Time-Lock Puzzles with Efficient Batch Solving

Jesko Dujmovic¹, Rachit Garg², and Giulio Malavolta³

¹Helmholtz Center for Information Security (CISPA) and Saarbrücken Graduate School of Computer Science Email:jesko.dujmovic@cispa.de ²UT Austin Email:rachg96@cs.utexas.edu ³Bocconi 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, in many cases, 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 keyhomomorphic 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	1
	1.1 Our Results Out Out <th< td=""><td>1</td></th<>	1
	1.2 Technical Overview	3
	1.3 Related Work	8
	1.4 Open Questions	8
2	Preliminaries	8
	2.1 Puncturable Pseudorandom Functions.	9
	2.2 Time-Lock Puzzles	10
	2.3 Cryptographic Groups	14
	2.4 Lattice Preliminaries	15
3	Time-Lock Puzzles with Batch Solving	15
4	Removing Coordination among Parties	19
5	Puncturable Key-Homomorphic PRFs	21
	5.1 Bounded Domain Puncturable Key-Homomorphic PRFs from Pairings	21
	5.2 (Almost) Key-Homomorphic Puncturable PRF from LWE	24
6	Rogue Puzzle Attacks	31
	6.1 Constructions	32
	6.2 An Efficient NIZK Protocol	34
7	Implementation and Evaluation	37
	7.1 Implementation and Experimental Setup	38
	7.2 Benchmarks	40
8	Acknowledgments	41
A	Analyzing alternative algorithms for matching	48
B	Alternative NIZK protocol for pairings	49
С	Formal proof for Construction 3.1	52
D	Formal construction and proof for rogue security	54

1 Introduction

A Time-Lock Puzzle (TLP) is a cryptographic primitive that allows one to hide a message for a pre-determined amount of time (denoted by *T*). TLPs possess two essential characteristics *efficency* and *sequentiality*. Efficiency requires that creating 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 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], or frontrunning prevention in cryptocurrencies [CDN20], 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.

¹https://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]. The computational cost in evaluating the KH-PRF is dominated by 3λ matrix multiplications. 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. In terms of communication, for batching 7000 puzzles, we only transmit a total size of 40 MB. We also discuss different tradeoffs between communication size and computational time 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 we 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 homomorphically 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 PRF(k, i^*). Furthermore, the PRF must be key homomorphic in the sense that for any two keys k₀ and k₁ 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

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

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

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

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

$$c_{i} + \sum_{j \neq i} \mathsf{PRF}(\mathsf{k}_{j}^{*}, i) - \mathsf{PRF}(\tilde{k}, i) = \mathsf{PRF}(\mathsf{k}_{i}, i) + m_{i} + \sum_{j \neq i} \mathsf{PRF}(\mathsf{k}_{j}^{*}, i) - \mathsf{PRF}(\tilde{k}, i)$$
$$= \mathsf{PRF}(\mathsf{k}_{i}, i) + m_{i} + \sum_{j \neq i} \mathsf{PRF}(\mathsf{k}_{j}, i) - \mathsf{PRF}\left(\sum_{i} \mathsf{k}_{i}, i\right)$$
$$\approx \mathsf{PRF}(\mathsf{k}_{i}, i) + m_{i} + \sum_{j \neq i} \mathsf{PRF}(\mathsf{k}_{j}, i) - \sum_{i} \mathsf{PRF}(\mathsf{k}_{i}, i)$$
$$= m_{i}$$

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 $S \subseteq [n]$ come together to batch a puzzle, each puzzle $i \in S$ 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 S come together, then the probability for any two parties to have a collision in their random sampling is $\leq |S|^2/n$. Since n is exponential, we only fail with negligible probability. Unfortunately this trivial solution fails when our scheme can only 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 $|V| = n_{\text{new}}$. Instead of sampling a single random index in the trivial solution, we assume that each party on the left samples d numbers randomly in $[n_{\text{new}}]$. 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 dwhere 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 $n_{\text{new}} \ge 3 \cdot n$ and $d = \lambda/\log(n_{\text{new}})$. Hall's marriage theorem states that for every subset $X \subseteq S$, there exists a perfect matching if $|\Gamma(X)| \ge |X|$, where $\Gamma(X)$ denotes the set of neighbouring vertices to 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 $\{A_i\}$, they show how to compute a new matrix A_F for some circuit *F*. Additionally, given LWE samples $\{s^TA_i + x_iG + e_i^T\}_{i \in [\ell]}$ over the modulus *q* for some $x = (x_1, \ldots, x_\ell)$, they give an algorithm to compute $s^TA_F + F(x)G + e^T$ for some small *e* and the gadget matrix *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 **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 **s** at point x^* computes,

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

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,

PuncturedEval(k*, x) =
$$\lfloor (\mathbf{s}^T (\mathbf{A}_{EQ_x} + EQ_x(x^*)\mathbf{G}) + \mathbf{e}^T)G^{-1}(\mathbf{D}) \rfloor_p$$

= $\lfloor \mathbf{s}^T \mathbf{A}_{EQ_x}G^{-1}(\mathbf{D}) + \mathbf{e}^TG^{-1}(\mathbf{D}) \rfloor_p$
= $\lfloor \mathbf{s}^T \mathbf{A}_{EQ_x}G^{-1}(\mathbf{D}) \rfloor_p + \{-1, 0, 1\}^m$

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

$$\mathbf{s}^T \mathbf{A}_{EO_x} G^{-1}(\mathbf{D}) + \mathbf{s}^T \mathbf{D} + \mathbf{e}^T G^{-1}(\mathbf{D})$$
(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. $PRF(s, x) + PRF(s', x) = PRF(s + s') + \{-1, 0, 1\}^m$), 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. We solve this by modifying the syntax of a PRF to sample a key such that we know the point at which the PRF will be punctured and evaluated.

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

$$PRF(k, i) = H(i)^{k}$$
 and $H(i)^{k_0} \cdot H(i)^{k_1} = H(i)^{k_0+k_1}$.

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 $n = poly(\lambda)$, we can precompute in the setup n group elements

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

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:

$$e(q,q)^{z_i\cdot k_0} \cdot e(q,q)^{z_i\cdot k_1} = e(q,q)^{z_i\cdot (k_0+k_1)}.$$

This construction is puncturable, and a punctured key, and a punctured key $k_{i^*}^*$ can be computed as $g^{x_{i^*} \cdot k}$. 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 roguepuzzle 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 adversary².

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 similar "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 $f(\lambda) = \operatorname{negl}(\lambda)$. We write

poly(λ) 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 *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. 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 $\max_{i \in [n]}(|a_i|)$. 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 $\mathcal{X} = \{\mathcal{X}_{\lambda,n}\}_{\lambda,n \in \mathbb{N}}$ and range $\mathcal{Y} = \{\mathcal{Y}_{\lambda}\}_{\lambda \in \mathbb{N}}$, consists of a tuple of PPT algorithms $\Pi_{\mathsf{PRF}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Puncture}, \mathsf{PRF}, \mathsf{PuncturedEval})$ defined as follows.

- pp ← Setup(1^λ, 1ⁿ) a probabilistic algorithm that takes as input the security parameter λ, domain index n and outputs public parameters pp.
- k ← KeyGen(pp) a probabilistic algorithm that takes in the public parameters and outputs a key k ∈ K_λ.
- $k^* \leftarrow \text{Puncture}(\text{pp}, k, i^*)$ a probabilistic algorithm that takes as input a key $k \in \mathcal{K}_{\lambda}$ 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 $i \in \chi_{\lambda,n}$ and returns a string y.
- $y \leftarrow \text{PuncturedEval}(\text{pp}, k^*, i^*, i)$ a deterministic algorithm that takes as input a punctured key k^* , a punctured index $i^* \in X_{\lambda,n}$, an index $i \in X_{\lambda,n}$ 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 for all $\lambda, n \in \mathbb{N}$, it holds that,

$$\Pr\left[\exists i^* \neq i \in X_{\lambda,n}, \mathsf{PRF}(\mathsf{pp},\mathsf{k},i) \neq \mathsf{PuncturedEval}(\mathsf{pp},\mathsf{Puncture}(\mathsf{pp},\mathsf{k},i^*),i^*,i) : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda},1^n) \\ \mathsf{k} \leftarrow \mathsf{KeyGen}(\mathsf{pp}) \end{array}\right]$$

is 0, where the probability is over the random coins of Setup, KeyGen and Puncture.

We say that a scheme is *almost* functionality preserving if for all $\lambda, n \in \mathbb{N}$, it holds that,

$$\Pr\left[\exists i^* \neq i \in X_{\lambda,n}, \|\mathsf{PRF}(\mathsf{pp},\mathsf{k},i) - \mathsf{PuncturedEval}(\mathsf{pp},\mathsf{Puncture}(\mathsf{pp},\mathsf{k},i^*),i^*,i)\|_{\infty} > 1: \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda},1^n) \\ \mathsf{k} \leftarrow \mathsf{KeyGen}(\mathsf{pp}) \end{array}\right]$$

is 0, where the probability is over the random coins of Setup, KeyGen and Puncture.

- Security: For a bit $b \in \{0, 1\}$, security parameter λ , we define the following security game between an adversary \mathcal{A} and a challenger as follows:
 - Adversary \mathcal{A} outputs the bound on the domain of the PRF 1^{*n*}.
 - Challenger outputs the public parameters $pp \leftarrow \text{Setup}(1^{\lambda}, 1^{n})$.
 - Adversary \mathcal{A} sees the public parameters and outputs an index $i^* \in [n]$.
 - Challenger samples a key $k \leftarrow \text{KeyGen}(pp)$, and punctures the key $k^* \leftarrow \text{Puncture}(pp, k, i^*)$.
 - If b = 0, the challenger computes $y \leftarrow \mathcal{Y}_{\lambda}$, else if b = 1, it computes $y \leftarrow \mathsf{PRF}(\mathsf{pp},\mathsf{k},i^*)$.
 - Adversary receives the punctured key k^* , and the computed value y and outputs a bit b', which is the output of the experiment.

We say that Π_{PRF} is secure if for any polynomially bounded adversaries $\mathcal{A} = \{\mathcal{A}_{\lambda}\}_{\lambda \in \mathbb{N}}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$, it holds that, $|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \le \mathsf{negl}(\lambda)$ in the game above.

(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 $\mathcal{X} = {\mathcal{X}_{\lambda,n}}_{\lambda,n \in \mathbb{N}}$ 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 $k_0, k_1 \in \mathcal{K}_{\lambda}$, 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 $k_0, k_1 \in \mathcal{K}_{\lambda}$, all indices $i \in \mathcal{X}_{\lambda,n}$, it holds that,

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

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 message space $\{\mathbb{S}_{\lambda}\}_{\lambda \in \mathbb{N}}$ is a tuple of three algorithms $\Pi_{\mathsf{TLP}} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sol})$ defined as follows:

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

$$Sol(pp, Gen(pp, s)) = s.$$

- Security: For a bit $b \in \{0, 1\}$, security parameter λ , polynomially bounded function $T(\cdot)$, we define the following security game between an adversary $(\mathcal{A}_1, \mathcal{A}_2)$ and a challenger as follows:
 - Challenger samples pp \leftarrow Setup $(1^{\lambda}, T(\lambda))$ and sends it to \mathcal{A}_1 .
 - \mathcal{A}_1 receives the public parameters and outputs state st, messages $s_0, s_1 \in \mathbb{S}_{\lambda}$.
 - Challenger computes $Z \leftarrow \text{Gen}(\text{pp}, s_b)$ and sends it to \mathcal{R}_2 .
 - \mathcal{A}_2 receives pp, challenge Z and state st and outputs a bit b', which is the output of the experiment.

We say Π_{TLP} is secure with gap $\varepsilon \in (0, 1)$, if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomially bounded functions where $T(\cdot) \geq \tilde{T}(\cdot)$, 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 $negl(\cdot)$, such that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \le \operatorname{negl}(\lambda)$$

in the game above.

- 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 $s \in \mathbb{S}_{\lambda}$, pp \leftarrow Setup $(1^{\lambda}, T)$, it holds that, $(--, -)) < p_{1}(\lambda \log |\mathbb{S}_{1}| \log T).$

Time (Gen(pp, s))
$$\leq p_1(\lambda, \log |\mathbb{S}_{\lambda}|, \log T)$$

(b) There exists a polynomial $p_2(\cdot, \cdot, \cdot)$ such that for all $\lambda, T \in \mathbb{N}$, pp \leftarrow Setup $(1^{\lambda}, T)$, inputs $s \in \mathbb{S}_{\lambda}$, it holds that,

 $\Pr\left[\text{Time}\left(\text{Sol}(pp, Z)\right) \le p_2(\lambda, \log |\mathbb{S}_{\lambda}|, T) : Z \leftarrow \text{Gen}(pp, s)\right] = 1,$

where the probability is taken over the coins of Gen.

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 Π_{hTLP} = (Setup, Gen, Sol, Eval) is homomorphic for the circuit family, $C = \{C_{\lambda,n}\}_{\lambda,n\in\mathbb{N}}$ and message space $\{\mathbb{S}_{\lambda}\}_{\lambda\in\mathbb{N}}$, if the syntax is augmented with the following algorithm:

• $Z' \leftarrow \text{Eval}(C, \text{pp}, Z_1, \dots, Z_n)$ a probabilistic algorithm that takes as input a circuit $C \in C_{\lambda,n}$ where $C: \mathbb{S}^n_{\lambda} \to \mathbb{S}_{\lambda}$, 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 $(s_1, \ldots, s_n) \in \mathbb{S}^n_{\lambda}$, for all $pp \leftarrow \text{Setup}(1^{\lambda}, T)$, it holds that,

 $\Pr\left[\mathsf{Sol}(\mathsf{pp},\mathsf{Eval}(C,\mathsf{pp},Z_1,\ldots,Z_n))=C(s_1,\ldots,s_n): \forall i\in[n], Z_i\leftarrow\mathsf{Gen}(\mathsf{pp},s_i)\right]=1,$

where the probability is taken over the coins of Gen.

• 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 $(s_1, \ldots, s_n) \in \mathbb{S}^n_{\lambda}$, pp \leftarrow Setup $(1^{\lambda}, T)$, it holds that,

 $\Pr\left[\text{Time}\left(\text{Eval}(C, \text{pp}, Z_1, \dots, Z_n)\right) \le p_1(\lambda, |C|, \log T): \forall i \in [n], Z_i \leftarrow \text{Gen}(\text{pp}, s_i)\right] = 1,$

where the probability is over the coins of Gen.

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 Hard Subgroup Membership (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 $\Pi_{batchTLP} = (Setup, Gen, BatchSol)$ supports *batch solving* with message space $\{\mathbb{S}_{\lambda}\}_{\lambda \in \mathbb{N}}$, if the syntax is augmented with the following algorithm:

- pp ← Setup(1^λ, *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.
- $Z \leftarrow \text{Gen}(\text{pp}, s)$ a probabilistic algorithm that takes as input public parameters pp, and a message $s \in \mathbb{S}_{\lambda}$, and outputs a puzzle *Z*. *Z* and outputs a message *s*.
- $\{(s_i, Z_i)\}_{i \in S} \leftarrow \text{BatchSol}(\text{pp}, \{Z_i\}_{i \in S})$ a deterministic algorithm that takes as input the combined public parameters pp, a set $S \subseteq [n]$ of puzzles Z_i and outputs for each puzzle a message $s_i \in \mathbb{S}_{\lambda}$.³

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

³Note that Sol is equivalent to running BatchSol on one index.

