# A Mempool Encryption Scheme for Ethereum via Multiparty Delay Encryption

Amirhossein Khajehpour[1], Hanzaleh Akbarinodehi[2], Mohammad Jahanara[3], and Chen Feng[4]

[1] =nil; Foundation
khajepour.amirhossein@gmail.com
[2] University of Minnesota Twin Cities
akbar066@umn.edu
[3] Scroll Foundation
mohammad@scroll.io
[4] The University of British Columbia, Kelowna
chen.feng@ubc.ca

**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. Furthermore, current block production architecture allows transaction censorship that compromises credible neutrality, a fundamental principle of Ethereums design philosophy. In this work, we present a novel *mempool encryption* scheme to alleviate the censorship and 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. 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 and prove its security under the presence of only one honest attestation aggregator per Ethereum slot.

**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, and immutability. This has enabled a broad range of emerging applications [43,1] such as *decentralized finance* (DeFi). Today's mainstream smart contract platforms

are designed in such a way that (1) pending transactions have no privacy, and (2) block proposers have unrestricted rights of inclusion and ordering of transactions within their block. More elaborately, users create their transactions in *plain-text* and broadcast it to the blockchain network where every node maintains its local view of pending transactions in a data structure called *mempool*[5]. Finally, it is up to a block proposer to include those transactions and decide the ordering. Thus, the block proposer can effectively censor certain transactions based on their content; or extract value from the ordering of transactions, a phenomenon frequently referred to as *miner/maximal extractable value* (MEV) in the literature [18].

If a substantial portion of block proposers actively censor transactions, motivated by financial gains or coerced through regulation, it can result in (i) undermining the credible neutrality of Ethereum and (ii) compromising the safety of time-sensitive DeFi applications operating on Ethereum (*e.g.* censoring oracle updates or liquidation transactions). Similarly, the ramifications of MEV ripple across the security and efficiency of the underlying blockchain platform [34,29], including (i) destabilization of the consensus mechanism [13], (ii) centralization of the network [47], and (iii) financial inefficiencies that deteriorate the user experience [18]. Specifically, in the last few months, almost 70% of blocks in Ethereum were actively censoring certain transactions [45]. Moreover, as of the time of this writing, the realized value attributed to MEV since The Merge has surged past an astonishing 300,000 ETH [20].

Motivated by the adverse implications of censorship attacks and MEV, the notion of mempool encryption (*a.k.a.,* mempool privacy) was introduced and studied in the literature [36,7,24]. Mempool encryption, at its core, is a scheme to encrypt transactions so that their content remains private until their inclusion is finalized, and inclusion guarantees decryption and execution. One can verify that an ideal mempool encryption scheme severely limits a proposer's ability to censor transactions based on their content or extract MEV. In addition, mempool encryption removes the burden of policing transactions for proposers under the risk of regulatory action.

Mempool encryption can be realized using a range of practical and theoretical approaches. In the following, we review the most prominent ones:

- **Threshold encryption**. This approach utilizes a threshold public-key encryption (TPKE) scheme. In a TPKE scheme, a committee publishes a public key, and each committee member holds a distinct decryption key. An encrypted message can be decrypted if and only if more than a threshold of the committee members cooperate.
  In a threshold encryption based mempool encryption scheme, users encrypt their transactions via the committee's public key. Hence, block proposers have to determine the order of the transactions without knowing their contents. The committee performs threshold decryption of included transactions, and subsequently, transactions will be executed. In practice, projects

[5] Alternatively users can directly send their transactions to trusted third-parties or a block builder

| | Number of messages | | | Sound | Security assumption |
|---|---|---|---|---|---|
| | Key Generation (per round) | Encryption (per transaction) | Decryption (per $\tau$ transactions) | | |
| Ferveo[7] | | $\mathcal{O}(1)$ | $\mathcal{O}(n \times \tau)$ | ✗ | $t$ out of $n$ honest players |
| ETHTID[42] | $\mathcal{O}(n^2)$ | $\mathcal{O}(1)$ | $\mathcal{O}(t \times \tau)$ | ✗ | $t$ out of $n$ honest players |
| F3B (TDH2)[50] | $\mathcal{O}(n^2)$ | $\mathcal{O}(1)$ | $\mathcal{O}(t \times \tau)$ | ✗ | $t$ out of $n$ honest players |
| F3B (PVSS)[50] | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(t \times \tau)$ | ✗ | $t$ out of $n$ honest players |
| tlock[21] | $\mathcal{O}(n^2)$ | $\mathcal{O}(1)$ | $\mathcal{O}(t)$ | ✗ | $t$ out of $n$ honest players |
| FairBlock[30] | $\mathcal{O}(n)$ | $\mathcal{O}(1)$ | $\mathcal{O}(t)$ | ✗ | $t$ out of $n$ honest players. |
| MDE | $\mathcal{O}(n)$ | $\mathcal{O}(1)$ | NA[6] | ✓ | 1 out of $n$ honest players, trusted setup |

**Table 1.** Comparison of communication complexity between MDE and number of other protocols that use threshold encryption approaches. The "Sound" column refers to whether the scheme is fully censorship-resistant or not. In general, the committee might decide not to reveal the shares, or the honest quorum might not be formed to reconstruct the shares, which results in denying the execution of certain transactions. However, in the case of time-lock encryption, such as MDE, all the encrypted transactions will eventually get decrypted and executed, unless the entire network stops operating.

such as Shutter network [39], FairBlock [30], and several others [49,4,32,40,7] utilize threshold mempool encryption. However, this approach has the following shortcomings: (i) It relies on a significant committee's honest majority trust assumption. That is, if a majority of committee members collude, they can violate the mempool's privacy guarantee by decrypting transactions before inclusion. Moreover, they can refuse to decrypt included transactions and stall their execution indefinitely [25,17,37]. (ii) The communication cost of these schemes is high and grows with the number of committee members; thus, the committee has to be relatively small. Table 1 compares the communication complexity of different threshold encryption based methods proposed by previous work. This exacerbates the aforementioned problem regarding the honest majority assumption.

– **Trusted Execution Environment (TEE)**. TEE is a secure area within a device's main processor, which guarantees secure storage and processing of data, and integrity of execution for applications executed on it. Note that the host processor can not manipulate or inspect data and programs on the TEE. Additionally, TEEs are shipped with attestation keys provided by the manufacturer that are required to prove their provenance. Some projects have attempted to leverage TEEs for mempool encryption [46]. In a TEE-based mempool encryption scheme, block proposers all operate within a TEE, and users encrypt their transactions with a public key that is created and shared by TEEs. The challenges of this approach are (i) the security guarantees of a TEE are contingent on a significant trust assumption about the manufacturer; (ii) numerous side-channel attacks were discovered against major TEEs such as Intel SGX, leaving users question the long-term security [5].

[6] The puzzle can be solved offline by everyone; therefore, no message is required.

- **Witness encryption**. This approach relies on a cryptographic primitive that is widely believed to be nonpractical and inefficient. Witness encryption, introduced by Garg *et al.* [22], allows for the encryption of data in such a way that the decryption key is not a traditional secret key, but rather a "witness" to a particular mathematical statement or problem. In this case, the witness for each encrypted transaction is an inclusion proof for that transaction in a block that has been attested to by a large enough number of node operators.
- **Time-lock encryption**. Time-lock puzzle (TLP) is a cryptographic primitive to encrypt messages into the future [35]. More precisely, the decryption procedure is inherently sequential and takes an inevitable number of computation levels, that can be utilized to impose a specific delay for which the message remains encrypted[7] Numerous works have utilized TLP to design mempool encryption schemes. We categorize these works into the following categories.
  (i) One TLP per transaction: In this approach, users have to put their transactions inside a separate TLP and then propagate it, such as the method proposed in [25]. At each round, the new block includes encrypted transactions and executes now-decrypted transactions from the previous blocks. Given that each transaction is a separate puzzle, the network participants have to solve many independent puzzles simultaneously, which renders this approach resource-intensive and uneconomical.
  (ii) One TLP per round: In this approach, the TLP is designed in such a way that there is only one puzzle to solve per round and it is independent of the number of transactions. It can be done in two ways: 1) Similar to the previous approach, one can have each TLP hide a single transaction while enabling TLPs to be aggregated into one. Consequently, with solving only the final puzzle, all the individual puzzles will be solved. Following this approach, Thyagarajan *et al.*[44] proposed a technique called *batch-solvable time-lock puzzles*; however, it scales each TLP's length linearly by the number of transactions. This limitation makes their approach impractical to use in a blockchain setting of thousands of transactions per block. In a subsequent work proposed by Srinivasan *et al.* [41], they overcame this issue at the expense of relying on purely theoretical cryptographic constructions. 2) Instead of hiding transactions themselves, another approach is to cleverly hide the secret key of a publicly known public key inside a time-lock puzzle [11]. The puzzle is constructed in a way that no one knows the secret key in advance. As a result, every transaction that is encrypted via the public key is guaranteed to be decrypted only once the time-lock puzzle is solved and the secret key is found. The ideas in this category will be discussed in more detail in Section 3.

[7] Note that utilizing time-lock encryption is only interesting for problems that require short-term delays. That is because with the advancement in hardware technologies such as CPUs and ASICs, performing basic math operations would become faster which results in solving the puzzles faster. In practice, we consider the fastest hardware technology present to choose the puzzle parameters and also increase the protocol's difficulty adaptively.

As it was pointed out, current mempool encryption schemes either fail in maintaining the communication or storage complexity, leverage highly theoretical approaches, or have significant trust issues. In this paper, by introducing a new time-lock encryption primitive, we present a mempool encryption scheme that operates under minimal trust assumptions. While our protocol utilizes a committee, unlike threshold encryption based solutions, it doesn't rely on an honest-majority assumption. Instead, it relies solely on the presence of at least one honest party. In addition, our protocol is highly scalable and bears minimal communication/storage overhead, only $\mathcal{O}(n)$ messages per round of the protocol for $n$ size of committee. Furthermore, our protocol seamlessly integrates with the current Ethereum architecture through a straightforward workflow.

Our proposed scheme is initialized with a one-time setup phase that generates a safe-prime RSA modulus. Then it randomly selects a committee of parties for each round; which collectively generates the current round's public key along with a time-lock puzzle concealing their shares of the secret key[8]. At each round, users encrypt their transactions using the public key of one of the recent rounds. Subsequently, block proposers build blocks including encrypted transactions while simultaneously deciphering transactions from the previous round's time-lock puzzles.

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

- **Formalization of** MempoolEncryption**.** Assuming a synchronous network model, we formally define the notion of mempool privacy that captures the minimum desired properties that we expect from a mempool encryption scheme.
- **Multiparty delay encryption.** The notion of *multiparty delay encryption* (MDE) is introduced. Then, by leveraging cryptographic time-lock puzzles, an MDE is designed with its security rigorously established under standard cryptographic assumptions.
- **Scalable** MempoolEncryption **gadget for Ethereum.** Using MDE, we propose a MempoolEncryption scheme for Ethereum. Importantly, the safety of this method relies on the presence of only one honest attestation aggregator, for each slot in today's Ethereum design, making it a trust-minimized solution. Additionally, the communication and storage costs of our method are negligible and it is highly scalable in terms of handling transactions. Table 1 outlines the communication cost involved in each phase of our protocol. The extra space our approach uses per block header is only 230 KB for 112-bit security.

