Time-Lock Puzzles with Efficient Batch Solving

Jesko Dujmovic[1], Rachit Garg[2], and Giulio Malavolta[3]

[1]Helmholtz Center for Information Security (CISPA) and Saarbrücken Graduate School of Computer Science
Email:jesko.dujmovic@cispa.de
[2]UT Austin
Email:rachg96@cs.utexas.edu
[3]Bocconi University and Max Planck Institute for Security and Privacy.
Email:giulio.malavolta@hotmail.it

Abstract

Time-Lock Puzzles (TLPs) are a powerful tool for concealing messages until a predetermined point in time. When solving multiple puzzles, it becomes crucial to have the ability to *batch-solve* puzzles, i.e., simultaneously open multiple puzzles while working to solve a *single one*. Unfortunately, all previously known TLP constructions that support batch solving rely on super-polynomially secure indistinguishability obfuscation, making them impractical.

In light of this challenge, we present novel TLP constructions that offer batch-solving capabilities without using heavy cryptographic hammers. Our proposed schemes are simple and concretely efficient, and they can be constructed based on well-established cryptographic assumptions based on pairings or learning with errors (LWE). Along the way, we introduce new constructions of puncturable key-homomorphic PRFs both in the lattice and in the pairing setting, which may be of independent interest. Our analysis leverages an interesting connection to Hall's marriage theorem and incorporates an optimized combinatorial approach, enhancing the practicality and feasibility of our TLP schemes.

Furthermore, we introduce the concept of "rogue-puzzle attacks", where maliciously crafted puzzle instances may disrupt the batch-solving process of honest puzzles. We then propose constructions of concrete and efficient TLPs designed to prevent such attacks.

Contents

1 Introduction

A Time-Lock Puzzle (TLP) is a cryptographic primitive that allows one to hide a message for a predetermined amount of time (denoted by T). TLPs possess two essential characteristics *efficency* and *sequentiality*. Efficiency requires that computing the puzzle is significantly faster, ideally in logarithmic time, relative to T. Sequentiality, on the other hand, demands that any potential adversary should not be able to solve the puzzle in less time than the stipulated duration T, even when employing parallel computational resources. Rivest, Shamir, and Wagner [RSW96] constructed the first TLP based on the conjectured sequentiality of repeated modular squaring in RSA groups. Ever since, TLPs have found a staggering variety of applications, including sealed-bid auctions [MT19], e-voting systems [MT19], fair contract signing [BN00], non-malleable commitments [LPS17], cryptocurrency payment systems [TMSS22], distributed consensus algorithms [WXDS20], and byzantine consensus protocols [SLM⁺23], to name a few. Time-lock puzzles have transitioned from theoretical constructs to practical tools and have been utilized in real-world protocols such as private blockchain voting [1].

Solve one, open many. The fundamental characteristic of time-lock puzzles (TLPs) is their reliance on a significant amount of sequential computation to be solved. However, this property can introduce challenges in protocols involving multiple puzzles. As the number of puzzles to be solved increases, the computational overhead required to complete the protocol can quickly become impractical. Moreover, this efficiency bottleneck can be exploited as an attack vector, potentially obstructing the successful termination of a protocol. For example, adversaries might flood the network with unopened puzzles, particularly in cases where an unfavourable outcome is expected.

This limitation has recently motivated new TLP constructions [MT19, BDGM19, SLM$^+$23, BF21] that offer a way around this problem. They design a cryptographic protocol that allows the solver to open many puzzles at the cost of a *single* puzzle opening. The work by [SLM$^+$23] is particularly interesting, which proposed the first construction of TLPs with *batched solving*. In this approach, when faced with multiple puzzles n, each with a time-lock duration of T, a solver can recover all n puzzles without solving all of them individually. Remarkably, the computational effort required remains the same as solving a single puzzle. Notably, the parties generating and computing these puzzles need not coordinate or even be aware of each other's participation.

While [SLM$^+$23] establishes the theoretical feasibility of batched solving, their scheme relies on the existence of general-purpose indistinguishability obfuscation [BGI$^+$01, GGH$^+$13]. Therefore, given the state of affairs of current obfuscation constructions [JLS21, GJLS21, WW21, GP21, BDGM22, JLS22], it is fair to say that their scheme is far from practically efficient and considered a heavyweight cryptographic primitive not ready for efficient deployment (there are certain *restricted* functionalities [LMA$^+$16, CMR17] but there are no general purpose implementations). This motivates the following question:

Can we build *concretely efficient* TLPs with batch solving?

1.1 Our Results

In this work, we propose a new approach to construct TLPs with batch solving. Our contributions are summarized below.

[1] https://cointelegraph.com/news/a16z-releases-anonymous-voting-system-for-ethereum.

(1) Generic transformation for batch solving. We present a generic method for constructing TLPs that support batch solving. Our construction builds upon and simplifies the concepts introduced in a prior work [SLM+23]. The construction is based on the combination of two key components: linearly homomorphic TLPs [MT19] and puncturable Key-Homomorphic PseudoRandom Functions (KH-PRFs). The resulting scheme is conceptually simple, based on well-understood computational assumptions, and *concretely efficient*. Depending on the number of homomorphic key operations allowed by our KH-PRF and the domain size, we consider two flexible settings. In the "unbounded" setting, the solver can batch an unlimited number of time-lock puzzles. In contrast, in the "bounded" setting, the setup phase of the TLP imposes an apriori limit on the size of the number of puzzles that can be batched. Notably, the runtime of the puzzle generation and the size of the puzzle are independent of this bound. Our solving algorithm for the bounded settings leverages a novel connection to Hall's marriage theorem. This connection allows us to enhance the concrete parameters of our scheme, contributing to its practical efficiency.

(2) New Puncturable Key-Homomorphic PRFs. We present two constructions of KH-PRFs.

- **Lattice-based puncturable KH-PRFs:** We propose a new construction of KH-PRF based on the hardness of the standard learning with errors (LWE) problem, with superpolynomial modulus to noise ratio. Compared with prior work [BV15], our scheme is conceptually simpler, practically more efficient, and does not need to assume the hardness of the 1D-SIS problem, which was required in [BV15]. 3λ matrix multiplications dominate the computational cost of evaluating our KH-PRF. Additionally, this puncturable key-homomorphic PRF incorporates a transparent setup.

  In the bounded setting (where the number of homomorphic operations is apriori bounded), we devise a puncturable PRF based on the LWE assumption with a polynomial modulus. Proving security requires care in resampling keys.

- **Pairing-based puncturable KH-PRFs:** We also show how to build the first puncturable KH-PRF from bilinear groups where the domain size is polynomially bounded. Prior to our work, group-based PRFs were either key-homomorphic [NPR99] or puncturable [SW14] but did not satisfy both properties. We present two constructions based on standard assumptions in bilinear groups featuring quadratic and linear public parameters, respectively. Notably, the evaluation of these PRFs requires just a single pairing operation.

  We note that our pairing-based construction requires a trusted setup. However, the setup is structured and more desirable than an "arbitrary" structured distribution. The structured reference string in the linear-CRS construction can be jointly sampled by mutually distrustful parties in an efficient manner [NRBB22]. Once the reference string has been sampled, we do not make additional trust assumptions. The same reference string can also be reused across multiple independent protocol instantiations. Furthermore, it can be updated if more parties wish to join the system using techniques in [GKM+18]. Additionally, batched TLPs also have applications in the setting with a private-coin setup. For instance, auctions and e-voting can also be realized using a TLP with batch solving and trusted setup.

(3) Security against rogue-puzzle attacks. We initiate the study of batch-solving algorithms secure against "rogue-puzzle attacks". In this scenario, we consider attackers capable of crafting malicious puzzles with the intent of disrupting the batch-solving process of legitimately generated puzzles. This notion is particularly relevant in large-scale scenarios, where one cannot trust users to generate their puzzles honestly, yet we want to guarantee correct termination for honest participants. Without this guarantee, batch-solving provides little advantage compared to the trivial solution since an adversary may stall the protocol by tampering with the output of the batch-solving procedure. Identifying and addressing this notion represents a primary conceptual contribution to the deployment of a batchable time lock puzzle.

In this context, we provide formal definitions of security against rogue-puzzle attacks and demonstrate how to enhance our TLP constructions to meet this security requirement. Along the way, we propose efficient zero-knowledge protocols for verifying the integrity of a puzzle to ensure that it is well-formed.

(4) Implementation and performance evaluation. To substantiate our claims for practicality, we present the first implementation of time-lock puzzles with batch solving. We consider two main parameters: batch-solving time and communication size. We present our results in Section 7.1 and mention some key takeaways below. For batching 500 puzzles where the hardness of the puzzle has to compute 500 million exponentiations, our batch-solving algorithm runs in 22.5 minutes. In comparison, a single puzzle takes 18.5 minutes to solve. For growing time parameter T, we expect the gap to narrow down. In terms of communication, for batching $7k$ puzzles, we only transmit a total size of 40 MB. We also discuss different tradeoffs between communication size and computational time (as highlighted in Remark 7.2) to cater to specific application requirements. Our code demonstrates that time-lock puzzles with batch solving can be implemented with currently available hardware, and have the potential for substantial savingss in large-scale protocols.

1.2 Technical Overview

In this section, we'll provide a technical overview of our solutions and the techniques developed within our work. This overview will encompass our main construction template, the efficient instantiation of underlying building blocks, and the concept of security against rogue-puzzle attacks.

A strawman solution. Before explain our construction, let us show how existing tools already give a weak form of batch solving. If we start from a homomorphic time-lock puzzle [MT19] over $\mathbb{Z}_N$ (for a large enough N), one way to batch puzzles is to homomorphically evaluate the packing function. In more details, given n puzzles $Z_1, \ldots, Z_n$ (of some linearly homomorphic time-lock puzzle) where each puzzle contains some λ-bit message, we can evaluate homomorphicaly the following linear function:

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

We can then solve the resulting puzzle Z^* to obtain all the n messages, encoded in different portions of the bit-string. While syntactically correct, this solution suffers from two important limitations:

- Bounded batching: Since the plaintext space needs to be large enough to accommodate all of the n messages, this means that at puzzle generation time one has to fix a bound on the number of batchable puzzles n.

- Quadratic overhead: In settings where n parties compute the puzzles separately, each puzzle must be of size at least n (for the reason specified above) and therefore the total communication of the protocol grows with $O(n^2)$.

Given this baseline, our objective is to improve on either of these properties (ideally both), without sacrificing the practical efficiency of the scheme.

Our Construction. Our generic construction is inspired by the work of [SLM+23], and our main observation is to decouple the task of assigning a unique identifier to each user from the batch-solving mechanism. We start by explaining our construction in the *simplified* settings where all parties computing a puzzle are associated with a *unique* index $i \in [n]$, and we assume that there are no collisions. Later in this overview, we will show how to remove this assumption. We will also assume the existence of a puncturable key-homomorphic PRF (KH-PRF) with domain at least n, where the adjective *puncturable* means that we can create a punctured version of the PRF key k at some point i^*, in such a way that the punctured key k^* allows one to evaluate the PRF at all points, except for $\mathsf{PRF}(\mathsf{k}, i^*)$. Furthermore, the PRF must be key homomorphic in the sense that for any two keys k_0 and k_1 and all points i it holds that

$$\mathsf{PRF}(\mathsf{k}_0, i) + \mathsf{PRF}(\mathsf{k}_1, i) \approx \mathsf{PRF}(\mathsf{k}_0 + \mathsf{k}_1, i).$$

We are now ready to describe how to augment a linearly homomorphic time-lock puzzle with the batch solving algorithm. We outline the algorithms below.

- Puzzle Generation: On input a message m_i and a unique index i, the puzzle generation algorithm samples a random PRF key k_i and computes the punctured key k_i^* at point i. Then it computes Z_i as the time-lock puzzle containing the key k_i and returns

$$\left\{Z_i, \mathsf{k}_i^*, i, c_i = \mathsf{PRF}(\mathsf{k}_i, i) + m_i\right\}.$$

- Batch Solving: To solve n puzzles as defined above, one can sum the puzzles homomorphically to obtain

$$(Z_1, \ldots, Z_n) \xrightarrow{\mathsf{Sum}} Z^* \in \mathsf{Gen}\left(\sum_i \mathsf{k}_i\right)$$

  and solve Z^* to recover $\tilde{k} = \sum_i \mathsf{k}_i$. The solver can recover each message m_i individually by computing

$$\begin{aligned} c_i + \sum_{j \neq i} \mathsf{PRF}(\mathsf{k}_j^*, i) + \mathsf{PRF}(\tilde{k}, i) &= \mathsf{PRF}(\mathsf{k}_i, i) + m_i + \sum_{j \neq i} \mathsf{PRF}(\mathsf{k}_j^*, i) - \mathsf{PRF}(\tilde{k}, i) \\ &= \mathsf{PRF}(\mathsf{k}_i, i) + m_i + \sum_{j \neq i} \mathsf{PRF}(\mathsf{k}_j, i) - \mathsf{PRF}\left(\sum_i \mathsf{k}_i, i\right) \\ &\approx \mathsf{PRF}(\mathsf{k}_i, i) + m_i + \sum_{j \neq i} \mathsf{PRF}(\mathsf{k}_j, i) - \sum_i \mathsf{PRF}(\mathsf{k}_i, i) \\ &= m_i \end{aligned}$$

  Where the (approximate) equalities follow from the puncturable correctness and the approximate key homomorphism of the PRF.

This should be contrasted with the scheme from [SLM+23], which is based on a similar principle, but instead of sending the puzzles in the plain, it sends an obfuscated circuit that samples a different puzzle for a given index i. Additionally, our work introduces a novel mechanism for uniquely assigning indices to parties (detailed below).

Batching without coordination. We observe that our batching algorithm requires the following property - when any subset of users $\mathcal{S} \subseteq [n]$ come together to batch a puzzle, each puzzle $i \in \mathcal{S}$ should have a unique identifier at which it is evaluated. If $n = 2^\lambda$, i.e., our batching scheme and the underlying key homomorphic PRF can support unbounded users, then simply sampling a random index of λ bits is enough. In such a setting, if any polynomial number of parties $\mathcal{S}$ come together, then the probability for any two parties to have a collision in their random sampling is $\leq |\mathcal{S}|^2/n$. Since n is exponential, we only fail with negligible probability. Unfortunately this trivial solution fails when our scheme can support bounded users. Specifically, we won't be able to batch with a non-negligible loss.

Our main observation is a connection between the existence of a unique identifier for each party and the problem of perfect matching in a bipartite graph. Let U and V be the two parts of the bipartite graph where U is the set of parties in a system i.e. $|U| = n$ and V is some expanded index set where $|V| = n_{\mathsf{new}}$. Instead of sampling a single random index in the trivial solution, we assume that each party on the left samples d numbers randomly in $[n_{\mathsf{new}}]$. Note that each party possessing a unique index is equivalent to the existence of a perfect matching in the bipartite graph. We ask what's the optimal setting for n_{new} and d where growing d will increase the time to generate puzzles and the communication cost between parties, while growing n_{new} will grow the public parameters pp of our batching scheme. In our main technical section, we apply Hall's marriage theorem in our probabilistic analysis to show that we can set $n_{\mathsf{new}} \geq 3 \cdot n$ and $d = \lambda/\log(n_{\mathsf{new}})$. Hall's marriage theorem states that for every subset $\mathcal{X} \subseteq \mathcal{S}$, there exists a perfect matching if $|\Gamma(\mathcal{X})| \geq |\mathcal{X}|$, where $\Gamma(\mathcal{X})$ denotes the set of neighbouring vertices to $\mathcal{X}$.

KH-PRFs: Lattice-Based Constructions. Brakerski and Vaikuntanathan [BV15] showed how to construct a constrained-key almost key-homomorphic PRF secure from lattice-based assumptions. However, this construction is designed for general constraints and hence impractical for our specific use case for puncturing. Their construction uses (1) a universal circuit for constraining general circuits, (2) makes non-black-box use of a cryptographic hash function, and additionally, (3) their security relies on LWE and 1D-SIS, which limit parameter choices and introduce additional security features. In contrast, we simplify their construction for the functionality and security we need. As a result, our construction is more efficient, makes black-box use of cryptography and eliminates the reliance on 1D-SIS. Our main changes include (1) replacing the universal circuit with a much simpler equality-check circuit, (2) removing the use of a hash function, and (3) not requiring 1D-SIS for our security proof. At a high level, last two modifications are possible because a puncturable PRF is a selective notion, whereas the construction of constrained-key PRF in [BV15] achieves adaptive security.

To gain some context, we first give a brief overview of the techniques from [BV15]. Given matrices $\{\mathbf{A}_i\}$, they show how to compute a new matrix $\mathbf{A}_F$ for some circuit F. Additionally, given LWE samples $\left\{\mathbf{s}^T\mathbf{A}_i + x_i\mathbf{G} + \mathbf{e_i}^T\right\}_{i\in[\ell]}$ over the modulus q for some $x = (x_1, \ldots, x_\ell)$, they give an algorithm to compute $\mathbf{s}^T\mathbf{A}_F + F(x)\mathbf{G} + \mathbf{e}^T$ for some small $\mathbf{e}$ and the gadget matrix $\mathbf{G}$. In our construction, we focus on the equality-check circuit $EQ_y(x)$ with a hardcoded string y. The circuit outputs 1 if and only if $x = y$. We compute our PRF as,

$$\mathsf{PRF}(\mathbf{s}, x) = \lfloor \mathbf{s}^T\mathbf{A}_{EQ_x} G^{-1}(\mathbf{D}) \rceil_p,$$

for some uniformly random matrix $\mathbf{D}$ and the binary decomposition function G^{-1}. The notation $\lfloor\cdot\rceil_p$ means we multiply each component with p/q and round to the next integer where the choice of p is elaborated later in the overview. Puncturing the key $\mathbf{s}$ at point x^* computes,

$$\mathsf{Puncture}(\mathbf{s}, x^*) = \left\{\mathbf{s}^T(\mathbf{A}_i + x_i^*\mathbf{G}) + \mathbf{e}_i^T\right\}_{i\in[\ell]}.$$

Given a punctured key k, we use the algorithm from [BV15] to compute $\mathbf{s}^T(\mathbf{A}_{EQ_x} + EQ_x(x^*)\mathbf{G} + \mathbf{e}^T)G^{-1}(\mathbf{D})$. Observe that if $x \neq x^*$ then, we can compute,

$$\begin{aligned}\mathsf{PuncturedEval}(\mathsf{k}^*, x) &= \lfloor (\mathbf{s}^T(\mathbf{A}_{EQ_x} + EQ_x(x^*)\mathbf{G}) + \mathbf{e}^T)G^{-1}(\mathbf{D})\rceil_p \\ &= \lfloor \mathbf{s}^T\mathbf{A}_{EQ_x}G^{-1}(\mathbf{D}) + \mathbf{e}^T G^{-1}(\mathbf{D})\rceil_p \\ &= \lfloor \mathbf{s}^T\mathbf{A}_{EQ_x}G^{-1}(\mathbf{D})\rceil_p + \{-1, 0, 1\}^m\end{aligned}$$

where the last equality holds with if we set our parameters such that q/p is bigger than $\|\mathbf{e}G^{-1}(\mathbf{D})\|_\infty$. Intuitively, security relies on the fact that when $x = x^*$, an adversary can only compute $\lfloor \mathbf{s}^T\mathbf{A}_{EQ_x}G^{-1}(\mathbf{D}) + \mathbf{s}^T\mathbf{G}\, G^{-1}(\mathbf{D}) + \mathbf{e}^T G^{-1}(\mathbf{D})\rceil_p = \lfloor \mathbf{s}^T\mathbf{A}_{EQ_x}G^{-1}(\mathbf{D}) + \mathbf{s}^T\mathbf{D} + \mathbf{e}^T G^{-1}(\mathbf{D})\rceil_p$. In our security proof, the intuition is to add extra noise $\mathbf{e}'$ to

$$\mathbf{s}^T\mathbf{A}_{EQ_x}G^{-1}(\mathbf{D}) + \mathbf{s}^T\mathbf{D} + \mathbf{e}^T G^{-1}(\mathbf{D}) \tag{1}$$

while maintaining the rounded expression. If we can do this, then $\mathbf{s}^T\mathbf{D} + \mathbf{e}'^T$ is a valid LWE sample , and we can use LWE security to make the term pseudorandom and completing the proof. In the case where q/p is superpolynomial, then adding error vector $\mathbf{e}'$ is unlikely to change the rounded value through a standard smudging argument.

Extending to a polynomial modulus-to-noise ratio. If we want the rely on LWE security that has a modulus-to-noise ratio that is polynomial, two issues arise - (1) The key-homomorphic operation of the PRF accumulates noise. Because our PRF is not perfectly key homomorphic but only almost key homomorphic (i.e. $\mathsf{PRF}(\mathbf{s}, x) + \mathsf{PRF}(\mathbf{s}', x) = \mathsf{PRF}(\mathbf{s} + \mathbf{s}') + \{-1, 0, 1\}^m$), summing these values accumulates noise. Our solution is to choose a sufficiently large p to minimize the impact of noise accumulation. In our application, this translates into an upper bound on the number of parties in the batch-solving algorithm, so that we can choose p accordingly. (2) If q/p is polynomial, then adding extra noise to the term in Eq. (1) is likely to change the rounded value. We resolve the second problem by resampling the key if adding noise to the term in Eq. (1) might change the rounded value. This is possible because at key generation time we know the point where we are going to puncture the PRF.

