Efficient Algorithms for Broadcast and Consensus Based on Proofs of Work

Lisa Eckey, Sebastian Faust, and Julian Loss

Ruhr University Bochum, Germany
{lisa.eckey, sebastian.faust, julian.loss}@ruhr-uni-bochum.de

Abstract. Inspired by the astonishing success of cryptocurrencies, most notably the Bitcoin system, several recent works have focused on the design of robust blockchain-style protocols that work in a peer-to-peer setting such as the Internet. In contrast to the setting traditionally considered in multiparty computation (MPC), in these systems, honesty is measured by computing power instead of requiring that only a certain fraction of parties is controlled by the adversary. This provides a potential countermeasure against the so-called Sybil attack, where an adversary creates fake identities, thereby easily taking over the majority of parties in the system. In this work we design protocols for Broadcast and Byzantine agreement that are secure under the assumption that the majority of computing power is controlled by the honest parties and for the first time have expected constant round complexity. This is in contrast to earlier works (Crypto'15, ePrint'14) which have round complexities that scale linearly with the number n of parties; an undesirable feature in a P2P environment with potentially thousands of users. In addition, our main protocol which runs in quasi-constant rounds, introduces novel ideas that significantly decrease communication complexity. Concretely, this is achieved by using an appropriate time-locked encryption scheme and by structuring the parties into a network of so-called cliques.

1 Introduction

In multiparty computation (MPC), a set of n fixed players aims to compute a function f by jointly running a distributed protocol such that nothing but the output of the function is revealed. A fundamental building-block for most MPC protocols is a so-called *broadcast channel*. Messages sent via the broadcast channel are guaranteed to be delivered consistently to all parties. More formally, a broadcast channel shall satisfy the following two requirements. First, the *consistency property*, which guarantees that all honest parties receive the same consistent message m. Second, the *validity property*, which ensures that if the sender is honest and sends a message m^*, then all honest parties receive m^*. The problem of constructing broadcast channels was first formulated as the "Byzantine Generals Problem" by Lamport in 1982 [21], and since then has been intensively studied in cryptographic research, and more generally, in the area of distributed computing [10, 32, 7, 34, 14, 9, 18].

A common assumption enabling particularly efficient protocols is the use of a trusted setup via a public key infrastructure (PKI). A PKI guarantees that the participants of the protocol are aware of each others public key, and has, e.g., been used in [18, 10] to construct efficient broadcast protocols. Unfortunately, constructing a PKI can prove particularly difficult to achieve in a fully decentralized peer to peer (P2P) setting like the Internet, where parties a-priori do not have any established relationship, and in fact may not even be aware of each other. An additional difficulty in such a setting arises from the fact that secure broadcast protocols typically require that the majority of the parties behaves honestly. This requirement is, however, hard to guarantee in a P2P setting, because potentially malicious parties can enter the system at any time. In fact, it is hard to even define the concept of a "party" as, for instance, a user of the protocol may create many "virtual" fake parties on a single machine, thereby easily taking over the majority of the parties in the system. Such an attack is called the *Sybil attack* [11], which is an important threat for designing secure distributed systems.

An approach to overcome the above challenges has been proposed in the seminal work of Satoshi Nakamoto with the Bitcoin cryptocurrency [25]. Bitcoin does not rely on a trusted PKI setup, and prevents the Sybil attack by operating under the assumption that the majority of computing power is controlled by the honest parties. At a technical level, Bitcoin (and other cryptocurrencies) guarantee the assumption that the majority of computing power is controlled by the honest parties via the so-called *Proof of Work* (PoW) – a concept initially proposed by Dwork and Naor at Crypto'92 [12]. Informally, a PoW is a resource intensive proof that guarantees that the prover invests computational effort in solving a puzzle, while verification that the puzzle was solved correctly can be done very efficiently. Using PoWs, we can now restrict the amount of adversarially controlled parties in a system, by requiring that participation in a protocol is only allowed when the party successfully completes a PoW.