The rest of this paper is organized as follows: Section 2 formally defines mempool encryption. In Section 3, we will discuss the limitations of current time-lock encryption methods and then give an overview of our method. Section 4 focuses on preliminaries and notations necessary for the rest of the paper. In

[8] In the context of Ethereum, the committee members are selected from the set of attestation aggregators in the current epoch.

Section 5, we will explain how we construct an MDE using our modified time-lock puzzle primitive based on the previous works. In Section 6, we will show the implementation details of adding MDE to Ethereum consensus. Later, Finally, Section 7 concludes this paper and suggests potential future directions.

### 1.1 Notation

The symbol $\wedge$ is being used as the logical "and" operator. $x \leftarrow\!\$\ \mathcal{S}$ is used to show that $x$ is uniformly sampled from the set $\mathcal{S}$. $|\mathcal{S}|$ for the set $\mathcal{S}$ indicates the size of $\mathcal{S}$. $|\mathbb{G}|$ for the group $\mathbb{G}$ denotes its order. For the group $\mathbb{G}$ and $\alpha \in \mathbb{G}$, the notion $ord_{\mathbb{G}}(\alpha)$ denotes the order of $\alpha$ in $\mathbb{G}$, and, $\langle\alpha\rangle$ is the subgroup generated by $\alpha$. The set $\mathbb{Z}_N^*$ is $\{x \mid gcd(x, N) = 1 \wedge x < N\}$. The group $\mathbb{J}_N = \{x | \left(\frac{x}{N}\right) = +1\}$ is all the elements whose Jacobi symbols with $N$ are $+1$. Note that $|\mathbb{J}_N| = 1/2|\mathbb{Z}_{N^*}|$. The operator $||$ is used for concatenation. The notion $\mathsf{negl}(\lambda)$ denotes a negligible function where $\mathsf{negl}(\lambda) < 1/p(\lambda)$ for every polynomial $p$. Two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are denoted as isomorphic via $\mathbb{G}_1 \cong \mathbb{G}_2$.

## 2 Problem Definition

The concept of mempool encryption has been studied before [7,30,39]. However, to the best of our knowledge, there is no formal and rigorous definition for mempool encryption. In this section, we formally specify the notion of mempool encryption.

There are two sets of participants in the network, validators (*i.e.* miners) and clients (*i.e.* users). Time is organized into rounds within a $\Delta$-synchronous network. This means that any messages between any two participants are guaranteed to be delivered within a maximum delay of $\Delta$ rounds. The parameter $\Delta$ represents the network's delay and is universally known to all parties involved. Each party following the protocol is called an honest player, and otherwise an adversary.

A state machine replication (SMR) protocol operates among the validators. This protocol takes transactions from the clients as input and generates a distributed ledger $\mathcal{L}$ as output. This ledger is an ordered sequence of transaction groups, each referred to as a block. Henceforth, we call $\mathcal{L}$ a *blockchain*.

Clients generate and broadcast transactions to the network. Furthermore, they request different validators for their versions of the blockchain. Upon receiving responses from the requested validators, clients aggregate and process them using a protocol $\mathcal{P}$, to create their local view of the blockchain. We denote this local view as $\mathcal{L}_r^c$, representing the blockchain in the view of client $c$ at round $r$. [9]

Blocks in $\mathcal{L}$ are labeled as $\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \cdots$ . Every block comprises two components: the body and the header. The body of each block contains an ordered sequence of transactions. Additionally, the header includes a reference

[9] For example, a client can do one of the following: (i) trust a single RPC provider, (ii) run a light client for the chain, or (iii) run a full node.

to the previous block within $\mathcal{L}$, and also incorporates some protocol-related metadata. We call a block is valid if it satisfies the protocol-related validity criteria*e.g.*signatures validations, par. We say $\mathcal{L}_1$ is a prefix of $\mathcal{L}_2$, denoted by $\mathcal{L}_1 \preceq \mathcal{L}_2$, if for all blocks $\mathbf{B} \in \mathcal{L}_1$, we have $\mathbf{B} \in \mathcal{L}_2$. Two ledgers, $\mathcal{L}_1$ and $\mathcal{L}_2$, are considered conflicting if neither of them is a prefix of the other.

For each security parameter $\lambda$, and $\ell$ as a polynomial function of $\lambda$, we say that $\mathcal{P}$ is $\ell$-secure, if $\mathcal{P}$ satisfies both of the following properties.

**Definition 1.** ***Safety:*** *For any honest clients $c_i, c_j$ and rounds $r, r'$ the ledgers $\mathcal{L}_r^{c_i}$ and $\mathcal{L}_{r'}^{c_j}$ are not conflicting. Also, for any client $c_i$ and rounds $r \leq r'$ the ledger $\mathcal{L}_r^{c_i}$ is a prefix of $\mathcal{L}_{r'}^{c_i}$.*

**Definition 2.** ***$\ell$-Liveness:*** *Assume the transaction $\mathsf{tx}$ is created by an honest client and is received by an honest validator at round $r$. Then $\mathsf{tx}$ should be included in $\mathcal{L}_{r'}^{c_i}$, for any honest client $c_i$, and any round $r' \geq r + \ell$.*

**Definition 3.** $\mathsf{MempoolEncryption}$*: We call the protocol $\Pi_{\mathsf{ME}} = (\mathsf{Encrypt}_{\mathcal{L}}, \mathsf{Decrypt}_{\mathcal{L}})$ is a $\mathsf{MempoolEncryption}$ protocol for the $\ell$-secure and $\Delta$-synchronous blockchain $\mathcal{L}$, if it provides a transaction encryption protocol such that*[10]*:*

- *Completeness: $\Pi_{\mathsf{ME}}.\mathsf{Encrypt}_{\mathcal{L}}$ takes as input a transaction $\mathsf{tx}$ and outputs its encryption $\mathsf{tx}^{\mathcal{E}}$. On the contrary, $\Pi_{\mathsf{ME}}.\mathsf{Decrypt}_{\mathcal{L}}$ takes as input a block $\mathbf{B} \in \mathcal{L}$ whose transactions are $[\mathsf{tx}_0^{\mathcal{E}}, \mathsf{tx}_1^{\mathcal{E}}, \cdots]$, and outputs $[\mathsf{tx}_0, \mathsf{tx}_1, \cdots]$ such that for all $j$, we have $\Pi_{\mathsf{ME}}.\mathsf{Encrypt}(\mathsf{tx}_j) = \mathsf{tx}_j^{\mathcal{E}}$. In addition, for any encrypted block, the parallel runtime*[11] *of $\Pi_{\mathsf{ME}}.\mathsf{Decrypt}_{\mathcal{L}}$ is bounded by $\rho(\ell)$, for some fixed polynomial $\rho$. Moreover, any encrypted transaction $\mathsf{tx}^{\mathcal{E}}$ included in $\mathcal{L}$ gets decrypted (and executed) before $\rho(\ell)$ delay.*
- *Soundness: For any transaction $\mathsf{tx}^{\mathcal{E}}$ that was encrypted at round $r$, and* propagated immediately*, there is no parallel algorithm to decrypt it in less than $\ell + \Delta$ rounds.* [12]

## 3 Technical Overview

The initial idea of time-locked puzzles was introduced by Rivest *et al.* [35]. Informally speaking, a time-lock puzzle hides a message such that it can only be revealed after performing a time-consuming computation that does not have any shortcut nor is parallelizable. A naive $\mathsf{MempoolEncryption}$ scheme design is to hide each transaction inside an independent time-lock puzzle with delay parameter $d$. However, in this approach, the computation needed to uncover all transactions, scales linearly with the number of transactions. Ideally, the objective is to reveal all encrypted transactions at the cost of solving one puzzle! To this end, we propose $\mathsf{MDE}$, a scheme to hide a secret key inside a time-lock puzzle, corresponding to a public key. The transactions will be encrypted by the

[10] We omit ledger $\mathcal{L}$ whenever it is clear from the context.
[11] Note that we denominate time in unit of round.
[12] In particular, there is no decryption circuit with depth less than $d - \Delta$ for $\mathsf{Encrypt}_{\mathcal{L}}$.

public key, and all of them can be decrypted once the puzzle hiding the secret key is solved. Our construction is inspired by [28] and utilizes homomorphic time-lock puzzles, to facilitate multi party generation of the secret key inside the puzzle, and the corresponding public key. More details of our construction are in Sections 3.2 and 5.

### 3.1 Related Work

In a parallel line of work, Doweck *et al.* [19] introduced the notion of *multi-party timed commitments*. Players use a multi-party computation (MPC) scheme to generate an ElGamal public key, for which the secret key is unknown to everyone. The secret key can be extracted from the public key by breaking the computational Diffie-Hellman (CDH) over a carefully chosen group. The protocol takes a delay parameter as input that specifies the *expected time* it takes to extract the secret key. Hence, everyone can encrypt their message by the public key and expect their message to be revealed with a delay. However, [19] is not suitable to instantiate a MempoolEncryption scheme, as it can not guarantee an exact delay time and also the initial MPC relies on multiple rounds of interaction. Additionally, this scheme is only secure against attackers with bounded parallelization power. More precisely, the speed of secret key extraction scales linearly with the parallelization power of the attacker.

Another way to instantiate a MempoolEncryption scheme is using delay encryption [16]. Burdges *et al.* introduced delay encryption [11]; in which, given a delay parameter $T$, the scheme uses a trusted setup to generate a master encryption key pk and an extraction key ek. Using pk and a session id, one can encrypt a message. Moreover, using ek and id any encrypted message corresponding to id can be decrypted, and this operation admits delay $T$. In [16] Chiang *et al.* propose a delay encryption-based MempoolEncryption scheme and study adversarial block delivery delays in this context, and propose a change to the consensus mechanism of proof-of-stake networks to circumvent them. Noteworthy, the underlying cryptographic primitive can be replaced trivially with MDE, which uses a significantly lighter cryptography construction. Moreover, it is not clear how [16] can be adopted by Ethereum, while our work provides a concrete design for seamless integration with Ethereum protocol. In addition, the storage requirements for delay encryption construction of [11], which has been used in [16], grows linearly with the delay parameter $T$, while the storage complexity of our construction for MDE is independent of the delay parameter $T$, that can be of independent interest.

An alternative way to design a MempoolEncryption is to use batch-solvable time-lock puzzles[44,41], in which independent puzzles can be aggregated together to generate a batched puzzle that hides all of the messages. Using a batch-solvable time-lock puzzle in the naive MempoolEncryption, the cost of revealing all of the transactions is the same as the cost of solving one puzzle. However, the existing constructions for batch solving either rely on in practical cryptographic primitives such as Indistinguishable Obfuscation (IO) [41,23], or

the size of puzzles (size of encrypted transactions) scales linearly with the number of maximum batchable puzzles (maximum number of transaction per block) [44]. The latter is particularly problematic for high-throughput blockchains.

### 3.2 Multiparty Delay Encryption: Informal Description

We introduce the notion of multiparty delay encryption (MDE), in which a group of $n$ non-trusting participants can generate and publish a pair $(\overline{\mathsf{pk}}, \overline{\mathsf{ek}})$, where $\overline{\mathsf{pk}}$ is a public key with its secret key $\overline{\mathsf{sk}}$ hidden inside a time-lock puzzle which can be obtained from the extraction key $\overline{\mathsf{ek}}$. Any message $m$ can be encrypted via $\overline{\mathsf{pk}}$, and is kept secret until the time-lock puzzle is solved and the value of $\overline{\mathsf{sk}}$ is extracted.