Batch solving correctness: We say Π_{batchTLP} satisfies batch solving correctness if for all *T*, *n* ∈ N, any subset S ⊆ [*n*], all messages s_i ∈ S_λ, all pp ← Setup(1^λ, *T*, *n*), there exists a negligible function negl(·), such that, for all λ ∈ N, it holds that,

 $\Pr\left[\mathsf{BatchSol}(\mathsf{pp}, \{Z_i\}_{i \in \mathcal{S}}) \neq \{(s_i, Z_i)\}_{i \in \mathcal{S}} : \forall i \in \mathcal{S}, Z_i \leftarrow \mathsf{Gen}(\mathsf{pp}, s_i) \right] \leq \mathsf{negl}(\lambda)$

where the probability is taken over the random coins of Gen.

• 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 $S \subseteq [n]$, for all messages $(s_1, \ldots, s_n) \in \mathbb{S}^n_{\lambda}$, all pp \leftarrow Setup $(1^{\lambda}, T, n)$, it holds that,

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

Definition 2.8 (Batching TLPs with unbounded number of parties). We say that our batched time lock puzzle scheme $\Pi_{\text{cobatchTLP}}$ supports an arbitrary polynomial number of parties if the algorithms Gen, Sol in Definition 2.7 run in time poly(λ , 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 $2^{\text{poly}(\lambda)}$ 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 BatchSol can depend on the size of the elements being batched i.e. |S|, and thus, run in time $p_1(\lambda, \log |\mathbb{S}_{\lambda}|, T) + p_2(\lambda, \log |\mathbb{S}_{\lambda}|, \log T, |S|)$.

In our work, we define a notion of TLPs with coordination. This is a relaxation of the general definition of TLPs with batch solving where each puzzle is associated with an index and you can batch puzzles of different indices. The definitions are straightforward modifications to the original algorithms.

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

- pp ← Setup(1^λ, *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.
- $Z \leftarrow \text{Gen}(\text{pp}, i, s)$ a probabilistic algorithm that takes as input public parameters pp, party index $i \in [n]$ and a message $s \in S_{\lambda}$, and outputs a puzzle Z.
- {(*i*, *s_i*)}_{*i*∈S} ← BatchSol(pp, S, {(*i*, *Z_i*)}_{*i*∈S}) a deterministic algorithm that takes as input the public parameters pp, a set S ⊆ [*n*], puzzles *Z_i* from each party *i* ∈ S, and outputs for each party *i* ∈ S, messages *s_i* ∈ S_λ.

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: For a bit b ∈ {0, 1}, security parameter λ, polynomially bounded function T(·), we define the following security game between an adversary (A₁, A₂) and a challenger as follows:

- Adversary \mathcal{A}_1 outputs a bound on the number of puzzles to batch $n(\lambda)$.
- Challenger samples $pp \leftarrow \text{Setup}(1^{\lambda}, T(\lambda), n(\lambda))$ and sends it to \mathcal{A}_1 .
- \mathcal{A}_1 receives the public parameters and outputs state st, messages $s_0, s_1 \in \mathbb{S}_\lambda$ and index $i \in [n]$.
- Challenger computes $Z \leftarrow \text{Gen}(\text{pp}, i, s_b)$ and sends it to \mathcal{A}_2 .
- \mathcal{A}_2 recieves pp, challenge Z and state st and outputs a bit b', which is the output of the experiment.

We say $\Pi_{\text{cobatchTLP}}$ is secure with gap $\varepsilon \in (0, 1)$, if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomially bounded functions where $T(\cdot) \geq \tilde{T}(\cdot)$, 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 negl(\cdot), such that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \le \operatorname{negl}(\lambda)$$

in the game above.

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 $\mathbf{h} = g^x$ 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 $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, g, e) \leftarrow \text{GroupGen}(1^{\lambda})$ 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 (Decisional Bilinear Diffie-Hellman (with $g^{1/x}$)⁴). Let GroupGen be a bilinear group generator. The decisional 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 $(p, \mathbb{G}, \mathbb{G}_T, g, e) \leftarrow \text{GroupGen}(1^{\lambda})$ and $(x, y, z, r) \leftarrow \mathbb{Z}_p^*$.

Assumption 2.12 (Decisional *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:

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

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

⁴When compared with the standard Decisional Bilinear Diffie-Hellman assumption, we additionally need a $g^{1/x}$ group element for our reduction. We do not introduce a new name as there is a lack of a naming convention for these assumptions.

[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 our constructions can be easily adapted to the asymmetric settings. Since we implement the asymmetric setting of Assumption 2.12, we restate the precise assumption below for completeness.

Assumption 2.13 (Asymmetric Decisional *n*-Power Diffie-Hellman). Let GroupGen be an asymmetric bilinear group generator. The asymmetric *n*-power Diffie-Hellman problem is hard for GroupGen if the following distributions are computationally indistinguishable:

$$\left(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, \left\{ g_1^{x^i} \right\}_{i \in [n]}, \left\{ g_2^{x^i} \right\}_{i \in [2n] \setminus \{n+1\}}, g_1^y, e(g_1, g_2)^{x^{n+1}y} \right)$$

$$\approx \left(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, \left\{ g_1^{x^i} \right\}_{i \in [n]}, \left\{ g_2^{x^i} \right\}_{i \in [2n] \setminus \{n+1\}}, g_1^y, e(g, g)^r \right)$$

where $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow \text{GroupGen}(1^{\lambda}) \text{ and } (x, y, r) \leftarrow \mathbb{Z}_p^*$.

2.4 Lattice Preliminaries

Assumption 2.14 (Learning With Errors [Reg05]). Let λ be the security parameter and $n = n(\lambda)$, $m = m(\lambda)$, $q = q(\lambda)$ be integers. Then the decisional Learning With Errors (LWE) assumption states that if we 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 assumption is hard if $-(\mathbf{A}, \mathbf{s}^T \mathbf{A} + \mathbf{e}^T) \approx_c (\mathbf{A}, \mathbf{r})$.

Lemma 2.15 (Leftover hash lemma [Reg05]). Let $n, q, m \in \mathbb{N}$ be natural numbers such that $m > (n + 1) \cdot \log q + \omega(\log n)$, then for any polynomial k = k(n), the following two distributions are computationally indistinguishable,

$$\left\{ (\mathbf{A}, \mathbf{A}\mathbf{R}) : \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{R} \leftarrow \{0, 1\}^{m \times k} \right\} \approx \left\{ (\mathbf{A}, \mathbf{S}) : \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{S} \leftarrow \mathbb{Z}_q^{n \times k} \right\}.$$

Gadget Matrix We call $\mathbf{g} = (2^0, 2^1, \dots, 2^{\lceil \log(q) \rceil})$ 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 When we use $\lfloor \mathbf{a} \rfloor_p$ 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 $i \in [n]$ (see Definition 2.10 for a formal definition), where *n* denotes a bound on the total number of parties. Consequently, we will modify the syntax of time-lock puzzles to add *i* to the puzzle generation algorithm Gen(pp, *i*, *m*) and we will assume that such an index is known to the puzzle solver. This relaxation will be removed through a generic transformation in Section 4.

We proceed by presenting our construction. We set the parameters of the scheme below and then instantiate it based on the required cryptographic building blocks.

- Bound on the maximum number of puzzles batched denoted by *n*. Note that setting $n = \lambda^{\omega(1)}$ allows our time-lock puzzle to support an unbounded number of parties.
- Integer *p* such that, p > 8n and integer *N* such that, $n \cdot p^{2\ell} < N^5$.

We mention the required cryptographic primitives below.

- A time-lock puzzle (denoted by Π_{TLP}) that is linearly homomorphic over \mathbb{Z}_N . Let (LHP.Setup, LHP.Gen, LHP.Eval, LHP.Sol) denote the algorithms of a linearly homomorphic puzzle. Algorithm LHP.Eval supports linear homomorphism, i.e. there exists an efficient summation circuit Σ that takes in inputs $s_1, \ldots, s_n \in \mathbb{Z}_N$ and computes $s_1 + \ldots + s_n \in \mathbb{Z}_N$. Thus, when LHP.Eval takes input Σ , public parameters pp and different puzzles Z_1, \ldots, Z_n , it can homomorphically add Z_1, \ldots, Z_n according to definition Definition 2.4.
- A puncturable almost key-homomorphic PRF (denoted by Π_{PRF}) with domain [*n*], additive key homomorphism over \mathbb{Z}_p^{ℓ} , where $\ell = \text{poly}(\lambda)$. The range of the PRF can be any additive group, for simplicity here we consider the range to be in \mathbb{Z}_p . Let (PRF.Setup, PRF.KeyGen, PRF, PRF.Puncture, PRF.PuncturedEval) denote the algorithms of a key-homomorphic puncturable PRF.

For notational convenience, we define the integer encoding and decoding algorithms, $\text{Encode}_{p,\ell} : \mathbb{Z}_p^{\ell} \to \mathbb{Z}_N$ and $\text{Decode}_{p,\ell} : \mathbb{Z}_N \to \mathbb{Z}_p^{\ell}$ as,

- Encode outputs an integer by computing, $\text{Encode}_{p,\ell}(x_1, \ldots, x_\ell) = \sum_{i=1}^{\ell} p^{2(i-1)} x_i$. Since the output of encode is less than *N* (due to how we set our parameters), the output of encode can be interpreted as an integer \mathbb{Z}_N .
- The decoding algorithm $\text{Decode}_{p,\ell}(y)$ is the reverse operation, i.e., vectorizing an integer by modular reduction and rounding. We can compute it by running the following algorithm. Interpret $y \in \mathbb{Z}$. Set i = 1. Loop till $i = \ell$.
 - Compute $x'_i = y \mod p^2$ and set $x_i = x'_i \mod p$.
 - Reset $y = (y x'_i)/p^2$ and increase *i* by 1.

Ourput (x_1, \ldots, x_ℓ) .

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.

- Setup $(1^{\lambda}, T, n)$:
 - $pp_{LHP} \leftarrow LHP.Setup(1^{\lambda}, T)$
 - $pp_{PRF} \leftarrow PRF.Setup(1^{\lambda}, n)$
 - Return $pp = (pp_{LHP}, pp_{PRF})$
- Gen(pp, *i*, *m*):

⁵These parameters have some slack which can be optimized for deploying in real-world systems.

- Sample a PRF key $k \leftarrow \mathbb{Z}_p^{\ell}$
- Time-lock the key by computing $Z \leftarrow LHP.Gen(pp_{LHP}, Encode_{p,\ell}(k))$
- Compute the punctured key $k^* \leftarrow PRF.Puncture(pp_{PRF}, k, i)$
- Mask the message $c \leftarrow \mathsf{PRF}(\mathsf{pp}_{\mathsf{PRF}},\mathsf{k},i) + m \cdot \lceil p/2 \rceil \mod p$.
- Return (i, Z, k^*, c)
- BatchSol $\left(pp, S, \left\{ i, Z_i, k_i^*, c_i \right\}_{i \in S} \right)$:
 - Sum the puzzles $\tilde{Z} \leftarrow LHP.Eval(\Sigma, pp_{LHP}, \{Z_i\}_{i \in S})$, where the evaluation algorithm computes the sum of the puzzles homomorphically
 - Solve the resulting puzzle $\tilde{k} \leftarrow LHP.Sol(pp_{1HP}, \tilde{Z})$
 - Compute $k' \leftarrow \text{Decode}_{p,\ell}(\tilde{k})$
 - For all $i \in S$, compute

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

and set m_i as $\lfloor \mu_i \rceil_{\lceil p/2 \rceil}$ by doing the rounding operation.

Analysis. Before we proceed with the formal analysis, it is worth highlighting that each puzzle consists of a tuple (i, Z_i, 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

Encode_{*p*,*l*}(k) =
$$\sum_{i=1}^{l} p^{(i-1)} k_i \le \sum_{i=1}^{l} p^{2i-1} \le p^{2l} < N$$

where the last inequality holds by how we set our parameters, and hence $\text{Decode}_{p,\ell}(\text{Encode}_{p,\ell}(k)) = k$ 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

$$\tilde{k} = \sum_{j \in S} \text{Encode}_{p,\ell}(k_j) = \sum_{j \in S} \sum_{i=1}^{\ell} p^{2(i-1)} k_{j,i}$$
$$= \sum_{i=1}^{\ell} p^{2(i-1)} \sum_{j \in S} k_{j,i} \le \sum_{i=1}^{\ell} p^{2i-1} \cdot n \le n \cdot p^{2\ell} < N,$$

where the last inequality holds by how we set our parameters. 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 (p is greater than 8n).

Plugging this into the solving equation, we have that

$$\mu_{i} = c_{i} + \sum_{j \in S \setminus \{i\}} PRF.PuncturedEval(pp_{PRF}, k_{j}^{*}, j, i) - PRF(pp_{PRF}, k', i)$$

$$= c_{i} + \sum_{j \in S \setminus \{i\}} PRF.PuncturedEval(pp_{PRF}, k_{j}^{*}, j, i) - PRF\left(pp_{PRF}, \sum_{j \in S} k_{j}, i\right)$$

$$= c_{i} + \sum_{j \in S \setminus \{i\}} PRF.PuncturedEval(pp_{PRF}, k_{j}^{*}, j, i) - \sum_{j \in S} PRF(pp_{PRF}, k_{j}, i) + e$$

$$= c_{i} + \sum_{j \in S \setminus \{i\}} PRF.PuncturedEval(pp_{PRF}, k_{j}, i) - \sum_{j \in S} PRF(pp_{PRF}, k_{j}, i) + e$$

$$= c_{i} - PRF(pp_{PRF}, k_{i}, i) + e + e'$$

$$= PRF(pp_{PRF}, k, i) + m_{i} \cdot \lceil p/2 \rceil - PRF(pp_{PRF}, k_{i}, i) + e + e'$$

$$= m_{i} \cdot \lceil p/2 \rceil + e + e'$$

where the third equality follows by the almost key-homomorphism of the PRF with some $||e||_{\infty} \le (n-1)$, and the fourth equality follows by almost functionality preservation of the PRF with some $||e'||_{\infty} \le (n-1)$. Thus μ_i is correctly rounded to m_i , since 4n < p/2.

Remark 3.4. For correctness, we crucially rely on our setting of parameters. 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 (Informal). Let Π_{LHP} be linearly-homomorphic time-lock puzzle secure against depth $T^{\varepsilon}(\lambda)$ -bounded adversaries and Π_{PRF} be an almost key-homomorphic puncturable PRF, then construction 3.1 is a batchable time-lock puzzle secure against $T^{\varepsilon}(\lambda)$ -bounded adversaries.

Proof. We include an informal proof below. Please see Appendix C for the complete formal theorem statement and reductions. 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., $(i, Z_i, k_i^*, c_i) \leftarrow \text{Gen}(\text{pp}, i, m).$
- H_1 : In this hybrid, we modify the Gen algorithm encode $0 \in \mathbb{Z}_N$ in the time-lock puzzle, as opposed to the key of the pseudorandom function. That is, we define

$$Z \leftarrow LHP.Gen(0)$$

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, $|\operatorname{Adv}_{H_1}(\mathcal{A}) - \operatorname{Adv}_{H_0}(\mathcal{A})| \le \operatorname{negl}(\lambda)$.