Inspired by the Bitcoin system, many recent academic works (see, e.g., [15, 26–28, 16] and many more) give PoW-based protocols solving the problem of state-machine replication [30, 20], in which an often (unknown) set of parties agrees on an ordered log of transactions – the so-called blockchain. At an abstract level, the blockchain allows users that do not trust each other to reliably maintain an append-only register that stores the state of a decentralized system. As the name suggests, the blockchain is built from a chain of blocks, where a block B_i is connected to the previous block B_{i-1} by containing the hash of B_{i-1}. A drawback of most of these works is that implicitly the security of these protocols rely on the trusted creation of a so-called *genesis block* B_0. Notice that the creation of the genesis block has to be trusted because we require B_0 to be unpredictable until the point of its publication.[1] An heuristic approach

[1] For readers familiar with the Bitcoin network, if the genesis block is known in advance of the official launch of the cryptocurrency, then an adversary could run a pre-mining attack. In such an attack the adversary mines a longest chain starting with the genesis

to guarantee this unpredictability is used by Bitcoin via letting B_0 contain an article of the New York Times published on the date the currency was launched.

Motivated by the problem of securely creating the genesis block, two recent works [2, 19] present protocols which can be used to replace the trusted setup with a distributed protocol, whose security solely relies on the assumption that some fraction of computing power is controlled by the honest parties. At a high-level, in these works the authors construct protocols for "one-shot" broadcast among an initially unknown number of parties[2] in a setting without any pre-established trust infrastructure. Concretely, it is shown that even when only a non-negligible fraction of computing power is controlled by the adversary, then the protocols of [2, 19] can realize a secure broadcast channel among the participating users. An important drawback of the above works, however, is that their round complexity increases linearly with the number of parties. This is in particular problematic in large-scale settings with possibly thousands of users.

In this work we study the question if the round complexity of broadcast – and of the related problem of byzantine agreement – can be reduced[3]. Our main result shows that this is indeed the case, by giving the *first protocol* that runs in a constant number of rounds when the majority of computing power is controlled by the honest parties. We emphasize that while the protocols of [2, 19] allow to realize broadcast in a setting that is more adversarial than the setting considered in our work (they even work when the majority of computing power is corrupted), the related problem of byzantine agreement is in fact meaningful *only* in a setting with an honest majority. Hence, in particular for one of the main applications of creating the genesis block (which indeed requires byzantine agreement rather than broadcast), our protocol improves previous works, while still working in the same adversarial model. Moreover, even for the broadcast setting, we believe that in large-scale settings with thousands of users improving on efficiency may be more important for many applications than considering a stronger adversarial model.[4] We provide further details and comparison with other related work in the next sections.

1.1 Our contributions

Expected constant round broadcast in the PoW model. Building on the earlier works of [18, 19, 2], we propose a novel protocol which realizes secure broadcast and runs in an expected constant number of rounds. More concretely, our protocol works in the model introduced in [2] and borrows ideas from the

block long before the system starts. This enables him – despite having only limited computational resources – to run, e.g., a double spending attack.

[2] This is in contrast to the traditional MPC setting, where the exact number of parties running the protocol is known to all players.

[3] In byzantine agreement, parties *agree* on a value x. If all the honest parties give the same input to the protocol, then agreement should be reached on this value.

[4] Notice that our protocols require that the adversary can take over the majority of computing power – which for large-scale settings will be hard to achieve anyways.

work of Katz and Koo [18]. The latter builds expected constant round broadcast protocols in the trusted PKI model when the majority of keys is controlled by the honest users. Following the approaches of [19, 2], we show how to adapt this protocol into an expected constant round protocol, for a setting without trusted PKI but when we can rely on the assumption that the majority of parties is controlled by the honest users. While we believe that this is an interesting feasibility result, we notice that our protocols have the drawback that they increase the communication complexity when compared to the works of [19, 2]. Improving on the communication complexity forms the second main contribution of our work outlined next.