MDE has an initial **setup** phase to generate the public parameters of the system. Following this, each participant $i \in [n]$, individually **generates** MDE shares $(\mathsf{pk}_i, \mathsf{ek}_i)$, which are subsequently **aggregated** into the final pair $(\overline{\mathsf{pk}}, \overline{\mathsf{ek}})$. Each participant is also required to publish a **verification** proof of their share's integrity. Any message can be **encrypted** by $\overline{\mathsf{pk}}$. Anyone can **extract** the secret key $\overline{\mathsf{sk}}$ from the extraction key $\overline{\mathsf{ek}}$, which can be used to **decrypt** all the encrypted messages.

An MDE scheme should satisfy the following properties. The formal definitions will be provided in Section 4.

**Definition 4.** MDE ***Correctness (Informal):*** *If all the shares $(\mathsf{pk}_i, \mathsf{ek}_i)$ pass the **verification**, then for all the messages, the **decryption** and the **encryption** must work properly.*

**Definition 5.** MDE ***Adaptive 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. Additionally, the malicious parties have the ability to see honest parties' messages before taking their actions.*

### 3.3 Construction

Our construction for the MDE is based on homomorphic time-lock puzzles (HTLPs). In [28], the HTLP is defined as 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 $T$. The puzzle can be solved by calculating $x = u^{2^T} \mod N$, and 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 $(u_1, v_1)$ hiding the secrets $s_0$ and $s_1$ respectively, then the puzzle $(u_0u_1, v_0v_1)$ hides the secret $s_0 + s_1$.

A natural approach to build an MDE is to use these HTLPs as MDE shares, that is $(\mathsf{pk}_i, \mathsf{ek}_i) \leftarrow (u_i, v_i)$. Note that each $u_i \leftarrow g^{r_i} \mod N$ is a valid ElGamal

public key in $\mathbb{J}_N$[13]. While the shares can be aggregated to obtain $\overline{\mathsf{pk}} = g^{\sum r_i} \mod N$ as desired, we can't extract $\overline{\mathsf{sk}} = \sum_i r_i$ from $\overline{\mathsf{ek}} = h^{\overline{\mathsf{sk}}N}(1+N)^{\sum s_i} \mod N^2$. More precisely, unlike $\sum s_i$, the puzzle structure does not provide any way to extract $\overline{\mathsf{sk}}$, other than solving a discrete log problem in $\mathbb{J}_N$. Therefore, we need to modify the construction of HTLP to meet all the necessary requirements of our MDE.

In our proposed construction for HTLP, each puzzle is defined as 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 $k \leftarrow\$ \mathbb{Z}_{N/2}$ is another randomness factor. In this construction, the portion $u = g^{r+s} \mod N$ is $\mathsf{pk}$, the tuple $(u, v, y, w)$ is $\mathsf{ek}$, and $r + s$ is $\mathsf{sk}$. To extract $\mathsf{sk}$, first the solver calculates $\bar{u} = u \times y = g^{r+s+k} \mod N$, $\bar{v} = v \times w = h^{(r+s+k)N}(1+N)^{r+s} \mod N^2$, and $x = \bar{u}^{2^T} \mod N$. One can verify that the value of $\mathsf{sk}$ is $((\bar{v}/x^N \mod N^2) - 1)/N$. Hence after, we call this protocol a modified linearly homomorphic time-lock puzzle ($\Pi_{\mathsf{MTLP}}$) and provide its detailed explanation and security proofs in Appendix D.

To build MDE, we use the proposed construction of MLHTLP. More precisely, each party $p_i$ publishes a share including public key $\mathsf{pk}_i = u_i$ and extraction key $\mathsf{ek}_i = (u_i, v_i, y_i, w_i)$, along with a zero-knowledge argument of knowledge[14] of $r_i$, $s_i$ and $k_i$. The aggregated public key and secret key is $\overline{\mathsf{pk}} = \Pi u_i \mod N$ and $\overline{\mathsf{sk}} = \sum r_i + s_i$, respectively. Similarly, the aggregated extraction key is $\overline{\mathsf{ek}_i} = (\Pi u_i \mod N, \Pi v_i \mod N^2, \Pi y_i \mod N, \Pi w_i \mod N^2)$. Later in Theorem 2, we prove that $\overline{\mathsf{sk}}$ will remain hidden as long as we have at least one honest party.

## 4 Multiparty Delay Encryption: Formal Description

In this section, we provide a formal description of MDE.

**Definition 6.** ***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$(1^\lambda, T)$*: A probabilistic algorithm that on receiving the security parameter $1^\lambda$ outputs the system's public parameters* $\mathsf{pp}$ *according to the delay $T$. Setup runs only once at the beginning and can be a potentially distributed algorithm.*
- Gen$(n, \mathsf{pp})$*: A probabilistic algorithm that, on receiving the system's public parameters* $\mathsf{pp}$*, and $n \in \mathbb{N}$ the maximum number of participants in generating the final public-key, outputs the tuple* $(\mathsf{ek}, \mathsf{pk})$ *such that* $\mathsf{ek}$ *is the extraction key corresponding to the encryption public-key* $\mathsf{pk}$*.*

[13] When $N$ is a safe-prime RSA modulus, the decisional Diffie-Hellman (DDH) problem is widely believed to be hard in $\mathbb{J}_N$ [38,9].

[14] This proof helps us to show correctness and adaptive security. The cryptographic primitive for the zero-knowledge proof is similar to the one used in [27].

- $\mathsf{Verify}(n, (\mathsf{ek}, \mathsf{pk}), \mathsf{pp})$*: A probabilistic protocol that checks the construction of* $\mathsf{ek}$ *and shows the prover poses the knowledge of* $\mathsf{sk}$ *corresponding to* $\mathsf{pk}$ *through a zero-knowledge protocol*[15]*. 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 verification.*
- $\mathsf{Aggregate}((\mathsf{ek}_1, \mathsf{pk}_1), (\mathsf{ek}_2, \mathsf{pk}_2) \dots (\mathsf{ek}_k, \mathsf{pk}_k), \mathsf{pp})$*: A deterministic algorithm that receives system's public parameters* $\mathsf{pp}$*, and many* $(\mathsf{ek}_i, \mathsf{pk}_i)$ *tuples. It returns an aggregated* $(\overline{\mathsf{ek}}, \overline{\mathsf{pk}})$*, which will later be used for message encryption.*
- $\mathsf{Encrypt}(m, \mathsf{pk})$*: A probabilistic algorithm which on receiving the message* $m \in \mathcal{M}$ *and the public-key* $\mathsf{pk}$*, outputs* $c \in C$*, the encryption of* $m$*, according to* $\mathsf{pk}$*.*
- $\mathsf{Decrypt}(c, \mathsf{sk})$*: A deterministic algorithm that decrypts the ciphertext* $c$ *via the decryption key* $\mathsf{sk}$ *and returns a message* $m \in \mathcal{M}$*.*
- $\mathsf{Extract}(\mathsf{ek}, T, \mathsf{pp})$*: A deterministic algorithm which receives the extraction key* $\mathsf{ek}$ *and system's public parameters* $\mathsf{pp}$ *along with delay* $T$*. Then, outputs the decryption key* $\mathsf{sk}$ *that corresponds to* $\mathsf{ek}$ *or* $\perp$*.*

**Definition 7.** MDE ***Correctness****. The* MDE *protocol* $\Pi_{\mathsf{MDE}} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Verify}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Extract})$ *is correct if given* $\lambda$*, for all polynomials* $T$ *in* $\lambda$*, all messages* $m \in \mathcal{M}$*, all* $n \in \mathbb{N}$ *that* $n = p_0(\lambda)$ *for a fixed polynomial* $p_0$*, all* $\mathsf{pp}$ *in the support set of* $\Pi_{\mathsf{MDE}}.\mathsf{Setup}(n, 1^\lambda, T)$*, and all* $(\mathsf{ek}_i, \mathsf{pk}_i)$ *where* $\Pi_{\mathsf{MDE}}.\mathsf{Verify}(n, (\mathsf{ek}_i, \mathsf{pk}_i), \mathsf{pp}) = \mathit{true}$*, given* $(\overline{\mathsf{ek}}, \overline{\mathsf{pk}}) \leftarrow \Pi_{\mathsf{MDE}}.\mathsf{Aggregate}(\mathsf{ek}_0, \mathsf{pk}_0) \dots (\mathsf{ek}_{n-1}, \mathsf{pk}_{n-1}), \mathsf{pp})$*, and* $\overline{\mathsf{sk}} \leftarrow \Pi_{\mathsf{MDE}}.\mathsf{Extract}(\overline{\mathsf{ek}}, T\ \mathsf{pp})$*, the running time of the algorithm* $\Pi_{\mathsf{MDE}}.\mathsf{Extract}$ *is bounded with* $p_1(\lambda, T)$ *for a fixed polynomial* $p_1$ *and we have:*

$$\Pi_{\mathsf{MDE}}.\mathsf{Decrypt}(\Pi_{\mathsf{MDE}}.\mathsf{Encrypt}(m, \overline{\mathsf{pk}}), \overline{\mathsf{sk}}) = m. \tag{1}$$

**Definition 8.** MDE ***Adaptive Security****. Let the* MDE *protocol* $\Pi_{\mathsf{MDE}} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Verify}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Extract})$*, where* $\mathcal{E}$ *is the extractor of the* Verify *algorithm which can extract the witness from the prover.* $\Pi_{\mathsf{MDE}}$ *is adaptively secure if for any probabilistic polynomial time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$*, where the depth of* $\mathcal{A}_2$*,* $\mathcal{A}_3$ *and* $\mathcal{E}$ *are 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:*

[15] More precisely, there exist an extractor $\mathcal{E}$ of the Verify algorithm which can extract the witness from the prover

$$\Pr\left[\begin{array}{l|c} & pp \leftarrow \Pi_{\mathsf{MDE}}.\mathsf{Setup}(n, 1^\lambda, T), \\ & \tau_1 \leftarrow \mathcal{A}_1(T, pp, 1^\lambda), \\ & (ek_n, pk_n) \leftarrow \Pi_{\mathsf{MDE}}.\mathsf{Gen}(n, pp), \\ \forall_{i\in\{1,\cdots,n-1\}} & \\ \Pi_{\mathsf{MDE}}.\mathsf{Verify}(n, & \\ (ek_i, pk_i), pp) & ((m_0, m_1), (ek_1, pk_1), \cdots \\ = \mathit{True} & (ek_{n-1}, pk_{n-1}), \tau_2) \leftarrow \\ & \mathcal{A}_2((ek_n, pk_n), \tau_1), \\ \wedge & \\ \mathcal{A}_3(c, \tau_2) = b & (\overline{ek}, \overline{pk}) \leftarrow \Pi_{\mathsf{MDE}}.\mathsf{Aggregate}( \\ & (ek_1, pk_1), \cdots, (ek_n, pk_n), pp), \\ & \\ & b \leftarrow\!\!\$\ \{0, 1\}, \\ & c \leftarrow \Pi_{\mathsf{MDE}}.\mathsf{Encrypt}(m_b, \overline{pk}) \end{array}\right]$$

$$\leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

## 5 Proposed Solution

In this section, we present the details of MDE and its security and correctness proofs.

