Mitigating MEV via Multiparty Delay Encryption

Amirhossein Khajehpour[1], Hanzaleh Akbarinodehi[2], Mohammad Jahanara[3], and Chen Feng[1]

[1] The University of British Columbia, Kelowna
akhajehp@mail.ubc.ca, chen.feng@ubc.ca
[2] University of Minnesota Twin Cities
akbar066@umn.edu
[3] Scroll Foundation
mohammad@scroll.io

Abstract. Ethereum is a decentralized and permissionless network offering several attractive features. However, block proposers in Ethereum can exploit the order of transactions to extract value. This phenomenon, known as *maximal extractable value* (MEV), not only disrupts the optimal functioning of different protocols but also undermines the stability of the underlying consensus mechanism.
In this work, we present a new method to alleviate the MEV problem by separating transaction inclusion and execution, keeping transactions encrypted before execution. We formulate the notion of *multiparty delay encryption* (MDE) and construct a practical MDE scheme based on time-lock puzzles. Unlike other encryption-based methods, our method excels in scalability (in terms of transaction decryption), efficiency (minimizing communication and storage overhead), and security (with minimal trust assumptions). To demonstrate the effectiveness of our MDE scheme, we have implemented it on a local Ethereum testnet. We also prove that with the presence of just one honest attestation aggregator per slot, the MEV threat can be significantly mitigated in a practical way.

Keywords: Maximal Extractable Value · Time-lock Puzzle · Multiparty Computation.

1 Introduction

Blockchain-based smart contract platforms such as Ethereum offer several attractive features, including integrity and transparency in execution, censorship resistance, and immutability. This has enabled a broad range of emerging applications [51,1] such as *decentralized finance* (DeFi). Today's mainstream blockchain platforms are designed in such a way that a block proposer has the authority to decide the ordering of transactions within a block. However, this authority enables block proposers to extract value, a phenomenon frequently referred to as *miner/maximal extractable* Value (MEV) in the literature [18]. The magnitude of MEV can, in certain instances, eclipse the conventional block rewards. As of

the time of this writing, the realized value attributed to MEV since The Merge has surged past an astonishing 300,000 ETH in price [23].

The ramifications of MEV ripple across the security and efficiency of the underlying blockchain platform [43,34]. Most notably, should a substantial portion of block proposers exploit MEV, the repercussions can be profound, including (i) destabilization of the consensus mechanism [13], (ii) centralization of the network [55], and (iii) financial inefficiencies that erode the user experience [18].

Motivated by the adverse implications of MEV, a myriad of MEV mitigation strategies have been proposed in the literature [6,55]. Among these, the concept of content-agnostic ordering [55] has emerged as one of the most promising avenues of research. In essence, this research paradigm endeavors to order transactions independently of their content. At its core, users can commit to their transactions initially and then reveal them until the order has been established. This concept can be realized through a commit-and-reveal protocol [55], leveraging either threshold encryption or time-lock encryption.

- **Threshold encryption**. This approach utilizes a key management committee with an honest majority (or super-majority). Users encrypt their transactions using the committee's public key, forcing block proposers to determine the transaction order without knowing the content. The committee then performs threshold decryption of the transactions and, subsequently, transactions will be executed. Noteworthy projects such as Shutter network [48], FairBlock [37], and several others [57,4,40,49] have embraced threshold encryption. However, this approach has been criticized due to fundamental vulnerabilities it may introduce into the system [46]. In particular, it relies on a considerable portion of parties acting honestly and suffers from communication/storage overhead.
- **Time-lock encryption**. In this approach, transactions remain encrypted until a predetermined future moment when they can be automatically decrypted through some mathematical computation. Unlike threshold encryption, time-lock encryption eliminates the need for an explicit reveal phase, thereby mitigating certain attack vectors (e.g., participants aborting the reveal phase [29,16,47]). Nevertheless, current time-lock encryption methods encounter challenges in efficiently handling transaction decryption [29,6,32], or impose additional communication/storage overhead on the network [10,21].

In this paper, we aim to push time-lock encryption to the next level by introducing a new protocol designed to overcome the limitations of current methods. As we will soon see, our protocol is highly scalable in terms of transaction decryption and bears minimal communication/storage overhead. Unlike threshold encryption, our protocol doesn't require the honest-majority assumption, relying solely on the presence of at least one honest party. Moreover, our protocol seamlessly integrates with the Ethereum ecosystem through a straightforward workflow. Commencing with a one-time setup phase, our protocol engages a randomly generated committee of attesters (attestation aggregators) for each slot. These attesters propose their shares of the current round public key along with a time-lock puzzle concealing their shares of the secret key. Subsequently,

block proposers build blocks from encrypted transactions while simultaneously deciphering transactions from previous slots' time-lock puzzles.

The main contributions of this work can be summarized as follows.

- **Formalization of k-MEV safe network.** The MEV problem is formally characterized building upon prior work [16,38] and the concept of k-MEV safe domain is introduced based on the notion of input causality [44].
- **Multiparty delay encryption.** The notion of *multiparty delay encryption* (MDE) is introduced. Then, with leveraging cryptographic time-lock puzzles, an MDE is designed with its security rigorously established under standard cryptographic assumptions.
- **Scalable and efficient MEV mitigation.** It is demonstrated that the proposed multiparty delay encryption can fully mitigate the MEV problem within the defined k-MEV safe domain. Importantly, the safety of this method endures even with the presence of just one honest attestation aggregator in today's Ethereum design. Additionally, the communication and storage costs of our method are negligible and its scalability in handling transactions is unrivaled.
- **Implementation and feasibility.** Our method is successfully implemented in an Ethereum local network, demonstrating the feasibility of the proposed solution.

The rest of this paper is organized as follows: In Section 2, we will discuss the limitations of current time-lock encryption methods and then give an overview of our method. In Section 3, we will formally introduce k-MEV safe network and define our threat model. Section 4 focuses on preliminaries and notations necessary for the rest of the paper. In Section 5, we will explain how we construct a new time-lock puzzle primitive based on the previous works, and then use this primitive to build our MDE. In Section 6, we will show the implementation details of adding MDE to Ethereum consensus. Later, Section 7 briefly touches upon other mitigation approaches and discusses their limitations compared to our work. Finally, Section 8 concludes this paper and suggests potential future directions.

2 Technical Overview

The initial idea of time-locked puzzles was introduced by Rivest *et al.* which utilizes the sequential squaring assumption [45]. Informally speaking, time-lock puzzles allow us to hide a message inside a mathematical structure and prove that the message content only can be recovered after performing a time-consuming calculation which does not have any shortcut nor is parallelizable. Our problem then can be informally described as finding an efficient way to hide an arbitrary number of messages (transactions in our case) via time-lock puzzles. Using the initial version, however, if we hide more than one message with each inside separate puzzles, then the computation needed to uncover all of them scales linearly with the number of messages. Ideally, we want to be able to recover

all the messages with the cost of only solving one puzzle. In what follows, we will explain why the existing approaches fail to achieve this objective and then introduce our new method.

2.1 Challenges With Using Existing Approaches

With recent advancements, the seminal work of [33] enabled participants to have homomorphic properties in the puzzle space which laid the basis for many future works. Homomorphism allows us to evaluate a circuit over the puzzle space; meaning that, only by solving one single puzzle, the final output of the circuit will be revealed. Nevertheless, even via homomorphism we can not recover all the messages simultaneously, a notion called, batch-solving. Thyagarajan *et al.* [52] proposed extending the message space from $\mathbb{Z}_N$ to $\mathbb{Z}_{N^n}$ to enable the puzzle batch-solving method, where n as the maximum number of puzzles to be batched together. In their method, the number n is fixed during setup phase which restricts the system's ability to accommodate additional participants after it is setup. Furthermore, using large puzzle sizes leads to increased communication costs by a linear factor as well [50].

Later, Srinivasan *et al.*[50] proposed another approach which generates compact puzzle; thus, supporting unbounded batch-solving and having final puzzle size independent of the batch size. Via their method one can solve many arbitrary puzzles at the cost of solving one puzzle; however, because of relying on Indistinguishable Obfuscation (IO) constructions, it is yet far from being practical [25].

In a recent study by [10], a fascinating concept of *delay encryption* was introduced. Instead of hiding messages inside separate time-lock puzzles and consolidating them into a single puzzle, the authors proposed a method that involves hiding a secret key inside a time-lock puzzle, which corresponds to a publicly known public key. Consequently, any participant within the system can encrypt their messages using the public key. Once the puzzle is solved, the solver gains the ability to decrypt all the messages simultaneously. Notably, the solving algorithm only needs to tackle a single puzzle. However, there are significant drawbacks to their approach. First, the storage requirements for computing the decryption key are substantial and increase with the amount of delay. Second, the setup time for their system increases proportionally with the delay as well, resulting in substantial costs [36].

Similarly, Doweck*et al.* [21] have used a combination of delay encryption and MPC where a committee of parties with the help of a coordinator create an ElGamal public key. Then everyone can encrypt their messages via the public key. Later, the coordinator will solve the discrete logarithm problem and find the value of the secret key. Once the secret key is found, the coordinator can decrypt all the messages. However, their approach relies on the computational and parallelization assumption of all the participants, and their construction only guarantees the expected mean value of the delay and does not guarantee the required delay for every case.

2.2 Our Approach

Similar to the approach taken by [10,21], we introduce the notion of Multiparty Delay Encryption (MDE). MDE's core idea is to generate and publish the pair (pk, ek), where pk is a public key with its secret key sk hidden inside a time-lock puzzle which can be solved with the help of the extraction key ek. Consequently, any message m that were encrypted via pk is kept secret until the time-lock puzzle is solved and the value of sk is extracted.

MDE requires an initial one-time setup phase that can be run in a trustless or distributed fashion. Subsequently, the protocol, when applied to the Ethereum network, for each slot commences with a puzzle generator committee $[p_0, .., p_{n-1}]$ consisting of n distinct validators. Each committee member p_i will then broadcast its puzzle shares through the network denoted by the pair (pk_i, ek_i) as described before. Once all the puzzle shares (pk_i, ek_i) are received by the block proposers, they will combine them into a single puzzle $(\overline{pk}, \overline{ek})$ and broadcast it. At this point on, all participants in the Ethereum network have the ability to encrypt their transactions via $\overline{pk}$. Additionally, the next block proposers include the encrypted transactions into the new blocks without executing them. Simultaneously, future block proposers start to solve the proposed aggregated puzzle using $\overline{ek}$. The encryption ensures that the transactions remain secure and confidential until the appropriate time for their execution. Finally, once the solution $\overline{sk}$ is revealed, the next block proposer in line unveils the transactions' contents using $\overline{sk}$ and executes them in order. This ensures that the execution order of the transactions within a block is done securely and efficiently. Figure 2.2 illustrates this process. In general, for each slot, its block proposer is in charge of two tasks: (i) including the transactions encrypted with the new public key and (ii) executing old blocks' transactions after solving their time-lock puzzles.

Multiparty Delay Encryption We now will provide a generalized overview of the process involved in designing an MDE upon the time-lock puzzle concepts introduced in the works of [33,31]. The informal definition of MDE is as follows (with its formal definition provided in Section 4).

Definition 1. ***Multiparty Delay Encryption. Informal.*** *MDE has an initial* ***setup*** *phase to generate the public parameters of the system. Following this, a committee consisting of untrusted members is responsible for individually* ***generating*** *MDE shares, which are subsequently* ***aggregated*** *into a final share. To ensure the integrity of the shares, each member is also required to publish a* ***verification*** *proof of their share generation, demonstrating adherence to the protocol and the absence of corruption. Each share comprises a public key and an extraction key, with the latter concealing the secret key. The public key serves the purpose of message* ***encryption*** *at a later stage. Within this protocol, any participant has the capability to* ***extract*** *the private key using the aggregated extraction key. Ultimately, the private key is employed to* ***decrypt*** *all the messages.*

We need to define two properties for MDE.

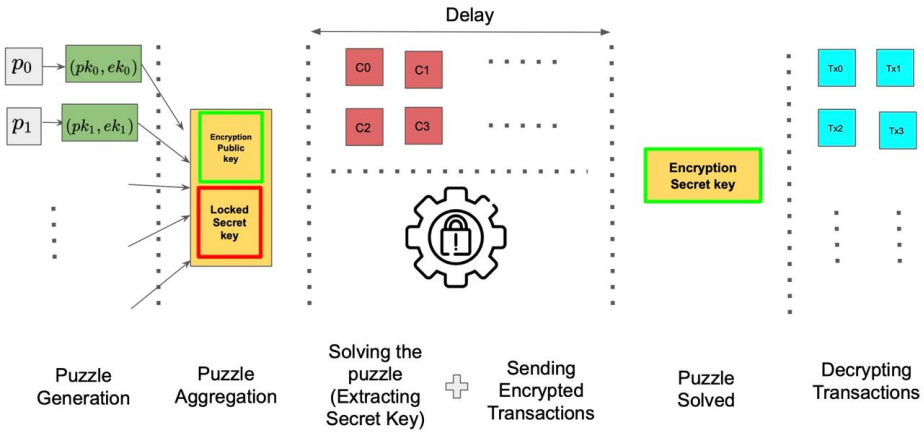

Fig. 1. An overview of the MDE protocol design in today's Ethereum model. In the beginning, puzzle generators (AttestionAggregators) broadcast their puzzle shares through the attestations. Then, the block proposer combines all the puzzles into one. Later, future block proposers, after observing the puzzle, start the solver procedure. Meanwhile, everyone in the network can encrypt their transactions for the time-lock public key. Finally, after revealing the solution, the next block proposer will execute and reveal the encrypted transactions.

Definition 2. ***MDE Correctness. Informal.*** *If all the shares pass the* ***verification****, then for all the messages, the* ***decryption*** *and the* ***encryption*** *must work properly.*

Definition 3. ***MDE Security. Informal.*** *With having at least one honest party in the committee which* ***generates*** *the shares, the result of* ***encryption*** *of any two messages must be indistinguishable any time earlier than the time is needed to* ***extract*** *the secret key.*

Construction To build the MDE, we start by studying time-lock puzzle itself. According to [33], the tuple $(u, v) \leftarrow (g^r \mod N, h^{rN}(1+N)^s \mod N^2)$ represents a time-lock puzzle hiding the secret s given the randomness r, g the generator of $\mathbb{J}_N$, $h = g^{2^T} \mod N$, RSA safe prime modulus N, and the delay parameter T. Later, the puzzle can be solved via calculating $x = u^{2^T} \mod N$ then $s = (v/x^N \mod N^2 - 1)/N$. Additionally, it is easy to verify that the puzzle pairs are linearly homomorphic. More precisely, assume that we have two puzzles (u_0, v_0) and (u_1, v_1) hide the secrets s_0 and s_1 respectively, then the puzzle (u_0u_1, v_0v_1) hides the secret $s_0 + s_1$.

Remember that we later need to generate the pair (pk, ek) where the secret key sk corresponding to the public key pk is hidden within a time-lock puzzle

which can be solved via the extraction key ek. The first portion of each puzzle $u = g^r$ is a valid ElGamal public key in $\mathbb{J}_N$ and can be considered as pk for $sk = r$. Unfortunately, via this construction, there is no way to extract sk after solving the puzzle. Therefore, we need to modify it to meet all the necessary requirements of our MDE.

In our modified version, each puzzle consists of the tuple $(u, v, y, w) \leftarrow (g^{r+s} \mod N, h^{r+s}N(1+N)^s \mod N^2, g^k \mod N, h^{kN}(1+N)^r \mod N^2)$ where k is another randomness factor. In this version, the portion $u = g^{r+s} \mod N$ is considered as the public-key pk and the tuple itself (u, v, y, w) as the extraction key ek. But this time, its secret key $sk = r + s$ can be extracted. First, the solver calculates $\bar{u} = u * y$, $\bar{v} = v * w = h^{(r+s)kN}(1 + N)^{r+s} \mod N^2$, and $x = \bar{u}^{2^T} \mod N$. Second, the value of $r + s$ can be accessed via $((\bar{v}/x^N \mod N^2) - 1)/N$. We call our modified construction Linearly Homomophic Time-lock Puzzle (LHTP) and its details are presented in Appendix D. Also, we showed that it is secure under the standard cryptographic assumptions, DCR and strong sequential squaring.