KH-PRFs: Pairing-Based Constructions. We also show a simple construction of key-homomorphic puncturable PRFs from groups. Our starting point is the existing construction [NPR99, BLMR13] in the random oracle model where

$$\mathsf{PRF}(\mathsf{k}, i) = \mathsf{H}(i)^{\mathsf{k}} \qquad \text{and} \qquad \mathsf{H}(i)^{\mathsf{k}_0} \cdot \mathsf{H}(i)^{\mathsf{k}_1} = \mathsf{H}(i)^{\mathsf{k}_0 + \mathsf{k}_1}.$$

Unfortunately it is not clear how to make this construction puncturable, without breaking the key homomorphism. Our observation is that, if we restrict ourselves to a bounded domain $n = \mathrm{poly}(\lambda)$, we can precompute in the setup n group elements

$$(g^{x_1}, \ldots, g^{x_n}) \qquad \text{and} \qquad \left\{g^{z_i/x_j}\right\}_{j \neq i}$$

where $x_i \leftarrow \mathbb{Z}_p^*$ and $z_i \leftarrow \mathbb{Z}_p^*$. For a uniformly sampled key k, we will then define the PRF output to be

$$\mathsf{PRF}(\mathsf{k}, i) = e\left(g^{x_j}, g^{z_i/x_j}\right)^{\mathsf{k}} = e(g, g)^{z_i \cdot \mathsf{k}}$$

for some $j \neq i$. Notably, this scheme preserves key homomorphism, satisfying:

$$e(g,g)^{z_i \cdot \mathsf{k}_0} \cdot e(g,g)^{z_i \cdot \mathsf{k}_1} = e(g,g)^{z_i \cdot (\mathsf{k}_0 + \mathsf{k}_1)}.$$

This construction is puncturable, and a punctured key, and a punctured key $\mathsf{k}^*_{i^*}$ can be computed as $g^{x_{i^*} \cdot \mathsf{k}}$. Observe that we can compute the PRF value at all points (by pairing it with the appropriate group element), except at point i^*, since the term $g^{z_{i^*}/x_{i^*}}$ is missing from the common reference string. It can be shown that this scheme is a secure (puncturable) PRF from standard assumptions in bilinear groups. One drawback of this construction is that the size of the common reference string is quadratic in n. We show how to overcome this efficiency limitation by adding some more structure to the common reference string, at the cost of relying on a q-type assumption. We refer the curious reader to the technical sections for more details.

Security against rogue-puzzle attacks. We introduce a new concept called security against rogue-puzzle attacks. This notion aims to ensure that the batch-solving algorithm correctly recovers the secret of honestly generated puzzles, even when the batch contains puzzles generated *adversarially*. To achieve this, we augment the syntax of the TLPs with an additional validity-check algorithm IsValid, that tests whether the puzzle was well-formed. The adversary is then allowed to sample puzzles arbitrarily (even adaptively) but contingent on passing this validity check. To build TLPs secure in this model, we have to worry about two main attacks:

- Malformed homomorphic puzzles: An adversary may tamper with the batch-solving algorithm by introducing malformed puzzles, leading to incorrect results upon homomorphic evaluation.
- Collision of indices: An adversary may attempt to force a collision of indices with an honest party, thereby disrupting the batch-solving algorithm, as it only works when there are no collisions.

While the former class of attacks can be prevented by simply augmenting the puzzle with a non-interactive zero-knowledge proof (NIZK). However, addressing the second type of attack is more intricate. Our solution is to sample the index deterministically using a hash function applied to the index-independent part of the puzzle. This approach reduces the collision of indices to a collision in the hash function, a computationally challenging problem. However, this outline hides a crucial detail, namely that for the case of *bounded identities*, the output domain of the hash function is of polynomial size. We carefully analyze the situation in the random oracle model. Interestingly, our bipartitate matching algorithm turns out to be *crucial* to derive a meaningful bound, whereas more crude approximations would yield trivial bounds on the success probability of the adversary[2].

As an additional contribution, we present efficient NIZK protocols tailored to our proposed constructions. These protocols optimize efficiency, considering that general-purpose NIZKs may not be suitable for our specific applications. In the pairing setting, the main idea is to use a variant of Schnorr protocol/Chaum Pedersen protocol where the prover proves knowledge of an exponent k in two different instances. In the LWE setting, we utilize the (almost) key homomorphic property of our PRFs along with efficient range proofs on time lock puzzles from [TBM+20].

[2] A trivial bound that handles malicious parties is by asking the degree to be equal to the number of puzzles batched. If every party samples a puzzle for each index, we setup a complete bipartite graph and hence a perfect matching in the malicious setting. We refer the interested reader to Appendix A for an alternate analysis.

1.3 Related Work

Key-homomorphic PRFs. Beside the constrained-key key-homomorphic PRF of [BV15], that we mentioned earlier, there is another constrained-key key-homomorphic PRF of [BP14, BFP+15] that, with some slight modifications, can be turned into puncturable key-homomorphic PRF. This construction accumulates much more noise than our modification of [BV15], which translates into much worse parameters. There are also multilinear map based constructions [BFP+15, CRV16]. Candidates of multilinear maps, however, are far from practical.

Timed cryptography. In addition to constructions based on sequential squaring, several other approaches have been proposed for creating time-lock puzzles, which we explore in this section. Bitansky et al. [BGJ+16] proposed a scheme based on succinct randomized encodings [BGL+15] and the existence of non-parallelizable languages. Recently, Burdges and De Feo [BF21] proposed the notion of delay encryption, which offers a simliar "solve one, open many" functionality as batchable time-lock puzzles and can be seen as an identity-based version of the standard time-lock puzzles. However, there are a few essential differences from our approach. First, delay encryption necessitates all parties to encrypt the puzzle with respect to the same identity, assuming some coordination among participants. Furthermore, the only known construction of delay encryption is based on hard problems related to isogenies, which have garnered considerably less attention than the sequential squaring problem.

Related to the notion of security against rogue puzzle attacks is the notion of non-malleable of time-lock puzzle [FKPS21]. While conceptually related (both notions consider an adversary that generates possibly corrupted puzzles), their objectives are quite different. Non-malleability aims to safeguard the confidentiality of a legitimately sampled puzzle, even when a solving oracle is present. In contrast, security against rogue puzzle attacks is concerned with ensuring the correctness of the batch-solving algorithm when maliciously generated puzzles are introduced.

Beyond time-lock puzzles, the other paradigm of accounting for time is to have a trusted party that regularly produces outputs and tie your cryptographic processes to that output [RSW96, CHSS02]. For example, Liu et al. [LKW15] combine (extractable) witness encryption [GGSW13] and a public reference clock, such as a blockchain. Given the heavy cryptographic machinery involved, we view these works as mainly feasibility results. Following the same idea, Döttling et al. [DHMW22] construct witness encryption for specific languages used in proof-of-stake blockchains, making it practically efficient. This work, however, also has substantial limitations in that it only allows for encrypting to the near future.

1.4 Open Questions

In this work, we leave an interesting question unanswered: is it possible to batch puzzles of varying levels of difficulty? Specifically, if two puzzles are generated such that one requires time T to open and the other requires time T', is there a way to combine them such that only a single puzzle needs to be solved, which will open the first puzzle at time T and the second puzzle at time T'? Addressing this question would necessitate a departure from the existing body of work, including homomorphic time-lock puzzles, as these conventional methods do not readily apply to this scenario.

2 Preliminaries

Throughout this work, we write λ to denote the security parameter. We say a function f is negligible in the security parameter λ if $f = o(\lambda^{-c})$ for all $c \in \mathbb{N}$. We denote this by writing $f(\lambda) = \mathsf{negl}(\lambda)$. We write

$\mathsf{poly}(\lambda)$ to denote a function that is bounded by a fixed polynomial in λ. We say an algorithm is efficient if it runs in probabilistic polynomial time (PPT) in the length of its input. A runtime of a PPT algorithm $\mathcal{A}$ on input x is denoted by $\mathit{Time}(\mathcal{A}(x))$. Throughout this work, we consider security against *non-uniform* adversaries (indexed by λ) that are represented by the circuit model of computation where the circuit size is polynomial in the length of their input.

For a positive integer $n \in \mathbb{N}$, we write $[n]$ to denote the set $\{1, \ldots, n\}$ and $[0, n]$ to denote the set $\{0, \ldots, n\}$. For a distribution D, we write $x \leftarrow D$ to denote that x is sampled from D. We now review the main cryptographic primitives we use in this work.

2.1 Puncturable Pseudorandom Functions.

A puncturable pseudorandom function (PRF) [BW13, KPTZ13, BGI14, SW14] is a PRF [GGM84] that has an additional puncturing algorithm, which produces a punctured version of the key. The punctured key can evaluate the PRF at all points except for the punctured one. For security, it is required that the PRF value at that specific point is pseudorandom, even given the punctured key.

Definition 2.1 (Puncturable PRFs)**.** A puncturable pseudorandom function family on key space $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda\in\mathbb{N}}$, domain $\mathcal{X} = \left\{\mathcal{X}_{\lambda,n}\right\}_{\lambda,n\in\mathbb{N}}$ and range $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$, consists of a tuple of PPT algorithms $\Pi_{\mathsf{PRF}} = (\mathsf{Setup}, \mathsf{Puncture}, \mathsf{PRF}, \mathsf{PuncturedEval})$ defined as follows.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$ a probabilistic algorithm that takes as input the security parameter λ, domain index n and outputs public parameters pp.
- $\mathsf{k}^* \leftarrow \mathsf{Puncture}_{\mathsf{pp}}(\mathsf{k}, i^*)$ a deterministic algorithm that takes as input a key $\mathsf{k} \in \mathcal{K}_\lambda$ and a position $i^* \in \mathcal{X}_{\lambda,n}$ and returns a punctured key k^*.
- $y \leftarrow \mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}, i)$ a deterministic algorithm that takes as input a key $\mathsf{k} \in \mathcal{K}_\lambda$ and an index $i \in \mathcal{X}_{\lambda,n}$ and returns a string y.
- $y \leftarrow \mathsf{PuncturedEval}_{\mathsf{pp}}(\mathsf{k}^*, i^*, i)$ a deterministic algorithm that takes as input a punctured key k^*, a punctured index $i^* \in \mathcal{X}_{\lambda,n}$, an index $i \in \mathcal{X}_{\lambda,n}$ and returns a string y.

In addition, Π_{PRF} must satisfy the following properties.

- **Functionality Preserving:** We say that Π_{PRF} satisfies functionality preserving if there exists a negl function such that for all $\lambda, n \in \mathbb{N}$, all keys $\mathsf{k} \in \mathcal{K}_\lambda$, all points $i^* \neq i \in \mathcal{X}_{\lambda,n}$, it holds that,

  $$\Pr\left[\mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}, i) \neq \mathsf{PuncturedEval}_{\mathsf{pp}}(\mathsf{Puncture}_{\mathsf{pp}}(\mathsf{k}, i^*), i^*, i) : \ \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)\ \right]$$

  is negligible in λ, where the probability is over the random coins of Setup. If functionality preserving holds with probability 1, we say Π_{PRF} is perfectly functionality preserving.

- **Security:** We say that Π_{PRF} is secure if for any polynomially bounded adversaries $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$, any polynomially bounded function $n(\lambda)$, for all $i^* \in \mathcal{X}_{\lambda,n}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$, it holds that,

  $$\left|\Pr\left[b \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{k}^*, y) : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n); \mathsf{k} \leftarrow \mathcal{K}_\lambda; b \leftarrow \{0,1\} \\ \mathsf{k}^* \leftarrow \mathsf{Puncture}_{\mathsf{pp}}(\mathsf{k}, i^*) \\ \text{if } b = 0 \text{ then } y \leftarrow \mathcal{Y}_\lambda, \text{else } y \leftarrow \mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}, i^*) \end{array}\right] - \frac{1}{2}\right|$$

  is negligible in λ

(Almost) Key-Homomorphism [NPR99, BLMR13, BP14, BV15, BFP+15]. We also require that the puncturable PRF satisfies a notion of key-homomorphism.

Definition 2.2 (Key-Homomorphism). Let $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family such that for every $\lambda \in \mathbb{N}$, $(\mathcal{K}_\lambda, +)$ is a finite group. We say Π_{PRF} defined on key space $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda\in\mathbb{N}}$, domain $\mathcal{X} = \left\{\mathcal{X}_{\lambda,n}\right\}_{\lambda,n\in\mathbb{N}}$ and range $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$, satisfies the key homomorphic property if for all $\lambda, n \in \mathbb{N}$ every $\mathsf{k}_0, \mathsf{k}_1 \in \mathcal{K}_\lambda$, all indices $i \in \mathcal{X}_{\lambda,n}$, it holds that,

$$\Pr\left[\mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}_0, i) + \mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}_0, i) = \mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}_0 + \mathsf{k}_1, i) : \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)\right] = 1.$$

We can also relax this notion to *almost key-homomorphism* by requiring that the above equality almost holds, for all $\lambda, n \in \mathbb{N}$ every $\mathsf{k}_0, \mathsf{k}_1 \in \mathcal{K}_\lambda$, all indices $i \in \mathcal{X}_{\lambda,n}$, it holds that,

$$\Pr\left[\|\mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}_0, i) + \mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}_0, i) - \mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}_0 + \mathsf{k}_1, i)\|_\infty \leq 1 : \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)\right] = 1.$$

2.2 Time-Lock Puzzles

We follow the syntax from Srinivasan et al, [SLM+23] where we consider the standard notation for time-lock puzzles except there is an additional setup phase that depends on the hardness parameter but not on the secret.

Definition 2.3 (Time-Lock Puzzles [RSW96]). A time-lock puzzle (TLP) with solution space $\{\mathbb{S}_\lambda\}_{\lambda\in\mathbb{N}}$ is a tuple of four algorithms $\Pi_{\mathsf{TLP}} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sol})$ defined as follows:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T)$ a probabilistic algorithm that takes as input a security parameter 1^λ and a time hardness parameter T, and outputs public parameters pp.
- $Z \leftarrow \mathsf{Gen}(\mathsf{pp}, s)$ a probabilistic algorithm that takes as input public parameters pp, and a solution $s \in \mathbb{S}_\lambda$, and outputs a puzzle Z.
- $s \leftarrow \mathsf{Sol}(\mathsf{pp}, Z)$ a deterministic algorithm that takes as input public parameters pp and a puzzle Z and outputs a solution s.

In addition, Π_{TLP} should satisfy the following properties:

- **Correctness:** We say Π_{TLP} is correct if for all $\lambda, T \in \mathbb{N}$, all secrets $s \in \mathbb{S}_\lambda$, it holds that,

$$\Pr\left[\mathsf{Sol}(\mathsf{pp}, \mathsf{Gen}(\mathsf{pp}, s)) = s : \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T)\right] = 1.$$

- **Security:** We say Π_{TLP} is secure with gap $\varepsilon \in (0, 1)$, if there exists a polynomial $\tilde{T}(\cdot)$ such that for for all polynomially bounded functions where $T(\cdot) \geq \tilde{T}(\cdot)$, any polynomially bounded adversaries, $(\mathcal{A}_1, \mathcal{A}_2) = (\left\{\mathcal{A}_{1,\lambda}\right\}_{\lambda\in\mathbb{N}}, \left\{\mathcal{A}_{2,\lambda}\right\}_{\lambda\in\mathbb{N}})$, where the depth of $\mathcal{A}_{2,\lambda}$ is atmost $T^\varepsilon(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$, it holds that,

$$\left|\Pr\left[\begin{array}{c} b \leftarrow \mathcal{A}_2(\mathsf{pp}, Z, \mathsf{st}) \\ \wedge\ (s_0, s_1) \in \mathbb{S}_\lambda^2 \end{array} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T(\lambda)) \\ (\mathsf{st}, s_0, s_1) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{pp}) \\ b \leftarrow \{0,1\}, Z \leftarrow \mathsf{Gen}(\mathsf{pp}, s_b) \end{array}\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

- **Efficiency:** We say Π_{TLP} satisfies efficiency if

(a) There exists a polynomial $p_1(\cdot,\cdot,\cdot)$ such that for all $\lambda, T \in \mathbb{N}$, inputs $s \in \mathbb{S}_\lambda$, it holds that,

$$\Pr\left[\mathit{Time}\left(\mathsf{Gen}(\mathsf{pp}, s)\right) \le p_1(\lambda, \log|\mathbb{S}_\lambda|, \log T) : \ \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T)\ \right] = 1.$$

(b) There exists a polynomial $p_2(\cdot,\cdot,\cdot)$ such that for all $\lambda, T \in \mathbb{N}$, inputs $s \in \mathbb{S}_\lambda$, it holds that,

$$\Pr\left[\mathit{Time}\left(\mathsf{Sol}(\mathsf{pp}, Z)\right) \le p_2(\lambda, \log|\mathbb{S}_\lambda|, T) : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ Z \leftarrow \mathsf{Gen}(\mathsf{pp}, s) \end{array}\right] = 1.$$

Homomorphic Time-Lock Puzzles. We also recall the definition of homomorphic time-lock puzzles [MT19], which allows one to compute functions on secrets homomorphically, without solving the puzzles first.

Definition 2.4 (Homomorphic TLPs [MT19]). We say $\Pi_{\mathsf{hTLP}} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sol}, \mathsf{Eval})$ is homomorphic for the circuit family, $C = \{C_{\lambda,n}\}_{\lambda,n\in\mathbb{N}}$ and solution space $\{\mathbb{S}_\lambda\}_{\lambda\in\mathbb{N}}$, if the syntax is augmented with the following algorithm:

- $Z' \leftarrow \mathsf{Eval}(C, \mathsf{pp}, Z_1, \ldots, Z_n)$ a probabilistic algorithm that takes as input a circuit $C \in C_{\lambda,n}$ where $C : \mathbb{S}_\lambda^n \to \mathbb{S}_\lambda$, public parameters pp and a set of n puzzles $(Z_1, \ldots, Z_n)$ and outputs a puzzle Z'.

In addition, Π_{hTLP} should satisfy the following evaluation property:

- **Evaluation Correctness:** We say Π_{hTLP} satisfies evaluation correctness if for all $\lambda, n, T \in \mathbb{N}$, for all circuits $C \in C_{\lambda,n}$, inputs $(s_1, \ldots, s_n) \in \mathbb{S}_\lambda^n$, it holds that,

$$\Pr\left[\mathsf{Sol}(\mathsf{pp}, \mathsf{Eval}(C, \mathsf{pp}, Z_1, \ldots, Z_n)) = C(s_1, \ldots, s_n) : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ \forall i \in [n], Z_i \leftarrow \mathsf{Gen}(\mathsf{pp}, s_i) \end{array}\right] = 1.$$

- **Evaluation Efficiency:** We say Π_{hTLP} satisfies evaluation efficiency if there exists a polynomial $p_1(\cdot,\cdot,\cdot)$ such that for all $\lambda, n, T \in \mathbb{N}$, circuits $C \in C_{\lambda,n}$, inputs $(s_1, \ldots, s_n) \in \mathbb{S}_\lambda^n$, it holds that,

$$\Pr\left[\mathit{Time}\left(\mathsf{Eval}(C, \mathsf{pp}, Z_1, \ldots, Z_n)\right) \le p_1(\lambda, |C|, \log T) : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ \forall i \in [n], Z_i \leftarrow \mathsf{Gen}(\mathsf{pp}, s_i) \end{array}\right] = 1.$$

We require homomorphic TLPs specifically that support homomorphic evaluations of linear functions over the puzzles, that are secure against depth bounded adversaries. We have such constructions from RSA groups [MT19] and class groups with imaginary quadratic order [TCLM21]. We also mention that both works show how to extend the message space, and therefore the linear space, to $\mathbb{Z}_{N^c}$ (and $\mathbb{Z}_{p^c}$, respectively) for any c without changing the atomic operation in the sequential computation; which is still repeated squaring over the base modulus.

Theorem 2.5 ([MT19]). *Assuming that the strong sequential squaring assumption in RSA groups, the DDH assumption, and the DCR assumption hold, there exists a time lock puzzle scheme that supports linear homomorphic evaluations over $\mathbb{Z}_N$, where N is an RSA modulus.*

Theorem 2.6 ([TCLM21]). *Assuming that the strong sequential squaring assumption in class groups and the HSM assumption hold, there exists a time lock puzzle scheme that supports linear homomorphic evaluations over $\mathbb{Z}_p$, where p is a prime.*

Time-Lock Puzzles with Batch Solving. We present a modified notion of TLPs with batched solving from [SLM+23] where Setup is allowed to take the maximum batch size as input.