**Algorithm 1** *Assuming $\Pi_{\mathsf{MTLP}}$ is our modified linearly homomorphic time-ock puzzle protocol and $\Pi_{\mathsf{ERP}}$ is any non-interactive zero-knowledge (NIZK) exponent range proof protocol such that $\Pi_{\mathsf{ERP}}.\mathsf{Verify}(b,\ x,\ a,\ N)$ returns true if there exists $e \in \mathbb{Z}_N$ such that $x = b^e \mod N$ and $e < a$. Our proposed solution is as follows:*

- $\mathsf{Setup}(1^\lambda, T)$*: Runs* $pp \leftarrow \Pi_{\mathsf{MTLP}}.\mathsf{Setup}(1^\lambda, T)$ *and returns* $pp$.
- $\mathsf{Gen}(n, pp)$*: It works similar to the* $\Pi_{\mathsf{MTLP}}.\mathsf{Gen}$ *with a difference in choosing the boundries of $s$ and $r$. 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 $w = h^{kN}(1+N)^r \mod N^2$. (See Figure 1 for further details.)*
- $\mathsf{Verify}(n, (ek, pk), pp)$*: 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 $\Pi_{\mathsf{ERP}}.\mathsf{Verify}(g, u, N/n, N)$. Next, the prover samples $x \leftarrow\!\!\$\ \mathbb{Z}_{(N/2+N/n)\times 2^{2\lambda}}$, $t \leftarrow\!\!\$\ \mathbb{Z}_{(N/n)\times 2^{2\lambda}}$ 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 $(\alpha, \beta)$ for $\alpha = (r+s+k)e + x$ and $\beta = (r+s)e + t$. Finally, to verify the puzzle $(u, v)$, returns $g^\alpha = (uy)^e \times a \ \wedge\ h^{\alpha N}(1+N)^\beta = (vw)^e b \ \wedge\ g^\beta = u^e \times \tau$. (See Figure 1 for further details.)*
- $\mathsf{Encrypt}(m, pk, pp)$*: Samples $r \leftarrow\!\!\$\ \mathbb{Z}_{N/2}$ and returns $(c_1, c_2)$ with $c_1 = g^r$ and $c_2 = m \times pk^r$.*

- $\mathsf{Decrypt}((c_1, c_2), \mathsf{sk}, \mathsf{pp})$: *Returns* $c_2/c_1^{\mathsf{sk}}$.
- $\mathsf{Aggregate}((\mathsf{ek}_1,\ \mathsf{pk}_1)\ldots(\mathsf{ek}_n, \mathsf{pk}_n), \mathsf{pp})$: *Runs* $\overline{\mathsf{ek}} = \Pi_{\mathsf{MTLP}}.\mathsf{Aggregate}(\mathsf{ek}_1\ \ldots\ \mathsf{ek}_n,\ \mathsf{pp})$, *and returns* $(\overline{\mathsf{ek}}, \overline{\mathsf{pk}})$ *for* $\overline{\mathsf{pk}} = \Pi\,\mathsf{pk}_i$.
- $\mathsf{Solve}(\overline{\mathsf{ek}}, \mathsf{pp})$: *Compiles* $\overline{\mathsf{ek}}$ *into the tuple* $(u, v, y, w)$ *and returns* $\Pi_{\mathsf{MTLP}}.\mathsf{Solve}((uy,\ vw,\ \perp,\ \perp),\ \mathsf{pp})$.

Note that with the Verify procedure, a party proves the knowledge of sk for a given ek in zero-knowledge. Its corresponding relation is:

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

Consider the following theorem.

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

*Proof.* Proof details are in Appendix F.

*Remark 1.* Note that the Verify procedure can be easily converted to *malicious-verifier zero-knowledge proof* via the Fiat-Shamir heuristic.

In the following theorem we show that our proposed algorithm satisfies correctness and adaptive security.

**Theorem 2.** *Given that* $\Pi_{\mathsf{MTLP}}$ *is a secure time-lock puzzle, Algorithm 1 is a correct and adaptively secure multiparty delay encryption.*

*Proof.* Proof details are in Appendix G.

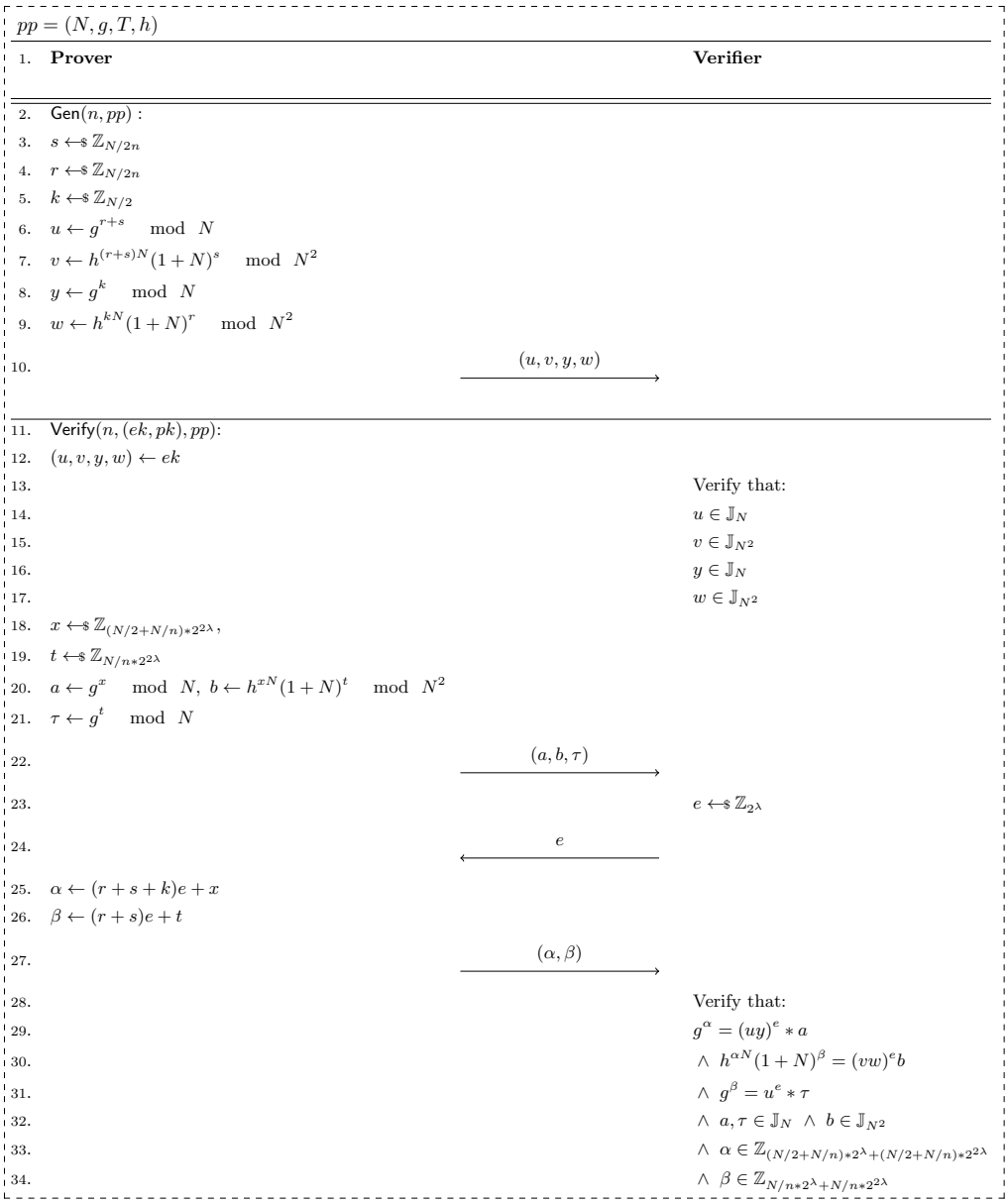


**Fig. 1.** In illustration of the Gen and Verify algorithms in MDE.

# 6 Mempool Encryption Scheme

In this section, we explain how to utilize a MDE to design a network that satisfies MempoolEncryption. We first give some background about Ethereum and then describe the changes that need to be made. Then, we will prove that our design is indeed an MempoolEncryption. Finally, we discuss encryption efficiency and spam protection.

## 6.1 Ethereum Background

In the 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 attestation committees[16] 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 for consensus mechanisms [12]. Attestations are signed with BLS signatures, 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 attestation aggregators.

| $\lambda$ | 512 | 1024 | 2048 | 4096 |
|---|---|---|---|---|
| Public parameters | $384 + \|T\|$ | $768 + \|T\|$ | $1,536 + \|T\|$ | $3,072 + \|T\|$ |
| Verification proof | 1,024 | 2,48 | 4,096 | 8,192 |
| Share | 768 | 1,536 | 3,072 | 6,144 |
| Verification proof + Share | 1,792 | 3,584 | 7,168 | 14,336 |

**Table 2.** Space complexities in bytes for different values of $\lambda$.

## 6.2 Protocol

Assume the Ethereum network is $\Delta$-synchronous and $l$-secure. Let $\Pi_{\mathsf{MDE}} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Verify}, \mathsf{Aggregate}, \mathsf{Solve})$ be an MDE protocol such that $\Pi_{\mathsf{MDE}}.\mathsf{Setup}(1^\lambda)$ was already done, and everyone knows the public parameters $\mathsf{pp} = (N, g, T, h)$. 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 $120M/32 \leq 4000000$. At each round we need to aggregate puzzles from all attestation aggregators. Given the aforementioned upper bound on the number of validators, setting the maximum number of puzzles we can aggregate to $n = 10000000$ is more than enough. Additionally, let $d = \ell + 2\Delta$ and assume the time needed to run $\Pi_{\mathsf{MDE}}.\mathsf{Solve}$ for the delay $T$ is equal

[16] They are often called Beacon committees link

to proposing $d$ slots. Note that even though the cryptographic assumption on which $\Pi_{\mathsf{MDE}}$ 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. We use the notation $\mathbf{B}^{-i}$ to represent the $i^{th}$ parent of the block $\mathbf{B}$ in the chain. Below are the actors involved in the $\Pi_{\mathsf{ME}}$ protocol and their corresponding responsibilities during each Ethereum protocol slot:

- **Attestations Aggregator**: In addition to their other responsibilities, aggregators collaboratively create the $(\overline{\mathsf{pk}}, \overline{\mathsf{ek}})$ pair for each slot. More precisely, each aggregator generates a new share via $\Pi_{\mathsf{MDE}}.\mathsf{Gen}$ and makes a validity proof for it using $\Pi_{\mathsf{MDE}}.\mathsf{Verify}$. Then, along with the aggregated attestation, it signs its MDE share and broadcasts it.
- **Block proposer**: To propose a new block $\mathbf{B}$, it performs the following:
  - It verifies all the MDE shares received from attestation aggregators via $\Pi_{\mathsf{MDE}}.\mathsf{Verify}$. If any of the verifications fail, it drops that attestation; otherwise, it includes the attestation in $\mathbf{B}$'s header. For convenience, we call the aggregated public key of shares in $\mathbf{B}$ via $\Pi_{\mathsf{MDE}}.\mathsf{Aggregate}$, the public key proposed in block $\mathbf{B}$.
  - Let $\overline{\mathsf{sk}}$ be the result of $\Pi_{\mathsf{MDE}}.\mathsf{Solve}$ on $\mathbf{B}^{-d}$'s aggregated MDE share. The block proposer must include $\overline{\mathsf{sk}}$ in $\mathbf{B}$'s header as well.
  - Decrypts all the encrypted transactions of the block $\mathbf{B}^{-d}$ via $\Pi_{\mathsf{MDE}}.\mathsf{Decrypt}$, and executes them.[17]
  - Fills $\mathbf{B}$'s transactions list with the pending transactions in the mempool that are encrypted via one of the $\mathbf{B}^{-1}, \mathbf{B}^{-2}, \ldots, \mathbf{B}^{-2\Delta}$ proposed public keys.
- **User**: Let $\mathbf{B}'$ be the latest block in the user's view and $\overline{\mathsf{pk}}$ be its proposed public key. The user encrypts $\mathsf{tx}$ via $\Pi_{\mathsf{MDE}}.\mathsf{Encrypt}(\mathsf{tx}, \overline{\mathsf{pk}})$ and broadcasts it.