 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 PRF(pp_{PRF}, k, *i*) is computationally indistinguishable from uniform, even given the punctured key k*, and therefore so is PRF(pp_{PRF}, k, *i*)+*m*. Thus, $|Adv_{H_2}(\mathcal{A}) - Adv_{H_1}(\mathcal{A})| \le negl(\lambda)$.

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

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. $\Gamma(S) = \{v \in V : \exists (u, v) \in E \land u \in U\}$.

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 $S \subseteq U$, $|\Gamma(S)| \ge |S|$.

Additionally, there are many algorithms to compute the perfect matching. One such algorithm is [HK73], (denoted in this document by 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 $\Pi_{batchTLP}$ below. Our transformation relies on the existence of a $\Pi_{cobatchTLP}$ scheme.

- pp ← Setup(1^λ, *T*, *n*). Let *n*_{new} and *d* be set according to parameters in Lemma 4.4. Sample pp ← cobatchTLP.Setup(1^λ, *T*, *n*_{new}). Output public parameters pp.
- $Z \leftarrow \text{Gen}(\text{pp}, m)$. For every $j \in [d]$, sample *d* choices randomly, i.e. $v_j \leftarrow [n_{\text{new}}]$ (without replacement⁶). Let $V = \{v_j\}_{j \in [d]}$, and we generate a puzzle, i.e. $Z_{v_j} \leftarrow \text{cobatchTLP.Gen}(\text{pp}, v_j, m)$.

⁶This means that we always sample a distinct set.

Output $Z = \left(V, \left\{(v_j, Z_{v_j})\right\}_{v_j \in V}\right).$

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

 $\mathcal{E} = \left\{ (i, v_j) : i \in \mathcal{S}, v_j \in [n_{\text{new}}], \text{ and } v_j \in V_i \right\}.$

Compute the maximal matching map ← FindMatch(G) where the matched vertices are denoted by the set S' and the mapping map = {(i, v_i^{*})}_{i \in S'}. Set S_{new} = {v_i^{*}}_{i \in S'}.

- Let
$$\{(v_i^*, s_i)\}_{v_i^* \in \mathcal{S}_{new}} \leftarrow \text{cobatchTLP.BatchSol}(pp, \mathcal{S}_{new}, \{(v_i^*, Z_{i, v_i^*})\}_{v_i^* \in \mathcal{S}_{new}}).$$

- For the unmatched vertices i.e. $i \in S \setminus S'$, set $s_i = \bot$. Output $\{(s_i, Z_i)\}_{i \in S}$.

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

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

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 BatchSol, our graph $G = (S, [n_{new}], \mathcal{E})$ 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 $\geq 1 - \operatorname{negl}(\lambda)$ 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,

$$\sum_{\ell=d}^{n} \binom{n}{\ell} \binom{n'}{\ell-1} \left(\frac{\binom{\ell-1}{d}}{\binom{n'}{d}}\right)^{\ell}.$$
(2)

By using the inequalities, $\frac{\binom{x}{d}}{\binom{y}{d}} \leq \frac{x \cdot (x-1) \dots (x-d+1)}{y \cdot (y-1) \dots (y-d+1)} \leq \left(\frac{x}{y}\right)^d$, and using the inequality that $\binom{x}{y} \leq \left(\frac{e \cdot x}{y}\right)^y$, the failure probability can be simplified to, $\sum_{\ell=d}^n \left(\frac{e \cdot n}{\ell}\right)^\ell \left(\frac{e \cdot n'}{\ell-1}\right)^{\ell-1} \left(\frac{\ell-1}{n'}\right)^{\ell-d}$. 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, $\frac{d}{d\ell} (f(\ell)) = f(\ell) \cdot ((d-2)(1+\ln \ell) - d\ln n' + \ln a)$. On setting $n' \ge 3n$, and $d \ge 4$ and since $\ell \le n$, the term $\left(\frac{\ell}{n'}\right)^{d \cdot \ell}$ will dominate and we can observe that $\frac{d}{d\cdot \ell} (f(\ell)) < 0$ 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 $d \ge (a + \omega(\log \lambda)) / b$, gives us that the probability of failure is $\le \operatorname{negl}(\lambda)$, 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 $\Pi_{cobatchTLP}$. 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 $\Pi_{cobatchTLP}$.

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

Theorem 4.6 (Security). If $\Pi_{\text{cobatchTLP}}$ 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 $\Pi_{\text{cobatchTLP}}$ 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-homomorphic PRFs from pairings, with different tradeoffs in terms of assumptions and parameter size. Importantly, both of these constructions only support of domain of size $n = poly(\lambda)$.

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

- Setup $(1^{\lambda}, 1^n)$:
 - $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, q, e)$ ← GroupGen (1^{λ})
 - 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 pp = $(\mathcal{G}, \{g^{x_i}\}_{i \in [n]}, \{g^{z_i/x_j}\}_{i \neq j})$
- KeyGen(pp) : Sample $k \in \mathbb{Z}_p^*$.
- PRF(pp, k, *i*):
 - Return $e(g^{z_i/x_j}, g^{x_j})^k = (e(g, g)^{z_i})^k$ for some $j \neq i$
- Puncture(pp, k, *i**):
 - Return $g^{x_{i^*}k} = (g^{x_{i^*}})^k$
- PuncturedEval(pp, k*, *i**, *i*):
 - Return \perp if $i = i^*$
 - Return $e(g, g)^{z_i k} = e(g^{x_{i^*} k}, g^{z_i/x_{i^*}})$

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

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

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

$$\prod_{j} \mathsf{PRF}(\mathsf{pp},\mathsf{k}_{j},i) = \prod_{j} e(g,g)^{z_{i}\mathsf{k}_{j}} = e(g,g)^{z_{i}\sum_{j}\mathsf{k}_{j}} = \mathsf{PRF}\left(\mathsf{pp},\sum_{j}\mathsf{k}_{j},i\right).$$

We now show that the scheme is secure against the decisional bilinear diffie-hellman assumption (with $g^{1/x}$, see Assumption 2.11).

Remark 5.2. When implementing this scheme, we can also post $\{e(g, g)^{z_i}\}_{i \in [n]}$ values as they are publicly computable from the public parameters. This makes the evaluation step faster at the cost of maintaining a bigger list of public parameters. Note that it doesn't change the security assumption under which we will prove our security below.

Theorem 5.3. If Assumption 2.11 holds, then Construction 5.1 satisfies security from Definition 2.1.

Proof. Let \mathcal{A} be an adversary where \mathcal{A} breaks the security of the underlying puncturable PRF with some non-negligible ε' . We construct an adversary \mathcal{B} that breaks Assumption 2.11 as follows.

- Algorithm \mathcal{B} receives $(U = g^x, V = g^{1/x}, W = g^y, Z = g^z, T)$ where *T* is either $e(g, g)^{xyz}$ or $e(g, g)^r$.
- Adversary \mathcal{A} outputs a bound on the domain of the PRF 1^{*n*}.

- Algorithm B samples a random index i^{*} ← [n] and (implicitly) sets x_{i^{*}} = 1/x and z_{i^{*}} = z. More explicitly, it computes the public parameters as follows,
 - For every $i \in [n]$, such that $i \neq i^*$, sample $x_i, z_i \in \mathbb{Z}_p$. Thus we can compute g^{x_i} and g^{z_i} respectively. For every $j \in [n]$, such that $j \neq i$, we divide the analysis into two cases, where $j = i^*$ and when $j \neq i^*$.
 - * When $j = i^*$, compute g^{z_i/x_j} as U^{z_i} .
 - * When $j \neq i^*$, compute g^{z_i/x_j} as we know both exponents.
 - For $i = i^*$, set $g^{x_{i^*}} = V$. For every $j \in [n]$, such that $j \neq i$, compute g^{z_i/x_j} as Z^{1/x_j} . Note that since $j \neq i^*$, we know the required exponents $1/x_j$.

Output pp = $(\mathcal{G}, \{g^{x_i}\}_{i \in [n]}, \{g^{z_i/x_j}\}_{i \neq j})$ to the adversary \mathcal{A} .

- Adversary \mathcal{A} sees the public parameters and outputs an index $i^* \in [n]$. If this index i^* does not match the i^* algorithm \mathcal{B} sampled, then algorithm \mathcal{B} simply aborts. Else, we continue with the execution.
- Algorithm \mathcal{B} implicitly sets $k = y \cdot x$ and sets the punctured key k^* as W. The computed value of the PRF point is set as the target T. It sends (k^*, T) to the adversary \mathcal{A} .
- Algorithm \mathcal{A} outputs a bit b' which is output by \mathcal{B} .

Note that the punctured key is $g^{x_{i^*} \cdot k} = g^{1/x \cdot (y \cdot x)} = g^y$. And the computation at the punctured point is $e(g, g)^{z_{i^*} \cdot k} = e(g, g)^{z \cdot y \cdot x}$. Thus when the target value is $e(g, g)^{xyz}$, we evaluate at the punctured point and when the target value is $e(g, g)^r$, we choose a random point in the range of the PRF evaluation. Since adversary \mathcal{B} chooses i^* randomly, it doesn't abort with probability 1/n and thus \mathcal{B} 's advantage in breaking the security of Assumption 2.11 is ϵ'/n which is non-negligible and we have a contradiction.

Construction 5.4 (Linear Setup). We specify the algorithms $\Pi_{PRF} = (Setup, PRF, Puncture, PuncturedEval) below.$

- Setup $(1^{\lambda}, 1^n)$:
 - $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, g, e)$ ← GroupGen (1^{λ})
 - Sample *x* uniformly at random for \mathbb{Z}_p^*
 - Return pp = $(\mathcal{G}, \{g^{x^i}\}_{i \in [2n] \setminus \{n+1\}})$
- PRF(pp, k, *i*):

- Return
$$e(g,g)^{x^{n+1+i}k} = (e(g,g)^{x^{n+1+i}})^k$$

• Puncture(pp, k, *i**):

- Return
$$q^{x^{i^*}k} = (q^{x^{i^*}})^k$$

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

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

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

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

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

Similar to Remark 5.2, here we can post $\{e(g,g)^{n+1+i}\}_{i \in [n]}$ in the public parameters. We now show that the scheme is secure against the decisional *n*-Power Diffie-Hellman assumption (see Assumption 2.12).

Theorem 5.5. If Assumption 2.12 holds, then construction Construction 5.4 satisfies security from Definition 2.1.

Proof. Let \mathcal{A} be an adversary where \mathcal{A} breaks the security of the underlying puncturable PRF with some non-negligible ε' . We construct an adversary \mathcal{B} that breaks Assumption 2.12 as follows.

- Algorithm \mathcal{B} receives $\left(g, \left\{X_i = g^{x^i}\right\}_{i \in [2n] \setminus \{n+1\}}, h, T\right)$ where *T* is either $e(g, h)^{x^{n+1}}$ or $e(g, g)^r$.
- Adversary \mathcal{A} outputs a bound on the domain of the PRF 1^{*n*}.
- Algorithm \mathcal{B} computes the public parameters as follows pp = $(\mathcal{G}, \{X_i\}_{i \in [2n] \setminus \{n+1\}})$ and outputs pp to adversary \mathcal{A} .
- Adversary \mathcal{A} sees the public parameters and outputs an index $i^* \in [n]$.
- Algorithm \mathcal{B} implicitly sets $k = DLog(h)/x^{i^*}$ and sets the punctured key k^* as *h*. The computed value of the PRF point is set as the target *T*. It sends (k^*, T) to the adversary \mathcal{A} .
- Algorithm \mathcal{A} outputs a bit b' which is output by \mathcal{B} .

Note that the punctured key is $g^{x^{i^*} \cdot k} = g^{\text{DLog}(h)} = h$. And the computation at the punctured point is $e(g, g)^{x^{n+1+i^*} \cdot k} = e(g, g)^{x^{n+1} \cdot \text{DLog}(h)} = e(g, h)^{x^{n+1}}$. Thus when the target value is $e(g, h)^{x^{n+1}}$, we evaluate at the punctured point and when the target value is $e(g, g)^r$, we choose a random point in the range of the PRF evaluation. Thus \mathcal{B} 's advantage in breaking the security of Assumption 2.11 is ε' which is non-negligible and we have a contradiction.

Remark 5.6. Observe that in both of our constructions, pp_{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. However, it provides more functionality and stronger security than we need. In the following sections, we show how to simplify the construction drastically for our security and functionality notions.

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

- ComputeA(F, A_0, \ldots, A_k) : Takes as input a circuit $F : \{0, 1\}^k \to \{0, 1\}$ and k matrices A_0, \ldots, A_k and outputs a matrix A_F .
- Compute C(F, x_1 , ..., x_k , A_0 , ..., A_k , a_0 , ..., a_k) : Takes as input a circuit $F : \{0, 1\}^k \rightarrow \{0, 1\}, k+1$ matrices $\mathbf{A}_0, \dots, \mathbf{A}_k$, a string of k bits x_1, \dots, x_k , and k + 1 vectors $\mathbf{a}_0, \dots, \mathbf{a}_k$ where $\mathbf{a}_i = \mathbf{s}^T (\mathbf{A} + x_i \mathbf{G}) + \mathbf{e}_i$ and $\mathbf{a}_0 = \mathbf{s}^T (\mathbf{A}_0 + \mathbf{G}) + \mathbf{e}_0$. It outputs a vector $\mathbf{a}_{F,x}$ such that $\mathbf{a}_{F,x} = \mathbf{s}^T (\mathbf{A}_F + F(x)\mathbf{G}) + \mathbf{e}_F$ associated with the output matrix A_F and the output bit F(x).

The runtime of both these algorithms is dominated by the matrix multiplication per AND gate and a matrix addition per NOT gate. We mention the formal lemma from [BV15].

Lemma 5.7 (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 $\mathbf{A}_0, \mathbf{A}_1, \dots, \mathbf{A}_k \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{a}_0, \mathbf{a}_1, \dots, \mathbf{a}_k$ be such that $\|\mathbf{a}_i - \mathbf{s}^T \mathbf{A}_i + x_i \mathbf{G}\|_{\infty} \leq \gamma$ for $i \in [k]$ and $-\mathbf{s}^T \mathbf{A}_0 + \mathbf{G}\|_{\infty} \leq \gamma$ for some $\mathbf{s} \in \mathbb{Z}_q^n$ and $\gamma = \gamma(\lambda)$. Let $\mathbf{A}_F \leftarrow \text{ComputeA}(F, \mathbf{A}_0, \dots, \mathbf{A}_k)$ and $\mathbf{a}_{F,x} \leftarrow \text{ComputeC}(F, x, \mathbf{A}_0, \mathbf{A}_1, \dots, \mathbf{A}_k, \mathbf{a}_0, \mathbf{a}_1, \dots, \mathbf{a}_k)$. Then, $\|\mathbf{a}_0 - \mathbf{s}^T \mathbf{A}_0 + \mathbf{G}\|_{\infty} \leq \gamma \text{ for some } \mathbf{s} \in \mathbb{Z}_q^n \text{ and } \gamma = \gamma(\lambda).$

we have that,

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

where E(F) is the noise growth estimation of the circuit F.

Formally, the noise growth estimation is computed as follows, $E(F) = E_F(w_o)$ with w_o being the output wire of F and E_F is a recursive function defined as follows.