Broadcast with significantly reduced message complexity. Our second contribution are secure broadcast protocols that provably run in expected logarithmic and quasi-constant number of rounds, but achieve significantly better communication complexity. To this end, we again rely on the protocols of Katz and Koo [18], but replace the most communication intensive parts with a novel protocol that enables to share consistently random coins assuming that the majority of computing power is honest. While the original protocol uses verifiable secret sharing (VSS) to achieve this goal, we present a protocol that makes use of *time-locked-encryption* [29] in a novel way. Time-locked encryption allows to "send a message into the future" by guaranteeing that the time-locked ciphertext c of message m can only be decrypted once a certain time, δ, has passed. To illustrate how we can use time-locked encryption to consistently share random coins let us consider the following strawman solution. In the first phase, each party P_i commits to a random secret coin x_i via a commitment c_i. In the second phase, these commitments are publicly opened by the parties.

Unfortunately, this simple approach runs into the following problem. Since the adversary is rushing, he can simply wait for the honest parties to open all their commitments, and then once all honest coins are known, choose whether or not it will provide the opening to its own commitments. Clearly, such behavior will bias any coin which is derived in some public manner from the available coin shares (for example, by adding them up or running them through a random oracle). However, by using time-locked encrypted ciphertexts rather than commitments, we can overcome this problem in the following way: If an adversary refuses to open its commitment, the honest parties can open them in at most time δ to complete the protocol in a deterministic way without the help of the adversary, and thereby preventing any adversarial bias. Notice that of course the above argument requires that the time-locked ciphertexts hide the coin shares of the honest parties until the protocol is completed, but this is naturally guaranteed by the security properties of the time-locked encryption scheme. Concretely, similar to earlier works that use time-locked encryption in protocol design [23] we can rely on non-parallelizable time-locked encryption from [4, 29]. We believe that our coin-sharing protocol and the use of time-locked encryption is of independent interest and may find applications in other settings to improve the efficiency of protocols that otherwise rely on heavy tools such as VSS protocols.

Efficient broadcast for large scale settings. Finally, we ask the question of how to design efficient broadcast protocols for a large-scale setting with (many) thousands of parties. The main idea is to partition the parties into disjoint subsets– so-called cliques. Each clique then emulates a party in our original protocol. Choosing the number of cliques as $\sqrt{n}$, we can decrease the communication complexity com to about $\sqrt{\mathsf{com}}$. A similar approach was recently also taken in [24]. However, in our protocol, we let the parties *choose* their cliques themselves, whereas in [24] parties are delegated to cliques randomly. To generate the randomness needed for this assignment of parties to cliques, the protocol presented in [24] requires an expensive first protocol round which requires that $\frac{5}{6}$ of the computing power is honest. After this first round, the protocol in [24] can tolerate a corruption rate of $\frac{1}{3}$ for running future broadcasts. Since in our protocol we let parties choose their cliques themselves, we do not need this expensive first round of the protocol, and thus can achieve the optimal corruption rate of $\frac{1}{3}$ for the entire protocol. We notice that [24] achieves better communication complexity than our protocol once the first round of generating the initial randomness has passed. The details of our protocol for large-scale broadcast and its analysis are moved to Appendix F.

1.2 Related Work

Broadcast and state-machine replication. Secure broadcast channels are a fundamental building block of cryptographic protocols and distributed systems in general, and hence have been studied extensively by the research community. Lately, motivated by the consensus mechanisms used in cryptocurrencies, there has been growing interest in the design of secure protocols for broadcast and byzantine agreement in the PoW model. Most important for our work are the already mentioned results of Katz et al. [19] and Andrychowicz and Dziembowski [2], which we compare in detail with our constructions in Section 6. Besides these two works, there has lately been a flurry of works that study the consensus mechanisms of Blockchain-based cryptocurrencies (see [15, 26–28] and more). While a detailed comparison of the models and assumptions considered in these works is beyond the scope of this paper, we emphasize that the main difference between protocols for Blockchain-based consensus mechanisms and our work is the fact that the former requires a trusted generation of the genesis block. As already outlined in the introduction our approach does not make such assumptions, and instead relies solely on the requirement that the majority of computing power is controlled by the honest users. On the other hand, once set-up, Blockchain-based protocols may achieve better efficiency (in particular, in terms of message complexity) than the constructions that we present in this work.