Definition 2.7 (TLPs with batch solving)**.** We say $\Pi_{\mathsf{batchTLP}} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{BatchPSol})$ supports *batch solving* with solution space $\{\mathbb{S}_\lambda\}_{\lambda \in \mathbb{N}}$, if the syntax is augmented with the following algorithm:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T, n)$ a probabilistic algorithm that takes as input a security parameter 1^λ, a time hardness parameter T, bound on the maximum batch size n, and outputs public parameters pp.
- $Z \leftarrow \mathsf{Gen}(\mathsf{pp}, s)$ a probabilistic algorithm that takes as input public parameters pp, and a solution $s \in \mathbb{S}_\lambda$, and outputs a puzzle Z. Z and outputs a solution s.
- $\{(s_i, Z_i)\}_{i \in \mathcal{S}} \leftarrow \mathsf{BatchPSol}(\mathsf{pp}, \{Z_i\}_{i \in \mathcal{S}})$ a deterministic algorithm that takes as input the combined public parameters pp, a set $\mathcal{S} \subseteq [n]$ of puzzles Z_i and outputs for each puzzle a solution $s_i \in \mathbb{S}_\lambda$.[3]

We require Π_{batchTLP} to hold the same correctness, security and efficiency properties from Definition 2.3 with the modified syntax. In addition, Π_{batchTLP} should satisfy the following property:

- **Batch solving correctness:** We say Π_{batchTLP} satisfies batch solving correctness if for all $T, n \in \mathbb{N}$, any subset $\mathcal{S} \subseteq [n]$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that, for all $\lambda \in \mathbb{N}$, solutions $s_i \in \mathbb{S}_\lambda$, it holds that,

  $$\Pr\left[\mathsf{BatchPSol}(\mathsf{pp}, \{Z_i\}_{i \in \mathcal{S}}) \neq \{(s_i, Z_i)\}_{i \in \mathcal{S}} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T, n) \\ \forall i \in \mathcal{S}, Z_i \leftarrow \mathsf{Gen}(\mathsf{pp}, s_i) \end{array}\right]$$

  is negligible in λ.

- **Batch solving efficiency:** We say Π_{batchTLP} satisfies batch solving efficiency if there exists polynomials $p_1(\cdot,\cdot,\cdot)$,$p_2(\cdot,\cdot,\cdot,\cdot)$, such that for all $\lambda, T, n \in \mathbb{N}$, any subset $\mathcal{S} \subseteq [n]$, for all solutions $(s_1, \ldots, s_n) \in \mathbb{S}_\lambda^n$, it holds that,

  $$\Pr\left[\begin{array}{c} \mathit{Time}\left(\mathsf{BatchPSol}(\mathsf{pp}, \{Z_i\}_{i \in \mathcal{S}})\right) \\ \leq p_1(\lambda, \log|\mathbb{S}_\lambda|, T) + p_2(\lambda, \log|\mathbb{S}_\lambda|, \log T, n) \end{array} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T, n) \\ \forall i \in \mathcal{S}, Z_i \leftarrow \mathsf{Gen}(\mathsf{pp}, s_i) \end{array}\right] = 1.$$

Definition 2.8 (Batching TLPs with unbounded number of parties)**.** We say that our batched time lock puzzle scheme $\Pi_{\mathsf{cobatchTLP}}$ supports an arbitrary polynomial number of parties if the algorithms Gen, Sol in Definition 2.7 run in time $\mathsf{poly}(\lambda, \log|\mathbb{S}_\lambda|, \log T, \log n)$. Similarly, our security property allows the adversary to submit a larger bound on the number of parties $n(\cdot)$ i.e. now the function could be bounded by $2^{\mathsf{poly}(\lambda)}$ instead of a polynomial in λ.

Remark 2.9. The syntax for Π_{batchTLP} can support public parameters that depend on n, thus the efficiency of Gen, Sol can depend on n. Our schemes will be more efficient where we only need to access a small subset of the public parameters. Thus, in the RAM model of computation, the efficiency of our algorithms Gen, Sol will not depend on n. Additionally, the efficiency of BatchPSol can depend on the size of the elements being batched i.e. $|\mathcal{S}|$, and thus, run in time $p_1(\lambda, \log|\mathbb{S}_\lambda|, T) + p_2(\lambda, \log|\mathbb{S}_\lambda|, \log T, |\mathcal{S}|)$.

In our work, we define a notion of TLPs with coordination, which is a straightforward modification to the original algorithms assuming that each user in the system posseses an index in $[n]$.

[3] Note that Sol is equivalent to running BatchPSol on one index.

Definition 2.10 (TLPs with coordination)**.** We say $\Pi_{\text{cobatchTLP}} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{BatchPSol})$ supports *batch solving* with solution space $\{\mathbb{S}_\lambda\}_{\lambda\in\mathbb{N}}$, defined as follows:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T, n)$ a probabilistic algorithm that takes as input a security parameter 1^λ, a time hardness parameter T, total number of parties n, and outputs public parameter pp.

- $Z \leftarrow \mathsf{Gen}(\mathsf{pp}, i, s)$ a probabilistic algorithm that takes as input public parameters pp, party index $i \in [n]$ and a solution $s \in \mathbb{S}_\lambda$, and outputs a puzzle Z.

- $\{(i, s_i)\}_{i\in\mathcal{S}} \leftarrow \mathsf{BatchPSol}(\mathsf{pp}, \mathcal{S}, \{(i, Z_i)\}_{i\in\mathcal{S}})$ a deterministic algorithm that takes as input the public parameters pp, a set $\mathcal{S} \subseteq [n]$, puzzles Z_i from each party $i \in \mathcal{S}$, and outputs for each party $i \in \mathcal{S}$, solutions $s_i \in \mathbb{S}_\lambda$.

Scheme $\Pi_{\text{cobatchTLP}}$ satisfies correctness, batch solving correctness, efficiency and batch solving efficiency similar to Definition 2.7 (with the appropriate syntax changes). We present the modified security definition.

- **Security:** We say $\Pi_{\text{cobatchTLP}}$ is secure with gap $\varepsilon \in (0, 1)$, if there exists a polynomial $\tilde{T}(\cdot)$ such that for for all polynomially bounded functions where $T(\cdot) \geq \tilde{T}(\cdot)$, any polynomially bounded (in λ) function $n(\cdot)$, any polynomially bounded adversaries, $(\mathcal{A}_1, \mathcal{A}_2) = (\{\mathcal{A}_{1,\lambda}\}_{\lambda\in\mathbb{N}}, \{\mathcal{A}_{2,\lambda}\}_{\lambda\in\mathbb{N}})$, where the depth of $\mathcal{A}_{2,\lambda}$ is atmost $T^\varepsilon(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$, it holds that,

$$\left| \Pr\left[\begin{array}{c} b \leftarrow \mathcal{A}_2(\mathsf{pp}, Z, \mathsf{st}) \\ \wedge (s_0, s_1) \in \mathbb{S}_\lambda^2 \wedge i \in [n] \end{array} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T(\lambda), n(\lambda)) \\ (\mathsf{st}, i, s_0, s_1) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{pp}) \\ b \leftarrow \{0, 1\}, Z \leftarrow \mathsf{Gen}(\mathsf{pp}, i, s_b) \end{array} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda).$$

If we want to support unbounded parties our definition is modified similarly to Definition 2.8. Additionally, Remark 2.9 holds in this setting as well.

2.3 Cryptographic Groups

For a cryptographic group $\mathbb{G}$ of order q we use multiplicative notation, meaning the group operation is $\cdot$. Then we use exponentiation to indicate repeated multiplication i.e. we define $g^x = \prod_{i\in[x]} g$ for $g \in \mathbb{G}$ and $x \in \mathbb{Z}_q$. To simplify notation when we do vector exponentiation with $\mathbf{x} \in \mathbb{Z}_q^n$ we write $\mathbf{h} = g^{\mathbf{x}}$ instead of $(h_i = g^{x_i})_{i\in[n]}$, similarly use the Hadamard product $\mathbf{g} \odot \mathbf{h}$ to indicate the component-wise multiplication between two vectors of group elements $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n$.

Let $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, g, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ be a generator of a (symmetric) bilinear group generated by g of prime order p, with an efficiently computable pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. We recall a few well-known assumptions in bilinear groups.

Assumption 2.11 (Bilinear Diffie-Hellman)**.** Let GroupGen be a bilinear group generator. The bilinear Diffie-Hellman problem is hard for GroupGen if the following distributions are computationally indistinguishable:

$$\left(p, \mathbb{G}, \mathbb{G}_T, g, e, g^x, g^{1/x}, g^y, g^z, e(g, g)^{xyz}\right) \approx \left(p, \mathbb{G}, \mathbb{G}_T, g, e, g^x, g^{1/x}, g^y, g^z, e(g, g)^r\right)$$

where $(p, \mathbb{G}, \mathbb{G}_T, g, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and $(x, y, z, r) \leftarrow \mathbb{Z}_p^*$.

Assumption 2.12 (n-Power Diffie-Hellman [BGW05]). Let GroupGen be a bilinear group generator. The n-power Diffie-Hellman problem is hard for GroupGen if the following distributions are computationally indistinguishable:

$$\left(p, \mathbb{G}, \mathbb{G}_T, g, e, \left\{g^{x^i}\right\}_{i\in[2n]\setminus\{n+1\}}, g^y, e(g,g)^{x^{n+1}y}\right)$$
$$\approx \left(p, \mathbb{G}, \mathbb{G}_T, g, e, \left\{g^{x^i}\right\}_{i\in[2n]\setminus\{n+1\}}, g^y, e(g,g)^{r}\right)$$

where $(p, \mathbb{G}, \mathbb{G}_T, g, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and $(x, y, r) \leftarrow \mathbb{Z}_p^*$.

[BBG05] show that this assumption holds in the bilinear generic group model. In favor of a simpler exposition, we only define and use symmetric pairings, however both the constructions and the assumptions can me easily adapted to the asymmetric settings.

2.4 Lattice Preliminaries

Assumption 2.13 (LWE). Sample $\mathbf{A} \leftarrow_\$ \mathbb{Z}_q^{n\times m}$, $\mathbf{s} \leftarrow_\$ \mathbb{Z}_q^n$, and $\mathbf{r} \leftarrow_\$ \mathbb{Z}_q^m$ uniformly random and $\mathbf{e} \leftarrow \chi_{\sigma,B}^m$ be component-wise sampled from the discrete gaussian distribution with standard deviation σ and truncated at $B = \sigma\omega(\sqrt{\log(\lambda)})$. The LWE assumption is hard if $(\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e}^T)$ is computationally indistinguishable from $(\mathbf{A}, \mathbf{r})$.

Gadget Matrix We call $\mathbf{g} = (2^0, 2^1, \ldots, 2^{\lceil\log(q)\rceil})$ the gadget vector and $\mathbf{G} = \mathbf{g}^T \otimes \mathbf{I}_n \in \mathbb{Z}_q^{n\times\lceil\log(q)\rceil n}$ the gadget matrix. And $G^{-1} : \mathbb{Z}_q^{n\times m} \to \mathbb{Z}_q^{\lceil\log(q)\rceil n\times m}$ is the binary decomposition function, which is not a linear operation but for any matrix $\mathbf{A} = \mathbf{G}G^{-1}(\mathbf{A})$.

Rounding and Norm When we use $\|\mathbf{a}\|_\infty$ on some vector $\mathbf{a} \in \mathbb{Z}_q^n$ we mean lift $\mathbf{a}$ to $\mathbb{Z}^n$ and then $max_{i\in[n]}(|a_i|)$. Similarly, when we use $\lfloor\mathbf{a}\rceil_p$ for some vector $\mathbf{a} \in \mathbb{Z}_q^n$ we lift $\mathbf{a}$ to Q^n then component-wise round $\mathbf{a}p/q$ to the closest element in $\mathbb{Z}_p$.

3 Time-Lock Puzzles with Batch Solving

In what follows we describe a generic construction of time-lock puzzle with batch solving. To make our presentation modular, we will initially assume that each party in the protocol is indexed by a unique identifier $i \in [n]$, where n denotes a bound on the total number of parties. Note that setting $n = \lambda^{\omega(1)}$ allows our time-lock puzzle to support an unbounded number of parties (provided that we instantiate it with suitable building blocks). Consequently, we will modify the syntax of time-lock puzzles to add i to the puzzle generation algorithm $\mathsf{Gen}(\mathsf{pp}, i, m)$ and we will assume that such an index is known to the puzzle solver. This assumption will be removed in Section 4.

We proceed by presenting our construction. We assume the existence of the following building blocks:

- A time-lock puzzle Π_{TLP} that is linearly homomorphic over $\mathbb{Z}_N$.
- A puncturable almost key-homomorphic PRF Π_{PRF} with domain $[n]$ and additive key homomorphism over $\mathbb{Z}_p^\ell$, where $\ell = \mathsf{poly}(\lambda)$.

We will set the parameters of the above schemes in such a way that,

$$n \cdot p^{2\ell} < N \tag{2}$$

$$p > n \tag{3}$$

For notational convenience, we define the integer encoding algorithm Encode : $\mathbb{Z}_p^\ell \to \mathbb{Z}$ as

$$\mathsf{Encode}_{p,\ell}(x_1, \ldots, x_\ell) = \sum_{i=1}^{\ell} p^{2(i-1)} x_i$$

and the decoding algorithm $\mathsf{Decode}_{p,\ell}$ as the reverse operation, i.e., vectorizing an integer by modular reduction and rounding.

Construction 3.1 (Batchable Time-Lock Puzzle)**.** We describe our algorithms below. For convenience we only consider messages $m \in \{0, 1\}$, but the construction can be easily extended to larger domains.

- $\mathsf{Setup}(1^\lambda, T, n)$:
    - $\mathsf{pp}_{\mathsf{LHP}} \leftarrow \mathsf{LHP.Setup}(1^\lambda, T)$
    - $\mathsf{pp}_{\mathsf{PRF}} \leftarrow \mathsf{PRF.Setup}(1^\lambda, n)$
    - Return $\mathsf{pp} = (\mathsf{pp}_{\mathsf{LHP}}, \mathsf{pp}_{\mathsf{PRF}})$
- $\mathsf{Gen}(\mathsf{pp}, i, m)$:
    - Sample a PRF key $\mathsf{k} \leftarrow \mathbb{Z}_p^\ell$
    - Time-lock the key by computing $Z \leftarrow \mathsf{LHP.Gen}(\mathsf{pp}_{\mathsf{LHP}}, \mathsf{Encode}_{p,\ell}(\mathsf{k}))$
    - Compute the punctured key $\mathsf{k}^* \leftarrow \mathsf{Puncture}_{\mathsf{pp}_{\mathsf{PRF}}}(\mathsf{k}, i)$
    - Mask the message $c \leftarrow \mathsf{PRF}_{\mathsf{pp}_{\mathsf{PRF}}}(\mathsf{k}, i) + m \cdot \lceil p/2 \rceil$
    - Return (i, Z, k^*, c)
- $\mathsf{BatchSol}\left(\mathsf{pp}, S, \left\{i, Z_i, \mathsf{k}_i^*, c_i\right\}_{i \in S}\right)$:
    - Sum the puzzles $\tilde{Z} \leftarrow \mathsf{LHP.Eval}\left(\sum, \mathsf{pp}_{\mathsf{LHP}}, \{Z_i\}_{i \in S}\right)$
    - Solve the resulting puzzle $\tilde{\mathsf{k}} \leftarrow \mathsf{LHP.Sol}(\mathsf{pp}_{\mathsf{LHP}}, \tilde{Z})$
    - Compute $k' \leftarrow \mathsf{Decode}_{p,\ell}(\tilde{k})$ and reduce each coordinate modulo p
    - For all $i \in S$, compute

$$\mu_i = c_i + \sum_{j \in S \setminus \{i\}} \mathsf{PuncturedEval}_{\mathsf{pp}_{\mathsf{PRF}}}(\mathsf{k}_j^*, j, i) - \mathsf{PRF}_{\mathsf{pp}_{\mathsf{PRF}}}(\mathsf{k}', i) \pmod{N}$$

    and set m_i as $\lfloor \mu_i \rceil_{\lceil p/2 \rceil}$.

Analysis. Before we proceed with the formal analysis, it is worth highlighting that each puzzle consists of a tuple $(i, Z_i, \mathsf{k}_i^*, c_i)$ where the size of each element is at most logarithmic in n. Furthermore, the sequential computation in the batch solving algorithm consists of solving a single puzzle, whereas all of the other operations do not depend on the time parameter T. Thus the scheme satisfies the desired efficiency requirements. Also notice that if Setup of both PRF and LHP are transparent then so is the setup of the batchable TLP.

Theorem 3.2 (Correctness)**.** *If* Π_{TLP} *satisfies correctness according to Definition 2.3, and* Π_{PRF} *satisfies correctness, then, Construction 3.1 satisfies batch solving correctness according to Definition 2.7.*

Proof. Observe that for all $k \in \mathbb{Z}_p^\ell$, as

$$\mathsf{Encode}_{p,\ell}(k) = \sum_{i=1}^{\ell} p^{2(i-1)} k_i \leq \sum_{i=1}^{\ell} p^{2i-1} \leq p^{2\ell} < N$$

where the last inequality holds by how we set our parameters Eq. (2), and hence $\mathsf{Decode}_{p,\ell}(\mathsf{Encode}_{p,\ell}(k)) = k$ as we're simply representing each element as an integer on a bigger base. Correctness of Construction 3.1 is straightforward from the correctness of Π_{TLP}. □

Theorem 3.3 (Batch Solving Correctness)**.** *If* Π_{TLP} *satisfies correctness according to Definition 2.3, and* Π_{PRF} *satisfies correctness, and* almost key-homomorphism*, then, Construction 3.1 satisfies batch solving correctness according to Definition 2.7.*

Proof. To show correctness, we first observe that, by the evaluation correctness of the time-lock puzzles, we have

$$\begin{aligned}\tilde{\mathsf{k}} &= \sum_{j \in S} \mathsf{Encode}_{p,\ell}(\mathsf{k}_j) = \sum_{j \in S} \sum_{i=1}^{\ell} p^{2(i-1)} \mathsf{k}_{j,i} \\ &= \sum_{i=1}^{\ell} p^{2(i-1)} \sum_{j \in S} \mathsf{k}_{j,i} \leq \sum_{i=1}^{\ell} p^{2i-1} \cdot n \leq n \cdot p^{2\ell} < N,\end{aligned}$$

where the last inequality holds by how we set our parameters Eq. (2). In particular, this implies that the summation happens without modular reduction. Additionally, observe that

$$\mathsf{k}' = \mathsf{Decode}_{p,\ell}(\tilde{k}) = \mathsf{Decode}_{p,\ell}\left(\sum_{i=1}^{\ell} p^{2(i-1)} \sum_{j \in S} \mathsf{k}_{j,i}\right) = \sum_{j \in S} \mathsf{k}_j$$

where the above sum is also over the integers, since each coordinate of the keys is at most p and $pn < p^2$, by how we set our parameters Eq. (3).

Plugging this into the solving equation, we have that

$$\begin{aligned}
\mu_i &= c_i + \sum_{j \in S \setminus \{i\}} \mathsf{PuncturedEval}_{\mathsf{pp}_{\mathsf{PRF}}}(\mathsf{k}_j^*, j, i) - \mathsf{PRF}_{\mathsf{pp}_{\mathsf{PRF}}}(\mathsf{k}', i) \\
&= c_i + \sum_{j \in S \setminus \{i\}} \mathsf{PuncturedEval}_{\mathsf{pp}_{\mathsf{PRF}}}(\mathsf{k}_j^*, j, i) - \mathsf{PRF}_{\mathsf{pp}_{\mathsf{PRF}}}\left(\sum_{j \in S} \mathsf{k}_j, i\right) \\
&= c_i + \sum_{j \in S \setminus \{i\}} \mathsf{PuncturedEval}_{\mathsf{pp}_{\mathsf{PRF}}}(\mathsf{k}_j^*, j, i) - \sum_{j \in S} \mathsf{PRF}_{\mathsf{pp}_{\mathsf{PRF}}}\left(\mathsf{k}_j, i\right) + e \\
&= c_i + \sum_{j \in S \setminus \{i\}} \mathsf{PuncturedEval}_{\mathsf{pp}_{\mathsf{PRF}}}(\mathsf{k}_j, i) - \sum_{j \in S} \mathsf{PRF}_{\mathsf{pp}_{\mathsf{PRF}}}\left(\mathsf{k}_j, i\right) + e \\
&= c_i - \mathsf{PRF}_{\mathsf{pp}_{\mathsf{PRF}}}\left(\mathsf{k}_i, i\right) + e \\
&= \mathsf{PRF}_{\mathsf{pp}_{\mathsf{PRF}}}(\mathsf{k}, i) + m_i \cdot \lceil p/2 \rceil - \mathsf{PRF}_{\mathsf{pp}_{\mathsf{PRF}}}\left(\mathsf{k}_i, i\right) + e \\
&= m_i \cdot \lceil p/2 \rceil + e
\end{aligned}$$

where the third equality follows by the almost key-homomorphism of the PRF, and the fourth equality follows by functionality preservation of the PRF.[4] Once again, by the almost homomorphism we can bound $\|e\|_\infty \leq (n-1)$ and thus μ_i is correctly rounded to m_i, since $n \leq p$. □

Remark 3.4. Notice, we crucially rely on the fact that $n \leq p$. We later show a construction of a key-homomorphic puncturable PRF that has a codomain with a (arbitrary but fixed) polynomial modulus p. In that case the number of puzzles we can batch is upperbounded by $\lfloor p/2 \rfloor$.

Theorem 3.5. *Let Π_{LHP} be linearly-homomorphic time-lock puzzle secure against depth $\mathbf{T}^\varepsilon(\lambda)$-bounded adversaries and Π_{PRF} be a almost key-homomorphic puncturable PRF, then construction 3.1 is a batchable time-lock puzzle secure against $\mathbf{T}^\varepsilon(\lambda)$-bounded adversaries.*

Proof. We proceed by defining a series of hybrids.