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

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 $\|\mathbf{Z}_i\|_{\infty} \le E(F).$

Remark 5.8. Our lemma differs in two places from the lemma presented by Brakerski and Vaikuntanathan. They estimate the noise that accumulates with $m^{depth(F)}$, while we bound it by the recursive function E(F). Secondly, they upper bound $\|\mathbf{Z}_i\|_{\infty}$ by $m^{depth(F)} \cdot \gamma$, while we upper bound by E(F). Both of these changes are derived exactly by following their proof.

In the following we describe a PRF that largely follows the blueprint of the constrained-key keyhomomorphic PRF of [BV15]. Our constructions differ from the construction in [BV15] in the following ways.

• The construction for a constrained PRF in Brakerski and Vaikuntanathan [BV15] (Theorem 5.1) assumes the hardness of decision LWE, 1D-SIS and the existence of an admissable hash function [BB04, CHKP12, BV15].

One-dimensional short integer solutions (1D-SIS) for parameters q, m, t states that given a vector $\mathbf{v} \leftarrow \mathbb{Z}_q^m$, it's hard to find a vector $\mathbf{z} \in \mathbb{Z}^m$ such that $\|\mathbf{z}\| \leq t$ and $\langle \mathbf{v}, \mathbf{z} \rangle \in [-t, t] + q\mathbb{Z}$. Admissible hash functions were introduced in [BB04] to convert selective secure IBE schemes into fully secure schemes. They can be constructed from any collusion resistant hash function.

Our specific application for (almost) key homomorphic PRF's does not need to satisfy the strong adaptive security guarantees provided by the construction of Brakerski and Vaikuntanathan and we only need to rely on LWE security.

 We only need to constrain the key so that it can puncture at a single point. This allows us to consider specialized constraining circuits and replace the universal circuit in the construction of [BV15] by a simpler equality check circuit. We define it below.

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

Observe that the circuit is a big AND of 2λ many clauses $(c_j)_{j \in [2\lambda]}$. The noise growth estimation of each clause, i.e. $E(c_i)$ is less than equal to (2m + 3). From the definition of the noise growth circuit, we can observe that the noise grows slowly if the heavy part of the computation is in the right spline. Thus, we can state our equality check circuit as, $EQ = (c_1 \wedge (c_2 \wedge (c_3 \wedge \dots (c_{2\lambda-1} \wedge c_{2\lambda}) \dots)))$ where $c_1, \dots, c_{2\lambda}$ are the clauses mentioned above and we can get $E(EQ) \leq (2\lambda - 1)m(2m + 3) \leq O(\lambda m^2)$.

- In our applications, we know the point at which we puncture before generating the corresponding key. Thus we can modify the key generation algorithm and allow it to depend on a punctured point. We mention the modifications to Definition 2.1 below (without repeating the entire definition for brevity).
 - Our key generation algorithm, KeyGen(pp, x^*) takes in public parameters pp and a punctured point $x^* \in X_{\lambda,n}$ and outputs a key $k \in \mathcal{K}_{\lambda}$.
 - Functionality preserving only holds when considering the key k to be punctured at point *i**.
 Similarly for almost functionality preserving.
 - For the security definition, the challenger knows the punctured point when generating the key and computes it by running the KeyGen operation. Thus, the interaction with the adversary in the security game stays the same.
 - (Almost) key-homomorphism requires that homomorphism for any $x^* \in X_{\lambda,n}$, and keys k_0, k_1 computed under the support of the key generation operation, the key homomorphism guarantee holds.

Even though our security requirements are similar, we point out that this restricts the applications of our primitive as a generic pseudo-random function. Specifically, we cannot puncture at multiple points in a given set and our construction can only guarantee pseudorandomness at a pre-determined point. Thus our security requirement no longer implies a pseudo-random function where the adversary can make oracle queries to the PRF and a random function at multiple points. This simplification is enough for our application and helps in concrete efficiency and simplifying the assumptions from which we can build our primitive.

We proceed by presenting our construction. We set the parameters of the scheme below and then instantiate it based on the required cryptographic building blocks.

• Let $n = n(\lambda)$, $m = m(\lambda)$, $q = q(\lambda)$ be the LWE parameters. We additionally define parameters $p = p(\lambda)$, $\gamma = \gamma(\lambda)$. Let $\gamma' = \gamma'(\lambda)$ be an upper bound on the noise that accumulates in our construction. Additionally, we set $m = \lceil \log q \rceil \cdot n$ and $\gamma' = (E(EQ) + 1) \cdot m\gamma$ where EQ is the equality check circuit.

- Let $\chi_{\sigma,\gamma}^m$ be the discrete gaussian distribution with parameter σ that is truncated at γ . We write χ_{σ} to denote that the discrete gaussian has not been truncated.
- We construct a puncturable PRF with domain $\{0, 1\}^{\lambda}$, key generation \mathbb{Z}_q^n and output space \mathbb{Z}_p^m .

Construction 5.9 (Almost key-homomorphic PRF from LWE). We describe our algorithms below.

Setup (1^{λ}) :

- Sample $A_0, A_1, \{B_i\}_{i \in [\lambda]}, C, D \leftarrow \mathbb{Z}_q^{n \times m}$ uniformly at random
- Return pp = $({A_\beta}_{\beta \in \{0,1\}}, {B_i}_{i \in [\lambda]}, C, D)$

KeyGen(pp, *x*^{*}):

• Sample $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ uniformly at random. Compute $\mathbf{y} = \mathbf{s}^T (\mathbf{B}_{EQ,x^*} + \mathbf{C}) G^{-1}(\mathbf{D})$. Keep resampling \mathbf{s} until \mathbf{y} has no entry in $[-\gamma', \gamma'] + (q/p)\mathbb{Z}$.

PRF(pp, k = $\mathbf{s} \in \mathbb{Z}_{q}^{n}, x$):

- Let $\mathbf{B}_{EQ,x} \leftarrow \text{ComputeA}(EQ, \mathbf{A}_1, \mathbf{B}_1, \dots, \mathbf{B}_{\lambda}, \mathbf{A}_{x_1}, \dots, \mathbf{A}_{x_{\lambda}})$
- Return $\left[\mathbf{s}^{T}(\mathbf{B}_{EQ,x}+\mathbf{C})G^{-1}(\mathbf{D})\right]_{p}$

Puncture(pp, $k = s \in \mathbb{Z}_q^n, x^*$):

- For each $\beta \in \{0, 1\}$:
 - Sample $\mathbf{e}_{1,\beta} \leftarrow \chi^m_{\sigma,\gamma}$ according to 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 $\mathbf{e}_{2,i} \leftarrow \chi^m_{\sigma,\gamma}$ according to error distribution
 - Let $\mathbf{b}_i = \mathbf{s}^T (\mathbf{B}_i + x_i^* \mathbf{G}) + \mathbf{e}_{2,i}$
- Sample $\mathbf{e}_3 \leftarrow \chi^m_{\sigma, \gamma}$ according to error distribution
- Let $\mathbf{c} = \mathbf{s}^T \mathbf{C} + \mathbf{e}_3$
- Return $k^* = (\{a_\beta\}_{\beta \in \{0,1\}}, \{b_i\}_{i \in [\lambda]}, c)$

PuncturedEval(pp, k^*, x^*, x):

• Compute

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

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

Claim 5.10. On setting $q \ge 2m \cdot (2\gamma' + 1) \cdot p$, and $m = n \cdot \lceil \log q \rceil$, the algorithm KeyGen runs in polynomial time with probability $1 - \operatorname{negl}(\lambda)$.

Proof. Observe that $\mathbf{s}^T \cdot \mathbf{C}$ is a uniform vector in $\mathbb{Z}_q^{1 \times m}$ because both \mathbf{s} and \mathbf{C} are sampled randomly. Since \mathbf{D} is a random matrix in $\mathbb{Z}_q^{n \times m}$, $G^{-1}(\mathbf{D})$ is a uniform matrix in $\{0, 1\}^{m \times m}$, and by leftover hash lemma, if $m > 2 \cdot \log q + \omega(\log \lambda)$, then $\mathbf{s}^T \cdot \mathbf{C} \cdot G^{-1}(\mathbf{D})$ is a uniform vector over $1 \times m$. As we set m appropriately, the total computation, $\mathbf{s}^T(\mathbf{B}_{EQ,x^*} + \mathbf{C})G^{-1}(\mathbf{D})$ is statistically close to uniform. We resample with probability $1 - (1 - (2\gamma' + 1)p/q)^m \le m(2\gamma' + 1)p/q$. Inputting our value of q, we get that we resample with probability $\le 1/2$. Thus, after atmost sampling λ times, the probability that we resample again is $2^{-\lambda}$, which is negligible.

Observe that if q is superpolynomial, then the probability of sampling is negligible and we do not need to resample. Thus when choosing LWE with super-polynomial modulus, our key generation algorithm need not depend on the punctured point x^* .

Theorem 5.11 (Almost Functionality Preserving). If, $q/p > \gamma' = (E(EQ) + 1) \cdot m\gamma$, our construction is almost functionality preserving, i.e. for all λ , for all pp, and inputs $x, x^* \in \{0, 1\}^{\lambda}$, such that $x \neq x^*$, $k \leftarrow \text{KeyGen}(\text{pp}, x^*)$, and setting $k^* \leftarrow \text{Puncture}(\text{pp}, k, x^*)$, we have that,

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

Proof. Let $\mathbf{B}_{EQ,x} \leftarrow \text{ComputeA}(EQ, \mathbf{A}_1, \mathbf{B}_1, \dots, \mathbf{B}_{\lambda}, \mathbf{A}_{x_1}, \dots, \mathbf{A}_{x_{\lambda}})$

$$PRF(pp, k, x) = \left\lfloor \mathbf{s}^{T} (\mathbf{B}_{EQ,x} + \mathbf{C}) G^{-1}(\mathbf{D}) \right\rfloor_{p}$$

$$= \left\lfloor (\mathbf{b}_{EQ,x,x^{*}} + \mathbf{c})^{T} G^{-1}(\mathbf{D}) + \mathbf{e}^{T} \right\rfloor_{p} \qquad \text{for some } \mathbf{e} \text{ with } |\mathbf{e}| \leq E(EQ) \cdot m\gamma$$

$$= \left\lfloor (\mathbf{b}_{EQ,x,x^{*}} + \mathbf{c})^{T} G^{-1}(\mathbf{D}) \right\rfloor_{p} + \{-1, 0, 1\}^{m} \qquad (3)$$

$$= PuncturedEval(pp, k^{*}, x^{*}, x) + \{-1, 0, 1\}^{m}$$

The first equality due to Lemma 5.7, and the second Eq. (3) holds because $|\mathbf{e}| \le (E(EQ) + 1) \cdot m\gamma$ and we choose $q/p > (E(EQ) + 1) \cdot m\gamma$.

Claim 5.12 (Almost Key Homomorphism). For all λ , keys $\mathbf{s}_1, \mathbf{s}_2 \in \mathcal{K}_{\lambda}$, all inputs $x \in \{0, 1\}^{\lambda}$, it holds that,

$$\|\mathsf{PRF}_{pp}(\mathbf{s}_{1} + \mathbf{s}_{2}, x) - (\mathsf{PRF}(pp, \mathbf{s}_{1}, x) + \mathsf{PRF}(pp, \mathbf{s}_{2}, x)))\|_{\infty} \le 1$$

Proof. 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 $\lfloor \mathbf{a} \rfloor_p + \lfloor \mathbf{b} \rfloor_p \leq \lfloor \mathbf{a} + \mathbf{b} \rfloor_p + \mathbf{e}$ where $\mathbf{e} \in \{-1, 0, 1\}^m$.

Remark 5.13. 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 $\mathbf{s}^T(\mathbf{B}_{EO,x^*} + \mathbf{C})G^{-1}(\mathbf{D})$ has no entry in $[-\gamma', \gamma'] + (q/p)\mathbb{Z}$.

Theorem 5.14 (Pseudorandom at Punctured Point). *Our construction above is a secure puncturable prf i.e.* for all polynomially bounded adversaries $\mathcal{A} = \{\mathcal{A}_{\lambda}\}_{\lambda \in \mathbb{N}}$, there exists a negligible function negl(·), such that for all $\lambda \in \mathbb{N}$, it holds that, $|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \le \text{negl}(\lambda)$ in the security game defined in *Definition 2.1*.

Proof. We begin by defining the original security game from Definition 2.1 (our construction can support any polynomial domain PRF and we do not preset the domain bound). For a bit $b \in \{0, 1\}$, security parameter λ , we play the following game between a challenger and an adversary.

- Challenger outputs the public parameters pp ← Setup(1^λ) where pp = ({A_β}_{β∈{0,1}}, {B_i}_{i∈[λ]}, C, D) and all matrices are sampled randomly.
- Adversary \mathcal{A} sees the public parameters and outputs an index $x^* \in \{0, 1\}^{\lambda}$.
- Challenger samples a key $k \leftarrow \text{KeyGen}(pp, x^*)$.
 - Sample $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ uniformly at random. Compute $\mathbf{y} = \mathbf{s}^T (\mathbf{B}_{EQ,x^*} + \mathbf{C}) G^{-1}(\mathbf{D})$. Keep resampling \mathbf{s} until \mathbf{y} has no entry in $[-\gamma', \gamma'] + (q/p)\mathbb{Z}$.

Compute the punctured key $k^* \leftarrow Puncture(pp, k, x^*)$.

- For each $\beta \in \{0, 1\}$:
 - * Sample $\mathbf{e}_{1,\beta} \leftarrow \chi^m_{\sigma,\gamma}$ according to 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 $\mathbf{e}_{2,i} \leftarrow \chi^m_{\sigma, \gamma}$ according to error distribution
 - * Let $\mathbf{b}_{i} = \mathbf{s}^{T}(\mathbf{B}_{i} + x_{i}^{*}\mathbf{G}) + \mathbf{e}_{2,i}$
- Sample $\mathbf{e}_3 \leftarrow \chi^m_{\sigma,\gamma}$ according to error distribution
- Let $\mathbf{c} = \mathbf{s}^T \mathbf{C} + \mathbf{e}_3$
- Set $k^* = (\{a_\beta\}_{\beta \in \{0,1\}}, \{b_i\}_{i \in [\lambda]}, c).$
- If b = 0, the challenger computes $y \leftarrow \mathbb{Z}_p^m$, else if b = 1, it computes $y \leftarrow \mathsf{PRF}(\mathsf{pp},\mathsf{k},x^*)$.
 - Let $\mathbf{B}_{EQ,x^*} \leftarrow \text{ComputeA}(EQ, \mathbf{A}_1, \mathbf{B}_1, \dots, \mathbf{B}_{\lambda}, \mathbf{A}_{x_1^*}, \dots, \mathbf{A}_{x_1^*})$
 - Return $[\mathbf{s}^T(\mathbf{B}_{EQ,x^*} + \mathbf{C})G^{-1}(\mathbf{D})]_n$
- Adversary receives the punctured key k^{*}, and the computed value *y* and outputs a bit *b'*, which is the output of the experiment.

We proceed by providing a security proof through a hybrid argument.

- H_0 : This is the original hybrid, where pp, k, k^{*}, and y are sampled as shown above.
- H_1 : In the first hybrid we change how we sample matrices \mathbf{A}_{β} 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 $\mathbf{B}_i = \hat{\mathbf{B}}_i x^* \mathbf{G}$.
- H_2 : We change how we compute the PRF evaluation *y*. If b = 1, we instead sample *y* in the following manner,
 - Sample $\mathbf{e} \leftarrow \chi^m_{\sigma,\gamma}$ according to error distribution and let $\mathbf{d} = \mathbf{s}^T \mathbf{D} + \mathbf{e}$.