Another difference of our protocols when compared to Blockchain-based protocols is that the latter considers the problem of so-called *state-machine replication* in which parties agree on an ordered log of transactions. Given a protocol for state-machine replication, it is easy to achieve byzantine agreement or broadcast. We notice that in the synchronous model, one may use the approach shown in [18] to run multiple instances of a consensus protocol back-to-back to agree on an ordered log. We can also use the approach of [18] to achieve state-machine replication, and thereby even achieving consensus in the so called *permissionless* setting, where the set of parties can grow (or decrease) during the execution of the protocol. The only assumption our protocol relies on is that during the multiple execution runs the majority of computing power is always controlled by the honest parties.

We remark that such a bootstrapping approach may lead to improvements over blockchain based consensus protocols in particular regarding the storage space needed to perform the protocol. If in every execution of our protocol, all parties in the system (also newly joined ones) agree on some set of transactions, there is no need to keep the entire history of the state. This is in contrast to many existing blockchain protocols which require a continuous chain that starts with the honestly generated genesis block. Of course, a downside of such an approach is that newly joined users can not check the validity of past transactions within the system, but instead rely on that the majority of computing power was always honest during the lifetime of the protocol. We leave a further comparison and improvements in this direction for future work.

Non-Parallelizable Functions. The versatile concept of non-parallelizable functions and puzzles has been the focus of some recent works [3, 23, 6, 8]. One of the main applications of such functions is the creation of unpredictable random beacons via so-called *delay functions* [12, 17, 1, 22]. The authors of [6, 8] show how such functions can be used to convert the headers of the Ethereum blockchain into an unpredictable random beacon by suitably applying them to the headers of the blocks and extracting the entropy from the resulting values. While these works show the practicality of non-parallelizability as a concept for randomness generation, they again rely on the existence of a blockchain structure for the creation of random beacons and thus require a trusted setup in form of a genesis block.

2 Model and Definitions

In this section we introduce some notation, cryptographic building blocks and the model for communication and computation that we use for our constructions.

2.1 Communication Model

We consider a setting in which some set of parties $P_1, \ldots, P_n$ engage in a distributed protocol Π. We follow the modeling approach of [2], and do not require any trusted setup and assume unlike in the traditional multiparty computation setting that neither the number of honest parties nor the exact number of parties engaging in Π is known. In particular, we emphasize that no public key infrastructure (PKI) needs to be shared among the parties, i.e., the parties do not initially know each others public keys. In contrast to standard MPC based solutions, we assume that the honest parties control a majority of the computing power in the system (instead of controlling an explicit number of parties), and that all parties in the system can send at most θ messages in a round of length Δ, where Δ denotes some measure of real time (say, one minute). As discussed in [2] some bound on the total number of messages that parties can send within one round is necessary. This is the case, because already accepting/processing messages takes time, and otherwise the adversary can launch a denial-of-service attack simple by sending too many messages.

We make the standard assumption of having bilateral channels between all parties. In addition, as in [2], we assume a public broadcast channel $\mathcal{C}$ that can be accessed by all parties. This channel $\mathcal{C}$ is solely needed to allow honest parties to send messages to an unknown number of parties. Note that in this case, the parties cannot use the bilateral channels, because the corresponding parties may not even be known yet.

We emphasize that we require only mild reliability guarantees from the channel $\mathcal{C}$. Concretely, honest parties can use $\mathcal{C}$ to broadcast a message m to all parties, and the adversary can delay the delivery of such messages for at most Δ. In particular, we require that the adversary cannot drop messages from any of these channels. On the other hand, similarly to bilateral channels the adversary is allowed to read all messages that are sent via $\mathcal{C}$, and it can use $\mathcal{C}$ to send messages only to a certain subset of honest parties. The latter property in particular allows the adversary to introduce inconsistencies between honest parties, which must be avoided by a reliable broadcast channel as we want to construct in this paper. A possible way to implement the channel $\mathcal{C}$ in practice are network gossiping protocols – similar as the one used by Bitcoin [2].