To build MDE, we can assume that the public key pk is u, the extraction key ek is (u, v, y, w), and the hidden secret key sk is $r + s$. Next, in order to satisfy the safety property, we will utilize the homomorphic property of the time-lock puzzle. In other words, assuming each party p_i proposes the share consisting of the public key $pk_i = u_i$ and extraction key $ek_i = (u_i, v_i, y_i, w_i)$, the aggregated public key and secret key will be $\overline{pk} = \Pi u_i$ and $\overline{sk} = \sum r_i + s_i$. Similarly, the aggregated extraction key will be $\overline{ek}_i = (\Pi u_i, \Pi v_i, \Pi y_i, \Pi w_i)$. This way, we can prove that the value of the aggregated secret key will remain hidden as long as we have at least one honest party. To satisfy the correctness property, we designed a non-interactive zero-knowledge protocol similar to the approach of [31] which is shown in Figure 5.

3 System Model

The concept of MEV was first formalized by [16]. Then, it was extended to cross-domain space in the work of [38]. In our work, we will adopt both of the mentioned formalizations. Finally, we will utilize *input causality* [44] to introduce the notion of k-MEV safe domain. First, we start by recalling the definition of *domain* from [38]:

Definition 4. ***Domain.*** *A* domain χ *is a tuple* $(\mathcal{P}, \mathcal{A}, \mathcal{S})$ *which is a self-contained system with a globally shared* state *where* $\mathcal{S}$ *shows the space of all the possible states in* χ*. This state is mutated by various players through actions (often referred to as "transactions"), that execute in a shared execution environment's semantics where* $\mathcal{A}$ *is the set of all possible actions and the set* $\mathcal{P}$ *contains all the players in* χ*.*

Then, we can define an *action* adopted from [38]:

Definition 5. ***Action.*** *Given the domain* $\chi = (\mathcal{P}, \mathcal{A}, \mathcal{S})$*, an action* $a \in \mathcal{A}$ *is a mapping of a state to another state. For* $n \in \mathbb{N}$ *we denote by* $\mathsf{apply}(s, [a_1, .., a_n])$

the result of applying the sequence of actions $[a_1, .., a_n]$ *to the state* s*, where* $a_i \in \mathcal{A}$*. We also show the set of available actions corresponding to the player* p *via* $\mathcal{A}_p$*.*

Since the systems that we are concerned about are blockchains, we also need to introduce the concept of *block* into our model:

Definition 6. ***Block****. Given the domain* $\chi = (\mathcal{P}, \mathcal{A}, \mathcal{S})$*, a block* $b \in \mathcal{A}^*$ *is a ordered sequence of actions* $[a_1^b, .., a_{\ell_b}^b]$ *for* $\ell_b \in \mathbb{N}$ *and* $a_i^b \in \mathcal{A}$*. Blocks have limited size and each might have their own metadata according to the domain constraints. We denote by* $\mathsf{bapply}(s, b)$ *the result of applying actions of the block* b *to the state* s*; i.e.* $\mathsf{apply}(s, [a_1^b, .., a_{\ell_b}^b])$*. Moreover, we use a similar notation* $\mathsf{bapply}(s, [b_1, ..., b_j])$ *to denote result of applying a sequence of blocks* $[b_1, ..., b_j]$ *on a state* s

Below, we will use the modified version of the *sequencer* from [38]:

Definition 7. ***Sequencer****. A* sequencer *is a player that orders actions and bundles them into blocks, and thus influences future states of the domain.*

Therefore, the current state can be seen as the result of applying a sequence of blocks over the domain's initial state.

Inspired by [16], we will define the notion k-MEV, which, informally speaking, refers to the profit a player can make, if they mine k blocks using a specific set of actions based upon a given state. In a formal sense:

Definition 8. k-MEV*. Let the domain* $\chi = (\mathcal{P}, \mathcal{A}, \mathcal{S})$*. Assume we show the balance of the party* $p \in \mathcal{P}$ *at the state* $s \in \mathcal{S}$ *via* $\mathsf{balance}(s, p)$*. Then, the* k-MEV *of* p *starting from the state* s *via the set of actions* $A' \subset \mathcal{A}$ *will be defined as follows:*

$$\mathsf{MEV}^k_{A'}(p, s) = \max\ \mathsf{balance}(\mathsf{bapply}(s, [b_1, .., b_k]),\ p) - \mathsf{balance}(s,\ p) \quad (1)$$

where b_i *for every* $i \in \mathbb{Z}_k$ *is a block with* $[a_1^{b_i}, .., a_{\ell_{b_i}}^{b_i}]$ *as its actions and the maximum is getting over every choice of* $a_j^{b_i} \in A'$ *for* $j \in \mathbb{Z}_{\ell_{b_i}}$*.*

We can now define our threat model and explain the power of the adversary in our system. We assume that an arbitrary number of players can collude and attempt to increase their payoff strategically; however, they have to follow the domain's constrains. For example, the stake of the players deviating from the protocol can not exceed 1/3 of the entire stake in the Ethereum network. Now, given the domain $\chi = (\mathcal{P}, \mathcal{A}, \mathcal{S})$, assume $P \subset \mathcal{P}$ is a coalition of players in χ. Then, we denote all the possible actions that the coalition has access to via $\mathcal{A}_P \triangleq \bigcup_{p \in P} \mathcal{A}_p$. We can also define the $\mathsf{balance}$ and k-MEV of P in a similar way. Furthermore, similar to the idea of *input causality*, we will define the k-MEV safe domain. In simple terms, we allow an adversarial coalition to corrupt the sequencer for a continuous sequence of k blocks. If, during this period, the coalition's MEV does not surpass what they would normally gain, the domain is referred to as k-MEV safe.

Definition 9. k-MEV ***Safe***. *Given the domain $\chi = (\mathcal{P}, \mathcal{A}, \mathcal{S})$, any coalition of players P where $P \subset \mathcal{P}$, any state s where $s \in \mathcal{S}$, and $A' \subset \mathcal{A}$ as all the actions that the sequencer has received from other players which non of them has yet been applied to s*[4]*. χ is a k-MEV safe domain if:*

$$\mathsf{MEV}^k_{\mathcal{A}_P}(P, s) = \mathsf{MEV}^k_{A' \cup \mathcal{A}_P}(P, s) \tag{2}$$

Now, we are ready to explain how an k-MEV safe domain can protect Ethereum from some of the popular MEV instances.

- Front-running: According to the k-MEV safe definition for $k \geq 1$, there is no motivation for any player to alter the transaction order within the current block. Consequently, there would be no benefit to prioritize one's transaction ahead of a target transaction. Therefore, front-running is no longer a concern.
- Back-running: Just like front-running, placing a transaction immediately after a target transaction is no longer considered a valid strategy. Furthermore, due to the unknown state of the preceding k blocks, there is a high likelihood that the back-running opportunity may no longer exist, rendering the attack economically unjustifiable. Additionally, in order to seize back-run opportunity, every player would strive to take the first position in the $k + 1^{th}$ block, which can be relatively expensive due to the competition. Other than that, the profit of back-running is already marginal compared to front-running attacks; thus, further discouraging players from engaging in this attack.
- Sandwich attack: Similar to the argument regarding front-running, within a k-MEV secure domain, there is no motivation for performing a sandwich attack.
- PGA: By removing the incentives for front-running, back-running, and sandwich attacks, thereby significantly reducing the competition for transaction slots within a block. This, in turn, eliminates the PGA threat as well.
- Time-bandit and undercutting attacks: As the value of k increases, the economic viability of conducting time-bandit and undercutting attacks diminishes exponentially, eventually rendering them infeasible [20,24].

4 Priliminaries

4.1 Notation

We use $x \leftarrow\!\!\$\ \mathcal{S}$ to denote that x is uniformly sampled from the set $\mathcal{S}$. $||$ is used for concatenation. $|\mathcal{S}|$ for the set $\mathcal{S}$ indicates the size of $\mathcal{S}$. $|N|$ for the number N refers to the binary bit length of N. $||G||$ for the group G refers to the order of G. For the group G and $\alpha \in G$, the notion $ord_G(\alpha)$ refers to the order of α in G, and, $\langle\alpha\rangle$ refers to the subgroup generated by α. The group $\mathbb{J}_N = \{x | \left(\frac{x}{N}\right) = +1\}$ refers to all the elements whose Jacobi symbols with N are $+1$. Note that $||\mathbb{J}_N|| = 1/2||\mathbb{Z}_{N^*}||$. The notion $\mathsf{negl}(\lambda)$ refers to a negligible function where $\mathsf{negl}(\lambda) < 1/p(\lambda)$ for every polynomial p. Two groups G_1 and G_2 are denoted as isomorphic via $G_1 \cong G_2$.

[4] More precisely, actions from A' has not been included into any blocks up until the state s, from the initial state.

4.2 Syntax

According to what was mentioned, below we will formally define the functionalities and requirements of an MDE scheme as follows:

Definition 10. ***Multiparty Delay Encryption****. Let $\mathcal{M}$ be the message space. Then, a multiparty delay encryption is a protocol that runs among an arbitrary number of parties which consists of the tuple (*Setup, Gen, Verify, Aggregate, Encrypt, Decrypt, Extract*) such that:*

- Setup$(1^\lambda, \mathcal{T})$*: A probabilistic algorithm that on receiving the security parameter 1^λ outputs the system's public parameters pp according to the delay $\mathcal{T}$. Setup runs only once at the beginning and can be a potentially distributed algorithm.*
- Gen(n, pp)*: A probabilistic algorithm that, on receiving the system's public parameters pp, and $n \in \mathbb{N}$ the maximum number of participants in generating the final public-key, outputs the tuple (ek, pk) such that ek is the extraction key corresponding to the encryption public-key pk.*
- Verify(n, ek, pp)*: A probabilistic protocol that checks the construction of ek. If it is according to the protocol, it returns true; otherwise, it returns false. It also needs n, the maximum number of participants, for some verifications.*
- Aggregate$((ek_1, pk_1), (ek_2, pk_2), .., (ek_k, pk_k), pp)$*: A deterministic algorithm that receives system's public parameters pp, and many (ek_i, pk_i) tuples. It returns an aggregated $(\overline{ek}, \overline{pk})$, which will later be used for message encryption.*
- Encrypt(m, pk)*: A probabilistic algorithm which on receiving the message $m \in \mathcal{M}$ and the public-key pk, outputs $c \in C$, the encryption of m, according to pk.*
- Decrypt(c, sk)*: A deterministic algorithm that decrypts the ciphertext c via the decryption key sk and returns a message $m \in \mathcal{M}$.*
- Extract$(ek, \mathcal{T}, pp)$*: A deterministic algorithm which receives the extraction key ek and system's public parameters pp along with delay $\mathcal{T}$. Then, outputs the decryption key sk that corresponds to ek or $\perp$.*

Definition 11. ***MDE Correctness****. The MDE protocol $\Pi =$ (Setup, Gen, Verify, Aggregate, Encrypt, Decrypt, Extract) is correct if given λ, for all polynomials $\mathcal{T}$ in λ, all messages $m \in \mathcal{M}$, all $n \in \mathbb{N}$ that $n = p_0(\lambda)$ for a fixed polynomial p_0, all pp in the support set of Π.Setup$(n, 1^\lambda, \mathcal{T})$, and all (ek_i, pk_i) where* Verify$(n, ek_i, pp) =$ *true, given* $(\overline{ek}, \overline{pk}) \leftarrow \Pi$.Aggregate$((ek_0, pk_0), .. (ek_{n-1}, pk_{n-1}), pp)$*, and* $\overline{sk} \leftarrow \Pi$.Extract$(\overline{ek}, pp)$*, the running time of the algorithm Π.*Extract$(1^\lambda, \mathcal{T}, \bar{z})$ *is bounded with $p_1(\lambda, \mathcal{T})$ for a fixed polynomial p_1 and we have:*

$$\Pi.\mathsf{Decrypt}(\Pi.\mathsf{Encrypt}(m, \overline{pk}), \overline{sk}) = m \tag{3}$$

Definition 12. ***MDE Security****. The MDE protocol $\Pi =$ (Setup, Gen, Verify, Aggregate, Encrypt, Decrypt, Extract) is secure if for any probabilistic polynomial*

time adversary $(\mathcal{A}_1, \mathcal{A}_2)$ *which the depth of* $\mathcal{A}_2$ *is bounded by* $T^\epsilon(\lambda)$ *from above where* T *is a polynomial,* $0 < \epsilon < 1$*, and all* $n = p_0(\lambda)$ *for a fixed polynomial* p_0 *such that:*

$$\Pr\left[\mathcal{A}_2(1^\lambda, \overline{ek}, c, pp, \tau) = b \;\middle|\; \begin{array}{c} pp \leftarrow \Pi.\mathsf{Setup}(n, 1^\lambda, \mathcal{T}), \\ ((m_0, m_1), (ek_1, pk_1), .. \\ (ek_{n-1}, pk_{n-1}), \tau) \leftarrow \mathcal{A}_1(T, pp), \\ (ek_n, pk_n) \leftarrow \Pi.\mathsf{Gen}(n, pp), \\ b \leftarrow\!\$\ \{0,1\} \\ (\overline{ek}, \overline{pk}) \leftarrow \Pi.\mathsf{Aggregate}((ek_1, pk_1), \\ .., (ek_{n-1}, pk_{n-1}), (ek_n, pk_n)), pp), \\ c \leftarrow \Pi.\mathsf{Encrypt}(m_b, \overline{pk}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

5 Multiparty Delay Encryption

In this section, we will present some details of MDE as well as its security and correctness proofs.

Algorithm 1 *Assuming* Π *is an LHTP protocol and* Ψ *is any zero-knowledge exponent range proof protocol such that* $\Psi.\mathsf{Verify}(b,\ x,\ a,\ N)$ *checks that given* $x = b^e \mod N$ *whether* e *is less than* a *or not. Then, the description of algorithm is as follows:*

- $\mathsf{Setup}(1^\lambda, T)$*: Runs* $pp \leftarrow \Pi.\mathsf{Setup}(1^\lambda, T)$ *and returns* pp*.*
- $\mathsf{Gen}(n, pp)$*: It works similar to the* $\Pi.\mathsf{Gen}$ *with a difference in choosing the boundries of* s *and* r*. Additionally, it constructs a valid public key as the multiplication of* u *and* y *as follows. More precisely, it samples* $s, r \leftarrow\!\$\ \mathbb{Z}_{N/2n}$ *and* $k \leftarrow\!\$\ \mathbb{Z}_{N/2}$*. Then, outputs* (ek, pk) *such that:* $ek = (u, v, y, w)$*,* $pk = u$ *for* $u = g^{r+s} \mod N$*,* $y = g^k \mod N$*,* $v = h^{(r+s)N}(1+N)^s \mod N^2$*, and* $w = h^{kN}(1+N)^r \mod N^2$*.*
- $\mathsf{Verify}(n, ek, pp)$*: Let* ek *compiles into* (u, v, y, w)*. First, it checks that* u *and* $y \in \mathbb{J}_N$*, and* $v,\ w \in \mathbb{J}_{N^2}$*. Then, checks the exponent range of* u *to be in* $\mathbb{Z}_{N/n}$ *via* $\Psi.\mathsf{Verify}(g, u, N/n, N)$*. Next, the prover samples* $x \leftarrow\!\$\ \mathbb{Z}_{(N/2+N/n)*2^{2\lambda}}$*,* $t \leftarrow\!\$\ \mathbb{Z}_{(N/n)*2^{2\lambda}}$ *and sends* $a = g^x \mod N$*,* $b = h^{xN}(1+N)^t \mod N^2$ *along with* $\tau = g^t \mod N$ *to the verifier. Then, on receiving* $e \leftarrow\!\$\ \mathbb{Z}_{2^\lambda}$ *from the verifier, returns* (α, β) *for* $\alpha = (r+s+k)e + x$ *and* $\beta = (r+s)e + t$*. Finally, to verify the puzzle* (u, v)*, returns* $g^\alpha = (uy)^e * a \ \wedge\ h^{\alpha N}(1+N)^\beta = (vw)^e b \ \wedge\ g^\beta = u^e * \tau$*.*
- $\mathsf{Encrypt}(m, pk, pp)$*: Samples* $r \leftarrow\!\$\ \mathbb{Z}_{N/2}$ *and returns* (c_1, c_2) *with* $c_1 = g^r$ *and* $c2 = m * pk^r$*.*
- $\mathsf{Decrypt}((c_1, c_2), sk, pp)$*: Returns* c_2 / c_1^{sk}*.*
- $\mathsf{Aggregate}((ek_1, pk_1), ..(ek_n, pk_n), pp)$*: Runs* $\overline{ek} = \Pi.\mathsf{Eval}(ek_1, ..ek_n, pp)$*, and returns* $(\overline{ek}, \overline{pk})$ *for* $\overline{pk} = \Pi pk_i$*.*
- $\mathsf{Solve}(\overline{ek}, pp)$*: Compiles* $\overline{ek}$ *into the tuple* (u, v, y, w) *and returns* $\Pi.\mathsf{Solve}((uy,\ vw,\ \perp,\ \perp),\ pp)$*.*

Now, we prove the main theorem of this section:

Theorem 1. *Given that Π is a secure time-lock puzzle, Algorithm 1 is a secure multiparty delay encryption.*

Proof. **Correctness**: First of all, we need to show that with the Verify procedure, a party proves the knowledge of sk for a given ek in zero-knowledge. Its corresponding relation is:

$$\mathcal{R} = \{(N, g, h, u, v, y, w) : \exists r, s, k \in \mathbb{N} \mid \ r + s \in \mathbb{Z}_{N/2n} \ \wedge u = g^{r+s} \\ \wedge\ y = g^k \ \wedge\ vw = h^{N(r+s+k)}(1+N)^{r+s} \mod N^2\}$$

Therefore, we state the below theorem whose proof is in Appendix F:

Theorem 2. *Algorithm 1's* Verify *procedure is a public-coin honest-verifier zero-knowledge proof corresponding to the relation $\mathcal{R}$.*

Note that the Verify is easily convertible to *malicious-verifier zero-knowledge proof* via the Fiat-Shamir heuristic. Figure 5 shows the Gen and Verify procedures of Algorithm 1.

