, N.: The Bitcoin Backbone Protocol: Analysis and Applications. In: Advances in Cryptology - EUROCRYPT 2015 (2015)
21. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits. In: 2013 IEEE 54th Annual Symposium on Foundations of Computer Science (2013)
22. Garg, S., Gentry, C., Sahai, A., Waters, B.: Witness encryption and its applications. In: Proceedings of the ACM symposium on Theory of computing (2013)
23. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles (2017)
24. Goldreich, O., Goldwasser, S., Micali, S.: How to construct randolli functions. In: 25th Annual Symposium on Foundations of Computer Science. pp. 464–479 (1984)
25. Guo, Y., Pass, R., Shi, E.: Synchronous, with a Chance of Partition Tolerance. In: Advances in Cryptology – CRYPTO 2019 (2019)
26. Hanke, T., Movahedi, M., Williams, D.: DFINITY Technology Overview Series, Consensus System (2018)
27. Jain, A., Lin, H., Sahai, A.: Indistinguishability obfuscation from well-founded assumptions. In: Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing. pp. 60–73 (2021)
28. Katz, J., Loss, J., Xu, J.: On the Security of Time-Lock Puzzles and Timed Commitments. In: Theory of Cryptography (2020)

29. Kiffer, L., Rajaraman, R., shelat, a.: A Better Method to Analyze Blockchain Consistency. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (2018)
30. Kim, J., Mehta, V., Nayak, K., Shrestha, N.: Making Synchronous BFT Protocols Secure in the Presence of Mobile Sluggish Faults. Cryptology ePrint Archive, Report 2021/603 (2021)
31. Lin, H., Pass, R., Soni, P.: Two-Round and Non-Interactive Concurrent Non-Malleable Commitments from Time-Lock Puzzles. In: 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS) (2017)
32. Liu, J., Jager, T., Kakvi, S.A., Warinschi, B.: How to build time-lock encryption. Designs, Codes and Cryptography pp. 2549–2586 (2018)
33. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic Time-Lock Puzzles and Applications. In: Advances in Cryptology – CRYPTO 2019 (2019)
34. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International conference on the theory and applications of cryptographic techniques. pp. 223–238 (1999)
35. Pass, R., Seeman, L., Shelat, A.: Analysis of the Blockchain Protocol in Asynchronous Networks. In: Advances in Cryptology – EUROCRYPT 2017 (2017)
36. Pass, R., Shi, E.: FruitChains: A Fair Blockchain. In: Proceedings of the ACM Symposium on Principles of Distributed Computing (2017)
37. Pass, R., Shi, E.: Rethinking Large-Scale Consensus. In: 2017 IEEE 30th Computer Security Foundations Symposium (CSF) (2017)
38. Ren, L.: Analysis of Nakamoto Consensus. Cryptology ePrint Archive, Report 2019/943 (2019)
39. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-Lock Puzzles and Timed-Release Crypto. Tech. rep. (1996)
40. Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. SIAM Journal on Computing (2021)
41. Shrestha, N., Abraham, I., Ren, L., Nayak, K.: On the Optimality of Optimistic Responsiveness. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (2020)
42. Srinivasan, S., Loss, J., Malavolta, G., Nayak, K., Papamanthou, C., Thyagarajan, S.A.: Transparent Batchable Time-lock Puzzles and Applications to Byzantine Consensus. Cryptology ePrint Archive, Paper 2022/1421 (2022), https://eprint.iacr.org/2022/1421
43. Thyagarajan, S.A.K., Bhat, A., Malavolta, G., Döttling, N., Kate, A., Schröder, D.: Verifiable timed signatures made practical. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (2020)
44. Thyagarajan, S.A.K., Castagnos, G., Laguillaumie, F., Malavolta, G.: Efficient cca timed commitments in class groups. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (2021)
45. Wan, J., Xiao, H., Devadas, S., Shi, E.: Round-Efficient Byzantine Broadcast Under Strongly Adaptive and Majority Corruptions. In: Theory of Cryptography (2020)
46. Wan, J., Xiao, H., Shi, E., Devadas, S.: Expected Constant Round Byzantine Broadcast Under Dishonest Majority. In: Theory of Cryptography (2020)
47. Yu, H., Nikolic, I., Hou, R., Saxena, P.: OHIE: Blockchain Scaling Made Simple. In: 2020 IEEE Symposium on Security and Privacy (SP) (2020)
48. Zhao, J., Tang, J., Li, Z., Wang, H., Lam, K.Y., Xue, K.: An Analysis of Blockchain Consistency in Asynchronous Networks: Deriving a Neat Bound. In: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS) (2020)