Due to network delay, users might not be able to see the latest block. Therefore, we allow them to encrypt their transactions via the public key included in one of the most recent $d-l$ blocks. The additional checks described below should be added to the validation procedure of the block $\mathbf{B}$:

- Verify signatures and validity proofs of all MDE shares included in $\mathbf{B}$. All shares should be properly signed by their corresponding attestation aggregator's public key and pass the $\Pi_{\mathsf{MDE}}.\mathsf{Verify}$ check, otherwise the block is deemed invalid.
- Check the validity of the solution $\overline{\mathsf{sk}}$. Given $(\overline{\mathsf{ek}}, \overline{\mathsf{pk}})$ the aggregated MDE share of $\mathbf{B}^{-d}$, if the solution is wrong, meaning that $\overline{\mathsf{pk}}$ and $\overline{\mathsf{sk}}$ does not match, the block $\mathbf{B}$ is invalid.

[17] The corresponding secret keys of the encrypted transactions in block $\mathbf{B}^{-d}$ must have been revealed in the current or preceding blocks.

### 6.3 Security Proof

In Theorem 3 we prove that the protocol described in the previous subsection is indeed a MempoolEncryption protocol for the Ethereum network.

**Theorem 3.** *$\Pi_{\mathsf{ME}}$ is a* MempoolEncryption *protocol for the Ethereum network if there is at least one honest aggregated attestation included in each slot and given standard cryptographic assumptions in Theorem 2.*

*Proof.* The correctness of $\Pi_{\mathsf{MDE}}$, proved in Theorem 2, implies correctness of $\Pi_{\mathsf{ME}}$ where $\rho = p_1$.
Now suppose an honest user creates an encrypted transaction $\mathsf{tx}^{\mathcal{E}}$ at round $r$, with proposed public key of the last block in its view and propagates it immediately. Furthermore, existing one honest attestation aggregation in each slot guarantees the adaptive security of $\Pi_{\mathsf{MDE}}$. Given that network is $\Delta$ synchronous, no parallel algorithm can decrypt $\mathsf{tx}^{\mathcal{E}}$ in less than $\ell + \Delta$ rounds; thus, implying the soundness of $\Pi_{\mathsf{ME}}$.

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

**Table 3.** Time complexity of all the operations for different values of $\lambda$ and $T = 0x093226$. Experiments were run on a Macbook Pro 2.3 GHz Quad-Core Intel Core i5 CPU.

### 6.4 Transaction Encryption

Notice that the encrypted transactions still have to pay the 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 the 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. 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 a data field surpassing $\lambda/4$ bytes, additional storage is required. Nevertheless, through the integration of symmetric key cryptography, optimization techniques can also be employed.

### 6.5 Spam Protection and Meta-Data Privacy

Permissionless blockchains fundamentally need transaction fee to protect themselves from spam and denial-of-service attacks. Moreover, it is vital for blockchain nodes to be able to identify pending invalid transactions and not propagate them in the network, which includes transactions that are not able to pay the fee or simply have an incorrect signature or nonce. In other words, nodes need to be able to verify the validity of pending transactions or else an attacker can flood the network with invalid transactions with virtually no cost.

A mempool encryption scheme can provide privacy at different levels, depending on which parts of the transaction it hides. For instance, one choice is to only encrypt fields `to`, `value`, and `input-data` leaving other fields such as `signature`[18], `from`, `nonce`, and `gas-limit` exposed. This way, it would be possible to determine whether the transaction has the minimum balance to cover the execution fee. It is easy to see that such a mempool encryption scheme, leaks a lot of meta-data about each transaction.

Another approach is to encrypt the whole transaction but ask users to submit a ZK-SNARK proof of validity for the plain-text transaction. While this approach provides complete privacy, it involves designing complex circuits, prune to bugs, and requires client-side proof generation that can be computationally expensive. Moreover, we have to make sure that each account only makes one transaction per round[19], otherwise a malicious user can flood the network with many transaction that are individually valid but only one of them can be included (e.g. they share the same nonce or there are enough funds in the signer account to cover fees only for one of the transactions).

We propose an alternative solution, to provide complete privacy. Let us introduce a new type of transaction called *transaction-carrying transaction* (TCT). A TCT transaction outer-tx carries an encrypted transaction $\mathsf{tx}^{\mathcal{E}}$ inside, which is going to be decrypted and executed with a delay. The intent is for the outer-tx to pay for the inclusion and decryption costs[20]. Note that the gas price of this type of transaction has to be adjusted relative to the resources consumed by such transactions. The fees paid by outer-tx provides protection against invalid transactions, because regardless of the validity of the $\mathsf{tx}^{\mathcal{E}}$, an attacker has to pay for costs incurred by the network.

## 7 Conclusion and Future Work

We reviewed mempool encryption and defined a new notion called MempoolEncryption. Our main contribution starts with the introduction of the mulitparty delay encryption (MDE). Then, we proposed a construction for MDE based on the idea

[18] Note that for the purpose of the signature, the hash of transaction is calculated with the encrypted fields rather than plain-text.

[19] This can be done by assigning nullifier to each account that gets used and renewed by each transaction.

[20] This cost can be thought of as the cost of data availability and effort necessary for decryption.

of time-lock puzzles. Next, we adopted our algorithm to minimally change the current specification of the Ethereum network to achieve censorship and MEV resistance. We showed that if there is at least one honest attestation aggregator per slot, and the basic consensus assumptions are held, then our protocol is a MempoolEncryption for the Ethereum. 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 a share already uses small space, enabling MDE still might put considerable storage overhead on the ledger when choosing larger $\lambda$. For example, choosing $\lambda = 1024$ which gives us 112 bit security will increase the block header's size by roughly 230 KB [21]. A potential solution for this issue would be to only store a SNARK proof along with the aggregation result of all the 64 verifiable shares. Additionally, our current design suggests using ECDSA signature schemes to prove the authenticity of the verifiable shares. Therefore, the block proposer to prove the signatures are valid can show that the aggregated share is the multiplication of different values that are signed by a set of known ECDSA public keys.
- Table 3 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 [26,31], 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 [22].
- The proposed Verify does not allow us to verify the aggregated share. Recent advancements in composable zero-knowledge proofs [2,8] might enable us to build composable and recursive proofs that can replace the Verify functionality.
- As was mentioned, BLS signatures are aggregatable when the messages to be signed are identical; however, when combining different shares we can not leverage this feature. Adopting BLS signature to our scheme to be used instead of ECDSA signature would be highly of interest.
- Efficiently designing a trustless distributed RSA modulus generation has its long literature [14,15,10]. 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.

[21] The average of Ethereum block size is almost $170,000$ bytes [48] and to store 64 aggregated attestations we need $3,584 \times 64 = 229,376$ bytes.

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

## References


1. Abou Jaoude, J., Saade, R.G.: Blockchain applications–usage in different domains. Ieee Access **7**, 45360–45381 (2019)
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. Babel, K., Ji, Y., Juels, Ari Kelkar, M.: Prof: Fair transaction-ordering in a profit-seeking world (2023), `https://initc3org.medium.com/prof-fair-transaction-ordering-in-a-profit-seeking-world-b6dadd71f086`
6. 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)
7. Bebel, J., Ojha, D.: Ferveo: Threshold decryption for mempool privacy in bft networks. Cryptology ePrint Archive (2022)
8. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. Algorithmica **79**, 1102–1160 (2017)
9. Boneh, D.: The decision diffie-hellman problem. In: International algorithmic number theory symposium. pp. 48–63. Springer (1998)
10. Boneh, D., Franklin, M.: Efficient generation of shared RSA keys. In: Annual international cryptology conference. pp. 425–439. Springer (1997)
11. Burdges, J., Feo, L.D.: Delay encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 302–326. Springer (2021)
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. Chiang, J.H.y., David, B., Eyal, I., Gong, T.: FairPoS: Input Fairness in Permissionless Consensus. In: Bonneau, J., Weinberg, S.M. (eds.) 5th Conference on Advances in Financial Technologies (AFT 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 282, pp. 10:1–10:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2023). https://doi.org/10.4230/LIPIcs.AFT.2023.10, `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.AFT.2023.10`

17. Cline, D., Dryja, T., Narula, N.: Clockwork: An exchange protocol for proofs of non front-running
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. Doweck, Y., Eyal, I.: Multi-party timed commitments. arXiv preprint arXiv:2005.04883 (2020)
20. Flashbots: Mev-explore pre-merge (2023), `https://explore.flashbots.net`
21. Gailly, N., Melissaris, K., Romailler, Y.: tlock: practical timelock encryption from threshold bls. Cryptology ePrint Archive (2023)
22. Garg, S., Gentry, C., Sahai, A., Waters, B.: Witness encryption and its applications. In: Proceedings of the forty-fifth annual ACM symposium on Theory of computing. pp. 467–476 (2013)
23. Kavousi, A., Abadi, A., Jovanovic, P.: Timed secret sharing. Cryptology ePrint Archive (2023)
24. Kavousi, A., Le, D.V., Jovanovic, P., Danezis, G.: Blindperm: Efficient mev mitigation with an encrypted mempool and permutation. Cryptology ePrint Archive (2023)
25. Khalil, R., Gervais, A., Felley, G.: Tex-a securely scalable trustless exchange. Cryptology ePrint Archive (2019)
26. Koc, C.K.: High-speed RSA implementation. Tech. rep., Technical Report TR-201, RSA Laboratories (1994)
27. Liu, Y., Wang, Q., Yiu, S.M.: Towards practical homomorphic time-lock puzzles: Applicability and verifiability. Cryptology ePrint Archive (2022)
28. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: Annual International Cryptology Conference. pp. 620–649. Springer (2019)
29. 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)
30. Momeni, P., Gorbunov, S., Zhang, B.: Fairblock: Preventing blockchain frontrunning with minimal overheads. In: International Conference on Security and Privacy in Communication Systems. pp. 250–271. Springer (2022)
31. Orup, H.: Simplifying quotient determination in high-radix modular multiplication. In: Proceedings of the 12th Symposium on Computer Arithmetic. pp. 193–199. IEEE (1995)
32. 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/`
33. 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)
34. 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)
35. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)