Next, to prove the correctness of Theorem 1, assume there are n number of shares with the extraction, and public keys $ek_i = (u_i, v_i, y_i, w_i)$ and pk_i, where $\forall_i$ $\mathsf{Verify}(n, ek_i, pk_i) = true$, we can conclude that there exists $r_i, s_i, k_i \in \mathbb{N}$ where $r_i + s_i \in \mathbb{Z}_{N/2n}$ such that $u_i = g^{r_i+s_i},\ y_i = g^{k_i},\ v_i w_i = h^{r_i+s_i+k_i}(1+N)^{r_i+s_i} \mod N^2$ as the result of Theorem 2. Let $\overline{ek} = (\bar{u}, \bar{v}, \bar{y}, \bar{w}) = \mathsf{Aggregate}((ek_1, pk_1), .. (ek_n, pk_n))$. Then, the output of $\mathsf{Solve}(\overline{ek}, pp)$ will be $\bar{s} \leftarrow \Pi.\mathsf{Solve}(z, pp)$ for $z = (\bar{u}\bar{y}, \bar{v}\bar{w}, \perp, \perp)$. Additionally, according to Lemma 12, $\bar{s} = \sum r_i + s_i \mod N$. Also, we know that $r_i, s_i \in \mathbb{Z}_{N/2n}$. Therefore, $\sum r_i + s_i < \mathbb{Z}_N$. Thus, Solve gives us $\sum r_i + s_i$. Moreover, $\overline{pk} = g^{\sum s_i, r_i}$ and consequently, $\overline{pk} = g^{\overline{sk}}$. Finally, given a message $m \in \mathcal{M}$, $\mathsf{Decrypt}((c_1, c_2), sk, pp)$ for $(c_1, c_2) = \mathsf{Encrypt}(m, pk, pp)$:

$$c_2 / c_1^{\overline{sk}} = m * \overline{pk}^{\,r} / g^{r * \overline{sk}} = m * g^{\overline{sk}} / g^{\overline{sk}} = m$$

Security: We start by stating another variant of the security for Algorithm 1 whose proof is in Appendix G:

Lemma 1. *Given a MDE algorithm Π, for any* PPT *adversary $\mathcal{A}$ whose depth is bounded by $T^\epsilon(\lambda)$ from above where T is a polynomial, $0 < \epsilon < 1$, and $n = p_0(\lambda)$ for a fixed polynomial p_0:*

$$\Pr\left[\mathcal{A}(1^\lambda, z_b, pp) = b \,\middle|\, \begin{array}{c} pp \leftarrow \Pi.\mathsf{Setup}(n, 1^\lambda, T), \\ b \leftarrow\!\$\ \{0,1\},\ z_0 \leftarrow \Pi.\mathsf{Gen}(n, pp), \\ z_1 \leftarrow\!\$\ \mathbb{J}_N \times \mathbb{J}_{N^2} \times \mathbb{J}_N \times \mathbb{J}_{N^2} \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda) \quad (4)$$

Lemma 1 shows that the tuple (ek, pk, pp) is indistinguishable from a uniform tuple for the bounded adversary $\mathcal{A}_2$. According to MDE security Definition 12,

MDE $pp = (N, g, T, h)$

	Prover		**Verifier**
	$\mathsf{Gen}(n, pp)$:		
	$s \leftarrow\$ \mathbb{Z}_{N/2n}$		
	$r \leftarrow\$ \mathbb{Z}_{N/2n}$		
	$k \leftarrow\$ \mathbb{Z}_{N/2}$		
	$u \leftarrow g^{r+s} \mod N$		
	$v \leftarrow h^{(r+s)N}(1+N)^s \mod N^2$		
	$y \leftarrow g^k \mod N$		
	$w \leftarrow h^{kN}(1+N)^r \mod N^2$		
		$\xrightarrow{(u,v,y,w)}$	
	Verify(n, ek, pp)**:**		
	$(u, v, y, w) \leftarrow ek$		
			Verify that:
			$u \in \mathbb{J}_N$
			$v \in \mathbb{J}_{N^2}$
			$y \in \mathbb{J}_N$
			$w \in \mathbb{J}_{N^2}$
	$x \leftarrow\$ \mathbb{Z}_{(N/2+N/n)*2^{2\lambda}}$,		
	$t \leftarrow\$ \mathbb{Z}_{N/n*2^{2\lambda}}$		
	$a \leftarrow g^x \mod N,\ b \leftarrow h^{xN}(1+N)^t \mod N^2$		
	$\tau \leftarrow g^t \mod N$		
		$\xrightarrow{(a,b,\tau)}$	
			$e \leftarrow\$ \mathbb{Z}_{2^\lambda}$
		$\xleftarrow{e}$	
	$\alpha \leftarrow (r+s+k)e + x$		
	$\beta \leftarrow (r+s)e + t$		
		$\xrightarrow{(\alpha,\beta)}$	
			Verify that:
			$g^\alpha = (uy)^e * a$
			$\wedge\ h^{\alpha N}(1+N)^\beta = (vw)^e b$
			$\wedge\ g^\beta = u^e * \tau$
			$\wedge\ a, \tau \in \mathbb{J}_N\ \wedge\ b \in \mathbb{J}_{N^2}$
			$\wedge\ \alpha \in \mathbb{Z}_{(N/2+N/n)*2^\lambda + (N/2+N/n)*2^{2\lambda}}$
			$\wedge\ \beta \in \mathbb{Z}_{N/n*2^\lambda + N/n*2^{2\lambda}}$

Fig. 2. MDE schematic

we know that $(\overline{ek}, \overline{pk}) \leftarrow \Pi.\mathsf{Aggregate}((ek_1, pk_1), (ek_2, pk_2), .. (ek_n, pk_n), pp)$. Following the result of Lemma 1, we can replace (ek_n, pk_n) with a random sample from $\mathbb{J}_N \times \mathbb{J}_{N^2} \times \mathbb{J}_N \times \mathbb{J}_{N^2}$. Therefore, the tuple $(\overline{ek}, \overline{pk})$ is also indistinguishable from a random tuple for any $\mathcal{A}_2$. It is obvious to show that the output of $\Pi.\mathsf{Encrypt}(m, pk, pp)$ will also remain random which completes the proof.

6 Implementation

In this section, we explain how we can adopt MDE to design an k-MEV Ethereum network. First, we will review some of the necessary sub-components of Ethereum post-merge network. Then, we will give a high-level overview of our implementation design and describe the changes needed to apply to Ethereum's consensus layer. Finally, we will prove that our design is indeed an k-MEV safe network and discuss how to choose an appropriate delay value.

6.1 Ethereum Background

In Ethereum network, after The Merge, time is partitioned into predetermined units called slots, where a single validator is randomly chosen in each slot to propose a block. At each slot [5] attestation committees are formed by randomly grouping validators together, and they collaborate to vote and provide attestations on blocks. Each validator of an attestation committee individually provides votes via attestations for consensus mechanisms [12]. Attestations are signed with BLS signatures 3, hence, the ones that share an identical vote can be instantly aggregated into a single attestation by BLS signature aggregation. In each committee a subset of validators are selected at random to perform the task of aggregation, and they are denoted as aggregators.

6.2 Design

In order to apply MDE to today's Ethereum network, we need to introduce several new data structures.

Data Structures Below are the necessary data structures for Algorithm 1.

```
class MDEPublicParams:
  N:      bytes # size:"lambda/4"
  G:      bytes # size:"lambda/4"
  T:      bytes # size:"8"
  H:      bytes # size:"lambda/4"
```

[5] They are often called Beacon committees link

```
class MDEVerifyProof:
  A:      bytes # size:"lambda/4"
  B:      bytes # size:"lambda/2"
  Alpha:    bytes # size:"lambda/2+1"
  Beta:   bytes # size:"lambda/2"
  Tau:    bytes # size:"lambda/4"
```

```
class MDEVerifiableShare:
  Share:      MDEShare
  Share:      MDEVerifyProof
```

```
class AggregatedAttestationData:
  # All the fields are signed via BLS signature
  Slot:               Integer
  Index:              Integer
  # LMD GHOST vote
  Beacon_Block_Root:  bytes
  # FFG vote
  Source:             Checkpoint
  Target:             Checkpoint
```

```
class MDEAggregatedAttestationData:
  # Signed via ECDSA
  VerifiableShare:        MDEVerifiableShare
  # The below field is signed via BLS signature
  Attestation:            AggregatedAttestationData
```

```
class MDEShare:
  U:      bytes # size:"lambda/4"
  V:      bytes # size:"lambda/2"
  Y:      bytes # size:"lambda/4"
  W:      bytes # size:"lambda/2"
```

```
class MDEShareInvalidityProof:
  W:      bytes # size:"lambda/4"
  Pi:     bytes # size:"lambda/4"
```

```
class MDEBlock:
  PP:                 MDEPublicParams
  Shares:             []MDEShare
  SK:                 bytes                          # size:"
                              lambda/4"
  InvalidityProof:    MDEShareInvalidityProof
  Attestations:       []AggregatedAttestationData

  ENC_TX:             []Transaction               #
                              Encrypted transactions
```

```
DEL_TX:                []Transaction                    #
                                    Delayed transactions
```

Note that the block header is extended with four additional fields, `PP`, `Shares`, `SK`, and `InvalidityProof`. Additionally, we introduce two transaction types to the ledger, `Encrypted` and `Delayed` transactions. The type `Encrypted` transactions are those which are encrypted for a MDE's public key, and `Delayed` transactions are the `Encrypted` ones that are being decrypted in this block.

Notice that the encrypted transactions still have to pay fee, otherwise the system will be susceptible to Denial-of-Service attacks. To avoid this issue, we split the fee into two parts, an inclusion fee and an execution fee. The encrypted transaction pays the inclusion fee by specifying a gas price and having a signature in clear. The gas cost of inclusion is a function of the size of the encrypted transaction. Note that the encrypted transaction will be considered valid only if its sender can pay for its inclusion. Moreover, the encrypted transaction carries a normal Ethereum transaction, that can be valid or not, and it pays for its execution. In interest of saving space, we ignore this detail in the following description of the protocol.

Protocol Let's denote the Algorithm 1 with the tuple ($\mathsf{Setup}, \mathsf{Gen}, \mathsf{Aggregate}, \mathsf{Verify}, \mathsf{Solve}$). Assume $\mathsf{Setup}(1^\lambda)$ was already done, and everyone knows the public parameters $pp = (N, g, T, h)$ where $h = g^{2^T}$ and g is a generator of $\mathbb{J}_N$. Note that the current total supply of ETH is roughly 120M, and every validator must stake at least 32 ETH, hence, the maximum number of validators at the time of this writing cannot exceed $120M/32 \leq 4000000$. Therefore, choosing $n = 10000000$ is more than enough for the current range of parameters in Ethereum. Additionally, assume the time needed to run $\Pi.\mathsf{Solve}$ for the delay T is equal to proposing d blocks, and let $\omega \in \mathbb{N}$ be a window parameter which is smaller than d. Note that even though the cryptographic assumption on which MDE is relied is not parallelizable, when choosing the value of T we need to consider the fastest hardware technology present and increase it adaptively over time. That is because with the advancement in hardware technologies such as CPUs and ASICs, performing basic math operations would become faster. For convenience, we define the function $\Lambda(\mathtt{b})$ such that given the block `b` returns $(\overline{ek}, \overline{pk}) \leftarrow \Pi.\mathsf{Aggregate}((\mathtt{b.Shares}_0, \mathtt{b.Shares}_0.\mathtt{U}), .., (\mathtt{b.Shares}_n, \mathtt{b.Shares}_n.\mathtt{U}))$. We also use the notation $\mathtt{b}^{-i}$ to represent the i^{th} parent of the block `b` in the chain. Below are the actors involved in Π and their corresponding responsibilities during each slot of the network:

– **Attestation Aggregation**: Aggregators take on an additional task to help collaboratively create the (public key, extraction key) pair for each slot. An aggregator upon receiving the current attestations from other attestors, runs $(pk, ek) \leftarrow \mathsf{Gen}(n, pp)$, creates a new object of type `MDEAggregatedAttestation Data` denoted by `ta` and sets `ta.VerifiableShare` $\leftarrow ek$, and fills `ta.Attestation` according to the protocol. Then, it signs `ta.VerifiableShare` via its ECDSA

private key and `ta.Attestation` with its BLS private key same as normal Ethereum. Finally, it broadcasts `ta`.

- **Block proposer**: To propose a new `TDEBlock` denotd by `b`, it performs the following:
  - First, sets $\mathtt{b.PP} \leftarrow pp$. Assuming $\mathtt{ta}_0, .., \mathtt{ta}_n$ are all the newly arrived `MDEAggregatedAttestationData`, verifies all of them via $\mathsf{Verify}(n, \mathtt{ta}_i\mathtt{.Share}, pp)$. If any of the verifications fail, it drops that attestation; otherwise it does append($\mathtt{b.Shares}, \mathtt{a}_i\mathtt{.VerifiableShare.Share}$) and append($\mathtt{b.Attestations}$, $\mathtt{a}_i\mathtt{.Attestation}$).
  - Let $(\overline{ek}, \overline{pk}) \leftarrow \Lambda(\mathtt{b}^{-d})$, and $sk \leftarrow \mathsf{Solve}(\overline{ek}, pp)$. Then, sets $\mathtt{b.SK} \leftarrow sk$. Additionally, if $\mathtt{b.SK} = \perp$ fills `b.InvalidityProof` according to the protocol Π_{ACorSol} as well.
  - If $\mathtt{p.SK} \neq \perp$, decrypts all the ($\mathtt{etx}_0$, $\mathtt{etx}_1$, .., $\mathtt{etx}_n$) $\leftarrow \mathtt{b}^{-d}\mathtt{.ENC\_TX}$ via $\mathsf{Decrypt}(\mathtt{etx}_i, \mathtt{p.SK}, pp)$, executes them in order, and puts them in $\mathtt{b.DEL\_TX}$.
  - Fills $\mathtt{b.ENC\_TX}$ with the pending transactions in the mempool, which are encrypted via one of the $\mathtt{b}^{-1}\mathtt{.U}$, $\mathtt{b}^{-2}\mathtt{.U}$,.., $\mathtt{b}^{-w}\mathtt{.U}$.
- **User**: Broadcasts the transaction `tx` in the following way. Let `b` be the latest block in their view and $(\overline{ek}, \overline{pk}) \leftarrow \Lambda(\mathtt{b})$, then for the transaction `tx`, encrypts it via $\mathtt{etx} \leftarrow \mathsf{Encrypt}(\mathtt{tx}, \overline{pk}, pp)$ and broadcasts `etx`.