• Compute

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

, and set $y \leftarrow \left\lfloor (\mathbf{b}_{EQ,x^*,x^*} + \mathbf{c})^T G^{-1}(\mathbf{D}) - \mathbf{d}^T \right\rceil_p$.

 H_3 : Finally, we replace the vectors $\mathbf{a}_0, \mathbf{a}_1, \mathbf{b}_1, \dots, \mathbf{b}_{\lambda}, \mathbf{d}$ by uniformly random vectors.

Lemma 5.15. For all padversaries \mathcal{A} , for all $\lambda \in \mathbb{N}$, $\Pr[H_1(\mathcal{A}) = 1] = \Pr[H_0(\mathcal{A}) = 1]$.

Proof. Distributions $A_0, A_1, \{B_i\}_{i \in [\lambda]}$ are both uniform. Therefore, $Adv_{H_1}(\mathcal{A}) = Adv_{H_0}(\mathcal{A})$.

Lemma 5.16. For all padversaries \mathcal{A} , for all $\lambda \in \mathbb{N}$, $\Pr[H_2(\mathcal{A}) = 1] = \Pr[H_1(\mathcal{A}) = 1]$.

Proof. We rewrite y in the following way:

$$\mathbf{y} = \left[\mathbf{s}^{T} (\mathbf{B}_{EQ,x^{*}} + \mathbf{C}) G^{-1}(\mathbf{D}) \right]_{p}$$

= $\left[\mathbf{s}^{T} (\mathbf{B}_{EQ,x^{*}} + \mathbf{C} + \mathbf{G}) G^{-1}(\mathbf{D}) - \mathbf{s}^{T} \mathbf{D} \right]_{p}$
= $\left[(\mathbf{b}_{EQ,x^{*},x^{*}} + \mathbf{c})^{T} G^{-1}(\mathbf{D}) - \mathbf{d}^{T} + \mathbf{e}^{\prime \prime T} \right]_{p}$ (4)

where in the second equality, $\mathbf{e}'' = (\mathbf{e}_3 + \mathbf{e}')G^{-1}(\mathbf{D}) - \mathbf{e}$. By lemma 5.7 we know that $\|\mathbf{e}''\|_{\infty} \leq \gamma'$ and thus the values in the two hybrids differ if the vector $\mathbf{s}^T(\mathbf{B}_{EQ,x^*} + \mathbf{C})G^{-1}(\mathbf{D})$ has an entry in $[-\gamma', \gamma'] + (q/p)\mathbb{Z}$. Because **s** has been sampled using KeyGen(pp, x^*) to avoid this condition, the two hybrids are identical. \Box

Lemma 5.17. For all polynomially bounded adversaries \mathcal{A} , there exists a negligible function negl(·) such that for all $\lambda \in \mathbb{N}$, $|\Pr[H_3(\mathcal{A}) = 1] - \Pr[H_2(\mathcal{A}) = 1]| = \operatorname{negl}(\lambda)$.

Proof. We can do this by decisional LWE because in H_2

$$\mathbf{a}_{\beta} = \mathbf{s}^{T} \hat{\mathbf{A}}_{\beta} + \mathbf{e}_{1,\beta} \qquad \text{for all } \beta \in \{0, 1\}$$
$$\mathbf{b}_{i} = \mathbf{s}^{T} \hat{\mathbf{B}}_{i} + \mathbf{e}_{2,i} \qquad \text{for all } i \in [\lambda]$$
$$\mathbf{c} = \mathbf{s}^{T} \mathbf{C} + \mathbf{e}_{3}$$
$$\mathbf{d} = \mathbf{s}^{T} \mathbf{D} + \mathbf{e}$$

where all the matrices are independent and uniform, while in H_3 each of these vectors is replaced with uniform. This means the $|Adv_{H_3}(\mathcal{A}) - Adv_{H_4}(\mathcal{A})| \le negl(\lambda)$. \Box

Lemma 5.18. For all padversaries \mathcal{A} , for all $\lambda \in \mathbb{N}$, $\Pr[H_3(\mathcal{A}) = 1] = 1/2$.

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

Combining all the proofs above and by a hybrid argument, we have the formal proof our claim above.

Choice of Parameters. Collecting the formal lemmas and claims, we have the following two constructions, one where modulus-to-noise ration is super-polynomial and one where the ratio is polynomial.

Theorem 5.19. Setting $m = \lceil \log q \rceil \cdot n$, and $q = \lambda^{\omega(1)}$ and $p = \lambda^{\omega(1)}$ (both super-polynomial), such that $q \ge 2m(2\gamma' + 1)p$, the modulus-to-noise ratio α is super polynomial, and LWE holds with parameters n, q, m, α , then KeyGen does not depend on the punctured point x^* and the construction above is an almost-key homomorphic puncturable PRF.

Theorem 5.20. Setting $m = \lceil \log q \rceil \cdot n$, and $q = \text{poly}(\lambda)$ and $p = \text{poly}(\lambda)$ (both polynomial), such that $q \ge 2m(2\gamma' + 1)p$ and p divides $q, \gamma = \alpha q \cdot \omega(\sqrt{\log(\lambda)})$ where α is the modulus-to-noise ratio, and LWE holds with parameters n, q, m, α , then the construction above is an almost-key homomorphic puncturable PRF.

Observe that when using this construction in Construction 3.1, our polynomial p needs to be set bigger than 8× the number of puzzles to be batched. Thus when combining the multiple theorems, we need ensure our chosen parameters satisfy all the constraints.

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. Informally, in the game an adversary wins if an honest puzzle is incorrectly evaluated when batch solving together with potentially maliciously generated puzzles. Finally, we provide a construction that satisfies this property in various settings.

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

{0, 1} ← IsValid(pp, Z) a probabilistic algorithm that takes as input the public parameters, a puzzle Z and returns a bit {0, 1}.

In addition, $\Pi_{batchTLP}$ should satisfy the properties:

• Validity Check: We say $\Pi_{batchTLP}$ satisfies validity check if for all $\lambda, n, T \in \mathbb{N}$, for all inputs $s \in \mathbb{S}_{\lambda}$ it holds that

$$IsValid(pp, Gen(pp, s)) = 1$$

where pp \leftarrow Setup $(1^{\lambda}, T)$.

- **Rogue Puzzle Security:** For a security parameter *λ*, we define the following security game between an adversary *A* and a challenger as follows:
 - Adversary \mathcal{A} outputs the time for locking puzzle *T* and a bound on the number of puzzles to be batched *n*.
 - Challenger outputs the public parameters $pp \leftarrow \text{Setup}(1^{\lambda}, T, n)$.
 - Adversary \mathcal{A} sees the public parameters and outputs a message *m*.
 - Challenger honestly generates a puzzle $Z \leftarrow \text{Gen}(\text{pp}, m)$ and outputs the puzzle Z.
 - Adversary receives the puzzle Z and outputs a set of puzzle $\{Z_j^*\}_{j \in S^*}$ such that $|S^*| \le n$.
 - Challenger receives the set of puzzles and runs $\left\{(s_j^*, Z_j^*)\right\}_{j \in S^*} \leftarrow \text{BatchSol}(\text{pp}, \left\{Z_j^*\right\}_{j \in S^*}).$
 - Adversary wins the game and the output of the experiment is b = 1, if $\forall j \in S^*$, IsValid(pp, Z_j^*) = 1 and for some $j \in S^*$, $Z_j^* = Z$, $s_j^* \neq m$. Else the experiment outputs 0.

We say that Π_{batchTLP} is secure if for any polynomially bounded adversary $\mathcal{A} = \{\mathcal{A}_{\lambda}\}_{\lambda \in \mathbb{N}}$, there exists a negligible function negl(·), such that for all $\lambda \in \mathbb{N}$, it holds that, $\Pr[b = 1] \leq \operatorname{negl}(\lambda)$ in the game above.

Remark 6.2. Even if the algorithm BatchSol doesn't run or use the procedure IsValid, a scheme can be secure against rogue puzzle attacks. The IsValid procedure is simply separated out for ease of exposition. In practice before running BatchSol, our algorithm would run the procedure IsValid on all puzzles and only batch puzzles that pass the validity check.

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 ← LHP.Gen(pp_{LHP}, Encode_{p,ℓ}(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}^{7} and compute the index $i \in \{0, 1\}^{\lambda} \leftarrow \text{Hash}(Z_{indep})$, 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}.

The 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). The fully formal construction and proof is proved in Appendix D.

Construction (Bounded Setting, Lattices). When we want to use a polynomial modulus for the lattice based PRF, our PRF key k depends on the punctured point, and we can no longer sample the PRF key k and the Z independent of i. This leads us to a circularity. Z needs to depend on i and i is computed from the Z in the unbounded setting above. We resolve this issue by sampling them together.

We briefly sketch how the key k depends on *i* in Construction 5.9. 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. Our construction repeats until $C_i(k) = 1$ and keeps sampling k's uniformly randomly until this holds. 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 ← TLP.Gen(k). Compute the index i ∈ {0, 1}^λ ← Hash(Z).

⁷In the unbounded setting, the degree is 1 and the right side of the bipartite graph is the same as the left side.

• 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 (key homomorphism and functionality preserving) holds even if C_i does not hold (see Remark 5.13). In the rogue puzzle security game, the adversaries are of any polynomial depth, and can already break the security of the underlying puzzle. Thus, sampling a key k such that $C_i(k) = 0$ does not effect the security proofs for rogue attackers.

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 $[n_{new}]^d$. For the puzzle generation algorithm we:

• We sample the puzzle indepedent indices $Z_{indep,1}, \ldots, Z_{indep,d}$ 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 zeroknowledge (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 $Z_{indep,i}$.

The IsValid algorithm simply checks that the two conditions above are met and is same as before. 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 against rogue puzzle attacks, 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 puzzle 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 IsValid hold. We show below that this is not possible.

Observe that in Eq. (2), the probability of choosing a set $S \subseteq U$ is now $\binom{q}{\ell}$ because the adversary \mathcal{R}_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,

$$\sum_{\ell=d}^{n} \binom{q}{\ell} \binom{n'}{\ell-1} \left(\frac{\binom{\ell-1}{d}}{\binom{n'}{d}}\right)^{\ell}.$$
(5)

0

A similar malicious expression was analyzed in [GLWW23] and built on top of our honest analysis. Our theorem statement gives better parameters than [GLWW23] and we mention the modified theorem statement below. The proof is very similar to the proof above.

Lemma 6.3. 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 $\geq 1 - \operatorname{negl}(\lambda)$.

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 6.4 (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 completeness⁸.

Algorithm TLP.Gen(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 $(u_1, v_1) \leftarrow \text{TLP.Gen}(\text{pp}, s_1; r_1)$, $(u_2, v_2) \leftarrow \text{TLP.Gen}(\text{pp}, s_2; r_2)$, and $(u_3, v_3) \leftarrow \text{TLP.Gen}(\text{pp}, s_1 + s_2; r_1 + r_2)$, 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 $\Pi = (\text{Prove, Verify})$ be a protocol for an instance $\chi = \left(\text{pp}, Z, g_1^{x^{i^*}} \in \mathbb{G}, y \in \mathbb{G}\right)$ and witness $\omega = (k \in \mathbb{Z}_N, r \in \mathbb{Z}_{N^2})$ such that, Z = TLP.Gen(pp, k; r) and $y = \left(g_1^{x^{i^*}}\right)^k \in \mathbb{G}$.

- Prove (χ, ω) :
 - 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.

⁸For brevity, we only show the puzzle generation algorithm.

- Compute $\hat{k} = k' + c \cdot k \in \mathbb{Z}_N$, and $\hat{r} = r' + c \cdot r \in \mathbb{Z}^9$.
- Send $(\hat{k} \in \mathbb{Z}_N, \hat{r} \in \mathbb{Z})$ to the verifier.
- Output $\pi = (Z', y' \in \mathbb{G}, \hat{k} \in \mathbb{Z}_N, \hat{r} \in \mathbb{Z})$ as the proof.
- Verify(χ):
 - 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 TLP.Gen(pp, $\hat{k}; \hat{r}$) $\stackrel{?}{=} Z' \cdot Z^c$.
 - Check if $\left(g_1^{x^{i^*}}\right)^{k} \stackrel{?}{=} y' \cdot y^c$.
 - If all checks pass, accept, else reject.

Completeness The scheme is complete, because TLP.Gen(pp, $\hat{k}; \hat{r}$) = TLP.Gen(pp, k'; r')·TLP.Gen(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. $(g_1^{x^{i^*}})^{\hat{k}} = (g_1^{x^{i^*}})^{k'} \cdot (g_1^{x^{i^*} \cdot k})^c = y' \cdot y^c$.

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 $\chi = (pp, Z, g_1^{x^{i^*}} \in \mathbb{G}, y \in \mathbb{G})$ and outputs a proof $\pi = (Z', y' \in \mathbb{G}, \hat{k} \in \mathbb{Z}_N, \hat{r} \in \mathbb{Z})$ 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 $Z' = (g^{r_0} \mod N, h^{r'_1 \cdot N} \cdot (1+N)^{k'_0} \mod N^2)$, $Z = (g^{r_0} \mod N, h^{r_1 \cdot N} \cdot (1+N)^{k_0} \mod N^2)$ 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,

- $(g_1^{x^{i^*}})^{\hat{k}} = y' \cdot y^c$. Thus, $\hat{k} = k_1' + c \cdot k_1 \mod N$.
- TLP.Gen(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 $\leq \frac{[N/\phi(N)]}{N} < 2/\phi(N)$, 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 $k_0 = k_1 \mod N$.

⁹For value *c* in \mathbb{Z}_N for some q, the prover considers it as a positive integer by setting the output in 1, ..., *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 $Z = \mathsf{TLP}.\mathsf{Gen}(\mathsf{pp}, k; r)$ and $y = \left(g_1^{\mathbf{x}^{i^*}}\right)^k$.

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 $\tilde{y} = \frac{\left(g_1^{x^{t^*}}\right)^{\tilde{k}}}{y^c} \in \mathbb{G}$ 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 $(Z', y', c, \tilde{k}, \tilde{r})$.

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 $\leq \frac{N^3}{N^4} = \text{negl.}$

Remark 6.5 (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 6.6 (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 s ∈ Zⁿ_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 completeness¹⁰.

Algorithm TLP.Gen(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 $\mathbf{v} = h^{\mathbf{r} \cdot N} \odot (1 + N)^{\mathbf{s}}$ mod 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 TLP.Gen(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).

¹⁰For brevity, we only show the puzzle generation algorithm.

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 $n, m, B \in \mathbb{Z}^{11}, B < N, \mathbf{k} \in \mathbb{Z}_N^n, \mathbf{A}_x \in \mathbb{Z}_N^{n \times m}$, and $\mathbf{e} \in [-B, B]^m$. \mathbf{A}_x is public and depends on *x*.

• A special-case NIZK (called range proof) $\Pi_{range} = (Setup, Prove, Verify)$ 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 Π = (Setup, Prove, Verify) is an argument for the statement, $\chi = (pp, Z, \mathbf{A}_x, \mathbf{b})$ 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, TLP.Gen(pp, $\mathbf{k}; r) \stackrel{?}{=} Z$, and, $\mathbf{k}^T \mathbf{A}_x + \mathbf{e} \stackrel{?}{=} \mathbf{b}$.