36. Rondelet, A., Kilbourn, Q.: Threshold encrypted mempools: Limitations and considerations. arXiv preprint arXiv:2307.10878 (2023)
37. Sekar, V.: Preventing front-running attacks using timelock encryption. Ph.D. thesis, University College London (2022)
38. Seurin, Y.: New constructions and applications of trapdoor ddh groups. In: Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26–March 1, 2013. Proceedings 16. pp. 443–460. Springer (2013)
39. 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
40. Sikka: Sikka projects (2022), https://sikka.tech/projects/
41. 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)
42. Stengele, O., Raiber, M., Müller-Quade, J., Hartenstein, H.: Ethtid: Deployable threshold information disclosure on ethereum. In: 2021 Third International Conference on Blockchain Computing and Applications (BCCA). pp. 127–134. IEEE (2021)
43. Tasatanattakool, P., Techapanupreeda, C.: Blockchain: Challenges and applications. In: 2018 International Conference on Information Networking (ICOIN). pp. 473–475. IEEE (2018)
44. 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)
45. Toni, W.: Ethereum censorship dashboard (2024), https://censorship.pics/
46. Woetzel, C.: Secret network: A privacy-preserving secret contract & decentralized application platform. Accessed: Jan **27** (2022)
47. Yang, S., Zhang, F., Huang, K., Chen, X., Yang, Y., Zhu, F.: SoK: MEV countermeasures: Theory and practice. arXiv preprint arXiv:2212.05111 (2022)
48. ycharts.com: Ethereum average block size (i:ebs) (2023), https://ycharts.com/indicators/ethereum_average_block_size
49. 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)
50. Zhang, H., Merino, L.H., Qu, Z., Bastankhah, M., Estrada-Galiñanes, V., Ford, B.: F3b: a low-overhead blockchain architecture with per-transaction front-running protection. In: 5th Conference on Advances in Financial Technologies (AFT 2023). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2023)

# A Lemmas

**Lemma 1.** *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,*

$$\begin{aligned} x^N \mod N^2 = (kN + r)^N \mod N^2 = r^N + \\ r^{N-1} \times kN \times N + ... \ (kN)^N \mod N^2 \\ = r^N \mod N^2 = (x \mod N)^N \mod N^2 \end{aligned}$$

**Lemma 2.** *Given a cyclic group $\mathbb{G}_c$, its generator $g$, and an arbitrary element $x \in \mathbb{G}_c$, 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 $\alpha$. Therefore, assume otherwise. Then, $ord_{\mathbb{G}_c}(x)$ should divide $\alpha$. Additionally, $ord_{\mathbb{G}_c}(x)$ must divide $|\mathbb{G}_c|$ according to the Lagrange theorem. Therefore, $gcd(\alpha, |\mathbb{G}_c|) \geq ord_{\mathbb{G}_c}(x) > 1$*

**Lemma 3.** *Given a cyclic group $\mathbb{G}_c$, its generator $g$, and arbitrary elements $x, y \in \mathbb{G}_c$ such that $x^\alpha = y^\beta$, if $gcd(|\mathbb{G}_c|, d) = 1$ and $d$ divides both $\alpha$ and $\beta$, 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 2 and considering the fact that $d$ and $|\mathbb{G}_c|$ are coprime, we can conclude that $x^{\alpha/d} y^{-\beta/d} = 1$ and finally $x^{\alpha/d} = y^{\beta/d}$*

**Lemma 4.** *Assume $g$ is a generator of the cyclic group $\mathbb{G}_c$. For an element $y \in \mathbb{G}_c$ where $g^\beta = y^\alpha$ such that $\alpha | \beta$, then the discrete logarithm of $y$ in $\mathbb{G}_c$ will be $\beta/\alpha$ if $gcd(\alpha, |\mathbb{G}_c|) = 1$.*

*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 $gcd(\alpha, |\mathbb{G}_c|) = 1$, via the result of Lemma 1 we can conclude that $g^k y^{-1} = 1$. Therefore, $y = g^{\beta/\alpha}$*

**Lemma 5.** *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 6.** *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 7.** *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 $\alpha$ are in $\mathbb{N}$ 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 $\alpha$ 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 3 we can assume that $d = 1$[23]. 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 6*
- *$d' = 2$: 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 3 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 $\rho$ 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 6 we can break the RSA strong assumption in $\mathbb{J}_N$.*
  - *$g^\omega = -u^\gamma$: Since $gcd(\omega, \gamma) = 1$, at least one of $\omega$ or $\gamma$ must be odd. Without loss of generality, assume $\gamma$ is odd. Then, $g^\omega = -u^\gamma = (-u)^\gamma$. Again, via Lemma 6, 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 $\omega$ is odd, we can use the equation $-g^\omega = u^\gamma$ and reach the same contradiction.*

**Lemma 8.** *There is an isomorphism $\mathbb{Z}_N^* \times \mathbb{Z}_N \cong \mathbb{Z}_{N^2}^*$ [33].*

**Lemma 9.** *For the group $\mathbb{F} \leftarrow \{(1 + N)^x \mod N^2 \mid x \in \mathbb{Z}_N\}$ and $\mathbb{Z}_N$ there is an isomorphism $\mathbb{F} \cong \mathbb{Z}_N$.*

*Proof. First*

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

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

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

*Clearly, $\psi$ is bijective. Additionally:*

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

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

[23] Via taking $e$ as $e \times d'/d$ and $\alpha$ as $\alpha \times d'/d$

**Lemma 10.** *For every odd* $N \in \mathbb{N}$, $\left(\frac{1+N}{N^2}\right) = +1$.

*Proof.*

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

## B Basic Definitions

**Definition 9.** ***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:*

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

*where* $X$ *and* $Y$ *are both in the domain* $D$.

***Definition 10. Statistically Indistinguishable*** *Two random variables* $X$ *and* $Y$ *in domain* $D$ *are said to be statistically indistinguishable by parameter* $\lambda$ *if:*

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

***Definition 11. Computationally Indistinguishable*** *Two random variables* $X$ *and* $Y$ *in domain* $D$ *are said to be computationally indistinguishable by the parameter* $\lambda$ *if for every* PPT *algorithm* $\mathcal{D}$*:*

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

*We show computational indistinguishability via* $X \approx_c Y$

***Definition 12. 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 13. Zero-knowledge*** *An interactive protocol* $\Pi = (P,V)$ *for the language* $\mathcal{L}$ *given* $view(P,V)(x)$ *as the transcript of executing* $\Pi$ *between* $P$ *and* $V$ *is said to be zero-knowledge if there exists a* PPT *algorithm* $\mathcal{S}$ *(Simulator) such that for the two probability ensembles* $view(P,V)(x)$ *and* $\mathcal{S}(x)$ *we have* $view(P,V)(x) \approx_c \mathcal{S}(x)$.

***Definition 14. Zero-knowledge Interactive Protocol*** *The protocol* $\Pi(P,V)$ *is zero-knowledge interactive if it is complete, sound, and zero-knowledge.*

***Definition 15. 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 16. 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 $\mathcal{E}$ such that $\forall_{x\in\mathcal{L}}\ R(x, \mathcal{E}^P(x)) = accept$.*

***Definition 17. Safe RSA Modulus*** *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.*

## C Cryptographic Assumptions

**Assumption 1** ***Strong RSA****[6] Given an RSA modulus $N$, and $e \in Z_{\phi(N)^*}$ when $e > 2$, no* PPT *algorithm can efficiently find the value of $m$ from $c = m^e \mod N$.*

**Assumption 2** ***Strong Sequential Squaring****[28] Given $N$ a safe RSA modulus, $g$ a generator of $\mathbb{J}_N$, for any* PPT *$\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ which the depth of $\mathcal{A}_2$ is bounded by $T^\epsilon(|N|)$ from above where $T$ is a polynomial and $0 < \epsilon < 1$:*

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

$$\leq \frac{1}{2} + \mathsf{negl}(|N|).$$

**Assumption 3** ***Decisional Composite Residuosity (DCR)****[33] Given a RSA modulus $N$ and any* PPT *adversary $\mathcal{A}$, the decisional composite residuosity assumption over $\mathbb{Z}^*_{N^2}$ states that:*

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

$$\leq \frac{1}{2} + \mathsf{negl}(|N|).$$

## D Modified Linearly Homomorphic Time-lock Puzzle

*We have borrowed the syntax of the homomorphic time-lock puzzle from [27,28]:*

***Definition 18. Homomorphic Time-Lock Puzzle (HTLP)*** *Let $\mathbb{C}_\lambda$ for the security parameter $\lambda \in \mathbb{N}$ be a class of circuits and $\mathbb{S}$ a finite domain. Then, a homomorphic time-lock puzzle is a tuple* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Solve}, \mathsf{Aggregate})$ *such that:*

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

***Definition 19. HTLP Correctness*** *The HTLP protocol $\Pi =$* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Solve}, \mathsf{Aggregate})$ *is correct if given $\mathbb{C}_\lambda$ a class of circuits, for all polynomials $T$ in $\lambda$, all circuits $c \in \mathbb{C}_\lambda$, all inputs $(s_0, .., s_n) \in \mathbb{S}^n$, all* $\mathit{pp}$ *in the support of* $\mathsf{Setup}(1^\lambda, T)$*, and all $z_i$ in the support of $\Pi.\mathsf{Gen}(s_i, \mathit{pp})$, assuming $\bar{z} \leftarrow \Pi.\mathsf{Aggregate}(c, z_0, .., z_n, \mathit{pp})$, the running time of the algorithm $\Pi.\mathsf{Solve}(1^\lambda, \bar{z})$ is bounded with $p_0(\lambda, T)$ for a fixed polynomial $p_0$ and for $\bar{s} \leftarrow \Pi.\mathsf{Solve}(\bar{z}, \mathit{pp})$:*

$$\bar{s} \;=\; c(s_0, .., s_n). \tag{2}$$

***Definition 20. HTLP Security*** *The HTLP protocol $\Pi =$* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Solve}, \mathsf{Aggregate})$ *is secure if for any* $\mathsf{PPT}$ *adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ 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:*

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

$$\leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

***Definition 21. HTLP Compactness*** *Given $\mathcal{C}_\lambda$ a class of circuits, the HTLP protocol $\Pi =$* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Aggregate}, \mathsf{Solve})$ *is compact if for all polynomials $T$ in $\lambda$, all circuits $C \in \mathbb{C}_\lambda$, all inputs $(s_0, .., s_n) \in \mathbb{S}^n$, all* $\mathit{pp}$ *in the support of $\Pi.\mathsf{Setup}(1^\lambda, T)$, and all $z_i$ in the support of $\Pi.\mathsf{Gen}(s_i, \mathit{pp})$, assuming $\bar{z} = \Pi.\mathsf{Aggregate}(C, z_0, .., z_n, \mathit{pp})$ we have:*

- *The running time of the algorithm $\Pi.\mathsf{Aggregate}(C, z_0, .., z_n, \mathit{pp})$ 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** ***Modified Linearly Homomorphic Time-lock Puzzle ($\Pi_{\mathsf{MTLP}}$)***

– $\mathsf{Setup}(1^\lambda, T)$*: Outputs system's public parameters* $\mathsf{pp} \leftarrow (N, g, T, h)$*. $N$ is a safe RSA number with $|N| = 2\lambda$, $g$ is a generator of the group $\mathbb{J}_N$, $h \leftarrow g^{2^T} \mod N$, and $T$ is the delay parameter (minimum number of modular squaring needed for solving the puzzle).*
– $\mathsf{Gen}(s, \mathsf{pp})$*: For the secret $s \in \mathbb{Z}_N$, samples the random numbers $r, k \leftarrow\$\ \mathbb{Z}_N$ 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 $w \leftarrow h^{kN}(1+N)^r \mod N^2$.*
– $\mathsf{Aggregate}((u_1, v_1, y_1, w_1), ..(u_n, v_n, y_n, w_n), \mathsf{pp})$*: Returns $(\bar{u}, \bar{v}, \bar{y}, \bar{w})$ for $\bar{u} \leftarrow \Pi\ u_i \mod N$, $\bar{v} \leftarrow \Pi\ v_i \mod N^2$, $\bar{y} \leftarrow \Pi\ y_i \mod N$, and $\bar{w} \leftarrow \Pi\ w_i \mod N^2$.*
– $\mathsf{Solve}((u, v, y, w), \mathsf{pp})$*: 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]. Now, let's prove the security and correctness of $\Pi_{\mathsf{MTLP}}$:*