Figure 6.2 shows an example block after applying all the changes. The reason why we added the parameter ω is that we have to consider network delay. Due to network delay, users might not be able to capture the last block's time-lock public key. Therefore, we allow them to encrypt their transactions via the time-lock public key of the most recent w blocks. As an example, a block that contains `Encrypted` transactions for three different public keys is shown in Figure 6.2. Experiments show that $\omega = 3$ is enough for $d = 5$ blocks. Additionally, the main reason for separating `MDEAggregatedAttestationData` from `AggregatedAttestationData` is to decrease the extra storage overhead on `MDEBlock`. Later we will show how we can still guarantee the verifiability of the puzzles. The additional checks described below should be added to the validation procedure of the block `b`:

- ECDSA signature verification of all the `b.Shares`. They should be signed by attestation aggregators' public keys of `b`. If any of the signatures are invalid, then `b` is a corrupted block and needs to be dropped.
- Validation of the puzzle of $\mathtt{b}^{-d}$. If $\mathtt{b.InvalidityProof} \neq \perp$ and passes the verification checks of Π_{ACorSol}, the block proposer of $\mathtt{b}^{-d}$ is faulty, and we can punish it in a crypto-economic way. Note that we can choose the number of d small enough such that validators cannot withdraw their stakes too early.
- Check the validity of the solution `SK`. If $\mathtt{b.InvalidityProof} = \perp$, then given $(\overline{ek}, \overline{pk}) \leftarrow \Lambda(\mathtt{b}^{-d})$, if the solution is wrong, meaning that $\overline{pk}$ and `b.SK` does not match, we must invalidate the block `b` and drop it.

Notice that when the block `b`'s puzzle, denoted as $\Lambda(\mathtt{b})$, is proven to be invalid, it indicates that at least one of the `p.Shares` is compromised. The `MDEBlock` only stores `MDEShare` and lacks the necessary information to efficiently identify corrupted shares. Nevertheless, it is known that the block proposer of `b` must

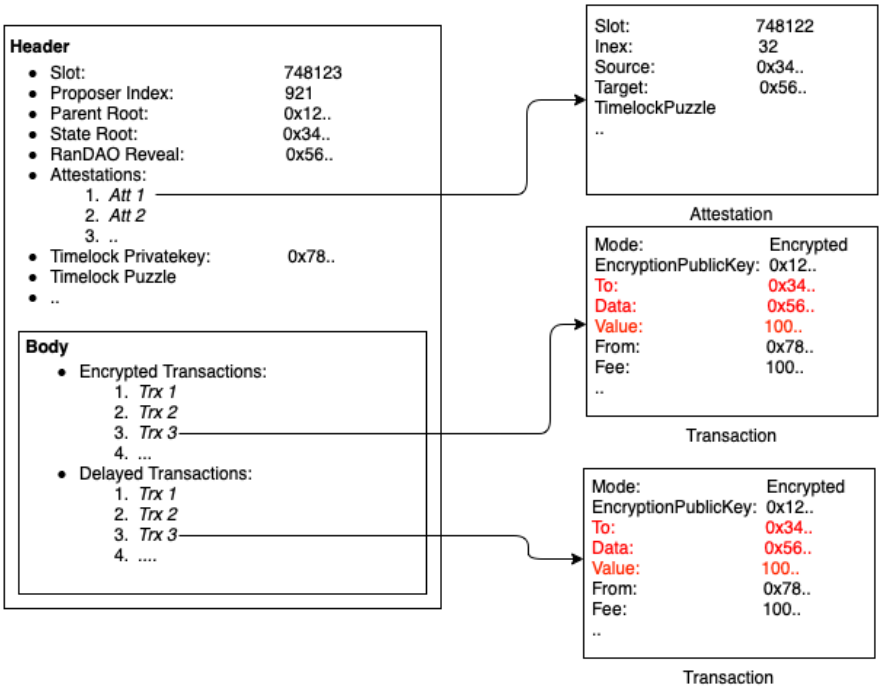

Fig. 3. This figure shows a beacon block after applying the time-lock changes. There are two additional fields in the block header (Time-lock Private key, and Time-lock Puzzle). Additionally, every attestation includes a puzzle share that will later be aggregated into the current block's time-lock puzzle. Other than that, regular transactions will no longer be supported, and the block body only contains Encrypted and Delayed transactions. Moreover, the fields shown in red in the transaction object are in the encrypted format.

have received the `MDEAggregatedAttestationData`s for all the `MDEShare`s and could have individually verified them before proposing the block. Therefore, if the invalidation is successful, the responsibility lies with the block proposer. As a result, no block proposer has the incentive to include an invalid `MDEShare` in the block, as they would eventually be detected. Consequently, attestation aggregators also have no incentive to publish an unverifiable `MDEVerifiableShare`.

What remains to be shown is that the block proposer cannot have prior knowledge of the solutions for all the `MDEShare`s. First, the block proposer needs to store the attestation aggregator's signature over `MDEShare`. All the `MDEShare`s are already guaranteed to be generated via the attestation aggregators. Additionally, being elected as an attestation aggregator is a random process, and collusion among all the aggregators is highly unlikely, given the assumption that

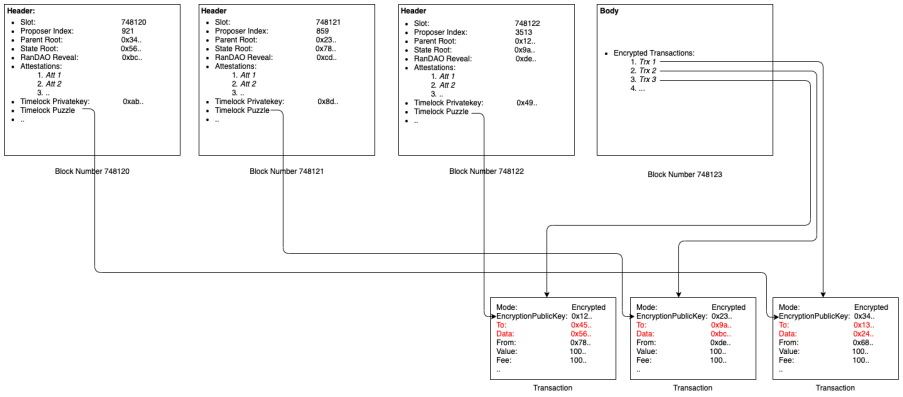

Fig. 4. A beacon block might contain Encrypted transactions that were encrypted via different time-lock public keys. This is allowed because of the inevitable network delay. For example, in this figure, block number 748123 has transactions that were encrypted via the time-lock public key of block numbers 748122, 748121, and 748120.

2/3 of the stake is allocated by honest parties. Thus, the only way the block proposer can attack the system is to only use the `MDEShare`s from the attestation aggregators with whom they collude. However, based on the network consensus and reward mechanism, it is known that if the block proposer does not include as many attestations as possible in their block, the block might not be finalized in the future. This can be further ensured by adding an additional slot validity requirement of having a minimum number of attestations.

Finally, the below theorem can be easily verified:

Theorem 3. *Given that the domain $\chi = (\mathcal{P}, \mathcal{A}, \mathcal{S})$ is the Ethereum network, our protocol produces a k-MEV safe network for any $k \in \mathbb{N}$, assuming at least one honest aggregated attestation is included in each slot and standard cryptographic assumptions in Theorem 1.*

Proof. Assume our network is in the state $s \in \mathcal{S}$ and $P \in \mathcal{P}$ is the set of block proposers of any consecutive k blocks. Following the designed protocol, the block proposer of each slot is enforced to include a valid aggregated puzzle into the block. Additionally, given the security of Theorem 1, no information will be leaked from current slot's transactions until next $k + 1^{th}$ future block. Therefore $\mathsf{MEV}^k_{A' \cup \mathcal{A}_P}(P, s)$ will be no greater than $\mathsf{MEV}^k_{\mathcal{A}_P}(P, s)$ when A' is consist of actions corresponding to using all the available transaction in the mempool. Consequently, our network becomes k-MEV safe.

Remark 1. In our protocol we adjust the values of d and w such that $k = d - \omega$. The windowing parameter ω ensures that the network propagation delay does

not prevent users from getting their transactions included in the blocks which are encrypted with relatively old public keys.

	Gen	Solve	Aggregate	Encrypt	Decrypt	Verify	Solve / Gen
$\lambda = 512$	43.157 ms	1.853 s	13.822 μs	1.691 ms	652.461 μs	42.158 ms	42.93
$\lambda = 1024$	187.265 ms	3.575 s	29.976 μs	8.654 ms	4.688 ms	180.471 ms	19.09
$\lambda = 2048$	908.347 ms	8.395 s	135.559 μs	27.370 ms	28.586 ms	919.904 ms	9.24
$\lambda = 4096$	4,613.85 ms	24.398 s	349.99 μs	398.824 ms	206.372 ms	4,626.1 ms	5.28

Table 1. Time complexity of all of all the operations for different values of λ and $T = 0x093226$. Experiments were run on a Macbook Pro 2.3 GHz Quad-Core Intel Core i5 CPU.

	$\lambda = 512$	$\lambda = 1024$	$\lambda = 2048$	$\lambda = 4096$
`MDEPublicParams`	$384 + \|T\|$	$768 + \|T\|$	$1,536 + \|T\|$	$3,072 + \|T\|$
`MDEVerifyProof`	769	1,537	3,073	6,145
`MDEShare`	$768 + \|T\|$	$1,536 + \|T\|$	$3,072 + \|T\|$	$6,144 + \|T\|$
`MDEVerifiableShare`	$1,537 + \|T\|$	$3,073 + \|T\|$	$6,145 + \|T\|$	$12,289 + \|T\|$
`MDEShareInvalidityProof`	256	512	1024	2048

Table 2. Space complexities in bytes for different values of λ. . Implementation available at https://github.com/RadNi/multiparty-delay-encryption.

6.3 Optimizing Delay

As mentioned before, the Ethereum network faces potential risks from MEV threats. These threats have adverse effects not only on the experience of regular users but also pose a significant risk to the overall safety of consensus. To provide an example, let's consider a scenario where a valuable frontrunning opportunity arises within block `b`. In such a situation, if the potential profit from frontrunning outweighs the cost of creating a fork, every miner would exert substantial effort to create a fork on top of block `b` in order to exploit that opportunity by themselves. However, in an k-MEV safe network, it can be informally stated that no participant has any incentive to modify the transaction list of a block, even after observing the state of the subsequent k blocks. Consequently, in order to profit from MEV, one would need to mine at least $k + 1$ consecutive blocks, either by forking the network or being selected as the legitimate miner for all $k + 1$ slots.

When determining the optimal value for k, two factors need to be taken into consideration. (i) Increasing the value of k enhances the network's immunity against two types of threats: reorganization attacks and MEV exploits. As

the ledger expands, it becomes increasingly challenging to attack older blocks. Moreover, profiting from MEV necessitates a miner to successfully mine at least $k+1$ subsequent blocks, a task that becomes highly unlikely for sufficiently large values of k. (ii) From the fact that no information about the block $\mathtt{b}$ will be revealed until k^{th} next block, we can conclude that it imposes an inevitable delay on the system.

6.4 Transaction Encryption

In this work, we introduce a new parameter within the transaction, denoted as $InclusionFee$. This parameter determines the gas amount that the sender is willing to allocate per byte for the purpose of ensuring their transaction inclusion in a block. It is noteworthy that our research exclusively concentrates on the encryption of the recipient and data fields within transactions. We defer further investigation of encrypting other fields such as value, sender, nonce, and fee to future research endeavors.

It is essential to highlight that, given the utilization of the ElGamal encryption scheme, representing every encrypted transaction requires two points in the group $\mathbb{J}_N$ which is $\lambda/2$ bytes. For transactions with data field surpassing $\lambda/4$ bytes, additional storage is required. Nevertheless, through the integration of symmetric key cryptography, optimization techniques can also be employed.

7 Related Work

MEV mitigation has already been well studied in the literature. Broadly speaking, there are three other classes of approaches other than content-agnostic ordering: (i) Incentive Structure Adjustment, (ii) Order Fairness, and (iii) MEV-aware Application Design.

7.1 Incentive Structure Adjustment

This method focuses on modifying the block reward or transaction fee structure to discourage MEV extraction. The primary goal of this category is to redirect the incentives of miners from selfish MEV extraction towards more socially beneficial behaviors. The core idea behind this concept is commoditizing block space. More specifically, block proposers sell block space to users who bid to have their transaction bundles atomically included. These methods ensure two key components in their design: (i) Transaction Semi-Privacy: Transaction content is only shown to a limited set of parties such as block builders, and (ii) Atomicity: The user pays only if the entire bundle is included. Flashbots MEV auction platform is one example of this class [22] similar to [42]. More recently, the MEV-Boost of Flashbots has gained significant attention which is an early implementation of the Proposer-Builder Separation (PBS) technique [11]. As an obvious drawback of this method, the block builders are trusted which can be seen as a single point of failure.

7.2 Order Fairness

In this approach, the order of transactions within a block is being fixed in a non-malleable fashion under a certain set of rules to ensure fairness. First In First Out (FIFO) model is the naive approach; however, not possible in a decentralized setting due to the propagation delay to different nodes in the network [47]. An alternative is to involve a trusted third party to determine the order, but this introduces trust issues into the system. Another option is to utilize consensus-based solutions, such as those described in [28,26]. Unfortunately, these approaches either operate only in a permissioned environment or rely on other unrealistic assumptions like network strong synchrony [27]. Some protocols have also employed Directed Acyclic Graph (DAG) ordering or the "gossip the gossip" method [19]. It's important to note that achieving perfect fairness in a distributed setting is impossible due to the Condorcet paradox from social choice theory [28].

7.3 MEV-aware Application Design

Depending on the functionality of the underlying DApp, many previous works have proposed application-specific designs. For example, CowSwap delegates the responsibility of settling a batch to a third-party solver who competes to provide the most efficient settlement that maximizes trade surplus [17], or FairTraDEX [35] where requires the blockchain to execute transactions in batch instead of sequentially. It also requires the participant to join an anonymity set for a sufficiently long time to become hidden within the anonymity set before submitting a transaction. This approach has two major disadvantages compared to other methods. First, analyzing their security is not DApp agnostic, and second, they mostly fail when there is a consensus-level issue such as transaction censorships and undercutting attacks.

8 Conclusion and Future Work

We reviewed the notions of MEV and k-MEV safe. Moreover, we explained why a k-MEV safe ledger maintains a strong resistance against MEV extraction. Our contribution starts with introduction of the notion of mulitparty delay encryption (MDE). Then, we proposed a construction for MDE based on the idea of time-lock puzzles which. Next, we adopted our algorithm to minimally change current specification of Ethereum network to achieve MEV resistance. We showed that if there is at least one honest attestation aggregator per slot, and the basic consensus assumptions are held, then Ethereum will become a k-MEV safe network for $k = d-\omega$, for an arbitrary windowing parameter ω and delay parameter d. We mention some of the potential future works to add to our contributions in the following:

– As it is pointed out in Table 2, even though the `MDEShare` already uses small space, enabling MDE still might put considerable storage overhead

on the ledger when choosing larger λ. For example, choosing $\lambda = 4096$ will increase the block size by 2.3 times [6]. A potential solution for this issue would be to only store a SNARK proof along with a single `MDEShare` that is the aggregation of all the 64 `MDEShare`s. Additionally, our current design suggests using ECDSA signature schemes to prove the authenticity of the `MDEShare`s. Therefore, the block proposer can only prove that the aggregated signature is the multiplication of different values which are signed by a set of known public keys.
- Table 1 shows the time needed for different operations of our MDE. We can apply many optimizations to accelerate the time complexity of the Gen algorithm operations [30,39], which we leave for future work. Note that our underlying math operations involve raising a fixed generator g and another fixed number h to powers of two [7].
- The proposed Verify does not allow us to verify the aggregated share. Another useful feature that our MDE construction could have would be using a suitable pseudo-random permutation to calculate the challenge e in such a way that challenges from different shares can be aggregated appropriately to preserve the verifiability of the aggregated share. Furthermore, recent advancements in composable zero-knowledge proofs [2,7] might enable us to build composable and recursive proofs which can replace the Verify functionality.
- As it was mentioned, BLS signatures are aggregatable when the messages to be signed are identical; however, when combining different `MDEShare`s we can not leverage this feature. Adopting BLS signature to our scheme to be used instead of ECDSA signature would be highly of interest.
- Our protocol can be augmented with zero-knowledge proof ideas to also hide the transaction's metadatas*e.g.* sender, gas limit, nonce, etc.
- Efficiently designing a trustless distributed RSA modulus generation has its long literature [14,15,8]. However, RSA safe prime modulus generation [3] has not been fully studied yet which can be another future contribution.