- Setup (1^{λ}) : Let crs \leftarrow range.Setup (1^{λ}) .
- Prove(crs, χ , ω):
 - 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 $\pi \leftarrow$ range.Prove (crs, (pp, Z'), ($\mathbf{e} \in \mathbb{Z}_N^m, \mathbf{r}^T \mathbf{A}_x \in \mathbb{Z}^m$)) where Z' is the puzzle and the witness is ($\mathbf{e} \in [-B, B]^m, \mathbf{r}^T \mathbf{A}_x \in \mathbb{Z}^m$).
- Verify(crs, *χ*, *π*):
 - 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 range. Verify (crs, (pp, Z'), π).

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. TLP.Gen(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, TLP.Gen(pp, $\mathbf{e}'; \mathbf{r}') = Z'$.

Soundness If the statement characterized by (pp, Z, A_x, 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 evaluation is to compare our solution with alternative solutions that solve the same problem. In our experiments, we consider 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 scheme by Malavolta and Thyagarajan [MT19]¹².

¹¹We're overloading the notation m from previous sections. It does not match the m in the PPRF construction.

¹²It 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.

• 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 the time lock puzzle using the linearly homomorphic time lock puzzle by Malavolta and Thyagarajan [MT19]. For messages longer than λ bits, we can use hybrid encryption.

• **Our Solution:** We initialize our time lock puzzle using the linearly homomorphic time lock puzzle by Malavolta and Thyagarajan [MT19]. We implement a pairing based key homomorphic PRF and use an assymetric group in Construction 5.4 for better concrete efficiency. Security of this construction is based on the asymmetric *n*-Power Diffie-Hellman assumption Assumption 2.13. Finally, we combine all the primitives using the transformations mentioned in Construction 3.1 and Construction 4.2.

We do not consider alternative constructions based on general purpose indistinguishability obfuscation iO [SLM⁺23]. As iO is a heavyweight cryptographic primitive that is not ready for efficient deployment (iO has been implemented for some *restricted* functionalities [LMA⁺16, CMR17], but there are no general purpose implementations).

Semi-honest setting The construction we implement is useful in a semi-honest security model and does not consider rogue puzzle attacks discussed in Section 6. For example, in an e-voting scenario where the attacker's power is limited to stalling the election by putting the honest users offline (e.g., by cutting the wires of their connection). The semi-honest attacker model is also motivated in settings such as permissioned blockchains, where parties are accountable for their actions.

We leave open the implementation based on our LWE based key homomorphic PRF Construction 5.9 and an implementation that is robust against rogue attackers 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 key-homomorphic schemes over the BLS-381 pairing group [BLS02, BGM17, SKSW20] and use the implementation from the herumi mcl library [Mit] (written in C++ language). 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] (the implementation is available at https://github.com/verifiable-timed-signatures/liblhtlp), which uses GNU Multi-Precision library [GMP] (version 6.2.1) and is implemented in C.

The bipartite matching algorithm in Construction 4.2 is implemented by a textbook Hopcroft Karp algorithm in C++.

Parameter selection for Construction 4.2. In order to compile our transformations, we need to set the correct values of the bipartite graph so that a matching exists with overwhelming probability. Recall from Theorem 4.3, for *n* denoting a maximum bound on the number of puzzles to be batched, we require setting the right side of the bipartite graph, $n' \ge 3n$ and the degree as, $d = O(1) + \frac{\omega(\log \lambda)}{\log n'}$.

In our experimentation, we choose the parameters so that Eq. (2) 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. We wish to choose the parameters so that the comminication size and the number of puzzles sent over is minimized. Thus, our main goal is to choose the parameters n', d that minimize the degree d.

We employ the following algorithm for choosing our parameters that guarantee 40 bits of statistical security for batching atmost $n \le 10,000$ puzzles.

- Set an initial size of n' = 100,000.
- Binary search for the minimum degree *d* between 1 and 128 that satisfies Eq. (2) with 40 bits of security. Lets call this degree d_{opt} .
- Perform a binary search for n' between n and 100,000 that satisfies Eq. (2) with 40 bits of security. We denote this by n'_{opt} .
- Output the right side of the bipartite graph to be of size $n' = n'_{ont}$, and degree $d = d_{opt}$.

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. (5) for an appropriate choice of a bound on the number of queries q.

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 higher communication complexity).
- Our Solution: We take $O(T) + O(n^2)$ where the latter takes quadratic time because each puzzle takes O(n) operations in Construction 3.1.

While the Hopcroft-Karp algorithm would yield a worst-case performance of $\tilde{O}(n^{1.5})$, we note that when the underlying graph is *random* (as in our setting), the running time is quasilinear in the size of the graph [Mot94, BMST06] in expectation (where the expectation is taken over the randomness for sampling the graph).

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. 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 different solutions). Parallelizing *n* time lock puzzles in the "trivial solution" involves runnning *n* threads where each thread has to perform *T* sequential computations. In contrast, in "our solution" we can profit from parallelism by computing the PRF operations in a new thread and then recovering the messages only involves O(n) operations. Thus, in total, we arrive at a sequential running time of $O(T) + \tilde{O}(n)$ (where the $\tilde{O}(n)$ is due to the matching algorithm).

Experimental setup. The implementation of our scheme consists of 2400 lines of code.¹³ We collect our benchmarks on a 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.

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.

Computational cost. We measure the computational cost of solving *n* puzzles. For the "trivial solution", we compute the average time to solve per puzzle by solving 10 puzzles and measure the total compute time by multiplying *n* to the average. We vary *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 wall clock time will change between machine implementations. We mainly compare between the "trivial solution" and "our solution" for algorithms BatchSol, Sol, Gen, Setup below. The computational cost for batch solving the "strawman solution" are similar to solving a single time lock puzzle in the "trivial solution" 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. The dotted line indicates the plot for "our trivial" solution while the solid line plots "our solution". We quote some concrete numbers below.

- Batch solve For T = 50 million, and n = 500, batching trivially takes $160 \times$ worse than our solution and would take about 15 hours of compute time.
- Solve Since the only difference when solving a single puzzle is a single pairing operation, the time to solve a single puzzle is the same.
- Puzzle generation The puzzle generation for "our solution" is worse, where for T = 50 million, per puzzle generation takes $3 \times$ time (as the degree of the graph is 3). Nevertheless, we generate the puzzle extremely efficiently and within 0.2 seconds.
- Puzzle setup Setup for "our solution" now involves sampling the CRS and involves some extra computation along with sampling a RSA integer. But even for n = 100,000, the pairing based puncturable PRF can be sampled within 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 both schemes are the same.

 $^{^{13}} The \ complete \ implementation \ is \ available \ here: \ https://github.com/RachitG54/batchtlpmcl \ .$

Communication size. In Fig. 2, we compare the setup size and the total communication size (in bytes) as a function of the number of puzzles batched. We compare the three solutions, the "trivial solution", the "strawman solution" and "our solution". We computed the numbers analytically, where we vary *n* between 100 and 7000. We mention some concrete statistics below.

- Puzzle setup "Our solution" is strictly worse in terms of setup size where the setup grows with the number of puzzles. For n = 7,000, the right side of the bipartite graph, n' = 11,000 and the setup size is 2200× worse, but still takes only 2.6MB size.
- Batch solve For n = 7,000, the "trivial solution" is the most efficient and takes 8MB of communication, and is approximately 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 the computation time for "our solution" between the pairing operations and the graph operations for $T = 5 \times 10^8$. The pairing operations grow linearly with *n*, 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 $1.2 \times$ slower than solving a single puzzle. 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. We presented two primary solutions for solving the problem of batch solving. The trivial solution is computation efficient, but requires $O(n^2)$ communication, while the solution presented by us needs to compute $O(n^2)$ quadratic operations, and is communication efficient. We can instead combine both solutions and combine both approaches. Specifically, we can imagine batching *B* puzzles together, where each of the puzzle encodes *A* puzzles using the "strawman solution". Using such parameters, we have that $n = A \times B$, communication cost is $O(n \cdot A)$ and computation cost is $O(B^2)$.

We do not explore rebalancing parameters in our prototype implementation, but it can already give us concretely better improvements. Note that in our experiments, the RSA modulus is 3072 bits, and the security parameter for symmetric key crypto is 128 bits. If we allow room for performing addition operations over each 128 bit block and consider a block size of 256 bits. We can set A = 3072/256 = 12, to save on our computation, while keeping the communication size of the puzzle exactly same. We leave open the problem of exploring these tradeoffs for a future work.

8 Acknowledgments

We thank George Lu, Brent Waters and David Wu for helpful discussions on removing coordination amongst parties. We thank Nico Döttling for helpful discussions on the lattice-based puncturable PRF.

J. D. was funded by the European Union (ERC, LACONIC, 101041207). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. G. M. was supported by the European Research Council through an ERC Starting Grant (Grant agreement No. 101077455, ObfusQation).

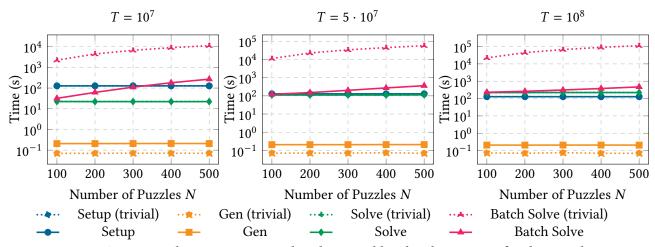
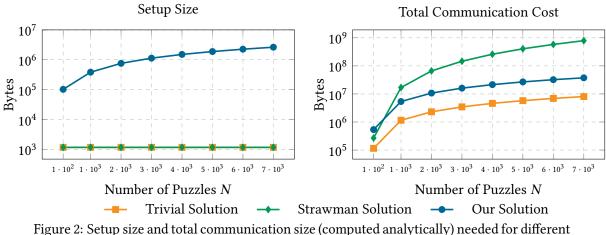


Figure 1: Setup, puzzle generation, puzzle solving and batch solving times for the trivial solution (indicated by dotted line) and our proposed solution (indicated by solid line). We vary the number of puzzle betwen 100 and 500 for different hardness of sequential computation, ranging from $T = 10^7$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

References

- [BB04] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *Annual International Cryptology Conference*, pages 443–459. Springer, 2004.
- [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.



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.

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

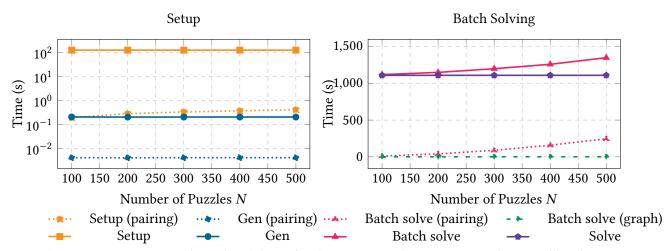


Figure 3: Computational cost breakdown for the setup, puzzle generation, solving and batch solving of our time lock puzzle. Batch solving involves (1) a combinatoric step (finding a matching in the bipartite graph) indicated by a dashed line; and (2) a cryptographic step for performing operations on 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.

- [BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. *Cryptology ePrint Archive*, 2017.
- [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.
- [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.

- [BMST06] Hannah Bast, Kurt Mehlhorn, Guido Schäfer, and Hisao Tamaki. Matching algorithms are fast in sparse random graphs. *Theory Comput. Syst.*, 2006.
- [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.
- [CDN20] Dan Cline, Thaddeus Dryja, and Neha Narula. Clockwork: An exchange protocol for proofs of non front-running. The Stanford Blockchain Conference 2020, 2020.
- [CHKP12] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. *Journal of cryptology*, 25:601–639, 2012.
- [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 nc^1. 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 n⁵/2 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 nc⁰. 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 timedrelease 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.
- [Mot94] Rajeev Motwani. Average-case analysis of algorithms for matchings and related problems. *J. ACM*, 41(6), nov 1994.
- [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.
- [Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pages 84–93, 2005.

- [RSW96] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.
- [SKSW20] Yumi Sakemi, Tetsutaro Kobayashi, Tsunekazu Saito, and Riad Wahby. Pairing-friendly curves. I-d, IETF, 9 2020.
- [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 parameter¹⁴) and $d(n, \lambda)$ be the degree of the bipartite graph. We analyze the different

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

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 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 $\ge 1 - \operatorname{negl}(\lambda)$ where the probability is taken over the random coins of sampling the bipartite graph.

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

$$\left(\frac{k-1}{n'}\right)^d,\tag{6}$$

where the probability is taken over the random coins of sampling the edges of the *k*th 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. \Box

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. (6), 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 $\leq {q \choose n} \cdot n \cdot 2^{-d}$. 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 6.4 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 6.4.
- 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 $\Pi_{range} = (Setup, Prove, Verify)$ 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 $\Pi = (\text{Prove, Verify})$ for an instance $\chi = (\text{pp}, Z, g_1^{x^{i^*}} \in \mathbb{G}, y \in \mathbb{G})$ and witness $\omega = (k \in \mathbb{Z}_p, r \in \mathbb{Z}_{N^2})$ such that, $Z = \text{TLP.Gen}(\text{pp}, k^{(0)}; r)$ and $y = (g_1^{x^{i^*}})^{k^{(1)}} \in \mathbb{G}$ and $k = k^{(0)} = k^{(1)} \mod p$.

- Prove (χ, ω) :
 - Sample randomly, $k' \leftarrow [p^2 \cdot N]$ and $r' \leftarrow [N^2 p^2]$.
 - Compute $Z' \leftarrow \mathsf{TLP.Gen}(\mathsf{pp}, k'; r'), y' \leftarrow \left(g_1^{x^{i^*}}\right)^{k'} \in \mathbb{G}.$
 - Compute $(\pi_{range}, \pi'_{range})$ by running range.Prove on *Z* and *Z'* respectively with the bound *p*. The prover sends $(Z', y', \pi_{range}, \pi'_{range})$ 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 $(\hat{k} \in \mathbb{Z}, \hat{r} \in \mathbb{Z})$ to the verifier.

- Output
$$\pi = \left(Z', y' \in \mathbb{G}, \pi_{\text{range}}, \pi'_{\text{range}}, \hat{k} \in \mathbb{Z}, \hat{r} \in \mathbb{Z}\right)$$
 as the proof.

- Verify(χ):
 - The verifier recieves information from the prover, verifies the range proof $(\pi_{\text{range}}, \pi_{\text{range}})$ 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 TLP.Gen(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.

¹⁵For 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, ..., q and treating them as integers.

Completeness The scheme is complete, because TLP.Gen(pp, $\hat{k}; \hat{r}$) = TLP.Gen(pp, k'; r')·TLP.Gen(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. $(g_1^{x^{i^*}})^{\hat{k}} = (g_1^{x^{i^*}})^{k'} \cdot (g_1^{x^{i^*} \cdot k})^c = y' \cdot y^c$. 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 $\chi = (pp, Z, g_1^{x^{t^*}} \in \mathbb{G}, y \in \mathbb{G})$ and outputs a proof $\pi = (Z', y' \in \mathbb{G}, \pi_{range}, \pi'_{range}, \hat{k} \in \mathbb{Z}, \hat{r} \in \mathbb{Z})$ 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 $Z' = (g^{r'_0} \mod N, h^{r'_1 \cdot N} \cdot (1+N)^{k'_0} \mod N^2)$, $Z = (g^{r_0} \mod N, h^{r_1 \cdot N} \cdot (1+N)^{k_0} \mod N^2)$ 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 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,

- $\left(g_1^{x^{i^*}}\right)^k = y' \cdot y^c$. 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 $\alpha \in [-p - 1, p + 1]$.
- TLP.Gen(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, $h^{((r'_0-r'_1)+c(r_0-r_1))\cdot N} = 1 \mod N$. 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 $Z = \text{TLP.Gen}(\text{pp}, k_0; r)$ and $y = \left(g_1^{x^{t^*}}\right)^{k_1}$.