H_0 : In the first hybrid, we compute the time-lock puzzle according to the original distribution, i.e., $(i, Z_i, \mathsf{k}_i^*, c_i) \leftarrow \mathsf{Gen}(\mathsf{pp}, i, m)$.

H_1 : In this hybrid, we modify the Gen algorithm encode some fixed 0-string in the time-lock puzzle, as opposed to the key of the pseudorandom function. That is, we define

$$Z \leftarrow \mathsf{LHP.Gen}(0)$$

Since the attacker is guaranteed to run in parallel time less than T, indistinguishability of the views follows immediately from the security of the time-lock puzzles. Therefore, $\left|\mathsf{Adv}_{H_1}(\mathcal{A}) - \mathsf{Adv}_{H_0}(\mathcal{A})\right| \leq \mathsf{negl}(\lambda)$.

H_2 : In the second hybrid, we we modify the Gen algorithm by sampling c uniformly from the range of the PRF. By the pseudorandomness of PRF we can establish that $\mathsf{PRF}(\mathsf{k}, i)$ is computationally indistinguishable from uniform, even given the punctured key k^*, and therefore so is $\mathsf{PRF}(\mathsf{k}, i) + m$. Thus, $\left|\mathsf{Adv}_{H_2}(\mathcal{A}) - \mathsf{Adv}_{H_1}(\mathcal{A})\right| \leq \mathsf{negl}(\lambda)$.

The proof is concluded by observing that in H_2 the adversary has probability $1/2$ of winning because the output of $\mathsf{Gen}(\mathsf{pp}, m, i)$ does not depend on b.

□

[4] If the PRF satisfies the weaker notion of statistical functionality preservation, then the fourth equality holds with all but negligible probability.

4 Removing Coordination among Parties

In this section, we show how to convert any batching scheme where parties possess a unique index to a batching scheme where parties do not have any coordination. Our main observation is for each party to sample a set of indices at random and ensure that the Hall's marriage condition holds with overwhelming probability. The perfect matching thus allows each party to hold a unique index on which we run our batch solving algorithm. We start with a few graph theory preliminaries. Let G be a bipartite graph with vertex sets U and V and edge set E. A complete matching $M \subseteq E$ from U to V is a set of $|U|$ independent edges in G. In a complete matching, each vertex in U is incident to a single edge in M. For a set $S \subseteq U$, we denote by $\Gamma(S) \subseteq V$, the neighbourhood set of S, i.e. $\Gamma(S) = \{v \in V : \exists (u, v) \in E \wedge u \in U\}$.

Theorem 4.1 (Hall's marriage theorem [Hal35])**.** *Given a bipartite graph G with vertex sets U and V and edge set E. The graph admits a perfect matching from U to V if and only if - for every subset $S \subseteq U$, $|\Gamma(S)| \geq |S|$.*

Additionally, there are many algorithms to compute the perfect matching. One such algorithm is [HK73], (denoted in this document by $\mathsf{FindMatch}$) that takes in $G = (U, V, E)$ and outputs a perfect matching in time $O(|E|\sqrt{|V|})$ where E denotes the number of edges. More formally, it outputs $\{(u, v_u)\}_{u \in U}$ where $v_u \in \Gamma(u)$ and $|\, \{v_u\}_{u \in U} \,| = |U|$. If a perfect matching does not exist, the algorithm outputs $\bot$.

Construction 4.2 (Transformation to remove coordination)**.** We describe our algorithms to construct Π_{batchTLP} below. Our transformation relies on the existence of a $\Pi_{\mathsf{cobatchTLP}}$ scheme.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T, n)$. Let n_{new} and d be set according to parameters in Lemma 4.4. Sample $\mathsf{pp} \leftarrow \mathsf{cobatchTLP.Setup}(1^\lambda, T, n_{\mathsf{new}})$. Output public parameters pp.
- $Z \leftarrow \mathsf{Gen}(\mathsf{pp}, m)$. For every $j \in [d]$, sample d choices randomly, i.e. $v_j \leftarrow [n_{\mathsf{new}}]$ (without replacement[5]). Let $V = \left\{v_j\right\}_{j \in [d]}$, and we generate a puzzle, i.e. $Z_{v_j} \leftarrow \mathsf{cobatchTLP.Gen}(\mathsf{pp}, v_j, m)$.

  Output $Z = \left(V, \left\{(v_j, Z_{v_j})\right\}_{v_j \in V}\right)$.
- $\{s_i, Z_i\}_{i \in \mathcal{S}} \leftarrow \mathsf{BatchPSol}(\mathsf{pp}, \{Z_i\}_{i \in \mathcal{S}})$.
  - For each $i \in \mathcal{S}$, parse each $Z_i = \left(V_i, \left\{(v_{i,j}, Z_{i,v_j})\right\}_{j \in V_i}\right)$.
  - Let $G = (\mathcal{S}, [n_{\mathsf{new}}], \mathcal{E})$ be a bipartite graph where

  $$\mathcal{E} = \left\{(i, v_j) : i \in \mathcal{S}, v_j \in [n_{\mathsf{new}}], \text{ and } v_j \in V_i\right\}.$$
  - Compute a perfect matching $\mathsf{map} \leftarrow \mathsf{FindMatch}(G)$ where $\mathsf{map} = \left\{(i, v_i^*)\right\}_{i \in \mathcal{S}}$. Set $\mathcal{S}_{\mathsf{new}} = \left\{v_i^*\right\}_{i \in \mathcal{S}}$. If a perfect matching doesn't exist, output $\bot$.
  - Let $\left\{(v_i^*, s_i)\right\}_{v_i^* \in \mathcal{S}_{\mathsf{new}}} \leftarrow \mathsf{cobatchTLP.BatchPSol}(\mathsf{pp}, \mathcal{S}_{\mathsf{new}}, \{(v_i^*, Z_{i,v_i^*})\}_{v_i^* \in \mathcal{S}_{\mathsf{new}}})$.
  - Output $\{(s_i, Z_i)\}_{i \in \mathcal{S}}$.

[5] This means that we always sample a distinct set.

Analysis. The correctness, efficiency of our scheme are straightforward from the correctness, efficiency of the underlying $\Pi_{\mathsf{cobatchTLP}}$.

Theorem 4.3. *If $\Pi_{\mathsf{cobatchTLP}}$ satisfies batch solving correctness according to Definition 2.10, then, Construction 4.2 satisfies batch solving correctness according to Definition 2.7 where $n_{\mathsf{new}} = 3n$ and $d = \frac{\omega(\log\lambda)}{\log(n_{\mathsf{new}})}$.*

Proof. In order to argue about the batch solving correctness, batch solving efficiency, we prove the following claim about FindMatch algorithm. Informally, we prove that, when running BatchPSol, our graph $G = (\mathcal{S}, [n_{\mathsf{new}}], \mathcal{E})$ computes a perfect matching with overwhelming probability.

Lemma 4.4. *Let $G = (U, V, E)$ be a random left regular bipartite graph where $|U| = n$, $|V| = n'$. Let the left regular degree be denoted by d. If $n' = 3n$, $d = O(1) + \frac{\omega(\log\lambda)}{\log(n')}$, then, the probability that there exists a perfect matching for G is $\geq 1 - \mathsf{negl}(\lambda)$ where the probability is taken over the random coins of sampling the bipartite graph.*

Proof. Let $S \subseteq U$ be some subset of size ℓ. Let T be the neighbourhood set of S, i.e. $T = \Gamma(S)$. Hall's condition is violated if $|T| \leq \ell - 1$. For fixed sets S, T, the probability that the hall's condition is violated is given by, $\left(\binom{\ell-1}{d}/\binom{n'}{d}\right)^{\ell}$, where the probability is taken over the random coins of sampling G - because the probability that the particular subset is chosen on a single vertex on the left is $\frac{\binom{\ell-1}{d}}{\binom{n'}{d}}$, and the condition holds for all vertices on the left.

Since the sets S can be sampled in $\binom{n}{\ell}$ ways, and the set T can be sampled in $\binom{n'}{\ell-1}$ ways, the probability of failure through a union bound is given by,

$$\sum_{\ell=d}^{n}\binom{n}{\ell}\binom{n'}{\ell-1}\left(\frac{\binom{\ell-1}{d}}{\binom{n'}{d}}\right)^{\ell}. \tag{4}$$

By using the inequalities, $\frac{\binom{x}{d}}{\binom{y}{d}} \leq \frac{x\cdot(x-1)\ldots(x-d+1)}{y\cdot(y-1)\ldots(y-d+1)} \leq \left(\frac{x}{y}\right)^{d}$, and using the inequality that $\binom{x}{y} \leq \left(\frac{e\cdot x}{y}\right)^{y}$, the failure probability can be simplified to, $\sum_{\ell=d}^{n}\left(\frac{e\cdot n}{\ell}\right)^{\ell}\left(\frac{e\cdot n'}{\ell-1}\right)^{\ell-1}\left(\frac{\ell-1}{n'}\right)^{\ell\cdot d}$. Observe that the dominating expression here is the $\frac{\ell-1}{n'}^{\ell\cdot d}$ expression. The expression can be succinctly written as $f(\ell) = \left(\frac{a}{\ell^2}\cdot\left(\frac{\ell}{n'}\right)^{d}\right)^{\ell}$, where a is some constant. Taking a derivative, $\frac{d}{d\ell}\left(f(\ell)\right) = f(\ell)\cdot((d-2)(1+\ln\ell) - d\ln n' + \ln a)$. On setting $n' \geq 3n$, and $d \geq 4$ and since $\ell \leq n$, the term $\left(\frac{\ell}{n'}\right)^{d\cdot\ell}$ will dominate and we can observe that $\frac{d}{d\cdot\ell}\left(f(\ell)\right) < 0$ and the function is decreasing. Thus we can upper bound our probability of failure by $(n-d+1)\cdot f(4)$. Plugging in the values for $n' = 3n$, and bounding loosely, we get the expression that the probability is upper bounded by $e^{a-b\cdot d}$, where $a = \ln(\frac{n}{3e}) + 4\ln(e^2n^2)$, $b = 4\ln(\frac{n'}{4})$ are some constants. Loosely setting $d \geq (a + \omega(log\lambda))/b$, gives us that the probability of failure is $\leq \mathsf{negl}(\lambda)$, hence completing the lemma proof. □

Since FindMatch outputs a perfect matching, the batch correctness and batch efficiency of our transformation holds from the batch correctness and batch efficiency of $\Pi_{\mathsf{cobatchTLP}}$. Note that from the analysis in [HK73], it takes $O(n\cdot d\sqrt{n})$ time to find the perfect matching. In Appendix A, we sketch an alternate analysis which can find a matching solution in time $O(n\cdot d)$ in the worst case, but requires a larger degree for the matching to exist with non-negligible probability. The alternate analysis is simpler, but leads to a larger degree bound, hence more communication, and slower puzzle generation. Additionally, the matching algorithm is blazingly efficient and will not be the bottleneck when compared to the cryptographic operations in the system. □

Remark 4.5. Notice that in the RAM model, the efficiency of our algorithms mirrors the efficiency of the underlying $\Pi_{\mathsf{cobatchTLP}}$.

- If cobatchTLP.Gen does not depend on n, our Gen then runs d (which is $\leq \lambda$) copies of cobatchTLP.Gen and hence will also not depend on n.

- Efficiency of Sol is exactly same to the efficiency of cobatchTLP.Sol.

- If the efficiency of cobatchTLP.BatchPSol does not depend on n i.e. equal to $p_1(\lambda, \log|\mathbb{S}_\lambda|, T) + p_2(\lambda, \log|\mathbb{S}_\lambda|, \log T, |\mathcal{S}|)$. Efficiency of BatchPSol will depend on finding a perfect matching where the number of edges are $|\mathcal{S}| \cdot d$ and thus will have the same efficiency.

Theorem 4.6 (Security)**.** *If $\Pi_{\mathsf{cobatchTLP}}$ satisfies security according to Definition 2.10, then, Construction 4.2 satisfies security according to Definition 2.7.*

Proof. The security of our construction follows from a standard hybrid argument where the reduction $\mathcal{B}$ given a puzzle $Z = \left(V, \left\{(v_j, Z_{v_j})\right\}_{v_j \in V}\right)$ guesses an index $v_j \in V$, breaks the underlying security of $\Pi_{\mathsf{cobatchTLP}}$ with a probability loss of $1/d$. □

Remark 4.7 (Special Case: Superpolynomial Indices)**.** The analysis becomes very simple as soon as n is superpolynomial. To remove coordination, we can sample one random index and produce the puzzle with respect to that index. The probability that two parties sample the same index is negligible. This also works if the amount of puzzles one can batch is bounded 3.4) but n is superpolynomial.

5 Puncturable Key-Homomorphic PRFs

5.1 Bounded Domain Puncturable Key-Homomorphic PRFs from Pairings

In the following we present two constructions of puncturable key-fomomorphic PRFs from pairings, with different tradeoffs in terms of assumptions and parameter size. Importantly, both of these constructions only support of domain of size $n = \mathsf{poly}(\lambda)$.

Construction 5.1 (Quadratic Setup)**.** We specify the algorithms $\Pi_{\mathsf{PRF}} = (\mathsf{Setup}, \mathsf{PRF}, \mathsf{Puncture}, \mathsf{PuncturedEval})$ below.

- $\mathsf{Setup}(1^\lambda, 1^n)$:
  - $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, g, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$
  - Sample x_i uniformly at random for $\mathbb{Z}_p^*$ for $i \in [n]$
  - Sample z_i uniformly at random for $\mathbb{Z}_p^*$ for $i \in [n]$
  - Return $\mathsf{pp} = (\mathcal{G}, \{g^{x_i}\}_{i \in [n]}, \{g^{z_i/x_j}\}_{i \neq j})$

- $\mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}, i)$:
  - Return $e(g^{z_i/x_j}, g^{x_j})^{\mathsf{k}} = (e(g,g)^{z_i})^{\mathsf{k}}$ for some $j \neq i$

- $\mathsf{Puncture}_{\mathsf{pp}}(\mathsf{k}, i^*)$:
  - Return $g^{x_{i^*}\mathsf{k}} = (g^{x_{i^*}})^{\mathsf{k}}$
- $\mathsf{PuncturedEval}_{\mathsf{pp}}(\mathsf{k}^*, i^*, i)$:
  - Return $\perp$ if $i = i^*$
  - Return $e(g, g)^{z_i \mathsf{k}} = e(g^{x_{i^*}\mathsf{k}}, g^{z_i/x_{i^*}})$

Analysis. To show that the scheme is indeed correct, it suffices to observe that for all $i \neq i^*$:

$$\mathsf{PuncturedEval}_{\mathsf{pp}}(\mathsf{k}^*, i^*, i) = e(g^{x_{i^*}\mathsf{k}}, g^{z_i/x_{i^*}}) = e(g, g)^{z_i \mathsf{k}} = \mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}, i).$$

It is similarly easy to show that the scheme is (perfect) key homomorphic over $\mathbb{Z}_p^*$ since for all $\in [n]$ we have,

$$\prod_j \mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}_j, i) = \prod_j e(g, g)^{z_i \mathsf{k}_j} = e(g, g)^{z_i \sum_j \mathsf{k}_j} = \mathsf{PRF}_{\mathsf{pp}}\left(\sum_j \mathsf{k}_j, i\right).$$

We now show that the scheme is secure against the Bilinear Diffie-Hellman assumption.

Theorem 5.2. *If the Bilinear Diffie-Hellman assumption holds Assumption 2.11, then Construction 5.1 satisfies security from Definition 2.1.*

Proof. The reduction is supplied by the challenger with the following group elements

$$(g^x, g^{1/x}, g^y, g^z, e(g, g)^{xyz}) \text{ or } (g^x, g^{1/x}, g^y, g^z, e(g, g)^r)$$

which, by a variable re-arrangement, we can rewrite as

$$(g^{1/x}, g^x, g^{kx}, g^z, e(g, g)^{kz}) \text{ or } (g^{1/x}, g^x, g^{kx}, g^z, e(g, g)^r).$$

The reduction sets (implicitly) $x_{i^*} = x$, $z_{i^*} = z$, and $\mathsf{k} = k$. For all $i \neq i^*$ the reduction samples $x_i \leftarrow \mathbb{Z}_p^*$ and $z_i \leftarrow \mathbb{Z}_p^*$. The reduction can then compute the public parameters

$$\mathsf{pp} = (\mathbb{G}, p, \{g^{x_i}\}_{i \in [n]}, \{g^{z_i/x_j}\}_{i \neq j})$$

by using the elements given by the challenger, along with the integers sampled locally. Finally, the reduction provides the adversary with the public parameters pp along with the punctured key g^{kx} and the element $R = \{e(g, g)^{kz}, e(g, g)^r\}$ as given by the challenger. It is easy to see that whenever $R = e(g, g)^{kz}$, then the reduction perfectly simulates the output of the PRF, whereas if R is uniform, then the view simulated by the reduction is identical to the random case. This shows that any distinguisher against the security of the PRF is also a solver for the Bilinear Diffie-Hellman problem. □

Construction 5.3 (Linear Setup)**.** We specify the algorithms $\Pi_{\mathsf{PRF}} = (\mathsf{Setup}, \mathsf{PRF}, \mathsf{Puncture}, \mathsf{PuncturedEval})$ below.

- $\mathsf{Setup}(1^\lambda, 1^n)$:
  - $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, g, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$
  - Sample x uniformly at random for $\mathbb{Z}_p^*$

  - Return $\mathsf{pp} = (\mathcal{G}, \{g^{x^i}\}_{i \in [2n] \setminus \{n+1\}})$

- $\mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}, i)$:
  - Return $e(g,g)^{x^{n+1+i}\mathsf{k}} = (e(g,g)^{x^{n+1+i}})^{\mathsf{k}}$

- $\mathsf{Puncture}_{\mathsf{pp}}(\mathsf{k}, i^*)$:
  - Return $g^{x^{i^*}\mathsf{k}} = (g^{x^{i^*}})^{\mathsf{k}}$

- $\mathsf{PuncturedEval}_{\mathsf{pp}}(\mathsf{k}^*, i^*, i)$:
  - Return $\perp$ if $i = i^*$
  - Return $e(g,g)^{x^{n+1+i}\mathsf{k}} = e(g^{x^{i^*}\mathsf{k}}, g^{x^{n+1+i-i^*}})$

Analysis. It is immediate to see that the scheme satisfies correctness since for all $i^* \neq i$:

$$\mathsf{PuncturedEval}_{\mathsf{pp}}(\mathsf{k}^*, i^*, i) = e(g,g)^{x^{n+1+i}\mathsf{k}} = (e(g,g)^{x^{n+1+i}})^{\mathsf{k}} = \mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}, i).$$

It is equally easy to see that the scheme is (perfect) linearly key-homomorphic over $\mathbb{Z}_p^*$:

$$\prod_j \mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}_j, i) = \prod_j e(g,g)^{x^{n+1+i}\mathsf{k}_j} = e(g,g)^{x^{n+1+i}\sum_j \mathsf{k}_j} = \mathsf{PRF}_{\mathsf{pp}}\left(\sum_j \mathsf{k}_j, i\right).$$

We now show that the scheme is secure against the n-Power Diffie-Hellman assumption.

Theorem 5.4. *If the n-Power Diffie-Hellman assumption Assumption 2.12 holds, then construction Construction 5.3 satisfies security from Definition 2.1.*

Proof. The reduction is provided by the challenger with the following group elements

$$\left(g^x, \ldots, g^{x^n}, g^{x^{n+2}}, \ldots, g^{x^{2n}}, h, e(g,h)^{x^{n+1}}\right) \text{ or } \left(g^x, \ldots, g^{x^n}, g^{x^{n+2}}, \ldots, g^{x^{2n}}, h, e(g,g)^r\right).$$

The reduction sets (implicitly) $\mathsf{k} = \mathsf{DLog}(h)/x^{i^*}$ and sets the public parameters to

$$\mathsf{pp} = (\mathbb{G}, p, \{g^{x^i}\}_{i \in [2n] \setminus \{n+1\}})$$

which are easy to compute given the elements supplied by the challenger. The reduction provides the distinguisher with the public parameters pp along with the punctured key h and the element $R = \{e(g,h)^{x^{n+1}}, e(g,g)^r\}$ as given by the challenger. Observe that

$$h = g^{\mathsf{DLog}(h)} = g^{\mathsf{k}x^{i^*}} \text{ and } e(g,h)^{x^{n+1}} = e(g,g)^{x^{n+1}\mathsf{k}x^{i^*}} = e(g,g)^{x^{n+1+i^*}\mathsf{k}}$$

are identically distributed as the real view, whereas if $R = e(g,g)^r$, then the view of the adversary is identical to the random case. It follows that any distinguisher against the pseudorandomness of the PRF is also a solver for the n-Power Diffie-Hellman problem. □

Remark 5.5. Observe that in both of our constructions, $\mathsf{pp}_{\mathsf{PRF}}$ depend polynomially in n, but our algorithms PRF, Puncture, PuncturedEval only look at a constant number of group elements, hence run very efficiently in the RAM model of computation. When these PRF's are plugged into Construction 3.1, they give us efficient batching algorithms according to Remark 2.9.