Acknowledgment

We like to express our gratitude to the Ethereum Foundation (Grant ID FY22-0719) and Aquanow for their generous support, which made this research possible. Additionally, we appreciate Professor Shahram Khazaei for his valuable comments and guidance.

References

1. Abou Jaoude, J., Saade, R.G.: Blockchain applications–usage in different domains. Ieee Access **7**, 45360–45381 (2019)

[6] The average of Ethereum block size is almost $170,000$ bytes [56] and to store 64 aggregated attestations we need $6144 \times 64 = 393,216$ bytes.

[7] In practice, we can almost always choose four as the generator g and simplify the arithmetic operations even more.

2. Albrecht, M.R., Cini, V., Lai, R.W., Malavolta, G., Thyagarajan, S.A.: Lattice-based snarks: Publicly verifiable, preprocessing, and recursively composable. In: Annual International Cryptology Conference. pp. 102–132. Springer (2022)
3. Algesheimer, J., Camenisch, J., Shoup, V.: Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In: Annual International Cryptology Conference. pp. 417–432. Springer (2002)
4. Asayag, A., Cohen, G., Grayevsky, I., Leshkowitz, M., Rottenstreich, O., Tamari, R., Yakira, D.: A fair consensus protocol for transaction ordering. In: 2018 IEEE 26th International Conference on Network Protocols (ICNP). pp. 55–65. IEEE (2018)
5. Baric, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: International Conference on the Theory and Application of Cryptographic Techniques (1997)
6. Baum, C., Hsin-yu Chiang, J., David, B., Frederiksen, T.K., Gentile, L.: Sok: Mitigation of front-running in decentralized finance. In: International Conference on Financial Cryptography and Data Security. pp. 250–271. Springer (2022)
7. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. Algorithmica **79**, 1102–1160 (2017)
8. Boneh, D., Franklin, M.: Efficient generation of shared RSA keys. In: Annual international cryptology conference. pp. 425–439. Springer (1997)
9. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: International conference on the theory and application of cryptology and information security. pp. 514–532. Springer (2001)
10. Burdges, J., Feo, L.D.: Delay encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 302–326. Springer (2021)
11. Buterin, V.: State of research: increasing censorship resistance of transactions under proposer/builder separation (pbs) (2022), `https://notes.ethereum.org/@vbuterin/pbs_censorship_resistance`
12. Buterin, V., Hernandez, D., Kamphefner, T., Pham, K., Qiao, Z., Ryan, D., Sin, J., Wang, Y., Zhang, Y.X.: Combining ghost and casper. arXiv preprint arXiv:2003.03052 (2020)
13. Carlsten, M., Kalodner, H., Weinberg, S.M., Narayanan, A.: On the instability of bitcoin without the block reward. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. pp. 154–167 (2016)
14. Chen, M., Doerner, J., Kondi, Y., Lee, E., Rosefield, S., Shelat, A., Cohen, R.: Multiparty generation of an RSA modulus. Journal of Cryptology **35**(2), 12 (2022)
15. Chen, M., Hazay, C., Ishai, Y., Kashnikov, Y., Micciancio, D., Riviere, T., Shelat, A., Venkitasubramaniam, M., Wang, R.: Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 590–607. IEEE (2021)
16. Cline, D., Dryja, T., Narula, N.: Clockwork: An exchange protocol for proofs of non front-running
17. CoWSwap: The smartest way to trade cryptocurrencies (2022), `https://docs.flashbots.net/flashbots-auction/overview`
18. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 910–927. IEEE (2020)

19. Danezis, G., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A.: Narwhal and tusk: a DAG-based mempool and efficient BFT consensus. In: Proceedings of the Seventeenth European Conference on Computer Systems. pp. 34–50 (2022)
20. Dembo, A., Kannan, S., Tas, E.N., Tse, D., Viswanath, P., Wang, X., Zeitouni, O.: Everything is a race and Nakamoto always wins. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 859–878 (2020)
21. Doweck, Y., Eyal, I.: Multi-party timed commitments. arXiv preprint arXiv:2005.04883 (2020)
22. Flashbots: Flashbots auction (2022), `https://docs.flashbots.net/flashbots-auction/overview`
23. Flashbots: Mev-explore pre-merge (2023), `https://explore.flashbots.net`
24. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 281–310. Springer (2015)
25. Kavousi, A., Abadi, A., Jovanovic, P.: Timed secret sharing. Cryptology ePrint Archive (2023)
26. Kelkar, M., Deb, S., Kannan, S.: Order-fair consensus in the permissionless setting. In: Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop. pp. 3–14 (2022)
27. Kelkar, M., Deb, S., Long, S., Juels, A., Kannan, S.: Themis: Fast, strong order-fairness in Byzantine consensus. Cryptology ePrint Archive (2021)
28. Kelkar, M., Zhang, F., Goldfeder, S., Juels, A.: Order-fairness for Byzantine consensus. In: Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III 40. pp. 451–480. Springer (2020)
29. Khalil, R., Gervais, A., Felley, G.: Tex-a securely scalable trustless exchange. Cryptology ePrint Archive (2019)
30. Koc, C.K.: High-speed RSA implementation. Tech. rep., Technical Report TR-201, RSA Laboratories (1994)
31. Liu, Y., Wang, Q., Yiu, S.M.: Towards practical homomorphic time-lock puzzles: Applicability and verifiability. Cryptology ePrint Archive (2022)
32. Loe, A.F., Medley, L., O'Connell, C., Quaglia, E.A.: Tide: A novel approach to constructing timed-release encryption. In: Australasian Conference on Information Security and Privacy. pp. 244–264. Springer (2022)
33. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: Annual International Cryptology Conference. pp. 620–649. Springer (2019)
34. McLaughlin, R., Kruegel, C., Vigna, G.: A large scale study of the Ethereum arbitrage ecosystem. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 3295–3312 (2023)
35. McMenamin, C., Daza, V., Fitzi, M., O'Donoghue, P.: Fairtradex: A decentralised exchange preventing value extraction. In: Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security. pp. 39–46 (2022)
36. Medley, L., Loe, A.F., Quaglia, E.A.: Sok: Delay-based cryptography. Cryptology ePrint Archive (2023)
37. Momeni, P., Gorbunov, S., Zhang, B.: Fairblock: Preventing blockchain front-running with minimal overheads. In: International Conference on Security and Privacy in Communication Systems. pp. 250–271. Springer (2022)

38. Obadia, A., Salles, A., Sankar, L., Chitra, T., Chellani, V., Daian, P.: Unity is strength: A formalization of cross-domain maximal extractable value. arXiv preprint arXiv:2112.01472 (2021)
39. Orup, H.: Simplifying quotient determination in high-radix modular multiplication. In: Proceedings of the 12th Symposium on Computer Arithmetic. pp. 193–199. IEEE (1995)
40. Osmosis: The osmosis blockchain is a decentralized network, ran by 100+ validators and full nodes, with many front-ends and development teams on it. explore our docs and examples to quickly learn, develop & integrate with the osmosis blockchain. (2022), https://docs.osmosis.zone/overview/
41. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Advances in Cryptology-EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18. pp. 223–238. Springer (1999)
42. Piatt, C., Quesnelle, J., Sheridan, C.: Eden network. Unpublished Manuscript (2021)
43. Raun, C., Estermann, B., Zhou, L., Qin, K., Wattenhofer, R., Gervais, A., Wang, Y.: Leveraging machine learning for bidding strategies in miner extractable value (mev) auctions. Cryptology ePrint Archive (2023)
44. Reiter, M.K., Birman, K.P.: How to securely replicate services. ACM Transactions on Programming Languages and Systems (TOPLAS) **16**(3), 986–1009 (1994)
45. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
46. Rondelet, A., Kilbourn, Q.: Threshold encrypted mempools: Limitations and considerations. arXiv preprint arXiv:2307.10878 (2023)
47. Sekar, V.: Preventing front-running attacks using timelock encryption. Ph.D. thesis, University College London (2022)
48. Shutter-Network: Introducing shutter network - combating front running and malicious mev using threshold cryptography (2021), https://blog.shutter.network/introducing-shutter-network-combating-frontrunning-and-malicious-mev-using-threshold-cryptography
49. Sikka: Sikka projects (2022), https://sikka.tech/projects/
50. Srinivasan, S., Loss, J., Malavolta, G., Nayak, K., Papamanthou, C., Thyagarajan, S.A.: Transparent batchable Time-lock Puzzles and Applications to Byzantine Consensus. In: IACR International Conference on Public-Key Cryptography. pp. 554–584. Springer (2023)
51. Tasatanattakool, P., Techapanupreeda, C.: Blockchain: Challenges and applications. In: 2018 International Conference on Information Networking (ICOIN). pp. 473–475. IEEE (2018)
52. Thyagarajan, S.A.K., Bhat, A., Malavolta, G., Döttling, N., Kate, A., Schröder, D.: Verifiable timed signatures made practical. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1733–1750 (2020)
53. Thyagarajan, S.A.K., Castagnos, G., Laguillaumie, F., Malavolta, G.: Efficient cca timed commitments in class groups. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 2663–2684 (2021)
54. Wesolowski, B.: Efficient verifiable delay functions. In: Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38. pp. 379–407. Springer (2019)
55. Yang, S., Zhang, F., Huang, K., Chen, X., Yang, Y., Zhu, F.: SoK: MEV countermeasures: Theory and practice. arXiv preprint arXiv:2212.05111 (2022)

56. ycharts.com: Ethereum average block size (i:ebs) (2023), https://ycharts.com/indicators/ethereum_average_block_size
57. Zhang, H., Merino, L.H., Estrada-Galinanes, V., Ford, B.: Flash freezing flash boys: Countering blockchain front-running. In: 2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW). pp. 90–95. IEEE (2022)

A Lemmas

Lemma 2. *For every $x \in \mathbb{N}$ and $N \in \mathbb{N}$, $x^N \mod N^2 = (x \mod N)^N \mod N^2$.*

Proof. Assume $x = kN + r$ for $k \in \mathbb{N}, 0 \leq r < N$. Then,

$$x^N \ mod\ N^2 = (kN + r)^N \ mod\ N^2 = r^N + r^{N-1} * kN * N + ... \ kN^N \ mod\ N^2$$
$$= r^N \ mod\ N^2 = (x \ mod\ N)^N \ mod\ N^2$$

Lemma 3. *Given a cyclic group G_c, its generator g, and an arbitrary element $x \in G_c$, if $x^\alpha = 1$, then either $gcd(\alpha, ||G_c||) \neq 1$ or x is the identity element.*

Proof. If x is the identity element, then clearly $x^\alpha = 1$ for any α. Therefore, assume otherwise. Then, $ord_{G_c}(x)$ should divide α. Additionally, $ord_{G_c}(x)$ must divide $||G_c||$ according to the Lagrange theorem. Therefore, $gcd(\alpha, ||G_c||) \geq ord_{G_c}(x) > 1$

Lemma 4. *Given a cyclic group G_c, its generator g, and arbitrary elements $x, y \in G_c$ such that $x^\alpha = y^\beta$, if $gcd(||G_c||, d) = 1$ and d divides both α and β, then $x^{\alpha/d} = y^{\beta/d}$.*

Proof. We have $x^\alpha y^{-\beta} = (x^{\alpha/d} y^{-\beta/d})^d = 1$. Via the result of Lemma 3 and considering the fact that d and $||G_c||$ are coprime, we can conclude that $x^{\alpha/d} y^{-\beta/d} = 1$ and finally $x^{\alpha/d} = y^{\beta/d}$

Lemma 5. *Assume g is a generator of the cyclic group G_c. For an element $y \in G_c$ where $g^\beta = y^\alpha$ such that $\alpha | \beta$, then the discrete logarithm of y in G_c will be β/α if $gcd(\alpha, ||G_c||) = 1$.*

Proof. Since $\alpha | \beta$, then there exists some k such that $g^{k\alpha} = y^\alpha$. Therefore, $g^{k\alpha} y^{-\alpha} = 1$ and $(g^k y^{-1})^\alpha = 1$. Finally, since $gcd(\alpha, ||G_c||) = 1$, via the result of Lemma 2 we can conclude that $g^k y^{-1} = 1$. Therefore, $y = g^{\beta/\alpha}$

Lemma 6. *Assume N is an RSA modulus. With finding a non-trivial square root of identity in $\mathbb{Z}_N^*$ an attacker can defactor N and break the RSA assumption.*

Proof. Assume $x \in \mathbb{Z}_N^*$ is the square root of identity which means that $x^2 = 1 \mod N$. Therefore, $(x-1)(x+1) = 0 \mod N$. Additionally, we know that $x \neq \pm 1$. Consequently, factors of N can be computed via $gcd(x+1, N)$ and $gcd(x-1, N)$.

Lemma 7. *Given an RSA modulus N in which the RSA strong assumption is held, and $y \in Z_N^*$, no PPT algorithm can find $x \in Z_N^*$ and e' such that $x^e = y^{e'}$ and $gcd(e, e') = 1$ for a given e.*

Proof. Proof by contradiction: Assume an adversary could find x and e' such that $x^e = y^{e'}$ where $gcd(e, e') = 1$. Therefore, there are some a and b such that $ea + e'b = 1$. Consequently, $x^{be} = y^{be'} = y^{1-ea} \implies (x^b y^a)^e = y$. Therefore, the attacker can find the e-th root of y, which is $x^b y^a$, and break the RSA strong assumption in N.

Lemma 8. *Given an RSA safe prime modulus $N = p'q'$ such that $p' = 2p + 1$ and $q' = 2q + 1$ for $p,\ q$ sufficiently large prime numbers, assume e and α are two numbers such that $e < p,\ q$, $e < \alpha$, and $g^\alpha = u^e$ for an arbitrary $u \in \mathbb{J}_N$ and g a generator of $\mathbb{J}_N$. Then, either e must divide α or the strong RSA assumption in $\mathbb{J}_N$ will be broken.*

Proof. Assume $d \leftarrow gcd(e, \alpha)$ and $d' \leftarrow gcd(d, ||\mathbb{J}_N||)$. We know that d' must be less than e; accordingly, it only can get the values 2 or 1. Below, each case is analyzed separately:

- *$d' = 1$: Via Lemma 4 we can assume that $d = 1$[8]. Then, given $y \leftarrow g$ and $e \leftarrow e$, $x \leftarrow u$ and $e' \leftarrow \alpha$ are found such that $x^{e'} = y^e$ where $gcd(e, e') = 1$ which contradicts the strong RSA assumption according to Lemma 7*
- *$d' = 2$: Assume $d \leftarrow 2^k\bar{d}$ and $g^{d\omega} = u^{d\gamma}$ such that $gcd(\bar{d}, ||\mathbb{J}_N||) = 1$ and $gcd(\omega, \gamma) = 1$; therefore, via Lemma 4 we can assume that $\bar{d} = 1$. Consequently, $g^{2^k\omega} = u^{2^k\gamma}$ and therefore $(g^\omega u^{-\gamma})^{2^k} = 1$. Because g is the generator, there is some ρ such that $g^\rho = g^\omega u^{-\gamma}$. Since $ord_{\mathbb{J}_N}(g) = ||\mathbb{J}_N|| = 2pq$, there must be some k' that $2^k\rho = k'2pq$. Thus, $2\rho = k'/2^{k-1}2pq = k''2pq$, and finally $g^{2\rho} = (g^\omega u^{-\gamma})^2 = 1$. Therefore, $(g^\omega + u^\gamma)(g^\omega - u^\gamma) = 0$. Clearly, if non of the $g^\omega + u^\gamma$ and $g^\omega - u^\gamma$ are zero, we could find two non-trivial factors of N via $gcd(N, g^\omega \pm u^\gamma)$. Otherwise:*
  - *$g^\omega = u^\gamma$: Considering $y \leftarrow g$, $e \leftarrow \gamma$, $e' \leftarrow \omega$, and $x \leftarrow u$ using Lemma 7 we can break the RSA strong assumption in $\mathbb{J}_N$.*
  - *$g^\omega = -u^\gamma$: Since $gcd(\omega, \gamma) = 1$, at least one of ω or γ must be odd. Without loss of generality, assume γ is odd. Then, $g^\omega = -u^\gamma = (-u)^\gamma$. Again, via Lemma 7, assuming $y \leftarrow g$, $e \leftarrow \gamma$, $e' \leftarrow \omega$, and $x \leftarrow -u$ we could break the RSA strong assumption. For the case that ω is odd, we can use the equation $-g^\omega = u^\gamma$ and reach the same contradiction.*

Lemma 9. *There is an isomorphism $\mathbb{Z}_N^* \times \mathbb{Z}_N \cong \mathbb{Z}_{N^2}^*$ [41].*

Lemma 10. *For the group $F \leftarrow \{(1+N)^x \mod N^2 \mid x \in \mathbb{Z}_N\}$ and $\mathbb{Z}_N$ there is an isomorphism $F \cong \mathbb{Z}_N$.*

Proof. First

$$(1+N)^x \mod N^2 = 1 + Nx + ... + N^x \mod N^2 = 1 + Nx,$$

Then, we define the isomorphism $\psi : F \to \mathbb{Z}_N$ such that:

$$\psi(1+N)^x = x$$

Clearly, ψ is bijective. Additionally:

$$\psi((1+N)^{x+y}) = x + y = \psi((1+N)^x) + \psi((1+N)^y),$$

which shows that ψ is an isomorphism between F and $\mathbb{Z}_N$

[8] Via taking e as $e * d'/d$ and α as $\alpha * d'/d$

Lemma 11. *For every odd $N \in \mathbb{N}$, $\left(\frac{1+N}{N^2}\right) = +1$.*

Proof.

$$\left(\frac{1+N}{N^2}\right) = \left(\frac{1+N}{N}\right) * \left(\frac{1+N}{N}\right) = \left(\frac{1+N}{N}\right)^2 = +1$$

B Basic Definitions

Definition 13. ***Statistical Distance*** *We show the statistical distance between two random variables X and Y with $\Delta(X,Y)$ which is coming from the following equation:*

$$\Delta(X,Y) = \sum_{a \in D} |\Pr[X = a] - \Pr[Y = a]|$$

Note that X and Y are both in domain D.