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 $\tilde{Z} \leftarrow \mathsf{TLP}.\mathsf{Gen}(\mathsf{pp}, \tilde{k}, \tilde{r})$ and $Z' \leftarrow \frac{\tilde{Z}}{Z^c}$. • The simulator outputs the transcript $(Z', y', c, \tilde{k}, \tilde{r})$.

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 $\leq \frac{N^2 p}{N^2 p^2} = \text{negl}$.

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.

C Formal proof for Construction 3.1

Theorem C.1. Suppose Π_{LHP} be a secure time-lock puzzle for gap $\varepsilon \in (0, 1)$ and polynomial $\tilde{T}(\cdot)$ defined according to Definition 2.3, Π_{PRF} be a secure puncturable PRF according to Definition 2.1, then for construction 3.1, for all polynomially bounded functions where $T(\cdot) \geq \tilde{T}(\cdot)$, 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 negl(\cdot) such that, $(\mathcal{A}_1, \mathcal{A}_2)$ wins the security game in Definition 2.10 with negligible advantage.

Proof. Let Π_{LHP} be a secure time-lock puzzle according to Definition 2.3, then, there exists some gap $\varepsilon \in (0, 1)$, and polynomial $\tilde{T}(\cdot)$. We define the sequence of hybrids similar to the main body where H_0 is the original hybrid where we play the following game between a challenger *C* and adversaries $(\mathcal{A}_1, \mathcal{A}_2)$. For any $T(\cdot) \geq \tilde{T}(\cdot)$,

- Adversary \mathcal{A}_1 outputs a bound on the number of puzzles to batch $n = n(\lambda)$.
- Challenger *C* samples $pp \leftarrow \text{Setup}(1^{\lambda}, T, n)$ and sends it to \mathcal{A}_1 .
- Adversary \mathcal{A}_1 receives the pp and outputs st, messages $s_0, s_1 \in \mathbb{S}_\lambda$ and index $i \in [n]$.
- Challenger *C* samples a bit $b \in \{0, 1\}$ and outputs puzzle $Z \leftarrow \text{Gen}(\text{pp}, i, s_b)$. Sample a PRF key $k \leftarrow \mathbb{Z}_p^{\ell}$. Time-lock the key by computing $Z' \leftarrow \text{LHP.Gen}(\text{pp}_{\text{LHP}}, \text{Encode}_{p,\ell}(k))$. Compute the punctured key $k^* \leftarrow \text{PRF.Puncture}(\text{pp}_{\text{PRF}}, k, i)$. Mask the message $c \leftarrow \text{PRF}(\text{pp}_{\text{PRF}}, k, i) + s_b \cdot \lceil p/2 \rceil$. Output $Z = (i, Z', k^*, c)$.
- Adversary \mathcal{R}_2 receives (pp, Z, st) and outputs a bit b'.

In H_1 , the challenger instead samples $Z' \leftarrow LHP.Gen(pp_{LHP}, 0)$. We argue that the change goes unnoticed to an adversary.

Lemma C.2. For all polynomially bounded functions where $T(\cdot) \geq \tilde{T}(\cdot)$, 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 at most $T^{\varepsilon}(\lambda)$, there exists a negligible function negl(\cdot) such that for all $\lambda \in \mathbb{N}$, $|\Pr[H_1(\mathcal{A}_1, \mathcal{A}_2) = 1] - \Pr[H_0(\mathcal{A}_1, \mathcal{A}_2) = 1]| = negl(\lambda)$.

Proof. Let $(\mathcal{A}_1, \mathcal{A}_2)$ be an adversary where

$$|\Pr[H_1(\mathcal{A}_1, \mathcal{A}_2) = 1] - \Pr[H_0(\mathcal{A}_1, \mathcal{A}_2) = 1]| \ge \varepsilon'$$

for some non-negligible ε' . We construct an adversary $(\mathcal{B}_1, \mathcal{B}_2)$ that breaks the security of Π_{LHP} as follows.

- Adversary \mathcal{A}_1 outputs a bound on the number of puzzles to batch $n = n(\lambda)$.
- Challenger *C* for Π_{LHP} outputs $pp_{LHP} \leftarrow Setup(1^{\lambda}, T)$.
- Algorithm B₁ samples pp_{PRF} ← PRF.Setup(1^λ, n) and outputs (pp_{LHP}, pp_{PRF}) to A₁. Algorithm A₁ outputs state st, messages s₀, s₁ and index i ∈ [n] to algorithm B₁.
 Algorithm B₁ samples a PRF key k ← Z^ℓ_p, samples a bit b ∈ {0, 1}. It computes the punctured key k*, mask c ← PRF(pp_{PRF}, k, i)+s_b · [p/2]. It sets state st' = (pp, st, i, k*, c) and messages (Encode_{p,ℓ}(k), 0) to the challenger.
- Challenger *C* computes the puzzle Z' and returns $(pp_{LHP}, Z', st' = (pp, st, i, k^*, c))$ to \mathcal{B}_2 .
- Algorithm \mathcal{B}_2 sends (pp, (i, Z', k^*, c) , st) to \mathcal{A}_2 and receives a bit b'. Algorithm \mathcal{B}_2 outputs b'.

If the challenger for Π_{LHP} chose $Encode_{p,\ell}(k)$ then we are in H_0 , and instead if it chose 0, then we are in H_1 . Thus $(\mathcal{B}_1, \mathcal{B}_2)$'s advantage in breaking the security of Π_{LHP} is non-negligible.

In H_2 , the challenger instead samples $c \leftarrow \mathcal{Y}_{\lambda}$ (where \mathcal{Y}_{λ} is the PRF range). We argue that the change goes unnoticed to an adversary.

Lemma C.3. For all polynomially bounded functions where $T(\cdot) \geq \tilde{T}(\cdot)$, 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 at most $T^{\varepsilon}(\lambda)$, there exists a negligible function negl(\cdot) such that for all $\lambda \in \mathbb{N}$, $|\Pr[H_2(\mathcal{A}_1, \mathcal{A}_2) = 1] - \Pr[H_1(\mathcal{A}_1, \mathcal{A}_2) = 1]| = negl(\lambda)$.

Proof. Let $(\mathcal{A}_1, \mathcal{A}_2)$ be an adversary where

$$|\Pr[H_2(\mathcal{A}_1, \mathcal{A}_2) = 1] - \Pr[H_1(\mathcal{A}_1, \mathcal{A}_2) = 1]| \ge \varepsilon'$$

for some non-negligible ε' . We construct an adversary \mathcal{B} that breaks the security of Π_{PRF} as follows.

- Algorithm \mathcal{A}_1 outputs a bound on the number of puzzles to batch $n = n(\lambda)$.
- Algorithm \mathcal{B} outputs a bound on the domain of the PRF 1^{*n*}.
- Challenger *C* for Π_{PRF} computes $pp_{\mathsf{PRF}} \leftarrow \mathsf{PRF}.\mathsf{Setup}(1^{\lambda}, 1^n)$.
- Algorithm \mathcal{B} computes $pp_{LHP} \leftarrow LHP.Setup(1^{\lambda}, T)$ and outputs $pp = (pp_{LHP}, pp_{PRF})$ to the algorithm \mathcal{R}_1 .
- Algorithm \mathcal{A}_1 outputs state st, messages $s_0, s_1 \in \mathbb{S}_{\lambda}$, index $i \in [n]$.
- Algorithm \mathcal{B} outputs the index $i^* = i \in [n]$.
- Challenger *C* for Π_{PRF} computes key k \leftarrow PRF.KeyGen(pp), punctures the key k* \leftarrow PRF.Puncture(pp_{PRF}, k, *i**) and computes $y \in \mathcal{Y}_{\lambda}$. It returns k*, *y* to algorithm \mathcal{B} .
- Algorithm \mathcal{B} samples a bit $b \in \{0, 1\}$, it computes the mask $c \leftarrow y + s_b \cdot \lceil p/2 \rceil$, and $Z' \leftarrow LHP.Gen(pp_{LHP}, 0)$ and outputs $Z = (i, Z', k^*, c)$.
- Algorithm \mathcal{A}_2 receives (pp, Z, st) and outputs a bit b'. Algorithm \mathcal{B} outputs the bit b'.

If the challenger for Π_{PRF} chose to evaluate using the PRF key, then we are in H_1 , else we are in H_2 .

In H_2 , the advantage of adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ in the security game is zero, $|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = 0$. This is because, the output to the adversary \mathcal{A}_2 does not depend on b.

D Formal construction and proof for rogue security

Construction (Unbounded Setting). We give the complete construction and proof below. Let Hash be a collision-resistant hash function with output space $\{0, 1\}^{\lambda}$ and NIZK be a secure non-interactive zero knowledge proof. Since this is the unbounded setting, we assume that the underlying PRF can support unbounded indices in space $\{0, 1\}^{\lambda}$.

 $pp \leftarrow \text{Setup}(1^{\lambda}, T, n)$:

- Sample $pp_{LHP} \leftarrow LHP.Setup(1^{\lambda}, T)$.
- Sample $pp_{PRF} \leftarrow PRF.Setup(1^{\lambda})$.
- Sample $pp_{Hash} \leftarrow Hash.Setup(1^{\lambda})$.
- Sample $pp_{NIZK} \leftarrow NIZK.Setup(1^{\lambda})$.
- Output $pp = (pp_{LHP}, pp_{PRF}, pp_{Hash}, pp_{NIZK})$.

Gen(pp, *m*):

- Sample the index independent values, i.e. sample a PRF key $k \leftarrow \mathbb{Z}_p^{\ell}$. Compute the puzzle $Z' \leftarrow LHP.Gen(pp_{LHP}, Encode_{p,\ell}(k))$.
- Calculate the index $i \leftarrow \text{Hash}(\text{pp}_{\text{Hash}}, Z')$.
- Compute the punctured key k^{*} ← PRF.Puncture(pp_{PRF}, k, *i*).
- Mask the message $c \leftarrow \mathsf{PRF}(\mathsf{pp}_{\mathsf{PRF}},\mathsf{k},i) + m \cdot \lceil p/2 \rceil$.
- Compute a NIZK proof π that certifies for (Z', k^*, i) that there exists a $k \in \mathbb{Z}_p^{\ell}$ such that $Z' = LHP.Gen(pp_{LHP}, Encode_{p,\ell}(k))$ and $k^* = PRF.Puncture(pp_{PRF}, k, i)$.
- Return (i, Z', k^*, c, π) .

IsValid(pp, *Z*):

- Parse pp and Z appropriately. If parsing fails, output 0.
- If $\text{Hash}(\text{pp}_{\text{Hash}}, Z') \neq i$, output 0.
- If π does not verify for (Z', k^*, i) , output 0.
- If all checks pass, output 1.

BatchSol $\left(\operatorname{pp}, \left\{ i_j, Z'_j, \mathsf{k}^*_j, c_j, \pi_j \right\}_{j \in \mathcal{S}} \right)$:

- Find the maximal set S' such that the resulting puzzles contain only unique indices $\{i_j\}_{j \in S'}$ (we can break ties arbitrarily) and the resulting puzzle set is denoted by $\{Z_j\}_{j \in S'}$ i.e. $\{i_j, Z'_j, k^*_j, c_j, \pi_j\}_{i \in S'}$.
- Compute the sum of puzzles, *Ž* ← LHP.Eval (∑, pp_{LHP}, {Z'_j}_{j∈S'}) where ∑ indicates that a sum of puzzles is computed homomorphically.
- Solve the resulting puzzle $\tilde{k} \leftarrow LHP.Sol(pp_{LHP}, \tilde{Z})$.
- Compute $k' \leftarrow \text{Decode}_{p,\ell}(\tilde{k})$.

• For all $j \in S'$, compute

$$\mu_j = c_j + \sum_{i \in \mathcal{S}' \setminus \{j\}} \mathsf{PRF}.\mathsf{PuncturedEval}(\mathsf{pp}_{\mathsf{PRF}},\mathsf{k}^*_i,i,j) - \mathsf{PRF}(\mathsf{pp}_{\mathsf{PRF}},\mathsf{k}',j) \pmod{p}$$

and set m_j as $\lfloor \mu_j \rceil_{\lceil p/2 \rceil}$.

• For all remaining puzzles $j \in S$, there exists some puzzle $j' \in S'$ such that $i_j = i_{j'}$, thus compute $m_j = \lfloor \mu_{j'} - c_{j'} + c_j \rceil_{\lceil p/2 \rceil}$.

Theorem D.1. If Hash is a collision resistant hash function, NIZK is a sound non-interactive zero-knowledge proof, then the construction above satisfies security against rogue puzzle attacks according to Definition 6.1.

Proof. We start by defining a sequence of hybrid experiments:

- Hyb₀ : This is the original rogue puzzle security game.
 - Adversary \mathcal{A} outputs the time for locking puzzle *T* and a bound on the number of puzzles to be batched *n*.
 - Challenger outputs the public parameters $pp \leftarrow \text{Setup}(1^{\lambda}, T, n)$.
 - Adversary \mathcal{A} sees the public parameters and outputs a message *m*.
 - Challenger honestly generates a puzzle $Z \leftarrow \text{Gen}(\text{pp}, m)$ and outputs the puzzle Z.
 - * Sample the index independent values, i.e. sample a PRF key k $\leftarrow \mathbb{Z}_p^{\ell}$. Compute the puzzle $Z' \leftarrow LHP.Gen(pp_{LHP}, Encode_{p,\ell}(k))$.
 - * Calculate the index $i^* \leftarrow \text{Hash}(\text{pp}_{\text{Hash}}, Z')$.
 - * Compute the punctured key k*, mask *c*, NIZK proof π and set $Z = (i^*, Z', k^*, c, \pi)$.
 - Adversary receives the puzzle *Z* and outputs a set of puzzle $\{Z_j^*\}_{j \in S^*}$ such that $|S^*| \leq n$.
 - Challenger receives the set of puzzles and runs $\left\{(s_j^*, Z_j^*)\right\}_{i \in S^*} \leftarrow \text{BatchSol}(\text{pp}, \left\{Z_j^*\right\}_{i \in S^*}).$
 - * For all $j \in S^*$, parse each $Z_j^* = (i_j, Z'_j, k_j^*, c_j, \pi_j)$.
 - * Let S' be the set such that the resulting puzzle set is denoted by $\{Z_j\}_{j \in S'}$ i.e. $\{i_j, Z'_j, k^*_j, c_j, \pi_j\}_{j \in S'}$.
 - Adversary wins the game and the output of the experiment is b = 1, if $\forall j \in S^*$, IsValid(pp, Z_j^*) = 1 and for some $j \in S^*$, $Z_j^* = Z$, $s_i^* \neq m$. Else the experiment outputs 0.
- Hyb₁: In this hybrid, the challenger outputs 0 if there exists an index $j \in S$ such that $i_j = i^*$, but $Z'_i \neq Z'$.
- Hyb₂: In this hybrid, the challenger outputs 0 if for some $j \in S^*$, IsValid(pp, Z_j^*) = 1, and let $k'_j = \text{Decode}_{p,\ell}(\text{LHP.Sol}(\text{pp}_{\text{LHP}}, Z'_j))$, but there does not exist randomness r such that the punctured key $k_i^* = \text{PRF.Puncture}(\text{pp}_{\text{PRF}}, k_j, i_j; r)$.