An important quality measurement of our protocols is the communication complexity of individual parties. To give a fair estimate of this parameter, we use a worst case analysis in the pure bilateral channels model assuming that each message sent via $\mathcal{C}$ corresponds to sending a single message via the corresponding bilateral channel. We will provide efficiency estimates of our protocols using the following two metrics commonly used in the MPC literature. The *message complexity* as the number of messages that every party must send in order to complete the protocol, and the *bit complexity* as the total number of bits that every party must send in order to complete the protocol.

2.2 Model of Computation

As in standard complexity-based cryptography, we assume that parties are PPT Turing machines. Since the security of our protocols relies on the assumption that the majority of computing power is controlled by honest parties, we need however a more fine-grained specification of computing power. To this end, we follow earlier works and model computing power by the number of queries that parties can make to certain types of oracles. In this work, we will be interested in two types of oracles to measure computing power. First, queries to a random oracle that models hash function evaluations and second, queries to an oracle O_R, which allows to perform ring operations and equality checks over the ring of integers $\mathbb{Z}_N$, for $N \in \mathbb{N}$. O_R answers queries of tuples of the form $\big((a_1^1, a_1^2, \mathsf{op}_1, N_1), \ldots, (a_l^1, a_l^2, \mathsf{op}_l, N_l)\big)$ where $\forall i : a_i^1, a_i^2 \in \mathbb{Z}_{N_i}, \mathsf{op}_i \in \{=, \times, +, -, \div\}$ and l is polynomially bounded (in some security parameter κ). Here, the operations $\{\times, +, -, \div\}$ denote the standard modular ring operations and "=" denote the modular equality relation. O_R's replies are of the form $b_1, ..., b_l$, where $b_i = a_i^1 \mathsf{op}_i a_i^2 \in \mathbb{Z}_{N_i}$ if $\mathsf{op}_i \in \{\times, +, -, \div\}$ and $b_i \in \{0, 1\}$, if $\mathsf{op}_i \in \{=\}$. We emphasize that we use the oracle O_R mainly as a way to formalize the amount of computational power that parties invest. One way to view access to O_R is that it provides the fastest way to compute operations over the ring $\mathbb{Z}_N$. This allows us to consider w.l.o.g. only adversaries that perform all such operations by calling O_R on appropriate inputs, and hence restricting the number of queries that they can make to O_R is a way to limit the adversary's computational efforts.

An alternative way to restrict the computational effort of the adversary was considered, e.g., in [3, 23]. Roughly speaking, these works restrict the computational power by representing adversaries as an arithmetic circuit. The depth of an arithmetic circuit corresponds in our modeling to the number of sequential queries to O_R, while the width of the circuits represents the number ℓ of operations that can be executed in parallel. We opted for modeling computational effort by queries to an oracle instead of representing adversaries as arithmetic circuits, because our modeling falls more in line with the hashrate limitation modeling used by earlier works [2, 19], on which our work is based.

We assume that every honest party in the system may query either oracle π times during real time δ.[5] Here, we follow the work of [2] and assume for simplicity that each honest party in the system can ask an equal amount of queries to the oracles during real time δ. We use $\pi_{\max}$ to denote the total amount of hash queries that can be asked to the random oracle during time δ by all participants in the system and $\pi_{\mathcal{A}}$ to refer to the total number of hash queries that can be made by the adversary. As above, we assume again for simplicity that the same parameters $\pi_{\max}$ and $\pi_{\mathcal{A}}$ also apply to queries to O_R. Our honest majority assumption immediately yields $\frac{\pi_{\max}}{\pi_{\mathcal{A}}} < \frac{1}{2}$. We further set $n := \lceil \frac{\pi_{\max}}{\pi} \rceil$ as an *upper bound* on the total number of parties in the system, and assume that $\pi_{\max}$ and therefore n is known to all honest parties.