Definition 14. Statistically Indistinguishable *Two random variables X and Y in domain D are said to be statistically indistinguishable by parameter λ if:*

$$\Delta(X,Y) = \mathsf{negl}(\lambda)$$

Definition 15. Computationally Indistinguishable *Two random variables X and Y in domain D are said to be computationally indistinguishable by the parameter λ if for every* PPT *algorithm $\mathcal{D}$:*

$$\Pr\left[\mathcal{D}(\alpha) = b \,\middle|\, \begin{array}{c} b \leftarrow \{0,1\}, \\ if\ b = 0 : \alpha \leftarrow\!\$\ X \\ else : \alpha \leftarrow\!\$\ Y \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda) \tag{5}$$

We show computational indistinguishability via $X \approx_c Y$

Definition 16. Safe RSA Number *The number $N = pq$ where p and q are prime numbers is a safe RSA number if $p = 2p\prime + 1$ and $q = 2q\prime + 1$ that both $p\prime$ and $q\prime$ are also large enough prime numbers.*

Definition 17. Interactive Proof of Argument *$\Pi = (P,V)$ is an interactive proof of argument for the language $\mathcal{L}$ for a* PPT *algorithm V in the size of elements in $\mathcal{L}$ if:*

1. ***Completness****: If $x \in \mathcal{L}$, $\Pr[(P,V)(x) = accept] = 1$.*
2. ***Soundness****: If $x \notin \mathcal{L}$ for every prover P^*, $\Pr[(P^*,V) = accept] \leq negl(|x|)$*

Definition 18. Zero-knowledge *An interactive protocol $\Pi = (P,V)$ for the language $\mathcal{L}$ given $view(P,V)(x)$ as the transcript of executing Π between P and V is said to be zero-knowledge if there exists a* PPT *algorithm S (Simulator) such that for the two probability ensembles $view(P,V)(x)$ and $S(x)$ we have $view(P,V)(x) \approx_c S(x)$*

Definition 19. Zero-knowledge Interactive Protocol *The protocol $\Pi(P,V)$ is zero-knowledge interactive if it is complete, sound, and zero-knowledge.*

Definition 20. Honest Verifier Interactive Zero-knowledge *An Interactive zero-knowledge protocol $\Pi = (P,V)$ is considered honest verifier if in the transcript between P and V, $view(P,V)(x)$ the randomness being used by V is also included.*

Definition 21. Proof Of Knowledge *Given $\mathcal{L}_R = \{x : \exists w s.t. R(x,w) = accept\}$ for a polynomial-time relation R and language $\mathcal{L}$, the protocol $\Pi = (P,V)$ is a proof of knowledge if there exists a PPT extractor algorithm E such that $\forall_{x\in\mathcal{L}}\ R(x,E(x)) = accept$.*

C Cryptographic Assumptions

Assumption 1 ***Strong RSA****[5] Given an RSA modulus N, and $e \in Z^*_{\phi(N)}$ when $e > 2$, no P.P.T algorithm can efficiently find the value of m from $c = m^e \mod N$.*

Assumption 2 ***Strong Sequential Squaring****[33] Given N a safe RSA modulus, g a generator of $\mathbb{J}_N$, for any PPT $(\mathcal{A}_1,\mathcal{A}_2)$ which the depth of $\mathcal{A}_2$ is bounded by $T^\epsilon(|N|)$ from above where T is a polynomial and $0 < \epsilon < 1$:*

$$\left[\mathcal{A}_2(x,y,\tau) = b \;\middle|\; \begin{array}{c} \tau \leftarrow \mathcal{A}_1(N,T,g) \\ x \leftarrow\$ \langle g\rangle,\ b \leftarrow\$ \{0,1\}, \\ if\ b = 0 : y \leftarrow x^{2^T} \mod N \\ else : y \leftarrow\$ \langle g\rangle \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(|N|) \quad (6)$$

Assumption 3 ***Decisional Composite Residuosity (DCR)****[41] Given a RSA modulus N and any PPT adversary $\mathcal{A}$, the decisional composite residuosity assumption over $\mathbb{Z}^*_{N^2}$ states that:*

$$\Pr\left[\mathcal{A}(N,y) = b \;\middle|\; \begin{array}{c} x \leftarrow\$ \mathbb{Z}^*_N, \\ b \leftarrow\$ \{0,1\}, \\ if\ b = 0 : y \leftarrow x^N \mod N^2 \\ else : y \leftarrow\$ \mathbb{Z}^*_{N^2} \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(|N|) \quad (7)$$

D Modified Linearly Homomorphic Time-lock Puzzle

We have borrowed the syntax of linearly homomorphic time-lock puzzle from [31,33]:

Definition 22. Homomorphic Time-Lock Puzzle (LHTP) *Let $\mathcal{C}_\lambda$ for the security parameter $\lambda \in \mathbb{N}$ be a class of circuits and $\mathcal{S}$ a finite domain. Then, a homomorphic time-lock puzzle is a tuple (Setup, Gen, Solve, Eval) such that:*

- $\mathsf{Setup}(1^\lambda, \mathcal{T})$*: A probabilistic algorithm that outputs the system's public parameter, pp, in a trusted/or distributed fashion according to the chosen delay value* $\mathcal{T}$*.*
- $\mathsf{Gen}(s, pp)$*: A probabilistic algorithm that, on receiving the system's public parameters pp and any arbitrary* $s \in \mathcal{S}$*, outputs a puzzle* z*.*
- $\mathsf{Aggregate}(C, z_0, z_1, .., z_n, pp)$*: A probabilistic algorithm that given a circuit* $C \in \mathcal{C}_k$*, a set of puzzles* $z_0, .., z_n$*, and system's public parameters pp, outputs an aggregated puzzle* $\bar{z}$ *according to* C*.*
- $\mathsf{Solve}(z, pp)$*: A deterministic algorithm which receives a puzzle* z *and system's public parameters pp and outputs a solution* $s \in \mathcal{S}$ *or* $\perp$*.*

Definition 23. LHTP Correctness *The LHTP protocol* $\Pi = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Solve}, \mathsf{Aggregate})$ *is correct if given* $\mathcal{C}_\lambda$ *a class of circuits, for all polynomials* $\mathcal{T}$ *in* λ*, all circuits* $c \in \mathcal{C}_\lambda$*, all inputs* $(s_0, .., s_n) \in \mathcal{S}^n$*, all pp in the support of* $\mathsf{Setup}(1^\lambda, \mathcal{T})$*, and all* z_i *in the support of* $\Pi.\mathsf{Gen}(s_i, pp)$ *, assuming* $\bar{z} \leftarrow \Pi.\mathsf{Aggregate}(c, z_0, .., z_n, pp)$*, where the running time of the algorithm* $\Pi.\mathsf{Solve}(1^\lambda, \bar{z})$ *is bounded with* $p_0(\lambda, \mathcal{T})$ *for a fixed polynomial* p_0*, for* $(\bar{s}, \bar{\pi}) \leftarrow \Pi.\mathsf{Solve}(\bar{z}, pp)$*:*

$$\bar{s} \;=\; c(s_0, .., s_n) \tag{8}$$

Definition 24. LHTP Security *The LHTP protocol* $\Pi = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Solve}, \mathsf{Aggregate})$ *is secure if for any* PPT *adversary* $(\mathcal{A}_1, \mathcal{A}_2)$ *which the depth of* $\mathcal{A}_2$ *is bounded by* $T^\epsilon(\lambda)$ *from above where* T *is a polynomial and* $0 < \epsilon < 1$ *such that:* Solvability*:*

$$\Pr\left[\mathcal{A}_2(1^\lambda, z, pp, \tau) = b \;\middle|\; \begin{array}{c} pp \leftarrow \Pi.\mathsf{Setup}(1^\lambda, T), \\ (m_0, m_1, \tau) \leftarrow \mathcal{A}_1(T, pp),\ b \leftarrow\!\$\ \{0,1\}, \\ z \leftarrow \Pi.\mathsf{Gen}(m_b, pp) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda) \tag{9}$$

Definition 25. LHTP Compactness *Given* $\mathcal{C}_\lambda$ *a class of circuits, the LHTP protocol* $\Pi = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Aggregate}, \mathsf{Solve})$ *is compact if for all polynomials* $\mathcal{T}$ *in* λ*, all circuits* $c \in \mathcal{C}_\lambda$*, all inputs* $(s_0, .., s_n) \in \mathcal{S}^n$*, all pp in the support of* $\Pi.\mathsf{Setup}(1^\lambda, \mathcal{T})$*, and all* z_i *in the support of* $\Pi.\mathsf{Gen}(s_i, pp)$*, assuming* $\bar{z} = \Pi.\mathsf{Aggregate}(c, z_0, .., z_n, pp)$ *we have:*

- *The running time of the algorithm* $\Pi.\mathsf{Aggregate}(c, z_0, .., z_n, pp)$ *is bounded by* $p_0(\lambda, |C|)$ *for a fixed polynomial* p_0*.*
- *The size of* $\bar{z}$ *is bounded by* $p_1(\lambda,\ |c(s_0, .., s_n)|)$ *for a fixed polynomial* p_1*.*

Algorithm 2 ***Linearly Homomorphic Time-Lock Puzzle (LHTP)***

- $\mathsf{Setup}(1^\lambda, T)$*: Outputs system's public parameters* $pp \leftarrow (N, g, T, h)$*.* N *a safe RSA number with* $|N| = 2\lambda$*,* g *a generator of the group* $\mathbb{J}_N$*,* $h \leftarrow g^{2^T} \mod N$*, and* T *the delay parameter (minimum number of modular squaring needed for solving the puzzle).*

- $\mathsf{Gen}(s, pp)$: *For the secret* $s \in \mathbb{Z}_N$, *samples the randomization* $r, k \leftarrow\$ \mathbb{Z}_N$ *and outputs* (u, v, y, w) *such that:* $u \leftarrow g^{r+s} \mod N$, $v \leftarrow h^{(r+s)N}(1+N)^s \mod N^2$, $y \leftarrow g^k \mod N$, *and* $w \leftarrow h^{kN}(1+N)^r \mod N^2$.
- $\mathsf{Aggregate}((u_0, v_0, y_0, w_0), (u_1, v_1, y_1, w_1), ..(u_n, v_n, y_n, w_n), pp)$: *Returns* $(\bar{u}, \bar{v}, \bar{y}, \bar{w})$ *for* $\bar{u} \leftarrow \Pi\ u_i \mod N$, $\bar{v} \leftarrow \Pi\ v_i \mod N^2$, $\bar{y} \leftarrow \Pi\ y_i$, *and* $\bar{w} \leftarrow \Pi\ w_i \mod N^2$.
- $\mathsf{Solve}((u, v, y, w), pp)$: *Calculates* $x \leftarrow u^{2^T} \mod N$ *and sets* $s \leftarrow \dfrac{v/x^N \mod N^2 - 1}{N}$. *If* $s \in \mathbb{Z}_N$ *returns* s, *otherwise returns* $\perp$.

Note that with having the puzzle $z = (u, v, y, w)$ *as the input, the* Solve *procedure does not use* y *and* w. *We will later see their usage when constructing our MDE. Additionally, to eliminate the need for trust during the* Setup *phase, we can use an untrusted distributed RSA modules generation protocol using MPC [3]. Alternatively, we can also leverage class groups of imaginary quadratic order, which only requires a public coin setup phase [53,50]. Now, lets prove the security and correctness theorem of LHTP:*

Theorem 4. *Let* N *be an RSA safe prime modulus. If the strong sequential squaring assumption is held in* $\mathbb{J}_N$ *and the DCR assumption is held in* $\mathbb{Z}^*_{N^2}$, *then LHTP is a time-lock puzzle entitled to the correctness, security, and compactness requirements.*

Proof. We start by proving the correctness first.
Correctness: *Lemma 12 directly implies the correctness condition.*

Lemma 12. *Given an LHTP protocol* Π *and the puzzle* $(u, v, y, w) \leftarrow \Pi.\mathsf{Gen}(s, pp)$ *for* $s \in \mathbb{N}$, *the output of* $\Pi.\mathsf{Solve}(u, v)$ *will be* $s \mod N$.

Proof.

$$x = u^{2^T} = (g^{r+s})^{2^T} = g^{(r+s)2^T}$$

According to Lemma 2:

$$x^N \mod N^2 = (x \mod N)^N \mod N^2$$

Assume $s = qN + k$ *for some* $q \in \mathbb{N}$ *and* $k < N$. *Therefore:*

$$\begin{aligned} v/x^N \mod N^2 &= h^{(r+s)N}(1+N)^s / g^{(r+s)N2^T} \mod N^2 \\ &= (1+N)^s \mod N^2 = 1 + sN \mod N^2 = 1 + kN \end{aligned}$$

Finally:

$$\frac{v/x^N \mod N^2 - 1}{N} = \frac{kN}{N} = s \mod N$$

Security*: To prove the security, we first prove the following lemma:*

Lemma 13. *Given a LHTP protocol Π and any* PPT *adversary $(\mathcal{A}_1, \mathcal{A}_2)$ which the depth of $\mathcal{A}_2$ is bounded by $T^\epsilon(\lambda)$ from above where λ is the security parameter, T is a polynomial and $0 < \epsilon < 1$:*

$$\Pr\left[\mathcal{A}_2(1^\lambda, z_b, pp, \tau) = b \;\middle|\; \begin{array}{c} pp \leftarrow \Pi.\mathsf{Setup}(1^\lambda, T), \\ (s, \tau) \leftarrow \mathcal{A}_1(T, pp), \\ b \leftarrow\!\!\$\ \{0,1\}, \\ z_0 \leftarrow \Pi.\mathsf{Gen}(s, pp),\ z_1 \leftarrow\!\!\$\ \mathbb{J}_N \times \mathbb{J}_{N^2} \\ \times \mathbb{J}_N \times \mathbb{J}_{N^2} \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda) \tag{10}$$

The proof of this lemma is in Appendix E. In other words, we proved that the tuple (u, v, y, w) does not reveal anything about the secret s. Now, it is time to prove the main security theorem. We do this by contradiction. Assume the adversary $(\bar{\mathcal{A}}_1, \bar{\mathcal{A}}_2)$ breaks the security of LHTP. Then we construct a new adversary $(\mathcal{A}_1, \mathcal{A}_2)$ to contradict the result of Lemma 13. $\mathcal{A}_1$ on receiving (T, pp) calls $(m_0, m_1, \tau) \leftarrow \overline{\mathcal{A}_1}(T, pp)$. Without lose of generality, assume $\overline{\mathcal{A}}_2$ can answer the experiment correctly with non-negligible advantage σ when it is called by the message m_0. Then, $\mathcal{A}_1$ returns (m_0, τ). Finally, $\mathcal{A}_2$ after receiving $(1^\lambda, z_b, pp, \tau)$ outputs whatever $\bar{\mathcal{A}}_2(1^\lambda, z_b, pp, \tau)$ returns. Observe $(\mathcal{A}_1, \mathcal{A}_2)$ has at least $\sigma/2$ which proves the security of LHTP.