5.2 (Almost) Key-Homomorphic Puncturable PRF from LWE

The constrained-key (almost) key-homomorphic PRF by [BV15] is already a (almost) key-homomorphic puncturable PRF. It, however, provides more functionality and stronger security than what we need. In the following we show how to simplify the construction drastically for our security and functionality notions.

We use the two algorithms from [BV15] (ComputeA, ComputeC) that allows us to embed circuits into matrices and LWE samples.

ComputeA$(F, \mathbf{A}_0, \ldots, \mathbf{A}_k)$: Takes as input a circuit $F : \{0,1\}^k \to \{0,1\}$ and k matrices $\mathbf{A}_0, \ldots, \mathbf{A}_k$ and outputs a matrix $\mathbf{A}_F$.

ComputeC$(F, x_1, \ldots, x_k, \mathbf{A}_0, \ldots, \mathbf{A}_k, \mathbf{a}_0, \ldots, \mathbf{a}_k)$: Takes as input a circuit $F : \{0,1\}^k \to \{0,1\}$, $k+1$ matrices $\mathbf{A}_0, \ldots, \mathbf{A}_k$, a k bits $x_1, \ldots, x_k$, and $k+1$ vectors $\mathbf{a}_0, \ldots, \mathbf{a}_k$ where $\mathbf{a}_i = \mathbf{s}^T(\mathbf{A}+x_i\mathbf{G})+\mathbf{e}_i$ and $\mathbf{s}^T(\mathbf{A}_0+\mathbf{G})+\mathbf{e}_0$. It outputs a vector $\mathbf{a}_{F,x}$ such that $\mathbf{a}_{F,x} = \mathbf{s}^T(\mathbf{A}_F + F(x)\mathbf{G}) + \mathbf{e}_F$

The runtime of both these algorithms roughly corresponds to a matrix multiplication per AND gate and a matrix addition per NOT gate.

Lemma 5.6 (Lemma 4.1 of [BV15]). *Let F be a Boolean circuit (of AND and NOT gates) on k input bits, and $x \in \{0,1\}^k$ be an input to the circuit. Let $\mathbf{A}_0, \mathbf{A}_1, \ldots, \mathbf{A}_k \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_k$ such that $\|\mathbf{a}_i - \mathbf{s}^T\mathbf{A}_i + x_i\mathbf{G}\|_\infty \leq B$ for $i \in [k]$ and $\|\mathbf{a}_0 - \mathbf{s}^T\mathbf{A}_0 + \mathbf{G}\|\infty \leq B$. Let $\mathbf{A}_F \leftarrow$ ComputeA$(F, \mathbf{A}_0, \ldots, \mathbf{A}_k)$ and $\mathbf{a}_{F,x} \leftarrow$ ComputeC$(F, x, \mathbf{A}_0, \mathbf{A}_1, \ldots, \mathbf{A}_k, \mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_k)$. Then with $E(F)$ being a noise growth estimation of the circuit F*

$$\|\mathbf{a}_{F,x} - \mathbf{s}^T\mathbf{A}_F + F(x)\mathbf{G}\|_\infty \leq E(F) \cdot B$$

where $E(F) = E_F(w_o)$ with w_o being the output wire of F and E_F is defined as follows.

$$E_F(w) = \begin{cases} 1 & \textit{if } w \textit{ is input wire} \\ 1 + E_F(w') & \textit{if } w \textit{ is the output wire of NOT gate with input } w' \\ m \cdot E_F(w_l) + E_F(w_r) & \textit{if } w \textit{ is the output wire of AND gate with left input } w_l \\ & \textit{ and right input } w_r \end{cases}$$

Furthermore, $\mathbf{a}_{F,x}$ is a "low-norm" linear function of $\mathbf{a}_0, \ldots, \mathbf{a}_k$. That is, there are matrices $\mathbf{Z}_0, \ldots, \mathbf{Z}_k$ (which depend on the circuit F, the input x, and the input matrices $\mathbf{A}_0, \ldots, \mathbf{A}_k$) such that $\mathbf{a}_{F,x} = \sum_{i=0}^{k} \mathbf{a}_i\mathbf{Z}_i$ and $\|\mathbf{Z}_i\|_\infty \leq E(F)$.

In the following we describe a PRF that largely follows the blueprint of the constrained-key key-homomorphic PRF of [BV15]. For our use-case the PRF does not have to be adaptively secure, i.e. the adversary is not allowed any queries to the PRF. This allows us to remove both the reliance on $1D - SIS$ and the admissible hash function.

We also do not need to be able to constrain the keys in arbitrary ways we just need the ability to puncture at a single point. This allows us to replace the universal circuit by the simpler equality circuit

$$EQ(x, x^*) = \bigwedge_{i \in [\lambda]} (x_i \overset{?}{=} x_i^*) = \bigwedge_{i \in [\lambda]} (\neg(x_i \wedge x_i^*) \wedge \neg(\neg x_i \wedge \neg x_i^*))$$

Notice that on the highest level this circuit is just a big and of 2λ many clauses $(c_j)_{j \in [2\lambda]}$ with each clause $E(c_i) \leq (2m+3)$. If we now arrange the big and such that the "heavy" part is in the right spline (i.e. $EQ = (c_1 \wedge (c_2 \wedge (c_3 \wedge \ldots (c_{2\lambda-1} \wedge c_{2\lambda}) \ldots))))$ we get $E(EQ) \leq (2\lambda - 1)m(2m+3) \leq O(\lambda m^2)$.

Construction 5.7. A PRF with domain $\{0,1\}^\lambda$ Let $\chi^m_{\sigma,B}$ be the discrete gaussian distribution with parameter σ trucated at B, if we write χ_σ then it is not truncated.

$\mathsf{Setup}(1^\lambda)$:

- Sample $\mathbf{A}_0 \leftarrow_{\$} \mathbb{Z}_q^{n\times m}$ uniformly at random
- Sample $\mathbf{A}_1 \leftarrow_{\$} \mathbb{Z}_q^{n\times m}$ uniformly at random
- Sample $\mathbf{B}_i \leftarrow_{\$} \mathbb{Z}_q^{n\times m}$ uniformly at random for each $i \in [\lambda]$
- Sample $\mathbf{D} \leftarrow_{\$} \mathbb{Z}_q^{n\times m}$ uniformly at random
- Sample $\mathbf{C} \leftarrow_{\$} \mathbb{Z}_q^{n\times m}$ uniformly at random
- Return $\mathsf{pp} = (\{\mathbf{A}_\beta\}_{\beta\in\{0,1\}}, \{\mathbf{B}_i\}_{i\in[\lambda]}, \mathbf{C}, \mathbf{D})$

$\mathsf{KeyGen}_{\mathsf{pp}}(x^*)$:

- Repeat:
  - Sample $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ uniformly at random
- Until $\mathbf{s}^T(\mathbf{B}_{EQ,x^*} + \mathbf{C})G^{-1}(\mathbf{D})$ has no entry in $[-B', B'] + (q/p)\mathbb{Z}$
- Return $\mathbf{s}$

$\mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}, x)$:

- Parse k into $\mathbf{s} \in \mathbb{Z}_q^n$
- Let $\mathbf{B}_{EQ,x} \leftarrow \mathsf{ComputeA}(EQ, \mathbf{A}_1, \mathbf{B}_1, \ldots, \mathbf{B}_\lambda, \mathbf{A}_{x_1}, \ldots, \mathbf{A}_{x_\lambda})$
- Return $\lfloor \mathbf{s}^T(\mathbf{B}_{EQ,x} + \mathbf{C})G^{-1}(\mathbf{D}) \rceil_p$

$\mathsf{Puncture}_{\mathsf{pp}}(\mathsf{k}, x^*)$:

- Parse k into $\mathbf{s} \in \mathbb{Z}_q^n$ and $\mathbf{e}_0 \in \mathbb{Z}_q^m$
- For each $\beta \in \{0,1\}$:
  - Sample $\mathbf{e}_{1,\beta} \leftarrow \chi^m_{\sigma,B}$ according error distribution
  - Let $\mathbf{a}_\beta = \mathbf{s}^T(\mathbf{A}_\beta + \beta\mathbf{G}) + \mathbf{e}_{1,\beta}$
- For each $i \in [\lambda]$:
  - Sample $\mathbf{e}_{2,i} \leftarrow \chi^m_{\sigma,B}$ according error distribution
  - Let $\mathbf{b}_i = \mathbf{s}^T(\mathbf{B}_i + x_i^*\mathbf{G}) + \mathbf{e}_{2,i}$
- Sample $\mathbf{e}_3 \leftarrow \chi^m_{\sigma,B}$ according error distribution
- Let $\mathbf{c} = \mathbf{s}^T\mathbf{C} + \mathbf{e}_3$
- Return $\mathsf{k}^* = (\{\mathbf{a}_\beta\}_{\beta\in\{0,1\}}, \{\mathbf{b}_i\}_{i\in[\lambda]}, \mathbf{c})$

$\mathsf{PuncturedEval}_{\mathsf{pp}}(\mathsf{k}^*, x^*, x)$:

- Compute

$$\begin{aligned}\mathbf{b}_{EQ,x,x^*} \leftarrow \mathsf{ComputeC}(&EQ, x^*, x, \mathbf{A}_1, \mathbf{B}_1, \ldots, \mathbf{B}_\lambda, \mathbf{A}_{x_1}, \ldots, \mathbf{A}_{x_\lambda},\\ &\mathbf{a}_1, \mathbf{b}_1, \ldots, \mathbf{b}_\lambda, \mathbf{a}_{x_1}, \ldots, \mathbf{a}_{x_\lambda})\end{aligned}$$

- Return $\lfloor (\mathbf{b}_{EQ,x,x^*} + \mathbf{c})^T G^{-1}(\mathbf{D}) \rceil_p$

Remark 5.8. The because the EQ circuit has $3\lambda - 1$ AND gates the runtime of PRF and PuncturedEval is dominated by the $3\lambda - 1$ matrix multiplications necessary to evaluate ComputeA or ComputeC.

Efficiency of Key Generation. Because $\mathbf{s}^T(\mathbf{B}_{EQ,x^*} + \mathbf{C})G^{-1}(\mathbf{D})$ is statistically close to uniform the resampling happens with probability $1 - (1 - (2B' + 1)p/q)^m \leq m(2B' + 1)p/q$. m, B', and p are polynomial and fixed. Therefore, we can choose $q \geq 2m(2B' + 1)p$, which is polynomial and will cause a resampling probability $\leq 1/2$.

If we choose q to be superpolyniomial the resampling probability is negligible. Therefore, $\mathbf{s}$ can be chosen uniformly at random without depending on i.

Theorem 5.9 (Pseudorandom at Punctured Point)**.** *For all x^* uniformly random* pp*. Let* $\mathsf{k} \leftarrow \mathsf{KeyGen}(x^*)$*, let* $\mathsf{k}^* \leftarrow \mathsf{Puncture}(\mathsf{k}, x^*)$ *and* $y = \mathsf{PRF}(\mathsf{k}, x^*)$ *and uniformly random u an adversary can not distinguish* $\mathcal{A}(\mathsf{pp}, \mathsf{k}^*, y)$ *from* $\mathcal{A}(\mathsf{pp}, \mathsf{k}^*, u)$*.*

Proof. We proof in hybrids

H_0 : In this hybrid we compute pp, k, k^*, and y according the original distribution.

H_1 : In the first hybrid we change how we sample matrices $\mathbf{A}_\beta$ for $\beta \in \{0, 1\}$ and $\mathbf{B}_i$ for $i \in [\lambda]$. We now sample $\hat{\mathbf{A}}_\beta$ and $\hat{\mathbf{B}}_i$ uniformly at random and then set $\mathbf{A}_\beta = \hat{\mathbf{A}}_\beta - \beta\mathbf{G}$ and $\mathbf{B}_i = \hat{\mathbf{B}}_i - x^*\mathbf{G}$. These two have the same distribution. Therefore, $\mathsf{Adv}_{H_1}(\mathcal{A}) = \mathsf{Adv}_{H_0}(\mathcal{A})$.

H_2 : Now we notice that if we sample $\mathbf{e} \leftarrow \chi^m_{\sigma,B}$ according to error distribution and let $\mathbf{d} = \mathbf{s}^T\mathbf{D} + \mathbf{e}$ and

$$\begin{aligned}\mathbf{b}_{EQ,x^*,x^*} \leftarrow \mathsf{ComputeC}(&EQ, x^*, x^*, \mathbf{A}_1, \mathbf{B}_1, \ldots, \mathbf{B}_\lambda, \mathbf{A}_{x^*_1}, \ldots, \mathbf{A}_{x^*_\lambda},\\ &\mathbf{a}_1, \mathbf{b}_1, \ldots, \mathbf{b}_\lambda, \mathbf{a}_{x^*_1}, \ldots, \mathbf{a}_{x^*_\lambda})\end{aligned}$$

Then we replace $\lfloor \mathbf{s}^T(\mathbf{B}_{EQ,x^*} + \mathbf{C})G^{-1}(\mathbf{D}) \rceil_p$ by $\lfloor (\mathbf{b}_{EQ,x^*,x^*} + \mathbf{c})^T G^{-1}(\mathbf{D}) - \mathbf{d}^T \rceil_p$. To see why this is valid we rewrite $\mathbf{y}$ in the following way:

$$\begin{aligned}\mathbf{y} &= \lfloor \mathbf{s}^T(\mathbf{B}_{EQ,x^*} + \mathbf{C})G^{-1}(\mathbf{D}) \rceil_p\\ &= \lfloor \mathbf{s}^T(\mathbf{B}_{EQ,x^*} + \mathbf{C} + \mathbf{G})G^{-1}(\mathbf{D}) - \mathbf{s}^T\mathbf{D} \rceil_p\\ &= \lfloor (\mathbf{b}_{EQ,x^*,x^*} + \mathbf{c})^T G^{-1}(\mathbf{D}) - \mathbf{d}^T + \mathbf{e}''^T \rceil_p \end{aligned} \tag{5}$$

with $\mathbf{e}'' = (\mathbf{e}_3 + \mathbf{e}')G^{-1}(\mathbf{D}) - \mathbf{e}$. By lemma 5.6 we know that $\|\mathbf{e}''\|_\infty \leq ((E(EQ) + 1) \cdot m + 1)B = B'$

This change only produces a different output if the vector $\mathbf{s}^T(\mathbf{B}_{EQ,x^*} + \mathbf{C})G^{-1}(\mathbf{D})$ has an entry in $[-B', B'] + (q/p)\mathbb{Z}$. Because $\mathbf{s}$ has been sampled using $\mathsf{KeyGen}(x^*)$ this does not happen.

H_3 : In the next hybrid we replace $\mathbf{a}_0, \mathbf{a}_1, \mathbf{b}_1, \ldots, \mathbf{b}_\lambda, \mathbf{d}$ by uniformly random vectors. We can do this by decisional LWE because in H_3

$$\begin{aligned} \mathbf{a}_\beta &= \mathbf{s}^T\hat{\mathbf{A}}_\beta + \mathbf{e}_{1,\beta} && \text{for all } \beta \in \{0, 1\}\\ \mathbf{b}_i &= \mathbf{s}^T\hat{\mathbf{B}}_i + \mathbf{e}_{2,i} && \text{for all } i \in [\lambda]\\ \mathbf{c} &= \mathbf{s}^T\mathbf{C} + \mathbf{e}_3 &&\\ \mathbf{d} &= \mathbf{s}^T\mathbf{D} + \mathbf{e} && \end{aligned}$$

where all the matrices are independent and uniform. This means the $|\mathsf{Adv}_{H_3}(\mathcal{A}) - \mathsf{Adv}_{H_4}(\mathcal{A})| \leq \mathsf{negl}(\lambda)$.

Because of $\mathbf{d}$'s uniformity we know that $\lfloor (\mathbf{b}_{EQ,x^*,x^*} + \mathbf{c})^T G^{-1}(\mathbf{D}) - \mathbf{d}^T \rceil_p$ is uniform if p divides q. □

Theorem 5.10 (Almost Functionality Preserving)**.** *For all x, $x^* \neq x$, sample* pp *uniformly at random* $\mathsf{k} \leftarrow \mathsf{KeyGen}(x^*)$ *correctly, let* $\mathsf{k}^* \leftarrow \mathsf{Puncture}_{\mathsf{pp}}(\mathsf{k}, x^*)$

$$\|\mathsf{PuncturedEval}_{\mathsf{pp}}(\mathsf{k}, x^*, x) - \mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}, x)\|_\infty \leq 1$$

Proof. Let $\mathbf{B}_{EQ,x} \leftarrow \mathsf{ComputeA}$

$$\begin{aligned}
\mathsf{PRF}_{\mathsf{pp}}(\mathsf{k}, x) &= \left\lfloor \mathbf{s}^T(\mathbf{B}_{EQ,x} + \mathbf{C})G^{-1}(\mathbf{D}) \right\rceil_p \\
&= \left\lfloor (\mathbf{b}_{EQ,x,x^*} + \mathbf{c})^T G^{-1}(\mathbf{D}) + \mathbf{e}^T \right\rceil_p && \text{for some } \mathbf{e} \text{ with } |\mathbf{e}| \leq E(EQ) \cdot mB \\
&= \left\lfloor (\mathbf{b}_{EQ,x,x^*} + \mathbf{c})^T G^{-1}(\mathbf{D}) \right\rceil_p + \{-1, 0, 1\}^m && (6) \\
&= \mathsf{PuncturedEval}_{\mathsf{pp}}(\mathsf{k}^*, x^*, x) + \{-1, 0, 1\}^m
\end{aligned}$$

Equation 6 holds because $|\mathbf{e}| \leq E(EQ) \cdot mB$ and we choose $q/p > E(EQ) \cdot mB$. □

Claim 5.11 (Almost Key Homomorphism)**.**

$$\left\| \mathsf{PRF}_{\mathsf{pp}}\left(\sum_{i \in [k]} \mathbf{s}_i, x \right) - \sum_{i \in [k]} \mathsf{PRF}_{\mathsf{pp}}(\mathbf{s}_i, x) \right\|_\infty \leq k - 1$$

This just follow from the fact that rounding is almost homomorphic. I.e., For any $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_q^m$ we have $\lfloor \mathbf{a} \rceil_p + \lfloor \mathbf{b} \rceil_p \leq \lfloor \mathbf{a} + \mathbf{b} \rceil_p + \mathbf{e}$ where $\mathbf{e} \in \{-1, 0, 1\}^m$.

Remark 5.12. Notice that almost functionality preservation and almost key homomorphism hold for any $\mathbf{s} \in \mathbb{Z}_q^n$ not only the ones sampled by KeyGen. This follows directly from the fact that the proofs of both these properties do not use the fact that $\mathbf{s}^T(\mathbf{B}_{EQ,x^*} + \mathbf{C})G^{-1}(\mathbf{D})$ has no entry in $[-B', B'] + (q/p)\mathbb{Z}$.

Choice of Parameters.

- Polynomial Modulus-to-Noise: For security and efficiency purposes it is advantagous if q is polynomial in λ. If one knows a polynomial upperbound on how often almost functionality preserving and almost key homomorphism is used then one can simply choose p to be more than double this upperbound in order to absorb all the errors that accumulate through these operations. Then the conditions towards all the are other parameters are $m \geq \log(q)n$, $q \geq 2m(2B' + 1)p$, $B = \alpha q \cdot \omega(\sqrt{\log(\lambda)})$, where α is the modulus-to-noise ratio and p divides q. This leads to a key-homomorphic puncturable PRF with exponential domain but a polynomial codomain modulus we mentioned in Remark 3.4.

- Superpolynomial Modulus-to-Noise: If one is willing to accept a superpolynomial modulus-to-noise ratio one can make q and p superpolynomial in λ. This has the advantage that now the key does not need to be rejection sampled as the rejection probability is negligible and p is big enough that it can absorb polynomially many key-homomorphic operations

6 Rogue Puzzle Attacks

In the following we formally consider the security of time-lock puzzles against *rogue-puzzle attacks*. First, we augment the syntax of our primitive with an additional algorithm that allows one to check that a puzzle is well-formed. Next, we formalize the security property as a cryptographic game. Finally, we provide a construction that satisfies this property in various settings.

Definition 6.1 (Rogue Puzzle Attacks)**.** We say $\Pi_{\mathsf{batchTLP}} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{BatchPSol}, \mathsf{IsValid})$ is secure against rogue puzzle attacks, if the syntax is augmented with the following algorithm:

- $\{0,1\} \leftarrow \mathsf{IsValid}(Z)$ a probabilistic algorithm that takes as input a puzzle Z and returns a bit $\{0,1\}$.

In addition, Π_{batchTLP} should satisfy the properties:

- **Validity Check:** We say Π_{batchTLP} satisfies validity check if for all $\lambda, n, T \in \mathbb{N}$, for all inputs $s \in \mathbb{S}_\lambda$ it holds that
$$\mathsf{IsValid}(\mathsf{Gen}(\mathsf{pp}, s)) = 1$$
where $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T)$.