2.3 Basic definitions

We denote protocols with the uppercase letter Π. We denote vectors with bold-faced letters $\boldsymbol{M}$ and write $\boldsymbol{M}[i]$ to denote the ith element of $\boldsymbol{M}$. We write $x \overset{\$}{\leftarrow} S$ to denote that a variable x is uniformly sampled from the finite set S. We write $[n]$ to refer to the set of numbers from 0 to $n-1$. To denote that the view of a party on some variable x is local, i.e., it could be different for every party, we write $\widehat{x}$ instead of x to keep recursive subindexing as low as possible.

[5] In reality, ring operations typically take more real time than hash function evaluations. We make the simplifying assumption that both oracles can only be queried π times during time δ which makes presentation simpler. Adjusting our model to capture the difference in evaluation time is straight-forward.

Proofs of Work. A proof of work is a proof that some amount of computational effort has been invested by a prover $\mathcal{P}$. Generating such a proof is resource intensive for $\mathcal{P}$, but can be verified efficiently by a verifier $\mathcal{V}$. Intuitively, such proofs can be useful to keep an adversary from flooding the honest parties with a large amount of 'spam messages'. This could be achieved by accepting messages only if they are accompanied with a valid proof of work. Since the adversary's computing power is limited, this mechanism essentially rules out such flooding attempts. Indeed, PoWs where first introduced in [12] to address the problem of handling spam messages. We will use PoWs to measure computational power of the parties and use it as a measure to prevent the so-called *sybill attack*. Sybill attacks allow the adversary to flood the protocol with many fake "dummy identities" and are problematic for large-scale multiparty protocols that rely on the assumption that some fraction of the protocol participants are honest. We use the modelling of PoWs as introduced by Andrychowicz and Dziembowski [2].

Definition 1 (Proof-of-Work Scheme). *Let $\mathcal{H}$ be a random oracle and let κ be the security parameter. A* Proof-of-Work Scheme *is a tuple of PPT algorithms* $(\mathsf{Solve}^{\mathcal{H}}, \mathsf{VerifyPow}^{\mathcal{H}})$ *with the following specification. The algorithm* $\mathsf{Solve}^{\mathcal{H}}$ *takes as input a challenge* $p \overset{\$}{\leftarrow} \{0,1\}^{\kappa}$ *and outputs a solution* $\psi \in \{0,1\}^*$. *The algorithm* $\mathsf{VerifyPow}^{\mathcal{H}}$ *takes as input challenge p and solution ψ and outputs either* 0 *or* 1. *For correctness we require that for all* $p \in \{0,1\}^{\kappa}$ *we have that* $\mathsf{VerifyPow}^{\mathcal{H}}(p, \mathsf{Solve}(p)) = 1$. *Moreover, we say that* $(\mathsf{Solve}^{\mathcal{H}}, \mathsf{VerifyPow}^{\mathcal{H}})$ *is π-secure if for all PPT algorithms* A*, the success probability in the following experiment is negligible in κ:*

- A *is allowed to make polynomial number of queries to the random oracle $\mathcal{H}$.*
- A *gets* $p \overset{\$}{\leftarrow} \{0,1\}^{\kappa}$ *and can make at most π further queries to the random oracle $\mathcal{H}$. Finally,* A *outputs ψ.* A *wins if* $\mathsf{VerifyPow}^{\mathcal{H}}(p, \psi) = 1$.

We will use the π-secure POW scheme from [2] and set δ, κ such that $\mathsf{Solve}^{\mathcal{H}}$ runs in time $\delta := 4\kappa^2 \Delta$ and $\mathsf{VerifyPow}^{\mathcal{H}}$ runs in time $\kappa \log(\pi \cdot \delta)/\pi$. To simplify notation, we will omit the parameterization of Solve and $\mathsf{VerifyPow}$.

Signature Schemes. Our constructions will use a digital signature scheme satisfying the standard security notion of *unforgeability under chosen message attacks* (UF-CMA-security). For ease of notation, in our protocols we will assume that each signature is accompanied with its corresponding public key.