Compactness*: It is easy to verify that the size of the aggregated puzzle via* Aggregate *function is polynomial to the circuit length and will not increase by the number of puzzles. Additionally, note that* Aggregate *runs in polynomial time to the circuit size.*

E Proof of Lemma 13

Proof. Via a set of experiments, we have:

Hybrid $\mathcal{H}_0$*: We begin by setting $\mathcal{H}_0$ as the original scheme. and sampling z_1 from $\mathcal{H}_0$. Therefore, $z_1 \leftarrow \Pi.\mathsf{Gen}(s, pp)$. Clearly, no adversary can distinguish between z_0 and z_1 since they share the exact same distribution.*

Hybrid $\mathcal{H}_1$*: In this hybrid, the tuple (u, v, y, w) is constructed as $r, s \leftarrow\!\!\$\ \mathbb{Z}_N$, $k \leftarrow\!\!\$\ \mathbb{Z}_{N/2}$, $u \leftarrow g^{r+s} \mod N$, $v \leftarrow h^{(r+s)N}(1+N)^s \mod N^2$, $y \leftarrow g^k \mod N$ and $w \leftarrow c^N(1+N)^r \mod N^2$ for $c \in \mathbb{J}_N$. One can verify that the only difference from $\mathcal{H}_0$ is that w instead of $h^{kN}(1+N)^r \mod N^2$ is $w \leftarrow c^N(1+N)^r \mod N^2$, but for the rest it is the same as $\mathcal{H}_0$. We want to show that given $z_0 \leftarrow\!\!\$\ \mathcal{H}_0$ $z_1 \leftarrow\!\!\$\ \mathcal{H}_1$ there can not be any efficient distinguisher $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ which can succeeds in distinguishing between z_0 and z_1 via a reduction against the strong sequential squaring assumption in $\mathbb{J}_N$. We prove by contradiction via constructing a distinguisher $\mathcal{R} = (\mathcal{R}_1, \mathcal{R}_2)$, which breaks the strong sequential squaring assumption. $\mathcal{R}_1$ on receiving (N, T, g) calculates $h \leftarrow g^{2^T}$, $pp \leftarrow (N, g, T, h)$ and calls $(s, \tau') \leftarrow \mathcal{A}_1(T, pp)$, then, returns $\tau \leftarrow (s, pp, \tau')$. Next, the challenger calls $\mathcal{R}_2$ with $(x, y, (s, pp, \tau'))$. Then, $\mathcal{R}_2$ constructs $\hat{z} \leftarrow (\hat{u}, \hat{v}, \hat{y}, \hat{w})$ where $\hat{r}, \leftarrow\!\!\$\ \mathbb{Z}_N$, $\hat{u} \leftarrow g^{\hat{r}+s}$, $\hat{v} \leftarrow h^{(\hat{r}+s)N}(1+N)^s \mod N^2$, $\hat{y} \leftarrow x$,*

and $\hat{w} \leftarrow (y^N \mod N^2)(1+N)^{\hat{r}} \mod N^2$. *Now, consider two following cases:*

- $x \leftarrow\$ \mathbb{J}_N,\ y \leftarrow x^{2^T}$: *The tuple* (N, g, g^{2^T}, x, y) *is a squared tuple. Therefore,* $(\hat{u}, \hat{v}, \hat{y}, \hat{w})$ *is a sample from* $\mathcal{H}_0$ *since* $\hat{y} = x$ *and* $\hat{w} = x^{N2^T}(1+N)^{\hat{r}} \mod N^2$.
- $x \leftarrow\$ \mathbb{J}_N,\ y \leftarrow\$ \mathbb{J}_N$: (N, g, g^{2^T}, x, y) *is a uniform tuple and, consequently,* $(\hat{u}, \hat{v}, \hat{y}, \hat{w})$ *is of the form* $\mathcal{H}_1$ *since* $\hat{w} = c^N(1+N)^{\hat{r}} \mod N^2$ *for* $c = y$.

Finally, $\mathcal{R}_2$ *invokes* $\mathcal{A}_2(1^\lambda, \hat{z}, pp, \tau')$ *and outputs whatever* $\mathcal{A}_2$ *returns. Therefore,* $\mathcal{R}$*'s advantage in breaking the sequential squaring assumption in* $\mathbb{J}_N$ *is equal to* $\mathcal{A}$*'s advantage in distinguishing between* $\mathcal{H}_0$ *and* $\mathcal{H}_1$

Hybrid $\mathcal{H}_2$: *In this hybrid, the tuple* (u, v, y, w) *is constructed such that* $r, s \leftarrow\$ \mathbb{Z}_N,\ u \leftarrow g^{r+s} \mod N,\ v \leftarrow h^{(r+s)N}(1+N)^s \mod N^2,\ y \leftarrow\$ \mathbb{J}_N$ *and* $w \leftarrow c^N(1+N)^r \mod N^2$ *for* $c \in \mathbb{J}_N$. *The only difference between* $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *is that* y *is replaced with a uniformly chosen element from* $\mathbb{J}_N$. *Clearly, the distribution of* $\mathcal{H}_2$ *and* $\mathcal{H}_1$ *are identical.*

Hybrid $\mathcal{H}_3$: *Now, in this hybrid, for the tuple* (u, v, y, w) *we have* $r, s \leftarrow\$ \mathbb{Z}_N,\ c \leftarrow\$ \mathbb{J}_{N^2},\ u \leftarrow g^{r+s} \mod N,\ v \leftarrow h^{r+s}(1+N)^s \mod N^2,\ y \leftarrow\$ \mathbb{J}_N,\ w \leftarrow c(1+N)^r \mod N^2$. *The only difference between* $\mathcal{H}_2$ *and* $\mathcal{H}_3$ *is the way* w *is constructed. We will focus on the case where* z_0 *is sampled according to* $\mathcal{H}_3$ *and* z_1 *is sampled from* $\mathcal{H}_2$. *We prove the an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *who can distinguish between* z_0, *and* z_1 *will break the DCR assumption over* $\mathbb{Z}^*_{N^2}$. *Via proof by contradiction, we construct the* PPT *adversary* $\mathcal{R}$ *such that on receiving the input* (N, y) *chooses* T *and constructs* $g \leftarrow\$ \mathbb{J}_N$ *and* $h \leftarrow g^{2^T} \mod N$ *and sets* $pp \leftarrow (N, g, T, h)$. *Then, runs* $(s, \tau') \leftarrow \mathcal{A}_1(T, pp)$. *Finally, calls* $\mathcal{A}_2(1^\lambda, z, pp, \tau)$ *and returns the output of* $\mathcal{A}_2$ *where* $z = (\hat{u}, \hat{v}, \hat{y},\ \hat{w}),\ \hat{r} \leftarrow\$ \mathbb{Z}_N,\ \hat{u} \leftarrow g^{\hat{r}+s} \mod N,\ \hat{v} \leftarrow h^{N(\hat{r}+\hat{s})}(1+N)^s \mod N^2,\ \hat{y} \leftarrow\$ \mathbb{J}_N,\ w \leftarrow y(1+N)^{\hat{r}} \mod N^2$. *Now, we show that the inputs of* $\mathcal{A}_2$ *are always according to either* $\mathcal{H}_2$ *or* $\mathcal{H}_3$:

- $x \leftarrow\$ \mathbb{Z}^*_N,\ y = x^N \mod N^2$: *Note that with probablity almost 1/2,* $x \in \mathbb{J}_N$. *In that case,* $\hat{w}$ *will be* $x^N(1+N)^r \mod N^2$ *which has exactly the same distribution as* $\mathcal{H}_2$.
- $y \leftarrow\$ \mathbb{Z}^*_{N^2}$: *Similarly,* $y \in \mathbb{J}_{N^2}$ *with 1/2 probability. Then,* $\hat{w} = y(1+N)^{\hat{r}} \mod N^2$ *for a randomly chosen* y *from* $\mathbb{J}_{N^2}$ *with 1/2 chance which is in the form of* $\mathcal{H}_3$.

Therefore, assuming the advantage of $\mathcal{A}$ *is* σ, $\mathcal{R}$ *will have* $\sigma/2$ *advantage in breaking DCR assumption over* $\mathbb{Z}^*_{N^2}$.

Hybrid $\mathcal{H}_4$: *In this hybrid,* w *is replaced with a randomly chosen element from* $\mathbb{J}_{N^2}$. *Therefore,* $\mathcal{H}_4$ *is consist of* (u, v, y, w) *for* $r, s \leftarrow\$ \mathbb{Z}_N,\ u \leftarrow g^{r+s} \mod N,\ v \leftarrow h^{2^T}(1+N)^s \mod N^2,\ y \leftarrow\$ \mathbb{J}_N$, *and* $w \leftarrow\$ \mathbb{J}_{N^2}$. *Then, we first prove the below lemma:*

Lemma 14. *For* $r \in \mathbb{Z}_N$ *call* X_r *and* Y *as random variables which represent* $x = c(1+N)^r \mod N^2$ *for* $c \leftarrow\$ \mathbb{J}_{N^2}$ *and* $y \leftarrow\$ \mathbb{J}_{N^2}$ *respectively. Then,* X_r *and* Y *are statistically indistinguishable.*

Proof. First, we know that $gcd((1+N)^r, N^2) = 1$. *Additionally,* $\left(\frac{(1+N)^r}{N^2}\right) = +1$ *according to the result of the Lemma 11 which implies that* $(1+N)^r \in \mathbb{J}_{N^2}$. *Therefore* $(1+N)^r$ *has an inverse element in* $\mathbb{J}_{N^2}$ *and consequently, given* $x \in \mathbb{J}_{N^2}$ *and* $r \in \mathbb{Z}_N$, *there exist a unique* $c \in \mathbb{J}_{N^2}$ *such that* $x = c(1+N)^r \mod N^2$. *Thus, for every* $\alpha \in \mathbb{J}_{N^2}$ *we have:*

$$\Pr_{c \in \mathbb{J}_{N^2}}[c(1+N)^r \mod N^2 = a|\ r] = \frac{1}{||\mathbb{J}_{N^2}||}$$

Therefore:

$$\Delta(X_r, Y) = \frac{1}{2}\sum_{a \in \mathbb{Z}^*_{N^2}} \Big|\Pr[X_r = a] - \Pr[Y = a]\Big|$$

$$= \frac{1}{2}\sum_{a \in \mathbb{J}_{N^2}} \Big|\Pr_{c \in \mathbb{J}_{N^2}}[c(1+N)^r \mod N^2 = a|\ r] - \frac{1}{||\mathbb{J}_{N^2}||}\Big|$$

$$= \frac{1}{2}\sum_{a \in \mathbb{J}_{N^2}} \Big|\frac{1}{||\mathbb{J}_{N^2}||} - \frac{1}{||\mathbb{J}_{N^2}||}\Big| = 0$$

As mentioned, the only difference between $\mathcal{H}_3$ *and* $\mathcal{H}_4$ *is in instantiating* w. *Given the random variables* X_r *and* Y *as the distribution of* w *in* $\mathcal{H}_3$ *and* $\mathcal{H}_4$ *respectively, the result of Lemma 14 directly implies that the hybrids* $\mathcal{H}_4$ *and* $\mathcal{H}_3$ *are statistically indistinguishable.*

Hybrid $\mathcal{H}_5$*: In this hybrid, the tuple* (u, v, y, w) *is constructed as* $r, s \leftarrow\$ \mathbb{Z}_N$, $c \leftarrow\$ \mathbb{J}_N$, $u \leftarrow g^{r+s} \mod N$, $v \leftarrow c^N(1+N)^s \mod N^2$, $y \leftarrow\$ \mathbb{J}_N$, $w \leftarrow\$ \mathbb{J}_{N^2}$. *The only difference with* $\mathcal{H}_4$ *is the way* v *is constructed. Instead of* $h^N(1+N)^s \mod N^2$ *for a random* $c \in \mathbb{J}_N$ *it is sampled as* $c^N(1+N)^s \mod N^2$. *We will show that if we sample* $z_0 \leftarrow\$ \mathcal{H}_4$ *and* $z_1 \leftarrow\$ \mathcal{H}_5$, *then an efficient distinguisher* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *between* z_0 *and* z_1 *with the advantage* σ, *has a non-negligible advantage in distinguishing a squaring tuple in* $\mathbb{J}_N$. *Similar to the approach we designed between the hybrids* $\mathcal{H}_0$ *and* $\mathcal{H}_1$, *we use the attacker* $\mathcal{R} = (\mathcal{R}_1, \mathcal{R}_2)$ *such that on receiving the tuple* (N, T, g), $\mathcal{R}_1$ *sets* $h \leftarrow g^{2^T}$ *and* $pp \leftarrow (N, g, T, h)$. *Then, calls* $(s, \tau') \leftarrow \mathcal{A}_1(T, pp)$ *and returns* $\tau \leftarrow (s, pp, \tau')$. *Then,* $\mathcal{R}_2$ *after receiving* (x, y, h) *samples the tuple* $z \leftarrow (\hat{u}, \hat{v}, \hat{y}, \hat{w})$ *such that* $\hat{u} \leftarrow x$, $\hat{v} \leftarrow y^N(1+N)^s \mod N^2$, $\hat{y} \leftarrow\$ \mathbb{J}_N$, *and* $\hat{w} \leftarrow\$ \mathbb{J}_{N^2}$. *Finally, returns the output of* $\mathcal{A}_2(1^\lambda, z, pp, \tau')$. *Now, we will show that with* $5/8$ *probability, the inputs to* $\mathcal{A}_2$ *are distributed according to either* $\mathcal{H}_4$ *or* $\mathcal{H}_5$*:*

- $x \leftarrow\$ \mathbb{J}_N$, $y \leftarrow x^{2^T}$*: Assume* $x = g^\alpha$, *Therefore,* $\hat{u} = g^\alpha$ *and* $\hat{v} = g^{\alpha N 2^T}(1+N)^s \mod N^2$. *Since the distribution of* α *is close to uniform distribution in* $\mathbb{Z}_{N/2}$, *and* s *also comes uniformly from* $\mathbb{Z}_N$, *with probability* $1/4$, $\alpha \geq s$ *and we can write* $\hat{u}$ *as* $g^{(\alpha-s)+s}$ *and* $\hat{v}$ *as* $g^{((\alpha-s)+s)N2^T}(1+N)^s \mod N^2$. *Consequently, the tuple* $(\hat{u}, \hat{v}, \hat{y}, \hat{w})$ *has identical distribution as* $\mathcal{H}_4$ *with probability 1/4.*
- $x \leftarrow\$ \mathbb{J}_N$, $y \leftarrow\$ \mathbb{J}_N$*: Then* $\hat{u} = x$ *and* $\hat{v} = y^N(1+N)^s \mod N^2$ *for a randomly chosen* y *from* $\mathbb{J}_N$ *which clearly implies that* $(\hat{u}, \hat{v}, \hat{y}, \hat{w})$ *represents a sample from* $\mathcal{H}_5$.

Therefore, with probability $1/2*1/4+1/2 = 5/8$, $\mathcal{A}_2$ *is fed with proper inputs and can produce* σ *advantage. Consequently,* $\mathcal{R}$ *must have at least* $5\sigma/8$ *advantage to break the strong sequential squaring assumption in* $\mathbb{J}_N$.
Hybrid $\mathcal{H}_6$*: Same as* $\mathcal{H}_3$*, we can use* $c(1+N)^s \mod N^2$ *for* $c \leftarrow\$ \mathbb{J}_{N^2}$ *via a reduction against the DCR assumption in* $\mathbb{Z}^*_{N^2}$.
Hybrid $\mathcal{H}_7$*: Similar to what we did for* $\mathcal{H}_4$*, according to Lemma 11 we can replace* v *with a random sample from* $\mathbb{J}_{N^2}$.
Hybrid $\mathcal{H}_8$*: Finally, it is easy to conclude that there is no information about the secret* s *left in* u*. Therefore, we sample* u *from* $\mathbb{J}_N$.