- **Rogue Puzzle Security:** We say Π_{batchTLP} satisfies rogue puzzle security if for all $\lambda, T \in \mathbb{N}$, any polynomially bounded adversaries, $(\mathcal{A}_1, \mathcal{A}_2) = (\{\mathcal{A}_{1,\lambda}\}_{\lambda\in\mathbb{N}}, \{\mathcal{A}_{2,\lambda}\}_{\lambda\in\mathbb{N}})$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr\left[\begin{array}{c} \{1 = \mathsf{IsValid}(Z_j^*)\}_{j\in\mathcal{S}^*} \\ \wedge \left(\exists\, j \in \mathcal{S}^*,\ i \in \mathcal{S} : \left(Z_j^* = Z_i \wedge s_j^* \neq s_i\right)\right) \end{array} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T(\lambda)) \\ (\mathsf{st}, \{s_i\}_{i\in\mathcal{S}}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{pp}) \\ \{Z_i \leftarrow \mathsf{Gen}(\mathsf{pp}, s_i)\}_{i\in\mathcal{S}} \\ \{Z_j^*\}_{j\in\mathcal{S}^*} \leftarrow \mathcal{A}_2(\mathsf{st}, \{Z_i\}_{i\in\mathcal{S}}) \\ \left\{(s_j^*, Z_j^*)\right\}_{j\in\mathcal{S}^*} \leftarrow \mathsf{BatchPSol}(\mathsf{pp}, \left\{(Z_j^*)\right\}_{j\in\mathcal{S}^*}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

6.1 Constructions

We present separate constructions in the settings where the public parameters are bounded and unbounded. We assume that the construction Construction 4.2 consists of the following structure.

- We assume that the coordinated scheme in Construction 4.2 consists of two parts, one that's dependent on the coordinated index, and the other that is independent of the index. In our concrete construction in Construction 3.1, this corresponds to Z computed as $Z \leftarrow \mathsf{LHP.Gen}(\mathsf{pp}_{\mathsf{LHP}}, \mathsf{Encode}_{p,\ell}(\mathsf{k}))$. The index dependent part consists of i, k^*, c, the index, the punctured key and the punctured point computation.

  We use Z_{indep} below to clearly indicate the puzzle independent part.

Construction (Unbounded Setting). We can achieve the above definition by modifying Construction 4.2 in the following manner. Let Hash be a collision-resistant hash function with output space $\{0,1\}^\lambda$. We assume that the underlying coordinated space can handle unbounded indices in space $\{0,1\}^\lambda$.

For the puzzle generation algorithm we:

- We sample the puzzle independent instance Z_{indep}[6] and compute the index $i \in \{0,1\}^\lambda \leftarrow \mathsf{Hash}(Z_{\mathsf{indep}})$, where Hash and Z_{indep} are defined above.

- Add a non-interactive zero-knowledge (NIZK) proof that certifies that the punctured key is consistent with the index attached to the puzzle, as well as the key encoded in Z_{indep}.

[6] In the unbounded setting, the degree is 1 and the right side of the bipartite graph is the same as the left side.

The $\mathsf{IsValid}$ algorithm simply checks that the two conditions above are met. The batch-solving algorithm is unchanged, except that it ignores puzzles with duplicate indices, i.e., it treats them as if they were the same puzzle Z and solve one of them (chosen arbitrarily). It is easy to see that the construction is still correct and secure, with a straightforward reduction to the zero-knowledge property of the NIZK.

Next, we argue that the construction satisfies security against rogue puzzle attacks, for the case of *unbounded batching*. We consider two cases: (i) If all indices are pairwise distinct, then the property follows from the soundness of the NIZK and, consequently, from the correctness of the puzzle. (ii) If there is a collision, then we argue that the puzzle of the colliding indices must be the same and therefore it suffices to solve one of them (otherwise, we have a contradiction to the collision-resistance property of Hash).

Construction (Bounded Setting, Lattices). When we want to use a polynomial modulus for the lattice based PRF we can no longer sample the PRF key k and therefore Z independent of i. This leads us to a circularity. Z needs to depend on i and i is computed from Z. We resolve this issue by sampling them together. To see how this works we will briefly go over how the key k depends on i. It is rejection sampled according to some condition C_i that depends on i and holds with probability $1/2$ over a uniformly random key:

- Repeat until $C_i(k) = 1$: Sample k uniformly at random.

So we modify the generation in the following way:

- Repeat until $C_i(k) = 1$: Sample k uniformly at random. Generate the linearly-homomorphic time-lock puzzle containing k as such $Z \leftarrow \mathsf{TLP.Gen}(k)$. Compute the index $i \in \{0,1\}^\lambda \leftarrow \mathsf{Hash}(Z)$.
- Add a non-interactive zero-knowledge (NIZK) proof π that certifies that the punctured key k^* is consistent with the index i, as well as the key encoded in Z.

The proof of this argument is the same as in the unbounded setting. That is because the condition C_i is only necessary to guarantee security while almost correctness holds even if C_i does not hold (see Remark 5.12). Therefore, the worst an adversary can do by choosing a k that does not meet condition C_i is to diminish security of its own puzzle.

Construction (Bounded Setting, Pairings). We can achieve the above definition by modifying Construction 4.2 in the following manner. We model Hash as a random oracle that has output space $[n_{\mathsf{new}}]^d$.

For the puzzle generation algorithm we:

- We sample the puzzle indepedent indices $Z_{\mathsf{indep},1}, \ldots, Z_{\mathsf{indep},d}$ and then compute the set

$$V \leftarrow \mathsf{Hash}(Z_{\mathsf{indep},1}, \ldots, Z_{\mathsf{indep},d}),$$

  where Hash and Z_{indep} are defined above.
- Sample the remaining puzzle dependent instances, and for all $i \in [d]$, add a non-interactive zero-knowledge (NIZK) proof π_i that certifies that the punctured key k_i^* (corresponding to Z_i) is consistent with the index attached to the puzzle, as well as the key encoded in $Z_{\mathsf{indep},i}$.

The $\mathsf{IsValid}$ algorithm simply checks that the two conditions above are met and is same as before. The batch-solving algorithm is unchanged, except that if a perfect matching doesn't exist, we move forward with the maximal matching found. It is easy to see that the construction is still correct and secure, with a straightforward reduction to the zero-knowledge property of the NIZK.

The only difference is that in the security game, the adversary can query the random oracle mulitple times and possibly either find a duplicate set or might influence the algorithm in a malicious way to cause the correctly setup puzzles to be incorrect. To argue this is not possible, we tweak the parameters of n_{new} and d and augment aur analysis to depend on $q = q(\lambda)$, the number of random oracle queries an adversary $\mathcal{A}_2$ makes in Definition 6.1. As before, we consider two cases: (i) If a perfect matching is computed, then the property follows from the soundness of the NIZK and, consequently, from the correctness of the puzzle. (ii) If a perfect matching doesn't exist, it can happen due to two reasons. (ii)(a) If the exact same puzzle and set are chosen. In this case, it suffices to solve one of them. (ii)(b) The adversary has found a list of queries that violate a perfect matching by arbitarily querying the random oracle and still having $\mathsf{IsValid}$ hold. We show below that this is not possible.

Observe that in Eq. (4), the probability of choosing a set $S \subseteq U$ is now $\binom{q}{\ell}$ because the adversary $\mathcal{A}_2$ might sample multiple different index independent puzzles and can choose to group any subset $\mathcal{S}^*$ of them. Thus the expression to be analyzed changes to the following analysis,

$$\sum_{\ell=d}^{n} \binom{q}{\ell}\binom{n'}{\ell-1}\left(\frac{\binom{\ell-1}{d}}{\binom{n'}{d}}\right)^{\ell}. \tag{7}$$

A similar malicious expression was analyzed in [GLWW23] and built on our honest analysis. We mention the modified theorem statement below. The proof is very similar to the proof above.

Lemma 6.2. *Let $G = (U, V, E)$ be a random left regular bipartite graph where $|U| = n$, $|V| = n'$. Let the left regular degree be denoted by d. If $n' = 3n$, $d = \frac{O(\log q)}{\log(n)} + \frac{\omega(\log \lambda)}{\log(n')}$, then, the probability that there exists a perfect matching for G is $\geq 1 - \mathsf{negl}(\lambda)$.*

The proof repeats along the lines of the proof of Lemma 4.4. Observe that our analysis depends on $\frac{O(\log q)}{\log(n)}$, i.e. we only depend on logarithmic factors in q.

6.2 An Efficient NIZK Protocol

While a general purpose NIZK suffices for our construction. We demonstrate how to efficiently instantiate a NIZK for our pairing based key homomomorphic PRF and LWE based key homomorphic PRF.

Pairing-based key homomorphic PRF The main idea is to use a variant of Schnorr protocol/Chaum Pedersen protocol where the prover proves knowledge of an exponent k in two different groups of the same order N. Since $\phi(N)$ is not known, we need to be careful in arguing zero knowledge for the randomness and apply a smudging argument, and the randomness is hidden. If the groups are coprime to each other, we need to additionally constrain the TLP to argue soundness (please see Appendix B).

Construction 6.3 (Sigma protocol for pairing based KH-PRF and RSA based TLP)**.** Our construction relies on the following primitives:

- A linearly homomorphic TLP scheme, where the TLP is homomorphic in the message and the random coins. We describe this property in the TLP scheme from [MT19] below, for completeness[7].

[7] For brevity, we only show the puzzle generation algorithm.

Algorithm $\mathsf{TLP.Gen}(\mathsf{pp}, s; r)$, samples $r \leftarrow \mathbb{Z}_{N^2}$. Computes $u = g^r \in \mathbb{Z}_N$ and $v = h^{r \cdot N} \cdot (1+N)^s \mod N^2$. Output, (u, v). Note that if $(u_1, v_1) \leftarrow \mathsf{TLP.Gen}(\mathsf{pp}, s_1; r_1)$, $(u_2, v_2) \leftarrow \mathsf{TLP.Gen}(\mathsf{pp}, s_2; r_2)$, and $(u_3, v_3) \leftarrow \mathsf{TLP.Gen}(\mathsf{pp}, s_1 + s_2 \ ; \ r_1 + r_2)$, then, we have that $u_3 = u_1 \cdot u_2 \mod N$ and $v_3 = v_1 \cdot v_2 \mod N^2$.

- A group $\mathbb{G}$ with composite order N and generator g_1. Boneh, Go and Nissim [BGN05] formalized how to generate a bilinear group of composite order N (their construction requires N is square free and not divisible by 3. As N is a product of two large primes, we satisfy these constraints).

  We rely on Assumption 2.12, holding in a group where the order is N and integers x, y, r are sampled randomly from $\mathbb{Z}_N$.

We define a interactive 3-round sigma protocol argument and then collapse rounds using a Fiat-Shamir transform for sigma protocols. Let $\Pi = (\mathsf{Prove}, \mathsf{Verify})$ be a protocol for an instance $\chi = \left(\mathsf{pp}, Z, g_1^{x^{i^*}} \in \mathbb{G}, y \in \mathbb{G}\right)$ and witness $\omega = (k \in \mathbb{Z}_N, r \in \mathbb{Z}_{N^2})$ such that, $Z = \mathsf{TLP.Gen}(\mathsf{pp}, k; r)$ and $y = \left(g_1^{x^{i^*}}\right)^k \in \mathbb{G}$.

- $\mathsf{Prove}(\chi, \omega)$:
  - Sample randomly, $k' \leftarrow \mathbb{Z}_N$ and $r' \leftarrow [N^4]$.
  - Compute $Z' \leftarrow \mathsf{TLP.Gen}(\mathsf{pp}, k'; r')$, $y' \leftarrow \left(g_1^{x^{i^*}}\right)^{k'} \in \mathbb{G}$. The prover sends (Z', y') to the verifier.
  - Receive $c \in \mathbb{Z}_N$ from the verifier.
  - Compute $\hat{k} = k' + c \cdot k \in \mathbb{Z}_N$, and $\hat{r} = r' + c \cdot r \in \mathbb{Z}$.[8]
  - Send $\left(\hat{k} \in \mathbb{Z}_N, \hat{r} \in \mathbb{Z}\right)$ to the verifier.
  - Output $\pi = \left(Z', y' \in \mathbb{G}, \hat{k} \in \mathbb{Z}_N, \hat{r} \in \mathbb{Z}\right)$ as the proof.
- $\mathsf{Verify}(\chi)$:
  - The verifier recieves information from the prover and sends a random value $c \in \mathbb{Z}_N$.
  - Recieve $(\hat{k} \in \mathbb{Z}_N, \hat{r} \in \mathbb{Z})$ from the prover, and perform the checks below.
  - Check if $\mathsf{TLP.Gen}(\mathsf{pp}, \hat{k}; \hat{r}) \stackrel{?}{=} Z' \cdot Z^c$.
  - Check if $\left(g_1^{x^{i^*}}\right)^{\hat{k}} \stackrel{?}{=} y' \cdot y^c$.
  - If all checks pass, accept, else reject.

Completeness The scheme is complete, because $\mathsf{TLP.Gen}(\mathsf{pp}, \hat{k}; \hat{r}) = \mathsf{TLP.Gen}(\mathsf{pp}, k'; r') \cdot \mathsf{TLP.Gen}(\mathsf{pp}, k; r)^c = Z' \cdot Z^c$ as our time lock puzzle is linearly homomorphic in the puzzle and the random coins. Similarly, it's easy to check that the second condition holds true i.e. $\left(g_1^{x^{i^*}}\right)^{\hat{k}} = \left(g_1^{x^{i^*}}\right)^{k'} \cdot \left(g_1^{x^{i^*} \cdot k}\right)^c = y' \cdot y^c$.

[8] For value c in $\mathbb{Z}_N$ for some q, the prover considers it as a positive integer by setting the output in $1, \ldots, N$.

Soundness We argue statistical soundness of our scheme, i.e. if a verifier accepts a proof, then the statement is in the language, i.e. there exists some witnesses $k \in \mathbb{Z}_N, r \in [N^2]$ that agree with the statement. Let's assume that Verify accepts statement $\chi = \left(\mathsf{pp}, Z, g_1^{x^{i^*}} \in \mathbb{G}, y \in \mathbb{G}\right)$ and outputs a proof $\pi = \left(Z', y' \in \mathbb{G}, \hat{k} \in \mathbb{Z}_N, \hat{r} \in \mathbb{Z}\right)$ such that the verifier accepts on a random input $c \in \mathbb{Z}_N$. Without loss of generality, we can assume that $y' = g^{k_1'} \in \mathbb{G}$, $y = g^{k_1} \in \mathbb{G}$ for some $k_1', k_1 \in \mathbb{Z}_N$. Similarly, we can expand the time lock puzzle, and assume $Z' = \left(g^{r_0'} \mod N, h^{r_1' \cdot N} \cdot (1+N)^{k_0'} \mod N^2\right)$, $Z = (g^{r_0} \mod N, h^{r_1 \cdot N} \cdot (1+N)^{k_0} \mod N^2)$ where $k_0', k_0 \in \mathbb{Z}_N$, and $r_1', r_1, r_0', r_0 \in \mathbb{Z}_{\phi(N)}$. Since the proof is adverserial, it is possible that these values are all different and maliciously generated.

Since Verify accepts, we have,

- $\left(g_1^{x^{i^*}}\right)^{\hat{k}} = y' \cdot y^c$. Thus, $\hat{k} = k_1' + c \cdot k_1 \mod N$.
- $\mathsf{TLP.Gen}(\mathsf{pp}, \hat{k}; \hat{r}) = Z' \cdot Z^c$.

  We have, $g^{\hat{r}} = g^{r_0' + c \cdot r_0} \mod N$, thus, $\hat{r} = r_0' + c \cdot r_0 \mod \phi(N)$.

  Finally, $h^{\hat{r} \cdot N} \cdot (1+N)^{\hat{k}} = h^{r_1' + c \cdot r_1} \cdot (1+N)^{k_0' + c \cdot k_0} \mod N^2$. Plugging in our expression for $\hat{r}$ from the previous evaluation, and analyzing the expression modulo N, $h^{\left((r_0' - r_1') + c(r_0 - r_1)\right) \cdot N} = 1 \mod N$. Since r_0, r_1, r_0', r_1' are all output by the prover in the first message, and $N, \phi(N)$ are coprime to each other. The expression holds true if $c = (r_1' - r_0') \cdot (r_0 - r_1)^{-1} \mod \phi(N)$. This happens with probability $\leq \frac{\lceil N/\phi(N) \rceil}{N} < 2/\phi(N)$, which is negligible. Thus $r_1' = r_0' \mod \phi(N)$ and $r_0 = r_1 \mod \phi(N)$.

  Simplifying, we have $N \cdot \hat{k} = N \cdot (k_0' + c \cdot k_0) \mod N^2$. Plugging in our expression for $\hat{k}$, $(k_1' - k_0') + c \cdot (k_1 - k_0) = 0 \mod N$. The expression holds if $c = (k_1' - k_0') \cdot (k_0 - k_1)^{-1} \mod N$. This happens with probability $\leq 1/N$. Thus, $k_0 = k_1 \mod N$ and we have $k_0 = k_1 \mod N$.

Combining the equalities, we have proved that there exists $r \in \mathbb{Z}_{\phi(N)} \in [N^2]$ such that $r = r_0 = r_1 \mod \phi(N)$, and there exists $k \in \mathbb{Z}_N$ where $Z = \mathsf{TLP.Gen}(\mathsf{pp}, k; r)$ and $y = \left(g_1^{x^{i^*}}\right)^k$.

Zero Knowledge We prove the honest verifier zero knowledge of the interactive protocol. The simulator given instance χ computes the transcript in the following order.

- Sample $\tilde{k} \leftarrow \mathbb{Z}_N$ and $\tilde{r} \leftarrow [N^4]$. Sample $c \leftarrow \mathbb{Z}_N$.
- Compute $\tilde{y} = \frac{\left(g_1^{x^{i^*}}\right)^{\tilde{k}}}{y^c} \in \mathbb{G}$ and compute $\tilde{Z} \leftarrow \mathsf{TLP.Gen}(\mathsf{pp}, \tilde{k}, \tilde{r})$ and $Z' \leftarrow \frac{\tilde{Z}}{Z^c}$.
- The simulator outputs the transcript $\left(Z', y', c, \tilde{k}, \tilde{r}\right)$.

Observe that (1) $\tilde{k}$ is distributed identical to $k' + c \cdot k$ because k' is sampled randomly from $\mathbb{Z}_N$. (2) $\tilde{r}$ is distributed statistically close to $r' + c \cdot r$ because $\tilde{r}$ and r' are both sampled uniformly from $[N^4]$. Since $c \cdot r$ is small, i.e. $\leq N^3$, the distributions are apart with a distance $\leq \frac{N^3}{N^4} = \mathsf{negl}$.

Remark 6.4 (Collapsing rounds)**.** We can collapse rounds to generate a NIZK scheme by computing the challenge $c \in \mathbb{Z}_N \leftarrow H(Z', y')$ where H is a random oracle and using the standard Fiat-Shamir transformation for sigma protocols.[FS86].

LWE-based key homomorphic PRF The main idea is to exploit the (almost) key homomorphic property of our PRF and the linearly homomorphic property of our TLP. Since our PRF is almost key homomorphic, we use the NIZK range proofs from [TBM⁺20] to prove that the error in our homomorphic operation is small. Informally sketching, assume that the TLP encodes key k, and the punctured key outputs $kA + e$, where A is a public matrix. Using the homomorphic property, we can compute the TLP encoding on error e attach a NIZK range proof proving that the value encoded is small.

Construction 6.5 (NIZK protocol for LWE based KH-PRF and RSA based TLP)**.** Our construction relies on the following primitives:

- A linearly homomorphic TLP scheme for messages $\mathbf{s} \in \mathbb{Z}_N^n$ where we can perform linear operations over the message space. We describe this property in parallel version of the TLP from [MT19] below, for completeness[9].

  Algorithm $\mathsf{TLP.Gen}(\mathsf{pp}, \mathbf{s} \in \mathbb{Z}_N^n)$, samples $\mathbf{r} \leftarrow \mathbb{Z}_{N^2}^n$. Computes $\mathbf{u} = g^{\mathbf{r}} \in \mathbb{Z}_N^n$ and $\mathbf{v} = h^{\mathbf{r}\cdot N} \odot (1+N)^{\mathbf{s}} \bmod N^2$. Output, $(\mathbf{u}, \mathbf{v})$.

  Let $f(\mathbf{s})$ be a linear map $\mathbb{Z}_N^n \to \mathbb{Z}_N^m$, let this be denoted by the operation $\mathbf{s}^T\mathbf{A} + \mathbf{b} \in \mathbb{Z}_N^m$, then we can compute $\mathsf{TLP.Gen}(\mathsf{pp}, f(\mathbf{s}); \mathbf{r})$, by computing $u_i' = \prod_{j\in[n]} u_j^{A_{j,i}}$ and $v_i' = \prod_{j\in[n]} v_j^{A_{j,i}} \cdot (1+N)^{b_i}$ for $i \in [m]$ and outputting $(\mathbf{u}', \mathbf{v}')$.

- Our lattice-based key-homomorphic puncturable PRF.

  As we want computation over the same ring for our puncturable PRF and our time lock puzzle, we rely on LWE holding in a ring where the modulus is a composite number N (same as the modulus of the time-lock puzzle).

  The important detail about the PPRF is that a key punctured at x has the form $\mathbf{k}^T\mathbf{A}_x + \mathbf{e}$, for some $n, m, B \in \mathbb{Z}$[10], $B < N$, $\mathbf{k} \in \mathbb{Z}_N^n$, $\mathbf{A}_x \in \mathbb{Z}_N^{n\times m}$, and $\mathbf{e} \in [-B, B]^m$. $\mathbf{A}_x$ is public and depends on x.