Broadcast and Consensus. In this work, we construct protocols that allow to implement secure broadcast or consensus protocols. A broadcast channel, as suggested by its name, allows a sender (usually called *dealer*) P_D to send a message to all parties in the system with the guarantee, that every honest party receives the same message which is the message that P_D intended to send. Formally, we have the following definition:

Definition 2. *A protocol run between n parties where a distinguished party P_D holds initial input m is a* Broadcast *protocol if the following properties hold for all honest parties $P_i, P_j, i, j \in [n]$*

– Consistency: *At the end of the protocol, P_i, P_j outputs $m_i = m_j$ are identical.*
– Validity: *If P_D is honest, then P_i outputs $m_i = m$.*

A consensus or *byzantine agreement protocol* allows n parties with certain inputs $m_1, \dots, m_n$ to jointly agree on a value.

Definition 3. *A protocol for n parties $P_1, \dots, P_n$ with inputs $m_1, \dots, m_n$, is a* Byzantine Agreement *protocol if the following properties hold for all honest parties $P_i, P_j, i, j \in [n]$*

– Agreement: *At the end of the protocol, P_i, P_j outputs $m_i = m_j$ are identical.*
– Validity: *If the inputs of all honest parties are identical $m_i = m_j = m$ then all honest parties output m.*

It is easy to see that byzantine agreement implies broadcast, since a simple broadcast protocol can be constructed by adding just one more step. The idea is to first let the dealer send his value m to all parties and then let the parties run a byzantine agreement protocol on the value they received from the dealer. If the dealer was honest, the properties of the byzantine agreement ensure validity, i.e., that all parties output m. The agreement requirement of broadcast follows directly from the agreement property of the byzantine agreement.

2.4 Time-Locked Encryption

In this section we introduce the notion of time-locked encryption which serves as a main building block in our schemes. Informally, a time-locked encrypted ciphertext that is encrypted at time t cannot be decrypted until time, say $t + \delta$.

Requirements for Our Scheme. We introduce a variant of time-locked encryption which allows to simultaneously encrypt multiple messages in such a way that they cannot be decrypted before some time bound δ has passed. To this end, we present a time-locked encryption scheme Π^{TL}, which allows the parallel encryption of k messages $(m_1, \dots, m_k)$ to ciphertexts $(c_1, \dots, c_k)$. Π^{TL} provides a (deterministic) procedure $\mathsf{Open}_{\mathsf{TL}}$, which on input of one of these ciphertexts $c_i, i \in [k]$ returns a token ψ after δ time steps. One may think of this token as a trapdoor that enables efficient decryption without knowledge of any private input. We require that each ciphertext can only be decrypted with one unique token ψ. Note however that once this token is known, it can be used to decrypt all ciphertexts of the tuple $(c_1, \dots, c_k)$.

Definition 4. *A* Time-Locked Encryption Scheme *in the O_R model is a tuple of PPT algorithms $(\mathsf{Enc}_{\mathsf{TL}}^{\mathsf{O}_R}, \mathsf{Open}_{\mathsf{TL}}^{\mathsf{O}_R}, \mathsf{Dec}_{\mathsf{TL}}^{\mathsf{O}_R})$. The randomized encryption algorithm $\mathsf{Enc}_{\mathsf{TL}}^{\mathsf{O}_R}$ takes as input k messages $m_1, \dots, m_k$ in the message space $\mathcal{M}$ and outputs k ciphertexts $c_1, \dots, c_k$ in the ciphertext space $\mathcal{C}$. The deterministic open algorithm $\mathsf{Open}_{\mathsf{TL}}^{\mathsf{O}_R}$ takes as input a single ciphertext c' and outputs a token ψ'. The deterministic decryption algorithm $\mathsf{Dec}_{\mathsf{TL}}^{\mathsf{O}_R}$ on input (c', ψ') outputs $m' \in \mathcal{M} \cup \{\bot\}$. We require the following properties for a time-lock encryption scheme Π^{TL} secure in the O_R model:*

```
Game $\mathbf{SemSec}^{\mathsf{A}}(1^\kappa)$ :
00 $\boldsymbol{m} \leftarrow \mathsf{A}^{\mathsf{O}_R,\tilde{H}}(1^\kappa)$
01 $(\boldsymbol{c}_0) \leftarrow \mathsf{Enc}_{\mathsf{TL}}^{\mathsf{O}_R,\tilde{H}}(\mathbf{0})$
02 $(\boldsymbol{c}_1) \leftarrow \mathsf{Enc}_{\mathsf{TL}}^{\mathsf{O}_R,\tilde{H}}(\boldsymbol{m})$
03 $b \leftarrow \{0,1\}$
04 $b' \leftarrow \mathsf{A}^{\mathsf{O}_R}(c_b)$
05 Return $b' = b$
```