F Proof of Theorem 2

Proof. Clearly, the scheme is public − coin since the verifier only broadcasts a randomly chosen e*. Additionally, we already know that* $\Psi.\mathsf{Verify}$ *is a succinct zero-knowledge argument of knowledge which guarantees that the exponent of* u*,* $r+s$ *is in* $\mathbb{Z}_N/n$.
Correctness: *The final step of* Verify *functionality in Algorithm 1 checks five conditions which are easy to verify their correctness:*

$$g^{\alpha} = g^{(r+s+k)e+x} = (g^{r+s}g^k)^e * g^x = (uy)^e * a$$
$$g^{\beta} = g^{(r+s)e+t} = (g^{r+s})^e * g^t = u^e * \tau$$
$$h^{\alpha N}(1+N)^{\beta} = h^{((r+s+k)e+x)N}(1+N)^{(r+s)e+t} \mod N =$$
$$h^{(r+s+k)eN}(1+N)^{(r+s)e} \mod N * h^{tN}(1+N)^x = (vw)^e b$$

Soundness: *We now try to build an emulator* E *that will extract the witness* $r+s$ *from a potentially malicious prover* P^**.* E *rewinds* P^* *to line number 23 of Figure 5 until it gets two accepting transcripts* $(\alpha_1, \beta_1, \tau_1)$ *and* $(\alpha_2, \beta 2, \tau_2)$ *for the challenges* $e_1,\ e_2 \leftarrow\$ \mathbb{Z}_{2^\lambda}$*. Such that* $g^{\alpha_1} = (uy)^{e_1} * a$*,* $g^{\alpha_2} = (uy)^{e_2} * a$.
Now, we want to find the discrete log of uy*. Assuming* $e_1 > e_2$*, take* $e' = e_1 - e_2$*,* $\alpha' = \alpha_1 - \alpha_2$*, and* $\beta' = \beta_1 - \beta_2$*. Therefore,* $g^{\alpha_1}/g^{\alpha_2} = (uy)^{e_1} * a/((uy)^{e_2} * a) = g^{\alpha'} = (uy)^{e'}$*. Next, according to Lemma 8,* e' *must divide* α'*. Therefore:*

- *If* e' *is odd: Since* $e' < p$ *and* $e' < q$*,* $gcd(e',\ |\mathbb{J}_N|)$ *will be one. Note that the order of the* $\mathbb{J}_N$ *(group generated by* g*) is* $\phi(N)/2 = 2pq$*. Then, via Lemma 5,* E *can extract the discrete log of* u *as* α'/e'.
- *If* e' *is even: We can write* e' *as* $2^d\bar{e}$*. Then given* $k = \alpha'/e'$*,* $g^{ke'} = (uy)^{e'} = g^{k2^d\bar{e}} = (uy)^{2^d\bar{e}}$*. Therefore, via Lemma 4 and given* $gcd(\bar{e},\ |\mathbb{J}_N|) = 1$ *we can say that* $g^{k2^d}(uy)^{-2^d} = 1$*. Additionally, there must be some* ρ *such that* $g^k(uy)^{-1} = g^{\rho}$*; therefore,* $\rho 2^d = k' * |\mathbb{J}_N| = k' * 2pq$ *for some* k' *since* $|g| = |\mathbb{J}_N|$*. Then,* 2^{d-1} *should divide* k' *since* p *and* q *are large prime numbers. Therefore, we can write* 2ρ *as* $k'/2^{d-1}2pq = k''2pq$ *for some integer* $k'' = k'/2^{d-1}$*. Subsequently, we can conclude that* $(g^k(uy)^{-1})^2 = (g^{\rho})^2 = g^{2\rho} = 1$*. Via Lemma 6, we know that* $g^k(uy)^{-1}$ *can be non-trivial with only negligible probability. Consequently, it must be* ± 1*, which implies that we can find the discreet logarithm of* uy*, which is* k.

Let $d_1 \leftarrow log_g(u)$ and $d_2 \leftarrow log_g(y)$. Therefore, so far, E could extract the value of $d_1 + d_2$. Using α_1 it can further find the value of x as well such that $g^x = a$. Via similar arguments, E can extract t. Additionally, following the result of Lemmas 10 and 9, for $F = \{(1+N)^x \mid x \in \mathbb{Z}_N\}$ we have $\mathbb{Z}_N^* \times F \stackrel{\sim}{=} \mathbb{Z}_{N^2}^*$. Note that the group $h >$ is only a subgroup of $\mathbb{Z}_N^*$ with exactly $1/4$ of the elements of $\mathbb{Z}_N^*$. Therefore, we can write v as $h^{v_1 N}(1+N)^{v_2} l_v \mod N^2$, w as $h^{w_1 N}(1+N)^{w_2} l_w \mod N^2$, and b as $h^{b_1 N}(1+N)^{b_2} l_b \mod N^2$ for some l_v, l_w and $l_b \in \mathbb{Z}_{N^2}^*$ such that for all $i, j \in \mathbb{Z}_N$, $h^{iN}(1+N)^j$ is co-prime to all of the l_v, l_w and l_b. Therefore, $(vw)_1^e b = h^{((v_1+w_1)e_1+b_1)N}(1+N)^{(v_2+w_2)e_1+b_2}(l_v l_w)^{e_1} l_b \mod N^2 = h^{\alpha_1 N}(1+N)^{\beta_1} \mod N^2$. It is easy to verify that $(l_v l_w)^{e_1} l_b$ must be 1. Since the prover does not know e_1 in advance, $l_v l_w = l_b = 1$; therefore, $(vw)^{e_1} b = h^{((v_1+w_1)e_1+b_1)N}(1+N)^{(v_2+w_2)e_1+b_2} = h^{\alpha_1 N}(1+N)^{\beta_1} \mod N^2$. As a result, $\alpha_1 = (v_1+w_1)e_1 + b_1$ and $\beta_1 = (v_2+w_2)e_1 + b_2$. Additionally, we knew that $\alpha_1 = (d_1+d_2)e_1 + x$; thus, $b_1 = x$ and $v_1 + w_1 = d_1 + d_2$. Additionally, $g^{\beta_1} = u^{e_1}\tau$ which implies that $\beta_1 = d_1 e + t$. Finally, we can show that $d_1 e + t = (v_2+w_2)e + b_2$, thus, $d_1 = v_2 + w_2$ and $b_2 = t$. Furthermore, E can easily extract the value of $v_2 + w_2$ as well using β_1. In conclusion, we showed that E can extract the exponent of u and j in $vw = h^{iN}(1+N)^j \mod N^2$ for some i, and i has to be equal to $log_g(u)$.

Zero-knowledge: *To show the zero-knowledge property, we use the method introduced in [31]. The transcript of protocol between the prover and the verifier in LTHP consists of $T = (a, b, \tau, e, \alpha, \beta)$. Now we build a simulator to re-construct a new transcript $T' = (a', b', e', \alpha', \beta', \tau)$ such that $\alpha' \leftarrow\$ \mathbb{Z}_{(N/2+N/n)*2^{2\lambda}}$, $\beta' \leftarrow\$ \mathbb{Z}_{N/n*2^{2\lambda}}$, $\tau' = g^{\beta'}/u^{e'}$, $a' \leftarrow g^{\alpha'}/u^{e'}$, $b \leftarrow h^{\alpha' N}(1+N)^{\beta'}/v^{e'}$, $e' \leftarrow\$ \mathbb{Z}_{2^\lambda}$ which is statistically indistinguishable from T with a set of hybrids:*

Hybrid *$\mathcal{H}_0$: It is the original transcript: $a \leftarrow g^x$, $b \leftarrow h^{xN}(1+N)^t \mod N^2$, $e \leftarrow \mathbb{Z}_{2^\lambda}$, $\alpha \leftarrow se + x$, $\beta \leftarrow se + t \mod N$, $\tau \leftarrow g^t \mod N$.*

Hybrid *$\mathcal{H}_1$: Unlike the real transcript, sample $\alpha \leftarrow\$ [(r+s+k)e, (r+s+k)e + (N/2+N/n) * 2^{2\lambda})$ and $a \leftarrow g^{\alpha-(r+s+k)e}$. Clearly, $\mathcal{H}_0$ and $\mathcal{H}_1$ are statistically indistinguishable.*

Hybrid *$\mathcal{H}_2$: Set $a \leftarrow g^\alpha/(uy)^e$. It is easy to verify that this new transcript is the same as $\mathcal{H}_1$.*

Hybrid *$\mathcal{H}_3$: This time, we extend the range of sampling α to $\alpha \leftarrow\$ \mathbb{Z}_{(N/2+N/n)*2^{2\lambda}}$. Then, we prove that the statistical distance between $\mathcal{H}_2$ and $\mathcal{H}_3$ is negligible. The only difference between these two is α. Therefore, with having X as the random variable of α in $\mathcal{H}_2$ and Y as the same variable in $\mathcal{H}_3$ below, we show that $\Delta(X, Y)$ is negligible in parameter λ.*

$$\Delta(X,Y) = \frac{1}{2} \sum_{\alpha\in[0,\ (r+s+k)e)} \frac{1}{(N/2+N/n)*2^{2\lambda}}$$
$$+\frac{1}{2} \sum_{\alpha\in[(r+s+k)e,\ (N/2+N/n)*2^{2\lambda})} \frac{1}{(N/2+N/n)*2^{2\lambda}} - \frac{1}{(N/2+N/n)*2^{2\lambda}}$$
$$+\frac{1}{2} \sum_{\alpha\in[(N/2+N/n)*2^{2\lambda},\ (r+s+k)e+(N/2+N/n)*2^{2\lambda})} \frac{1}{(N/2+N/n)*2^{2\lambda}}$$
$$= \frac{1}{(N/2+N/n)*2^{2\lambda}} \times (r+s+k)e$$
$$\leq \frac{1}{(N/2+N/n)*2^{2\lambda}} \times (N/2+N/n)*2^{\lambda} = 2^{-\lambda}$$

Hybrid $\mathcal{H}_4$*: Similarly, sample* $\beta \leftarrow\$ [(r+s)e, (r+s)e+N/n*2^{2\lambda})$ *and* $b \leftarrow h^{\alpha}(1+N)^{\beta-(r+s)e}$*. It is easy to verify that* $\mathcal{H}_3$ *and* $\mathcal{H}_4$ *are statistically indistinguishable.*
Hybrid $\mathcal{H}_5$*: Set* $b \leftarrow h^{\alpha N}(1+N)^{\beta}/(vw)^{e}$ *and* $\tau = g^{\beta}/u^{e}$ *This new transcript is clearly the same as* $\mathcal{H}_4$*.*
Hybrid $\mathcal{H}_6$*: Finally, we extend the range of sampling* α *to* $\beta \leftarrow\$ \mathbb{Z}_{N/n*2^{2\lambda}}$*. Same as before, we can prove that the statistical distance between* $\mathcal{H}_5$ *and* $\mathcal{H}_6$ *is negligible. The only difference between these two is* β*. Therefore, with having* X *as the random variable of* β *in* $\mathcal{H}_5$ *and* Y *as the same variable in* $\mathcal{H}_3$ *below we show that* $\Delta(X,Y)$ *is negligible in parameter* λ *as it is shown below:*

$$\Delta(X,Y) = \frac{1}{2} \sum_{\beta\in[0,\ (r+s)e)} \frac{1}{N/n*2^{2\lambda}} + \frac{1}{2} \sum_{\alpha\in[(r+s)e,\ N/n*2^{2\lambda})} \frac{1}{N/n*2^{2\lambda}} - \frac{1}{N/n*2^{2\lambda}} +$$
$$\frac{1}{2} \sum_{\beta\in[N/n*2^{2\lambda},\ (r+s)e+N/n*2^{2\lambda})} \frac{1}{N/n*2^{2\lambda}}$$
$$= \frac{1}{N/n*2^{2\lambda}} \times (r+s)e \leq \frac{1}{N/n*2^{2\lambda}} \times N/n*2^{\lambda} = 2^{-\lambda}$$

Therefore, we proved that the real transcript T *is statistically indistinguishable from the transcript generated by the simulator,* T'*.*

G Proof of Lemma 1

Proof. Note that the main difference between Lemmas 13 and 1 is the sampling range of s *and* r *which are now instead from* $\mathbb{Z}_{N/2n}$*. Let* σ *be the advantage of* $\mathcal{A}$ *in the experiment of Lemma 1. Then, using* $\mathcal{A}$*, we construct the adversary* $\overline{\mathcal{A}} = (\overline{\mathcal{A}_1}, \overline{\mathcal{A}_2})$ *that engages in the experiment of Lemma 13.* $\overline{\mathcal{A}_1}$ *upon receiving the* (T, pp) *returns a randomly generated* $s \leftarrow\$ \mathbb{Z}_{N/2n}$ *and an empty advice* $\tau \leftarrow \perp$*. Then,* $\overline{\mathcal{A}_2}$ *on receiving* $(1^{\lambda}, z_b, pp, \tau)$ *returns the output of* $\mathcal{A}(1^{\lambda}, z_b, pp)$*. It is easy to verify that the advantage of* $\overline{\mathcal{A}}$ *will be* $\bar{\sigma} = \sigma(1/2 * 1/2n + 1/2)$*. Additionally,*

the result of Lemma 13 shows that $\bar{\sigma} = \mathsf{negl}(\lambda)$. *Therefore,* $\sigma(1/2 * 1/2n + 1/2) = \mathsf{negl}(\lambda) = \sigma(1/4p(\lambda) + 1/2)$*; thus,* $\sigma = \mathsf{negl}(\lambda) * 4p(\lambda)/(1 + 2p(\lambda)) = \mathsf{negl}(\lambda)$. *Consequently, there cannot be any adversary that succeeds in the experiment of Lemma 1.*

H BLS Signature

Algorithm 3 ***BLS signature [9]*** *Given* $e : \mathrm{G}_0 \times \mathrm{G}_1 \to \mathrm{G}_T$ *a pairing operation between three algebraic groups* G_0, G_1, *and* G_T *with prime order* q *and generators* g_0, g_1, g_T *respectively, and a hash function* $H : \mathcal{M} \to \mathrm{G}_0$*:*

- Gen*: Returns* (pk, sk) *where* $sk \leftarrow\!\!\$\ \mathcal{Z}_q$ *is the private key and* $pk \leftarrow g_0^{sk}$ *is the public key.*
- Sign(sk, m)*: For message* $m \in \mathcal{M}$ *outputs* $H(m)^{sk}$.
- Verify(pk, m, σ)*: Outputs* $e(\sigma, g_1) = e(pk, H(m))$
- AggregateAndVerify$((pk_0, m_0, \sigma_0), ..(pk_n, m_n, \sigma_n))$*: Returns* $e(\sigma, g_1) = \Pi\ e(pk_i, H(m_i))$
- AggregateAndVerifySame$(m, ((pk_0, \sigma_0), ..(pk_n, \sigma_n)))$*: Returns* $e(\sigma, g_1) = e(\Pi\ pk_i, H(m))$

The function AggregateAndVerifySame *can only be used when all the signatures share the same message.*

I Puzzle Invalidity Check

We recall the Π_{ACorSol} *from [31]. This protocol uses the VDF proposed by [54] to allow a puzzle solver to prove the invalidity of a puzzle* $(u, v) \in \mathbb{J}_N \times \mathbb{J}_{N^2}$. *Its corresponding relation is:*

$$\mathcal{R} = \{(N, T, u, v) : \forall s \in \mathbb{Z}_N,\ v \neq u^{2^T}(1 + N)^s \mod N^2\}$$

Let Prime(2λ) *be the set of the first* $2^{2\lambda}$ *primes. Then, the description of this protocol between the prover* P *and the verifier* V *is as follows:*

1. P *sends* $w \leftarrow u^{2^T} \mod N$ *to* V.
2. V *picks a random prime number* l *from* Prime(2λ) *and sends* l *to* P.
3. *Let* $2^T = ql + r$ *for* $0 \leq r < l$ *and* $q \in \mathbb{N}$. *Then, efficiently computes* $\pi \leftarrow u^q \mod N$ *and sends* π *to* V.
4. *Finally,* V *outputs accept if* $w = \pi^l u^r \mod n$ *and* $\pi \in \mathbb{Z}_N$ *and* N *does not divide* $(vw^{-N} \mod N^2) - 1$, *and rejects otherwise.*

Theorem 5. *The protocol* Π_{ACorSol} *is an honest-verifier argument for the relation* $\mathcal{R}$.