- A special-case NIZK (called range proof) $\Pi_{\mathsf{range}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ that proves the plaintext of a time-lock puzzle Z is in range $[-B, B]$. A construction of such a range proof was given by [TBM⁺20].

We define a NIZK scheme $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is an argument for the statement, $\chi = (\mathsf{pp}, Z, \mathbf{A}_x, \mathbf{b})$ and witness $\omega = (\mathbf{r} \in \mathbb{Z}_{N^2}^n, \mathbf{k} \in \mathbb{Z}_N^n, \mathbf{e} \in [-B, B]^m)$, where the NP verifier checks if, $\mathsf{TLP.Gen}(\mathsf{pp}, \mathbf{k}; r) \overset{?}{=} Z$, and, $\mathbf{k}^T\mathbf{A}_x + \mathbf{e} \overset{?}{=} \mathbf{b}$.

- $\mathsf{Setup}(1^\lambda)$: Let $\mathsf{crs} \leftarrow \mathsf{range.Setup}(1^\lambda)$.
- $\mathsf{Prove}(\mathsf{crs}, \chi, \omega)$:
  - Homomorphically evaluate $f : \mathbf{s} \mapsto \mathbf{s}^t\mathbf{A}_x - \mathbf{b}$ on Z to get a new puzzle Z'.
  - We output a NIZK range proof $\pi \leftarrow \mathsf{range.Prove}\left(\mathsf{crs}, (\mathsf{pp}, Z'), \left(\mathbf{e} \in \mathbb{Z}_N^m, \mathbf{r}^T\mathbf{A}_x \in \mathbb{Z}^m\right)\right)$ where Z' is the puzzle and the witness is $\left(\mathbf{e} \in [-B, B]^m, \mathbf{r}^T\mathbf{A}_x \in \mathbb{Z}^m\right)$.
- $\mathsf{Verify}(\mathsf{crs}, \chi, \pi)$:
  - Homomorphically evaluate $f : \mathbf{s} \mapsto \mathbf{s}^t\mathbf{A}_x - \mathbf{b}$ on Z to get a new puzzle Z'.
  - Output the result of $\mathsf{range.Verify}(\mathsf{crs}, (\mathsf{pp}, Z'), \pi)$.

[9] For brevity, we only show the puzzle generation algorithm.

[10] We're overloading the notation m from previous sections. It does not match the m in the PPRF construction.

Completeness By definition we have N and $\mathbf{b} \equiv \mathbf{k}^T \mathbf{A}_x + \mathbf{e} \ mod \ N$ with $\mathbf{e} \in [-B, B]^m$. Therefore, $\exists \mathbf{r}' \in \mathbb{Z}^m$ s. t. $\mathsf{TLP.Gen}(\mathsf{pp}, -\mathbf{e}; \mathbf{r}') = Z'$. Also, the order of $g \in \mathbb{Z}_N$ and $h^N \in \mathbb{Z}_{N^2}$ is $\phi(N)$. Thus, if the equation $\mathbf{r}' = \mathbf{r}^T \mathbf{A}_x$ holds over the integers, it also holds modulo $\phi(N)$. Therefore, $\mathsf{TLP.Gen}(\mathsf{pp}, \mathbf{e}'; \mathbf{r}') = Z'$.

Soundness If the statement characterized by $(\mathsf{pp}, Z, \mathbf{A}_x, \mathbf{b})$ is not in the language, then evaluating $f : \mathbf{s} \mapsto \mathbf{s}^t \mathbf{A}_x - \mathbf{b}$ on Z will not yield a puzzle Z' that is in the range $[-B, B]^m$. Therefore, the range proof will fail.

Zero Knowledge Zero knowledge straightforwardly follows from the zero knowledge of the range proof.

7 Implementation and Evaluation

In this section, we describe the implementation and evaluation of our efficiently batchable time lock scheme. The goal of our experimental evaluation is to compare the different tradeoffs offered by our solution in computation time and communication size for different values of the number of puzzles batched. In our experiments, we provide a comparison of the following alternative approaches.

- **Trivial Solution:** Batch solving a time lock puzzle involves solving each of these puzzles individually. We initialize our time lock puzzle using the linearly homomorphic time lock puzzle [MT19][11].
- **Strawman Solution:** Given n puzzles $Z_1, \ldots, Z_n$ (of some linearly homomorphic time-lock puzzle) where each puzzle contains some λ-bit message, evaluate homomorphically the following linear function:

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

  to compute puzzle Z^*. Solve the resulting puzzle Z^* to obtain x^*, and recover all the n messages encoded in different blocks of the string (where each block is of length security parameter). We initialize our time lock puzzle using the linearly homomorphic time lock puzzle [MT19]. We assume that our messages can be λ bits long as for longer messages, we can use hybrid encryption and encrypt the keys for the symmetric encryption scheme. As an example, for 128 bit security, the RSA modulus is 3072 bits, and the security parameter for symmetric key encryption scheme is 128 bits, we can batch $3072/128 = 24$ puzzles for free using the strawman solution.
- **Our Solution:** We analyze our uncoordinated scheme that uses the pairing based key homomorphic PRF in Construction 5.3 as the building primitive in Construction 3.1 and applies our generic transformation from Construction 4.2. We initialize our time lock puzzle using the linearly homomorphic time lock puzzle [MT19]. We make small adjustments to Construction 5.3 for better concrete efficiency i.e. we use asymmetric groups for better efficiency. Security of this construction is based on the asymmetric n-Power Diffie-Hellman assumption.

We do not consider alternative constructions based on general purpose indistinguishability obfuscation iO [SLM+23] as iO is a heavyweight cryptographic primitives, not ready for efficient deployment (there are certain *restricted* functionalities [LMA+16, CMR17] implementations of iO, but there are no general

[11] It is possible to use a time lock puzzle that is not linearly homomorpic for this evaluation. We chose the TLP from [MT19] for a more direct comparison with the other two solutions.

purpose implementations). We leave open the implementation based on our LWE based key homomorphic PRF Construction 5.7 for future work.

7.1 Implementation and Experimental Setup

We instantiate the cryptographic building blocks that offer 128 bits of security, as follows:

- **Pairing group:** We instantiate the pairing-based broadcast encryption schemes over the BLS-381 pairing group [BLS02] and use the implementation from the herumi `mcl` library [Mit]. We used the C++ api to perform our pairing operations. The BLS-381 pairing group is asymmetric, and the (serialized) representations of an element of the base groups $\mathbb{G}_1$, $\mathbb{G}_2$, and the target group $\mathbb{G}_T$ are 48 bytes, 96 bytes, and 576 bytes, respectively.

- **RSA group:** We use the RSA assumption where the modulus is 3072 bits. We used the implementation present in the paper [TBM+20] (we use the implementation at `https://github.com/verifiable-timed-signatures/liblhtlp`), which uses GNU Multi-Precision library [GMP] (version 6.2.1) and is implemented in C.

Parameter selection for Construction 4.2. Recall from Theorem 4.3, for n denoting a maximum bound on the number of puzzles to be batched, we require setting our pairing scheme $n' \geq 3n$ and $d = O(1) + \frac{\omega(\log \lambda)}{\log n'}$. In our experimentation, we choose the parameters so that Eq. (4) satisfies 40 bits of statistical security i.e. the probability of a matching not existing is 2^{-40}. Combining this with our pairing and RSA implementation, we satisfy 40 bits of statistical security and 128 bits of computational security. Our main goal is to choose the parameters n', d that minimize the degree d. Achieving the right set of parameters is tricky, as for each configuration n', there exists a minimum degree d that satisfies Eq. (4) with 40 bits of security, and for each degree there exists a configuration n' that minimizes the setup and public parameter size and satisfies Eq. (4) with 40 bits of security.

We chose our parameters in the following way. In our experiments, the maximum number of puzzles ranged from $1 - 10k$.

- Set an initial $n' = 100k$. Our pairing based key homomorphic PRF only took 50 seconds to execute.

- Minimize the degree by testing the values of d between 1 and 128 that satisfy Eq. (4) with 40 bits of security. Lets call this degree d_{opt}.

- Perform a binary search where the range of n' is between n and $100k$, that result in the minimum value of n' such that we satisfy Eq. (4) with 40 bits of security. We denote this by n'_{opt}.

In our prototype implementation, we do not focus on the malicious setting. For the malicious setting, we would instead optimize on the expression in Eq. (7) for an appropriate choice of a bound on the number of queries q. Finally, we implemented the textbook Hopcroft Karp bipartite matching algorithm in C++.

Time Complexity. We analyze the time complexity of our batch solving algorithm for different approaches.

- **Trivial Solution:** The total compute time grows with $O(n \cdot T)$ where T is the number of repeated squaring exponentiations performed.

- **Strawman Solution:** The strawman solution performs best and takes $O(T)$ time to compute the result of all the opeinngs (though it leads to extremely high communication complexity).
- **Our Solution:** We take $O(T) + O(n^2)$ where the latter is because each puzzle takes $O(n)$ operations in Construction 3.1. Since n and T differ by several orders of magnitude as discussed in Section 1. Our solution is concretely efficient.

Parallel Computation. We use a single-threaded execution environment for all measurements. For our running time measurements, the trivial solution and our solution are easily parallelizable operations. Instead, we focus on the total CPU computation performed by the two schemes and do not exploit parallelization. Throughout this text we refer to the running time in seconds, but this can be interpreted as linearly related to total CPU cycles needed to perform the complete experiment. When reporting parameter sizes (e.g., setup size and puzzle size), we compute them *analytically* based on the number of group elements and the measured size of each group element.

Remark 7.1 (Parallelizing the key homomorphic PRF and the time lock puzzle)**.** For large values of n, parallelizing n time lock puzzles in the trivial solution involves very heavy parallelization and compute resources. In contrast, in our solution, if we use two threads in our implementation of Construction 3.1, on one thread we can homomorphically add the TLP and solve it, and on the other thread, we can compute a perfect matching and perform quadratic operations on the punctured key. Our implementation takes wall clock time similar to solving exactly one TLP! Observe that in our solution, after we solve the homomorphic puzzle, we only need to perform linear operations to recover the messages.

Experimental setup. Our implementation of our scheme consists of 2400 lines of code.[12] We collect our benchmarks on an client side MacBook Pro (13-inch, M1, 2020) running macOS Big Sur Version 11.5.2. The machine has a 8-core CPU @ 2.90GHz and 16 GB of RAM [13].

7.2 Benchmarks

In this section, we describe the main benchmarks (in terms of running time and communication size) for our batchable time lock puzzle. For our trivial solution, since solving n puzzles would take a bunch of time, instead, we simulate our numbers by solving 10 puzzles and measuring the total compute time by multiplying n to the average.

Computational cost. To show advantages of our scheme, we test our prototype implementation on the puzzle values n between 1 and 500 and the exponent T between 10^7, $5 \cdot 10^7$ and 10^8, roughly corresponding to 10 seconds, 50 seconds, and 100 seconds respectively on the test machine Fig. 1. In practice, the time for the puzzle lock will change between machine implementations. Hence the amount of sequential computation usually accounts for these variations.

We mainly compare between the trivial and our solution (the statistics for the strawman solution are the same as running solve denoted by the green line, and we do not add it into the graphs to prevent over crowding). Our experiments show that for even such small values of T, the trivial solution takes a longer compute time, while puzzle generation and setup become slightly worse Fig. 1. The dotted line indicates the plot for our trivial solution while the solid line plots our solution. Quoting concrete numbers, for $T = 50$

[12] The complete implementation is available here: `https://github.com/RachitG54/batchtlpmcl` .

[13] Our experiments did not rely on any heavy RAM usage.

million, and $n = 500$, the degree of our graph is 3. Batching trivially takes is 160× worse and would take about 15 hours of compute time. Our puzzle generation becomes slightly worse, for $T = 50$ million, per puzzle generation takes 3× time (as the degree of the graph is 3), but still generate the puzzle extremely efficiently and within 0.2 seconds. Setup for the trivial solution now grows linearly with the number of puzzles. But even for $n = 100k$ puzzles would only cause an overhead of 50 seconds while sampling a RSA integer that is a product of two strong primes takes about 2 minutes. Since n is at most 500, the setup time for all three schemes coincide.

Communication size. In Fig. 2, we compare the setup size and the total communication size as a function of the number of puzzles batched. We computed the numbers analytically, where we varied n between 100 and 7000. For setup size, our solution is strictly worse in terms of setup size where the setup grows with the number of puzzles. For $n = 7k$, our $n' = 11k$, and the setup size is 2200× worse, but still takes only 2.6MB size. For total communication size, the trivial solution is the most efficient and takes 8MB of communication, and is atmost 5× better than our solution that takes 37MB. The strawman solution becomes quadratically inefficient with increasing values of n where the communication takes 790MB.

Microbenchmarks. In Fig. 3, we compare the distribution of our computation time between the pairing operations and the graph operations for our solution for $T = 5 \times 10^8$. For our setup, the pairing operations grow linearly, but are extremely efficient and only take 0.4 seconds for $n = 500$. Puzzle generation involves an extra pairing operation that computes the punctured key in 4 milliseconds. For our batch solving algorithm, most of the time is dominated by the puzzle solving. Our pairing operations grow quadratically in the number of puzzles, while the graph algorithm is blazingly efficient and runs in a few micro seconds. For $n = 500$, the time to batch solve takes $1.2x$ slower than solving a single puzzle (it's also equal to the time taken to batch solve using the strawman solution). It takes 22 minutes to batch solve, 18.5 minutes to solve a single puzzle and 3.5 minutes to compute the quadratic pairing operations.

Rebalancing parameters. Using a standard rebalancing technique we can obtain different tradeoff for our batch solving algorithm. Our strawman solution gives parameters efficient in computation, but grow quadratically in total communication cost. On the other hand, our solution gives parameters very efficient in communication, but we need to compute quadratically many pairing operations. If our TLP is locked for a longer time such as few hours or days, even for large parameters, time to compute the pairing will be much smaller. Nevertheless, we can imagine combining these approaches to batch B puzzles together where each of these puzzles encodes A messages such that $n = A \times B$. Asymptotically, this leads to communication cost $O(n \cdot A)$ and computation cost $O(B^2)$.

Remark 7.2. Note that in our experiments, the RSA modulus is 3072 bits, and the security parameter for symmetric key crypto is 128 bits. Hence we can set $B = 3072/128 = 24$, to save on our computation, while keeping the communication size of the puzzle exactly same! In our prototype implementation, we do not explore the tradeoffs associated with combining our solutions. Since the hardness of our time lock puzzle and the quantity to optimize depends from user to user, we defer this task to the system designer when designing the scheme.

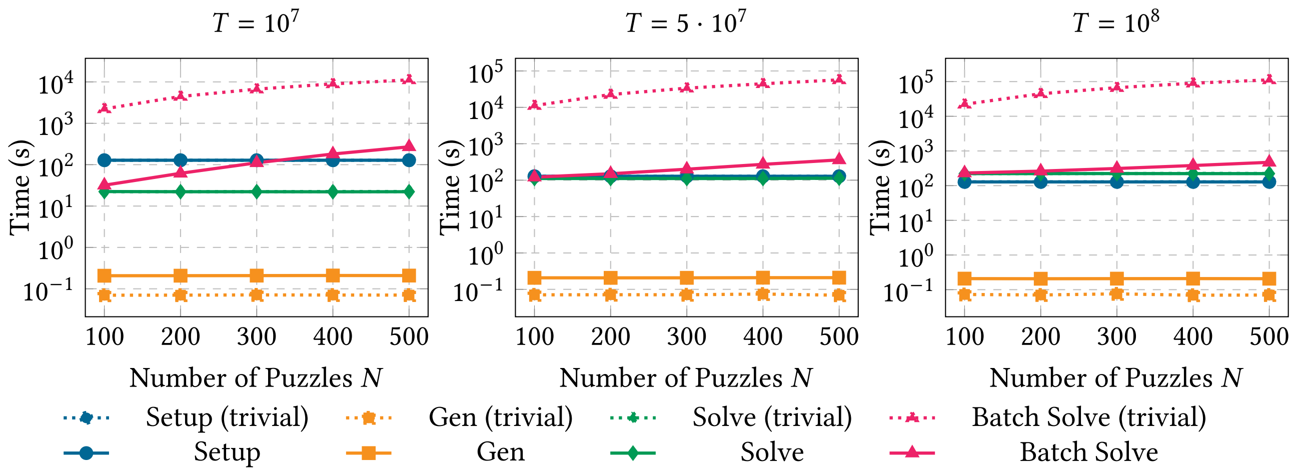

Figure 1: Setup, puzzle generation, puzzle solving and batch solving times for the trivial solution (indicated by dotted line) and our proposed solution (indicated by solid line). We vary the number of puzzle betwen 100 and 500 for different hardness of sequential computation, ranging from $T = 10^7$ to 10^8. The y-axis is over a log scale.

References

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2005.

[BDGM19] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 407–437. Springer, 2019.

[BDGM22] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Factoring and pairings are not necessary for IO: circular-secure LWE suffices. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 28:1–28:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[BF21] Jeffrey Burdges and Luca De Feo. Delay encryption. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 302–326. Springer, 2021.

[BFP+15] Abhishek Banerjee, Georg Fuchsbauer, Chris Peikert, Krzysztof Pietrzak, and Sophie Stevens. Key-homomorphic constrained pseudorandom functions. In Yevgeniy Dodis and Jesper Buus

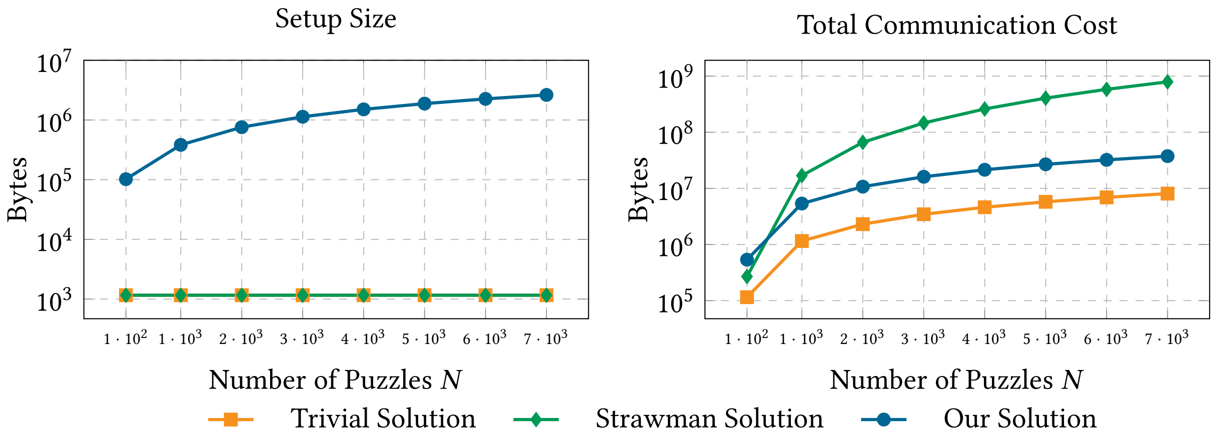

Figure 2: Setup size and total communication size (computed analytically) needed for different schemes that support batch solving. We compare the trivial solution, the strawman solution and our proposed solution. The total communication size includes the size of each puzzle multiplied by the number of parties in the system. The graph is independent of the hardness of the time lock puzzle. The y-axis is over the log scale.

Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 31–60. Springer, 2015.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2001.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, volume 8383 of *Lecture Notes in Computer Science*, pages 501–519. Springer, 2014.

[BGJ+16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 345–356. ACM, 2016.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 439–448. ACM, 2015.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2*, pages 325–341. Springer, 2005.

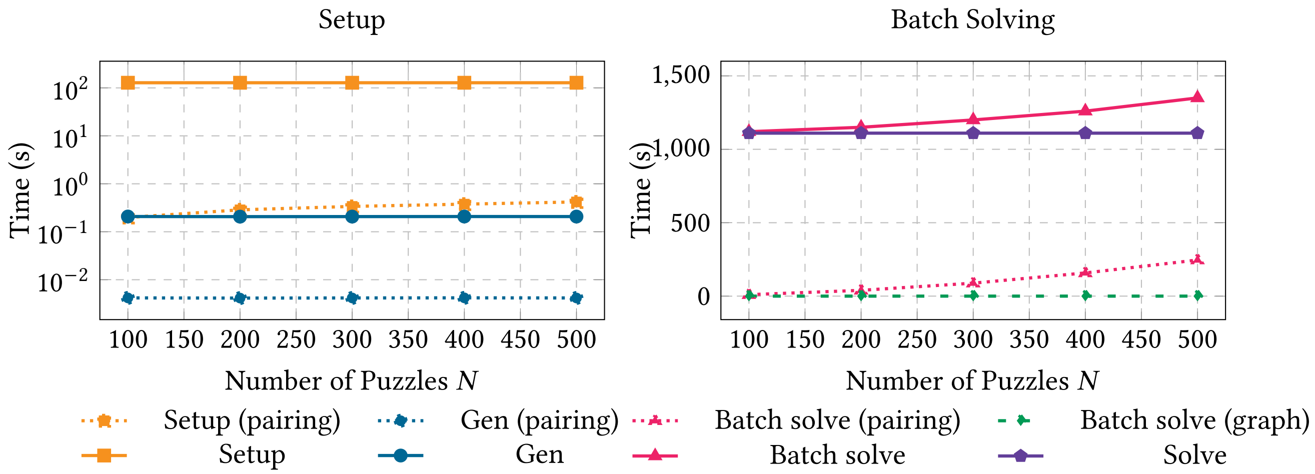

Figure 3: Computational cost breakdown for the setup, puzzle generation, solving and batch solving of our time lock puzzle. Batch solving involves (1) a combinatoric step (finding a matching in the bipartite graph) indicated by a dashed line; and (2) a cryptographic step for performing operations on the pairing group indicated by a dotted line and (3) repeated squaring operations on the RSA group. Algorithms, setup, puzzle generation do not include the combinatorial step and the complete computation is indicated by the solid line. We report the running times as a function of the maximum number of puzzles matched N and the exponent of sequential computation equal to $T = 5 \times 10^{8}$. For our first graph, the y-axis is over the log scale to display the different tradeoffs, while for the second graph, the y-axis shows the more fine-grained dependence.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 258–275. Springer, 2005.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428. Springer, 2013.