***Theorem 4.*** *Let $N$ be a safe RSA 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 $\Pi_{\mathsf{MTLP}}$ is a time-lock puzzle entitled to the correctness, security, and compactness requirements.*

*Proof. We start by proving the correctness first.*
***Correctness:*** *Lemma 11 directly implies the correctness condition.*

***Lemma 11.*** *Given the puzzle $(u, v, y, w) \leftarrow \Pi_{\mathsf{MTLP}}.\mathsf{Gen}(s, \mathsf{pp})$ for $s \in \mathbb{N}$, the output of $\Pi_{\mathsf{MTLP}}.\mathsf{Solve}(u, v)$ will be $s \mod N$.*

*Proof.*

$$x = u^{2^T} \mod N = (g^{r+s})^{2^T} \mod N = g^{(r+s)2^T} \mod N.$$

*According to Lemma 1:*

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

*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 12.*** *For any* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *which the depth of* $\mathcal{A}_2$ *is bounded by* $T^{\epsilon}(\lambda)$ *from above where* $\lambda$ *is the security parameter,* $T$ *is a polynomial and* $0 < \epsilon < 1$*:*

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

$$\leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

*The proof of this lemma is in Appendix E. In other words, we show 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* $\overline{\mathcal{A}} = (\overline{\mathcal{A}}_1, \overline{\mathcal{A}}_2)$ *breaks the security of* $\Pi_{\mathsf{MTLP}}$*. Then we construct a new adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *to contradict the result of Lemma 12.* $\mathcal{A}_1$ *on receiving* $(1^\lambda, T, pp)$ *calls* $(m_0, m_1, \tau) \leftarrow \overline{\mathcal{A}}_1(1^\lambda, T, pp)$*. Without loss of generality, assume* $\overline{\mathcal{A}}_2$ *can answer the experiment correctly with non-negligible advantage* $\sigma$ *when it is called by the message* $m_0$*. Then,* $\mathcal{A}_1$ *returns* $(m_0, \tau)$*. Finally,* $\mathcal{A}_2$ *after receiving* $(z_b, \tau)$ *outputs whatever* $\overline{\mathcal{A}}_2(z_b, \tau)$ *returns. Observe* $(\mathcal{A}_1, \mathcal{A}_2)$ *has at least* $\sigma/2$ *which proves the security of* $\Pi_{\mathsf{MTLP}}$*.*

***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 12

*Proof. Via a set of hybrids, we have:*

***Hybrid*** $\mathcal{H}_0$*: We begin by setting* $\mathcal{H}_0$ *as the original scheme and sampling* $z_1$ *from* $\mathcal{H}_0$*. Hence,* $z_1 \leftarrow \Pi_{\mathsf{MTLP}}.\mathsf{Gen}(s, pp)$*. 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* $c \leftarrow\!\$\, \mathbb{J}_N$*. 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* $\mathcal{H}_0$*. Via a reduction against the strong sequential squaring assumption in* $\mathbb{J}_N$ *we want to show that given* $z_0 \leftarrow\!\$\, \mathcal{H}_0$ *and* $z_1 \leftarrow\!\$\, \mathcal{H}_1$ *there can not be any efficient distinguisher* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *who can succeed in distinguishing between* $z_0$ *and* $z_1$*. We now construct the distinguisher* $\mathcal{R} = (\mathcal{R}_1, \mathcal{R}_2)$ *who breaks the strong sequential squaring assumption as follows.* $\mathcal{R}_1$ *on receiving* $(N, T, g)$ *calculates*

$h \leftarrow g^{2^T} \mod N$, $\mathsf{pp} \leftarrow (N, g, T, h)$ *and calls* $(s, \tau') \leftarrow \mathcal{A}_1(1^\lambda, T, \mathsf{pp})$*, then, returns* $\tau \leftarrow (s, \mathsf{pp}, \tau')$*. Next, the challenger calls* $\mathcal{R}_2$ *with* $(x, y, (s, \mathsf{pp}, \tau'))$*. 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:*

- $x \leftarrow\$ \mathbb{J}_N$, $y \leftarrow x^{2^T} \mod N$*: 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* $\mathcal{A}_2(\hat{z}, \tau')$ *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* $r, s \leftarrow\$ \mathbb{Z}_N$*,* $c \leftarrow\$ \mathbb{J}_{N^2}$*,* $u \leftarrow g^{r+s} \mod N$*,* $v \leftarrow h^{(r+s)N}(1+N)^s \mod N^2$*,* $y \leftarrow\$ \mathbb{J}_N$*,* $w \leftarrow c(1+N)^r \mod N^2$*. 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* $\mathsf{pp} \leftarrow (N, g, T, h)$*. Then, runs* $(s, \tau') \leftarrow \mathcal{A}_1(1^\lambda, T, \mathsf{pp})$*. Finally, calls* $\mathcal{A}_2(z, \tau)$ *and returns the output of* $\mathcal{A}_2$ *where* $z = (\hat{u}, \hat{v}, \hat{y}, \hat{w}), \hat{r} \leftarrow\$ \mathbb{Z}_N, \hat{u} \leftarrow g^{\hat{r}+s} \mod N, \hat{v} \leftarrow h^{N(\hat{r}+s)}(1+N)^s \mod N^2, \hat{y} \leftarrow\$ \mathbb{J}_N, w \leftarrow y(1+N)^{\hat{r}} \mod N^2$*. 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* $\sigma$*,* $\mathcal{R}$ *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* $r, s \leftarrow\$ \mathbb{Z}_N, u \leftarrow g^{r+s} \mod N, v \leftarrow h^{(r+s)N}(1+N)^s \mod N^2, y \leftarrow\$ \mathbb{J}_N$*, and* $w \leftarrow\$ \mathbb{J}_{N^2}$*. Then, we first prove the below lemma:*

***Lemma 13.*** *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 10 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 13 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 $h^{(r+s)N}(1+N)^s \mod N^2$ it is sampled as $c^N(1+N)^s \mod N^2$ for a random $c \in \mathbb{J}_N$. 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 $\sigma$, 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 $\mathsf{pp} \leftarrow (N, g, T, h)$. Then, calls $(s, \tau') \leftarrow \mathcal{A}_1(1^\lambda, T, \mathsf{pp})$ and returns $\tau \leftarrow (s, \mathsf{pp}, \tau')$. 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 $\mathcal{A}_2(z, \tau')$. Now, we will show that with $5/8$ probability, the inputs to $\mathcal{A}_2$ are distributed according to either $\mathcal{H}_4$ or $\mathcal{H}_5$:*

- *$x \leftarrow\$ \mathbb{J}_N, y \leftarrow x^{2^T} \mod N$: Assume $x = g^\alpha \mod N$ for some $\alpha \in \mathbb{Z}_N$. Therefore, $\hat{u} = g^\alpha \mod N$ and $\hat{v} = g^{\alpha N 2^T}(1+N)^s \mod N^2$. Since the distribution of $\alpha$ 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* $1/2\times1/4+1/2 = 5/8$, $\mathcal{A}_2$ *is fed with proper inputs and can produce* $\sigma$ *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* $v \leftarrow c(1+N)^s \mod N^2$ *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 10 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 1

*Proof. Clearly, the scheme is public − coin since the verifier only broadcasts a randomly chosen* $e$*. Additionally, we already know that* $\Pi_{\mathsf{ERP}}.\mathsf{Verify}$ *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:*

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

***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* $g^{\alpha_1} = (uy)^{e_1} \times a$, $g^{\alpha_2} = (uy)^{e_2} \times a$.
*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,* $g^{\alpha_1}/g^{\alpha_2} = (uy)^{e_1} \times a/((uy)^{e_2} \times a) = g^{\alpha'} = (uy)^{e'}$*. Next, according to Lemma 7,* $e'$ *must divide* $\alpha'$*. 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 4,* $E$ *can extract the discrete log of* $u$ *as* $\alpha'/e'$.
- *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 3 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* $\rho$ *such that* $g^k(uy)^{-1} = g^\rho$*; therefore,* $\rho 2^d = k' \times |\mathbb{J}_N| = k' \times 2pq$ *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 $2\rho$ 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 5, we know that $g^k(uy)^{-1}$ can be non-trivial with only negligible probability. Consequently, it must be $\pm 1$, 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 $\alpha_1$ 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 9 and 8, 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 $\beta_1$. 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 [27]. The transcript of protocol between the prover and the verifier in* Verify *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 $\alpha' \leftarrow\$ \mathbb{Z}_{(N/2+N/n)\times 2^{2\lambda}}$, $\beta' \leftarrow\$ \mathbb{Z}_{N/n\times 2^{2\lambda}}$, $\tau' = g^{\beta'}/u^{e'}$, $a' \leftarrow g^{\alpha'}/u^{e'}$, $b \leftarrow h^{\alpha' N}(1+N)^{\beta'}/v^{e'}$, $e' \leftarrow\$ \mathbb{Z}_{2^\lambda}$ 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 $\alpha \leftarrow\$ [(r+s+k)e, (r+s+k)e + (N/2+N/n)\times 2^{2\lambda})$ 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 $\alpha$ to $\alpha \leftarrow\$ \mathbb{Z}_{(N/2+N/n)\times 2^{2\lambda}}$. Then, we prove that the statistical distance between $\mathcal{H}_2$ and $\mathcal{H}_3$ is negligible. The only difference between these two is $\alpha$. Therefore, with having $X$ as the random variable of $\alpha$ in $\mathcal{H}_2$ and $Y$ as the same variable in $\mathcal{H}_3$. We show that $\Delta(X, Y)$ is negligible in parameter $\lambda$, as follows.*

$$\begin{aligned}\Delta(X,Y) &= \frac{1}{2}\sum_{\alpha\ \in\ I_1}\frac{1}{(N/2+N/n)\times 2^{2\lambda}}\\ &+\frac{1}{2}\sum_{\alpha\ \in\ I_2}\frac{1}{(N/2+N/n)\times 2^{2\lambda}}-\frac{1}{(N/2+N/n)\times 2^{2\lambda}}\\ &+\frac{1}{2}\sum_{\alpha\ \in\ I_3}\frac{1}{(N/2+N/n)\times 2^{2\lambda}}\\ &=\frac{1}{(N/2+N/n)\times 2^{2\lambda}}\times(r+s+k)e\\ &\leq\frac{1}{(N/2+N/n)\times 2^{2\lambda}}\times(N/2+N/n)\times 2^{\lambda}=2^{-\lambda},\end{aligned}$$

*where* $I_1=[0,(r+s+k)e)$, $I_2=[(r+s+k)e,(N/2+N/n)\times 2^{2\lambda})$, *and* $I_3=[(N/2+N/n)\times 2^{2\lambda},\ (r+s+k)e+(N/2+N/n)\times 2^{2\lambda})$.
***Hybrid*** $\mathcal{H}_4$*: Similarly, sample* $\beta\leftarrow\$\,[(r+s)e,(r+s)e+N/n\times 2^{2\lambda})$ *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* $\alpha$ *to* $\beta\leftarrow\$\ \mathbb{Z}_{N/n\times 2^{2\lambda}}$*. 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* $\beta$*. Therefore, with having* $X$ *as the random variable of* $\beta$ *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* $\lambda$ *as it is shown below:*

$$\begin{aligned}\Delta(X,Y) &= \frac{1}{2}\sum_{\beta\in I_1}\frac{1}{N/n\times 2^{2\lambda}}\\ &+\frac{1}{2}\sum_{\alpha\in I_2}\frac{1}{N/n\times 2^{2\lambda}}-\frac{1}{N/n\times 2^{2\lambda}}\\ &+\frac{1}{2}\sum_{\beta\in I_3}\frac{1}{N/n\times 2^{2\lambda}}\\ &=\frac{1}{N/n\times 2^{2\lambda}}\times(r+s)e\leq\frac{1}{N/n\times 2^{2\lambda}}\times N/n\times 2^{\lambda}\\ &=2^{-\lambda},\end{aligned}\tag{3}$$