Lemma D.2. Let Hash be a collision resistant hash function. Then for every adversary \mathcal{A} , there exists a negligible function negl(·) such that for all $\lambda \in \mathbb{N}$, $|\Pr[Hyb_1(\mathcal{A}) = 1] - \Pr[Hyb_0(\mathcal{A}) = 1]| = negl(\lambda)$.

Proof. The only difference in the hybrids is if the adversary was able to find a collision to i^* . Thus the adversary after seeing the public parameters pp, and Z', can come up with a puzzle $Z'_j \neq Z'$ such that $i^* = \text{Hash}(\text{pp}_{\text{Hash}}, Z) = \text{Hash}(\text{pp}_{\text{Hash}}, Z'_j)$ and hence breaking collision resistance.

Lemma D.3. Let NIZK be a sound NIZK proof. Then for every adversary \mathcal{A} , there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|Pr[Hyb_2(\mathcal{A}) = 1] - Pr[Hyb_1(\mathcal{A}) = 1]| = negl(\lambda)$.

Proof. The only difference in the hybrids is if the adversary was able to break NIZK security by having IsValid output 1, but the NIZK statement is not in the language.

Lemma D.4. The probability that an adversary wins in Hyb_2 is 0.

Proof. We begin with two cases.

- Case 1: If Z is included in the set of puzzles to be matched, i.e. $Z \in \{Z_j\}_{j \in S'}$
- Case 2: If Z is not included in the set of puzzles to be matched, i.e. $Z \notin \{Z_j\}_{j \in S'}$.

In this case, there exists some index $j' \in S'$ such that $i_{j'} = i^*$ and $Z_{j'}^* = Z$ (if the puzzles are unequal, the output of the experiment is 0 due to definition of Hyb₁).

Additionally, for both cases, if the condition checked in Hyb_2 is such that IsValid passes but the keys in Z'_j and k^*_j don't match, then the adversary cannot win. Thus we assume that there is an underlying key that agrees with the puzzle and the punctured key.

- Since the NIZK proofs verify for all *j* ∈ S^{*}, we know that have that k'_j = Decode_{p,ℓ}(LHP.Sol(pp_{LHP}, Z'_j)) and that the punctured key k^{*}_i is a punctured key to the same k'_j.
- Thus, we have that, $k' = \sum_{i \in S'} k'_i$.
- Since c_{i^*} was computed honestly, we have, $c_{i^*} = \mathsf{PRF}(\mathsf{pp}_{\mathsf{PRF}},\mathsf{k},i^*) + m\lceil p/2 \rceil$.

In **Case 1**, we have, $i^* \in S'$ and we perform the computation,

$$\mu_{i^*} = c_{i^*} + \sum_{i \in \mathcal{S}' \setminus \{i^*\}} \mathsf{PRF}.\mathsf{PuncturedEval}(\mathsf{pp}_{\mathsf{PRF}},\mathsf{k}^*_i,i,i^*) - \mathsf{PRF}(\mathsf{pp}_{\mathsf{PRF}},\mathsf{k}',i^*) \pmod{p}$$

and set m_{i^*} as $\lfloor \mu_{i^*} \rceil_{\lceil p/2 \rceil}$. Simplifying the computation, we have,

$$\mu_{i^*} = \mathsf{PRF}(\mathsf{pp}_{\mathsf{PRF}},\mathsf{k},i^*) + m \cdot \lceil p/2 \rceil + \sum_{i \in \mathcal{S}' \setminus \{i^*\}} \mathsf{PRF}.\mathsf{PuncturedEval}(\mathsf{pp}_{\mathsf{PRF}},\mathsf{k}_i^*,i,i^*) - \mathsf{PRF}(\mathsf{pp}_{\mathsf{PRF}},\mathsf{k}',i^*) \pmod{p}$$

$$= \mathsf{PRF}(\mathsf{pp}_{\mathsf{PRF}},\mathsf{k},i^*) + m \cdot \lceil p/2 \rceil + \sum_{i \in \mathcal{S}' \setminus \{i^*\}} \mathsf{PRF}.\mathsf{PuncturedEval}(\mathsf{pp}_{\mathsf{PRF}},\mathsf{k}_i^*,i,i^*) - \sum_{j \in \mathcal{S}'} \mathsf{PRF}(\mathsf{pp}_{\mathsf{PRF}},\mathsf{k}'_j,i^*) + e \pmod{p}$$

$$= m \cdot \lceil p/2 \rceil + e'$$

The first equation holds from the almost key homomorphism, and the second equation holds from the almost functionality preserving property. Thus the message computed after rounding is exactly *m*. In **Case 2**, we have, for some $j' \in S'$, such that $i^* = i_{j'}, Z' = Z'_{j'}$ and we perform the computation, such that $\mu_{j'} - c_{j'} = -\text{PRF}(\text{pp}_{\text{PRF}}, k_{j'}, i^*) + e$ as stated above. Since, $Z' = Z'_{j'}$ and the NIZK verifies, we have that $k_{j'} = k$. Computing $m_{i^*} = \lfloor \mu_{j'} - c_{j'} + c_{i^*} \rceil_{\lceil p/2 \rceil}$ and setting the honest computation of c_{i^*} , we have that $m_{i^*} = \lfloor m \cdot \lceil p/2 \rceil + e \rceil_{\lceil p/2 \rceil} = m$ and thus the adversary cannot win.

The proof follows from the previous three lemmas.

Construction (Bounded Setting, Pairings). We give the complete construction and proof below. Let Hash be a random oracle and NIZK be a secure non-interactive zero knowledge proof. The construction is a slight modification to the previous construction.

 $pp \leftarrow \text{Setup}(1^{\lambda}, T, n)$:

- Let n_{new} and d be set according to parameters in Lemma 6.3.
- Sample $pp_{1HP} \leftarrow LHP.Setup(1^{\lambda}, T)$.
- Sample $pp_{PRF} \leftarrow PRF.Setup(1^{\lambda}, 1^{n_{new}}).$
- Sample $pp_{NIZK} \leftarrow NIZK.Setup(1^{\lambda})$.
- Output $pp = (pp_{LHP}, pp_{PRF}, pp_{NIZK})$.

Gen(pp, *m*):

- For each $i \in [d]$, sample the index independent values, i.e. sample a PRF key $k^{(i)} \leftarrow \mathbb{Z}_p^{\ell}$. Compute the puzzle $Z'^{(i)} \leftarrow LHP.Gen(pp_{1HP}, Encode_{p,\ell}(k^{(i)}))$.
- Compute the set $V \in [n_{\text{new}}]^d$ by applying the random oracle i.e. $V = \text{Hash}(Z'^{(1)}, \ldots, Z'^{(d)})$. Let $V = \{v_i\}_{i \in [d]}$.
- For each $i \in [d]$, compute the punctured key $k^{*,(i)} \leftarrow PRF$.Puncture $(pp_{PRF}, k^{(i)}, v_i)$. Mask the message $c^{(i)} \leftarrow PRF(pp_{PRF}, k^{(i)}, v_i) + m \cdot \lceil p/2 \rceil$. Compute a NIZK proof $\pi^{(i)}$ that certifies for $(Z', k^{*,(i)}, v_i)$ that there exists a $k^{(i)} \in \mathbb{Z}_p^{\ell}$ such that $Z'^{(i)} = LHP.Gen(pp_{LHP}, Encode_{p,\ell}(k^{(i)}))$ and $k^{*,(i)} = PRF$.Puncture $(pp_{PRF}, k^{(i)}, v_i)$.
- Output $\left(\left\{(v_i, Z'^{(i)}, \mathsf{k}^{*,(i)}, c^{(i)}, \pi^{(i)})\right\}_{i \in [d]}\right)$.

IsValid(pp, *Z*):

- Parse pp and *Z* appropriately. If parsing fails, output 0.
- If $\operatorname{Hash}(Z'^{(1)}, \ldots, Z'^{(d)}) \neq V$, output 0.
- If for some $i \in [d]$, $\pi^{(i)}$ does not verify, output 0.
- If all checks pass, output 1.

BatchSol $\left(pp, \left\{ Z_j \right\}_{j \in \mathcal{S}} \right)$:

- For each $j \in \mathcal{S}$, parse each Z_j as $\left(\left\{(v_{i,j}, Z'_j^{(i)}, \mathsf{k}_j^{*,(i)}, c_j^{(i)}, \pi_j^{(i)})\right\}_{i \in [d]}\right)$.
- Let $G = (\mathcal{S}, [n_{new}], \mathcal{E})$ be a bipartite graph where

$$\mathcal{E} = \left\{ (j, v_{i,j}) : j \in \mathcal{S}, v_{i,j} \in [n_{\text{new}}] \right\}.$$

- Compute the maximal matching map ← FindMatch(G) where the matched vertices are denoted by the set S' and the mapping map = {(j, v_j^*)}_{i \in S'}. Set S_{new} = {v_j^*}_{i \in S'}.
- Let the resulting puzzle set be denoted by $\{Z_j\}_{j \in S'}$ i.e. $\{v_j^*, Z'_j, k_j^*, c_j, \pi_j\}$ (values Z'_j, k_j^*, c_j, π_j are the corresponding puzzle values associated with index v_i^*).

- Compute the sum of puzzles, *Ž* ← LHP.Eval (∑, pp_{LHP}, {Z'_j}_{j∈S'}) where ∑ indicates that a sum of puzzles is computed homomorphically.
- Solve the resulting puzzle $\tilde{k} \leftarrow LHP.Sol(pp_{LHP}, \tilde{Z})$.
- Compute $k' \leftarrow \text{Decode}_{p,\ell}(\tilde{k})$.
- For all $j \in S'$, compute

$$\mu_{j} = c_{j} + \sum_{i \in \mathcal{S}' \setminus \{j\}} \mathsf{PRF}.\mathsf{PuncturedEval}(\mathsf{pp}_{\mathsf{PRF}},\mathsf{k}_{i}^{*},i,j) - \mathsf{PRF}(\mathsf{pp}_{\mathsf{PRF}},\mathsf{k}',j) \pmod{p}$$

and set m_j as $\lfloor \mu_j \rceil_{\lceil p/2 \rceil}$.

• For all remaining puzzles $j \in S$, if there exists some puzzle $j' \in S'$ such that the index set and the puzzle match, i.e $\left(\left\{v_{i,j}\right\}_{i \in [d]}, \left\{Z'^{(i)}_{j}\right\}\right) = \left(\left\{v_{i,j'}\right\}_{i \in [d]}, \left\{Z'^{(i)}_{j'}\right\}\right)$, then compute $m_j = \lfloor \mu_{j'} - c_{j'} + c_j \rceil_{\lceil p/2 \rceil}$.

Theorem D.5. If Hash is a random oracle, NIZK is a sound non-interactive zero-knowledge proof, then the construction above satisfies security against rogue puzzle attacks according to Definition 6.1.

Proof. We start by defining a sequence of hybrid experiments:

- Hyb₀ : This is the original rogue puzzle security game.
 - Adversary \mathcal{A} outputs the time for locking puzzle *T* and a bound on the number of puzzles to be batched *n*.
 - Challenger outputs the public parameters $pp \leftarrow \text{Setup}(1^{\lambda}, T, n)$.
 - Adversary \mathcal{A} sees the public parameters and outputs a message *m*.
 - Challenger honestly generates a puzzle $Z \leftarrow \text{Gen}(\text{pp}, m)$ and outputs the puzzle Z.
 - * Let Z be parsed as $\{(v_i, Z'^{(i)}, k^{*,(i)}, c^{(i)}, \pi^{(i)})\}_{i \in [d]}$
 - Adversary receives the puzzle Z and outputs a set of puzzle $\{Z_j^*\}_{j \in \mathbb{S}^*}$ such that $|\mathcal{S}^*| \leq n$.
 - Challenger receives the set of puzzles and runs $\left\{(s_j^*, Z_j^*)\right\}_{j \in S^*} \leftarrow \text{BatchSol}(\text{pp}, \left\{Z_j^*\right\}_{j \in S^*}).$
 - * For all $j \in S^*$, parse each Z_j^* as $\left\{ (v_{i,j}, Z_j^{\prime(i)}, \mathsf{k}_j^{*,(i)}, c_j^{(i)}, \pi_j^{(i)}) \right\}_{i \in [d]}$.
 - * Let $j^* \in S^*$ be the index such that $Z_{j^*}^* = Z$ where Z is the honest puzzle.
 - * Let $G = (S, [n_{new}], \mathcal{E})$ be the bipartite graph computed.
 - * Let S' be the set such that the resulting puzzle set is denoted by $\{Z_j\}_{j \in S'}$ i.e. $\{v_j^*, Z_j', k_j^*, c_j, \pi_j\}_{j \in S'}$.
 - Adversary wins the game and the output of the experiment is b = 1, if $\forall j \in S^*$, IsValid(pp, Z_j^*) = 1 and for some $j \in S^*$, $Z_j^* = Z$, $s_j^* \neq m$. Else the experiment outputs 0.
- Hyb₁: In this hybrid, we analyze what happens if j^{*} ∈ S^{*} (the index corresponding to the honest puzzle) is not included in the maximal matching. There are two possible cases. This happens because there exists some j' ∈ S' such that ({v_i}_{i∈[d]}, {Z'⁽ⁱ⁾}) = ({v_{i,j'}}_{i∈[d]}, {Z'⁽ⁱ⁾}) or ∀j' ∈ S', ({v_i}_{i∈[d]}, {Z'⁽ⁱ⁾}) ≠ ({v_{i,j'}}_{i∈[d]}, {Z'⁽ⁱ⁾}). If it's the former we continue the game. If its the latter, the challenger outputs 0.

• Hyb₂: In this hybrid, the challenger outputs 0 if for some $i \in [d], j \in S^*$, IsValid(pp, Z_j^*) = 1, and let $k'^{(i)}_j$ = Decode_{*p*, ℓ}(LHP.Sol(pp_{LHP}, $Z'^{(i)}_j$)), but there does not exist randomness *r* such that the punctured key $k_j^{*,(i)}$ = PRF.Puncture(pp_{PRF}, $k_j^{(i)}, v_{i,j}; r$).

Lemma D.6. Let Hash be a random oracle. Then for every adversary \mathcal{A} , there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|Pr[Hyb_1(\mathcal{A}) = 1] - Pr[Hyb_0(\mathcal{A}) = 1]| = negl(\lambda)$.

Proof. The only difference in the hybrids is if the adversary is able to exclude the honest puzzle Z from the perfect matching by sampling a bunch of malicious puzzles $\{Z_j^*\}_{j \in S^*}$ and doesn't simply copy the puzzle sets $\{v_i\}_{i \in [d]}, \{Z'^{(i)}\}$. Thus to not have $j^* \in S^*$ be included in the maximal matching for graph G, and since IsValid holds, we must have sampled bad indices to break Lemma 6.3 security where q is the number of queries the adversary \mathcal{A} can make to the random oracle. Since we choose the parameters according to Lemma 6.3, this is not possible.

The rest of the proof follows identically to Theorem D.1 and we do not repeat here for brevity.