[BLS02] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In *SCN*, 2002.

[BN00] Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254. Springer, 2000.

[BP14] Abhishek Banerjee and Chris Peikert. New and improved key-homomorphic pseudorandom functions. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO*

2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I, volume 8616 of *Lecture Notes in Computer Science*, pages 353–370. Springer, 2014.

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic prfs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 1–30. Springer, 2015.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 280–300. Springer, 2013.

[CHSS02] Liqun Chen, Keith Harrison, David Soldera, and Nigel P. Smart. Applications of multiple trust authorities in pairing based cryptosystems. In George I. Davida, Yair Frankel, and Owen Rees, editors, *Infrastructure Security, International Conference, InfraSec 2002 Bristol, UK, October 1-3, 2002, Proceedings*, volume 2437 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2002.

[CMR17] Brent Carmer, Alex J. Malozemoff, and Mariana Raykova. 5Gen-C: Multi-input functional encryption and program obfuscation for arithmetic circuits. In *ACM CCS*, 2017.

[CRV16] Nishanth Chandran, Srinivasan Raghuraman, and Dhinakaran Vinayagamurthy. Reducing depth in constrained prfs: From bit-fixing to \mathbf ncˆ1. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part II*, volume 9615 of *Lecture Notes in Computer Science*, pages 359–385. Springer, 2016.

[DHMW22] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. Mcfly: Verifiable encryption to the future made practical. *IACR Cryptol. ePrint Arch.*, page 433, 2022.

[FKPS21] Cody Freitag, Ilan Komargodski, Rafael Pass, and Naomi Sirkin. Non-malleable time-lock puzzles and applications. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part III*, volume 13044 of *Lecture Notes in Computer Science*, pages 447–479. Springer, 2021.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49. IEEE Computer Society, 2013.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 276–288. Springer, 1984.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. *IACR Cryptol. ePrint Arch.*, page 258, 2013.

[GJLS21] Romain Gay, Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from simple-to-state hard problems: New assumptions, new techniques, and simplification. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 97–126. Springer, 2021.

[GKM^{+}18] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-snarks. In *Annual International Cryptology Conference*, pages 698–728. Springer, 2018.

[GLWW23] Rachit Garg, George Lu, Brent Waters, and David J. Wu. Realizing flexible broadcast encryption: How to broadcast to a public-key directory, 2023. CCS.

[GMP] The gnu multiple precision arithmetic library.

[GP21] Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 736–749. ACM, 2021.

[Hal35] P Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 1935.

[HK73] John E Hopcroft and Richard M Karp. An nˆ5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 60–73. ACM, 2021.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over \$\mathbb {F}_p\$, dlin, and prgs in nc^{0}. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 670–699. Springer, 2022.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 669–684. ACM, 2013.

[LKW15] Jia Liu, Saqib A. Kakvi, and Bogdan Warinschi. Extractable witness encryption and timed-release encryption from bitcoin. *IACR Cryptol. ePrint Arch.*, page 482, 2015.

[LMA+16] Kevin Lewi, Alex J. Malozemoff, Daniel Apon, Brent Carmer, Adam Foltzer, Daniel Wagner, David W. Archer, Dan Boneh, Jonathan Katz, and Mariana Raykova. 5Gen: A framework for prototyping applications using multilinear maps and matrix branching programs. In *ACM CCS*, 2016.

[LPS17] Huijia Lin, Rafael Pass, and Pratik Soni. Two-round concurrent non-malleable commitment from time-lock puzzles. *IACR Cryptol. ePrint Arch.*, page 273, 2017.

[Mit] Shigeo Mitsunari. mcl: a portable and fast pairing-based cryptography library.

[MT19] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic time-lock puzzles and applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 620–649. Springer, 2019.

[NPR99] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and kdcs. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 327–346. Springer, 1999.

[NRBB22] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. *IACR Cryptol. ePrint Arch.*, 2022.

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

[SLM+23] Shravan Srinivasan, Julian Loss, Giulio Malavolta, Kartik Nayak, Charalampos Papamanthou, and Sri AravindaKrishnan Thyagarajan. Transparent batchable time-lock puzzles and applications to byzantine consensus. *Public Key Cryptography*, 2023.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484. ACM, 2014.

[TBM+20] Sri Aravinda Krishnan Thyagarajan, Adithya Bhat, Giulio Malavolta, Nico Döttling, Aniket Kate, and Dominique Schröder. Verifiable timed signatures made practical. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1733–1750, 2020.

[TCLM21] Sri Aravinda Krishnan Thyagarajan, Guilhem Castagnos, Fabien Laguillaumie, and Giulio Malavolta. Efficient CCA timed commitments in class groups. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 2663–2684. ACM, 2021.

[TMSS22] Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, Fritz Schmid, and Dominique Schröder. Verifiable timed linkable ring signatures for scalable payments for monero. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26-30, 2022, Proceedings, Part II*, volume 13555 of *Lecture Notes in Computer Science*, pages 467–486. Springer, 2022.

[WW21] Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious LWE sampling. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 127–156. Springer, 2021.

[WXDS20] Jun Wan, Hanshen Xiao, Srinivas Devadas, and Elaine Shi. Round-efficient byzantine broadcast under strongly adaptive and majority corruptions. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 412–456. Springer, 2020.

A Analyzing alternative algorithms for matching

Let n be the size of the left bipartite set, $n'(n, \lambda)$ be the size of the right bipartite set (as functions of n and the security parameter[14]) and $d(n, \lambda)$ be the degree of the bipartite graph. We analyze the different algorithms that ensure that a perfect matching is found below.

- **Trivial Algorithm:** In order to compute a perfect matching always, we can set the graph to be a complete bipartite graph. The parameters are $n' = n$, $d = n$. Note that this also means that even if a malicious party tries to sample puzzles, we will guarantee existence of a perfect matching.
- **Greedy Algorithm:** Algorithm $\mathsf{FindMatch}$ performs a greedy analysis as follows, for each vertex on the left (on a lexicographic ordering of the vertices), it goes through each edge on the right, if it finds an unmatched vertex, it adds it to the matching. If for some vertex, all vertices on the right are matched, it outputs $\perp$. We present the analysis of the algorithm below where we can set $n' = 2n$, $d = O(\log n) + \omega(\log \lambda)$. Note that our matching analysis through hall's theorem in Lemma 4.4 gives a better theoretical bound by a factor of $\log n$ and in general a more optimal expression to analyze. Interestingly, we also show that when analyzing malicious parties, our greedy algorithm does not help us.

The main takeaway is that running a bipartite matching algorithm leads to more concretely efficient parameters for our transformation both in the honest and the rogue setting.

Lemma A.1. *Let $G = (U, V, E)$ be a random left regular bipartite graph where $|U| = n$, $|V| = n'$. Let the left regular degree be denoted by d. If $n' = 2n$, $d = O(\log n) + \omega(\log \lambda)$, then, the probability that the greedy algorithm outputs a perfect matching for G is $\geq 1 - \mathsf{negl}(\lambda)$ where the probability is taken over the random coins of sampling the bipartite graph.*

[14] Observe that this is the statistical security parameter and can hence be set as 40 or 60 in practice.

Proof. Algorithm FindMatch goes through each vertex on the left and tries to match them greedily. Let FindMatch output $\perp$ on the kth iteration, i.e. we're trying to match the kth vertex on the left. The probability that FindMatch outputs $\perp$ in this iteration, if all the vertices already matched are on the kth vertices edge set. This is given by,

$$\left(\frac{k-1}{n'}\right)^d, \tag{8}$$

where the probability is taken over the random coins of sampling the edges of the kth vertex. The probability that we output $\perp$ on any iteration, through a union bound is given by, $\sum_{k=1}^{n}\left(\frac{k-1}{n'}\right)^d$. Since $\frac{k-1}{n'} \leq 1/2$, we have the expression is bounded by $n \cdot 2^{-d}$, thus setting $d = \log n + \omega(\log \lambda)$ gives us the required bound. □

In the rogue setting in Section 6, the expression to analyze in this scenario depends on the number of queries made by the adversary. Let this be denoted by $q = q(\lambda)$. Observe that in Eq. (8), the probability of choosing an element on the left can be decided by the adversary, and hence, the probability that FindMatch outputs $\perp$ by a union bound is now $\leq \binom{q}{n} \cdot n \cdot 2^{-d}$. Since q can grow with any arbitrarily polyonimal in λ, the degree will have to grow linearly with n, thus worse than the trivial bound.

B Alternative NIZK protocol for pairings

In this section, we show an alternate protocol how to construct a NIZK for showing consistency between our pairing based key homomorphic prf and the key embedded inside a time lock puzzle. The main idea is to use a variant of Schnorr protocol/Chaum Pedersen protocol where the prover proves knowledge of an exponent k in two different groups. One is the pairing group $\mathbb{G}$ of order p on which punctured key computations are performed, and the other is the group $\mathbb{Z}_N$ where $\phi(N)$ is the order of the group. Since p, N are coprime, the construction in Construction 6.3 does not work. Specifically, because of the chinese remainder theorem, if $x = g_1^{k_1} \in \mathbb{G}$ and $y = k_2 \in \mathbb{Z}_N$ where $k_1 \in \mathbb{Z}_p$, then there exists an integer $k \in \mathbb{Z}_{p\cdot N}$ such that $x = g_1^k$ and $y = (k \mod N) \in \mathbb{Z}_N$. In order to ensure that the statement is sound, we restrict the value inside the time locked puzzle to a $k \in \mathbb{Z}_p$ by using the range proof in [TBM+20].

Construction B.1 (Sigma protocol for pairing based KH-PRF and RSA based TLP)**.** Our construction relies on the following primitives:

- A linearly homomorphic TLP scheme, where the TLP is homomorphic in the message and the random coins. Similar to Construction 6.3.
- A group $\mathbb{G}$ with prime order p and generator g_1.

  Additionally for ease of analysis, we assume that $p < \phi(N)$ and $3p^2 < N$ where N is the RSA prime in the TLP scheme from [MT19].
- A special-case NIZK $\Pi_{\text{range}} = (\text{Setup}, \text{Prove}, \text{Verify})$ that proves the plaintext of a time-lock puzzle Z is in range $[-B, B]$. A construction of such a range proof was given by [TBM+20].

We define our interactive 3-round sigma protocol argument $\Pi = (\text{Prove}, \text{Verify})$ for an instance $\chi = \left(\text{pp}, Z, g_1^{x^{i^*}} \in \mathbb{G}, y \in \mathbb{G}\right)$ and witness $\omega = \left(k \in \mathbb{Z}_p, r \in \mathbb{Z}_{N^2}\right)$ such that, $Z = \text{TLP.Gen}(\text{pp}, k^{(0)}; r)$ and $y = \left(g_1^{x^{i^*}}\right)^{k^{(1)}} \in \mathbb{G}$ and $k = k^{(0)} = k^{(1)} \mod p$.

- $\mathsf{Prove}(\chi, \omega)$:
  - Sample randomly, $k' \leftarrow [p^2 \cdot N]$ and $r' \leftarrow [N^2p^2]$.
  - Compute $Z' \leftarrow \mathsf{TLP.Gen}(\mathsf{pp}, k'; r')$, $y' \leftarrow \left(g_1^{x_{i^*}}\right)^{k'} \in \mathbb{G}$.
  - Compute $\left(\pi_{\mathsf{range}}, \pi'_{\mathsf{range}}\right)$ by running range.Prove on Z and Z' respectively with the bound p. The prover sends $(Z', y', \pi_{\mathsf{range}}, \pi'_{\mathsf{range}})$ to the verifier.
  - Receive $c \in \mathbb{Z}_p$ from the verifier.
  - Compute $\hat{k} = k' + c \cdot k \in \mathbb{Z}$, and $\hat{r} = r' + c \cdot r \in \mathbb{Z}$.[15]
  - Send $\left(\hat{k} \in \mathbb{Z}, \hat{r} \in \mathbb{Z}\right)$ to the verifier.
  - Output $\pi = \left(Z', y' \in \mathbb{G}, \pi_{\mathsf{range}}, \pi'_{\mathsf{range}}, \hat{k} \in \mathbb{Z}, \hat{r} \in \mathbb{Z}\right)$ as the proof.
- $\mathsf{Verify}(\chi)$:
  - The verifier recieves information from the prover, verifies the range proof $\left(\pi_{\mathsf{range}}, \pi_{\mathsf{range}}\right)$ and sends a random value $c \in \mathbb{Z}_p$. If range.Verify rejects, then reject.
  - Recieve $(\hat{k} \in \mathbb{Z}, \hat{r} \in \mathbb{Z})$ from the prover, and perform the checks below.
  - Check if $\mathsf{TLP.Gen}(\mathsf{pp}, \hat{k}; \hat{r}) \overset{?}{=} Z' \cdot Z^c$.
  - Check if $\left(g_1^{x_{i^*}}\right)^{\hat{k}} \overset{?}{=} y' \cdot y^c$.
  - If all checks pass, accept, else reject.

Completeness The scheme is complete, because $\mathsf{TLP.Gen}(\mathsf{pp}, \hat{k}; \hat{r}) = \mathsf{TLP.Gen}(\mathsf{pp}, k'; r') \cdot \mathsf{TLP.Gen}(\mathsf{pp}, k; r)^c = Z' \cdot Z^c$ as our time lock puzzle is linearly homomorphic in the puzzle and the random coins. Similarly, it's easy to check that the second condition holds true i.e. $\left(g_1^{x_{i^*}}\right)^{\hat{k}} = \left(g_1^{x_{i^*}}\right)^{k'} \cdot \left(g_1^{x_{i^*} \cdot k}\right)^{c} = y' \cdot y^c$. Additionally, we rely on the completeness of our range proof.

Soundness We argue statistical soundness of our scheme, i.e. if a verifier accepts a proof, then the statement is in the language, i.e. there exists some witnesses $k \in \mathbb{Z}_p, r \in \mathbb{Z}_{N^2}$ that agree with the statement. Let's assume that Verify accepts statement $\chi = \left(\mathsf{pp}, Z, g_1^{x_{i^*}} \in \mathbb{G}, y \in \mathbb{G}\right)$ and outputs a proof $\pi = \left(Z', y' \in \mathbb{G}, \pi_{\mathsf{range}}, \pi'_{\mathsf{range}}, \hat{k} \in \mathbb{Z}, \hat{r} \in \mathbb{Z}\right)$ such that the verifier accepts on a random input $c \in \mathbb{Z}_p$. Without loss of generality, we can assume that $y' = g^{k'_1} \in \mathbb{G}$, $y = g^{k_1} \in \mathbb{G}$ for some $k'_1, k_1 \in \mathbb{Z}_p$. Similarly, we can expand the time lock puzzle, and assume $Z' = \left(g^{r'_0} \mod N, h^{r'_1 \cdot N} \cdot (1+N)^{k'_0} \mod N^2\right)$, $Z = \left(g^{r_0} \mod N, h^{r_1 \cdot N} \cdot (1+N)^{k_0} \mod N^2\right)$ where $k'_0, k_0 \in \mathbb{Z}_N$, and $r'_1, r_1, r'_0, r_0 \in \mathbb{Z}_{\phi(N)}$. Since the proof is maliciously generated, it is possible that these values are all different and maliciously generated.

Since the range proof is sound, we can conclude that $k'_0, k_0 \in [-p, p]$. Since Verify accepts, we have,

[15] For each value (c, r, r') in $\mathbb{Z}_q$ for some q, the prover considers them as positive integers by setting the output in $1, \ldots, q$ and treating them as integers.

- $\left(g_1^{x^{i^*}}\right)^{\hat{k}} = y' \cdot y^c$. Thus, $\hat{k} = k_1' + c \cdot k_1 \mod p$. Let α be some integer, we have, $\hat{k} = k_1' + c \cdot k_1 + \alpha \cdot p$. Since $\hat{k}, k_1'$ are between $[-p, p]$, we have that $\alpha \in [-p-1, p+1]$.

- $\mathsf{TLP.Gen}(\mathsf{pp}, \hat{k}; \hat{r}) = Z' \cdot Z^c$.

  We have, $g^{\hat{r}} = g^{r_0' + c \cdot r_0} \mod N$, thus, $\hat{r} = r_0' + c \cdot r_0 \mod \phi(N)$.

  Finally, $h^{\hat{r} \cdot N} \cdot (1+N)^{\hat{k}} = h^{r_1' + c \cdot r_1} \cdot (1+N)^{k_0' + c \cdot k_0} \mod N^2$. Plugging in our expression for $\hat{r}$ from the previous evaluation, and analyzing the expression modulo N, $h^{\left((r_0' - r_1') + c(r_0 - r_1)\right) \cdot N} = 1 \mod N$. Since r_0, r_1, r_0', r_1' are all output by the prover in the first message, and $N, \phi(N)$ are coprime to each other. The expression holds true if $c = (r_1' - r_0') \cdot (r_0 - r_1)^{-1} \mod \phi(N)$. Since $p < \phi(N)$, this happens only with probability $\leq 1/p$, which is negligible. Thus $r_1' = r_0' \mod \phi(N)$ and $r_0 = r_1 \mod \phi(N)$.

  Simplifying, we have $N \cdot \hat{k} = N \cdot (k_0' + c \cdot k_0) \mod N^2$. Plugging in our expression for $\hat{k}$, $(k_1' - k_0') + c \cdot (k_1 - k_0) + \alpha \cdot p = 0 \mod N$. Note that k_0, k_1, k_0', k_1 are all small and between $[-p, p]$. Thus if $N > 3p^2$, then, $(k_1' - k_0') + c \cdot (k_1 - k_0) + \alpha \cdot p = 0 \in \mathbb{Z}$. Thus $(k_1' - k_0') + c \cdot (k_1 - k_0) = 0 \mod \mathbb{Z}_p$, and we have that $k_1' = k_0' \mod p$ and $k_0 = k_1 \mod p$ with probability $1 - 1/p$.

Combining the equalities, we have proved that there exists $r \in \mathbb{Z}_{\phi(N)} \in \mathbb{Z}_{N^2}$ such that $r = r_0 = r_1 \mod \phi(N)$, and there exists $k \in \mathbb{Z}_p$ such that $k = k_1 = k_0 \mod p$ where $Z = \mathsf{TLP.Gen}(\mathsf{pp}, k_0; r)$ and $y = \left(g_1^{x^{i^*}}\right)^{k_1}$.

Zero Knowledge We prove the honest verifier zero knowledge of the interactive protocol. The simulator given instance χ computes the transcript in the following order.

- Sample $\tilde{k} \leftarrow [p^2 \cdot N]$ and $\tilde{r} \leftarrow [N^2 p^2]$. Sample $c \leftarrow \mathbb{Z}_p$.

- Compute $\tilde{y} = \frac{\left(g_1^{x^{i^*}}\right)^{\tilde{k}}}{y^c} \in \mathbb{G}$ and compute $\tilde{Z} \leftarrow \mathsf{TLP.Gen}(\mathsf{pp}, \tilde{k}, \tilde{r})$ and $Z' \leftarrow \frac{\tilde{Z}}{Z^c}$.

- The simulator outputs the transcript $\left(Z', y', c, \tilde{k}, \tilde{r}\right)$.

Observe that (1) $\tilde{k}$ is statistically close to $k' + c \cdot k$ because $\tilde{k}, k'$ are sampled randomly from $[p^2 \cdot N]$. Since $c \cdot k$ is small, i.e less than equal to $p \cdot N$, the distributions are apart with a distance $\frac{1}{p}$. (2) $\tilde{r}$ is distributed statistically close to $r' + c \cdot r$ because $\tilde{r}$ and r' are both sampled uniformly from $[N^2 p^2]$. Since $c \cdot r$ is small, i.e. $\leq N^2 p$, the distributions are apart with a distance $\leq \frac{N^2 p}{N^2 p^2} = \mathsf{negl}$.

Remark B.2 (Collapsing rounds)**.** We can collapse rounds to generate a NIZK schene by computing the challenge c using a random oracle and using the standard Fiat-Shamir transformation for sigma protocols, [FS86]. Since the first round message is already non-interactive, we need not collapse with our sigma protocol, and can just attach it separately.