*where* $I_1=[0,\ (r+s)e)$, $I_2=[(r+s)e,\ N/n\times 2^{2\lambda})$, *and* $I_3=[N/n\times 2^{2\lambda},\ (r+s)e+N/n\times 2^{2\lambda})$*. Therefore, we proved that the real transcript* $T$ *is statistically indistinguishable from the transcript generated by the simulator,* $T'$*.*

## G Proof of Theorem 2

*Proof.* ***Correctness****: To the correctness of Theorem 2, assume there are* $n$ *number of shares with the extraction, and public keys* $\mathsf{ek}_i=(u_i,v_i,y_i,w_i)$ *and*

$\mathsf{pk}_i$, *where* $\forall_i\ \mathsf{Verify}(n, (\mathsf{ek}_i, \mathsf{pk}_i), \mathsf{pp}) = \mathit{true}$, *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* $u_i = g^{r_i+s_i}$, $y_i = g^{k_i} \mod N$, $v_i w_i = h^{(r_i+s_i+k_i)N}(1+N)^{r_i+s_i} \mod N^2$ *as the result of Theorem 1. Let* $\overline{\mathsf{ek}} = (\bar{u}, \bar{v}, \bar{y}, \bar{w}) = \mathsf{Aggregate}((\mathsf{ek}_1,\ \mathsf{pk}_1),\ ..\ (\mathsf{ek}_n, \mathsf{pk}_n))$. *Then, the output of* $\mathsf{Solve}(\overline{\mathsf{ek}}, \mathsf{pp})$ *will be* $\bar{s} \leftarrow \mathit{\Pi}.\mathsf{Solve}(z, \mathsf{pp})$ *for* $z = (\bar{u}\bar{y}, \bar{v}\bar{w}, \perp, \perp)$. *Additionally, according to Lemma 11,* $\bar{s} = \sum r_i + s_i \mod N$. *Also, we know that* $r_i, s_i \in \mathbb{Z}_{N/2n}$. *Therefore,* $\sum r_i + s_i < \mathbb{Z}_N$. *Thus,* $\mathsf{Solve}$ *gives us* $\sum r_i + s_i$. *Moreover,* $\overline{\mathsf{pk}} = g^{\sum s_i, r_i}$ *and consequently,* $\overline{\mathsf{pk}} = g^{\overline{\mathsf{sk}}}$. *Finally, given a message* $m \in \mathcal{M}$, $\mathsf{Decrypt}((c_1, c_2),\ \mathsf{sk}, \mathsf{pp})$ *for* $(c_1, c_2) = \mathsf{Encrypt}(m,\ \mathsf{pk},\ \mathsf{pp})$:

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

***Security****: We start by proving the security of* $\mathsf{MDE}$ *under non-adaptive adversary setting via the below lemma. Later, through reduction, we will prove that our scheme is also adaptively secure.*

***Lemma 14.*** $\mathsf{MDE}$ ***Non-Adaptive Security****. Let the* $\mathsf{MDE}$ *protocol* $\mathit{\Pi}_{\mathsf{MDE}} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Verify}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Extract})$. *Then, for every probabilistic polynomial time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ *which the depth of* $\mathcal{A}_2$ *and* $\mathcal{A}_3$ *are 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* $p_0$:

$$\Pr\left[\mathcal{A}_3(c, \tau_2) = b \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathit{\Pi}_{\mathsf{MDE}}.\mathsf{Setup}(n, 1^\lambda, \mathcal{T}), \\ ((\mathsf{ek}_1, \mathsf{pk}_1) \ldots (\mathsf{ek}_{n-1}, \mathsf{pk}_{n-1}), \\ \tau_1) \leftarrow \mathcal{A}_1(T, \mathsf{pp}, 1^\lambda), \\ (\mathsf{ek}_n, \mathsf{pk}_n) \leftarrow \mathit{\Pi}_{\mathsf{MDE}}.\mathsf{Gen}(n, \mathsf{pp}), \\ ((m_0, m_1), \tau_2) \leftarrow \\ \mathcal{A}_2((\mathsf{ek}_n, \mathsf{pk}_n), \tau_1), \\ (\overline{\mathsf{ek}}, \overline{\mathsf{pk}}) \leftarrow \\ \mathit{\Pi}_{\mathsf{MDE}}.\mathsf{Aggregate}((\mathsf{ek}_1, \mathsf{pk}_1) \\ \ldots (\mathsf{ek}_n, \mathsf{pk}_n), \mathsf{pp}), \\ b \leftarrow\!\!\$\ \{0,1\}, \\ c \leftarrow \mathit{\Pi}_{\mathsf{MDE}}.\mathsf{Encrypt}(m_b, \overline{\mathsf{pk}}) \end{array}\right]$$

$$\leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

*The proof of Lemma 14 is in Appendix H. Next, to show the* $\mathsf{MDE}$*'s security under adaptive adversary, we prove it via contradiction. Assume there exist the adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ *who can break the security in the game of Definition 8 with non-negligible advantage* $\sigma$. *Then, we construct the adversary* $\mathcal{R} = (\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3)$ *to break the security in Lemma 14. First,* $\mathcal{R}_1$ *on receiving* $(T, \mathsf{pp}, 1^\lambda)$ *calls* $\mathcal{A}_1(T, \mathsf{pp}, 1^\lambda)$ *to receive* $\tau_1$ *and then generates* $n-1$ *dummy shares via* $\forall\ (\mathsf{ek}_i, \mathsf{pk}_i) \leftarrow \mathit{\Pi}_{\mathsf{MDE}}.\mathsf{Gen}(n, \mathsf{pp})$. *Subsequently, returns* $((\mathsf{ek}_1, \mathsf{pk}_1), \ldots, (\mathsf{ek}_{n-1}, \mathsf{pk}_{n-1}), (\mathsf{sk}_d, \tau_1))$ *for* $\mathsf{sk}_d = \Sigma_{i \in \{1 \ldots n-1\}}(r_i + s_i)$. *Next,* $\mathcal{R}_2$ *on receiving* $((\mathsf{ek}_n,\ \mathsf{pk}_n), (\mathsf{sk}_d, \tau_1))$ *calls* $((m_0, m_1),\ (\hat{\mathsf{ek}}_1, \hat{\mathsf{pk}}_1)\ \ldots (\hat{\mathsf{ek}}_{n-1}, \hat{\mathsf{pk}}_{n-1}), \tau_2) \leftarrow \mathcal{A}_2((\mathsf{ek}_n,\ \mathsf{pk}_n),\ \tau_1)$. *Since*

$\forall_{i\in\{1\ldots n-1\}}\ \Pi_{\mathsf{MDE}}.\mathsf{Verify}(n,(\mathsf{ek}_i,\mathsf{pk}_i),\mathsf{pp}) = \mathtt{True}$, *therefore there is an extractor* $\mathcal{E}$ *which can extract the values of* $\hat{r}_i$ *and* $\hat{s}_i$ *that* $\mathcal{A}_2$ *used in generating* $(\hat{\mathsf{ek}},\hat{\mathsf{pk}})$. *Since* $\mathcal{R}_2$ *has access to the source code of* $\mathcal{A}_2$ *as well, it can run* $\mathcal{E}$ *to find* $\hat{\mathsf{sk}} = \Sigma_{i\in\{1\ldots n-1\}}\hat{r}_i + \hat{s}_i$ *with probability* $1-\delta$ *where* $\delta$ *is negligible. Finally,* $\mathcal{R}_2$ *returns* $((m_0,m_1),(\hat{\mathsf{sk}},\mathsf{sk}_d,\tau_2))$. *Next, when* $\mathcal{R}_3$ *is called with the inputs* $(c,(\hat{\mathsf{sk}},\mathsf{sk}_d,\tau_2))$, *assuming* $c=(c_0,c_1)$, $\mathcal{R}_3$ *constructs* $\bar{c}=(\bar{c}_0,\bar{c}_1)$ *as* $\bar{c}_0 = c_0$ *and* $\bar{c}_1 = c_1 \times c_0^{\hat{\mathsf{sk}}}/c_0^{\mathsf{sk}_d} \mod N$. *Finally,* $\mathcal{R}_3$ *returns the output of* $\mathcal{A}_3(\bar{c},\tau_2)$. *The advantage of* $\mathcal{R}$ *will be the non-negligible value* $(1-\delta)\sigma$ *which finishes the proof.*

## H Proof of Lemma 14

*Proof. We start by stating another variant of the security for Algorithm 1 whose proof is in Appendix I:*

***Lemma 15.*** *Given a* MDE *protocol* $\Pi_{\mathsf{MDE}}$, *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* $p_0$*:*

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

$$\leq \frac{1}{2}+\mathsf{negl}(\lambda).$$

*Lemma 15 shows that the tuple* $(\mathsf{ek},\mathsf{pk})$ *is indistinguishable from a uniform tuple for the bounded adversary* $\mathcal{A}_3$. *Following the result of Lemma 15, we can replace* $(\mathsf{ek}_n,\mathsf{pk}_n)$ *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* $(\overline{\mathsf{ek}},\overline{\mathsf{pk}})$ *also becomes indistinguishable from a random tuple for any* $\mathcal{A}_3$. *It is obvious to show that the output of* $\Pi_{\mathsf{MDE}}.\mathsf{Encrypt}(m,\mathsf{pk},\mathsf{pp})$ *will also remain random which completes the proof.*

## I Proof of Lemma 15

*Proof. Note that the main difference between Lemmas 12 and 15 is the sampling range of* $s$ *and* $r$ *which are now instead from* $\mathbb{Z}_{N/2n}$. *Let* $\sigma$ *be the advantage of* $\mathcal{A}$ *in the experiment of Lemma 15. 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 12.* $\overline{\mathcal{A}}_1$ *upon receiving the* $(1^\lambda,T,\mathsf{pp})$ *returns a randomly generated* $s\leftarrow\$\ \mathbb{Z}_{N/2n}$ *and an empty advice* $\tau\leftarrow\perp$. *Then,* $\overline{\mathcal{A}}_2$ *on receiving* $(z_b,\tau)$ *returns the output of* $\mathcal{A}(1^\lambda,z_b,\mathsf{pp})$. *It is easy to verify that the advantage of* $\overline{\mathcal{A}}$ *will be* $\bar{\sigma}=\sigma(1/2\times 1/2n+1/2)$. *Additionally, the result of Lemma 12 shows that* $\bar{\sigma}=\mathsf{negl}(\lambda)$. *Therefore,* $\sigma(1/2\times 1/2n+1/2)=\mathsf{negl}(\lambda)=\sigma(1/4p(\lambda)+1/2)$*; thus,* $\sigma=\mathsf{negl}(\lambda)\times 4p(\lambda)/(1+2p(\lambda))=\mathsf{negl}(\lambda)$. *Consequently, there cannot be any adversary that succeeds in the experiment of Lemma 15.*