Fig. 1. Semantic security game for time-locked encryption in the O_R model. $\mathbf{0}$ denotes a vector containing only zero strings, which has the same length as $\boldsymbol{m}$.

Correctness *For the encryption* $(c_1, \ldots, c_k) = \mathsf{Enc}_{\mathsf{TL}}^{\mathsf{O}_R}(m_1, \ldots, m_k)$ *of message vector* $(m_1, \ldots, m_k)$ *it holds* $\forall i, j \in [k]$ *that* $\mathsf{Dec}_{\mathsf{TL}}^{\mathsf{O}_R}(c_i, \mathsf{Open}_{\mathsf{TL}}^{\mathsf{O}_R}(c_j)) = m_i$.

Uniqueness *For all c, we have that* $\mathsf{Dec}_{\mathsf{TL}}^{\mathsf{O}_R}(c, \psi') = \perp$ *unless* $\psi' = \mathsf{Open}_{\mathsf{TL}}^{\mathsf{O}_R}(c)$.

Secrecy *We say that* Π^{TL} *satisfies* semantic security *if for all PPT adversaries* A *playing Game* **SemSec** *depicted in Figure 1 and making at most* π *calls to* O_R*, we have that* $\Pr[\mathbf{SemSec}^{\mathsf{A}}(1^\kappa) = 1] = \mathsf{negl}(\kappa)$.

BUILDING TIME-LOCKED ENCRYPTION. We now show how to instantiate a time-lock encryption scheme in the O_R model under Conjecture 1 which was originally introduced in [29] and is based on the non-parallelizability of repeated squaring. The concept of non-parallelizable functions have recently drawn considerable interest [3, 23, 6, 8]. Also, the recent work of Lin et al. [23] considers time-locked puzzles based on a hardness conjecture similar to Conjecture 1.

Conjecture 1. [29] Let $N = rq$, where $r, q \leftarrow \mathsf{PrimeGen}(1^\kappa)$ are primes, let $a \xleftarrow{\$} \mathbb{Z}_N$, and let A be an algorithm that calls O_R less than π times. Then $\Pr[\psi \equiv_N a^{2^\pi} \mid \psi \leftarrow \mathsf{A}^{\mathsf{O}_R}(a, \pi, N)] = \mathsf{negl}(\kappa)$.

In the following, let $\tilde{H} : \{0,1\}^* \to \{0,1\}^\kappa$ be a random oracle. We prove in Lemma 1 that the resulting scheme satisfies **SemSec** security. A detailed proof of Lemma 1 can be found in Appendix A.

Time-locked encryption scheme Π^{TL}

```
$\mathsf{Enc}_{\mathsf{TL}}(\mathbf{m_1, ..., m_n})$

$q, r \xleftarrow{\$} \mathsf{PrimeGen}$
$N = rq, \varphi(N) = (r-1)(q-1)$
$a \xleftarrow{\$} \mathbb{Z}_N$
$s \equiv_{\varphi(n)} 2^\pi, \psi \equiv_N a^s$
$K = \tilde{H}(\psi)$
$R_1, ..., R_n \xleftarrow{\$} \in \{0,1\}^\kappa$
$\forall i \in [n] : c_i := R_i || \left(\tilde{H}(K||R_i) \oplus (m_i, r, q)\right)$
Return $((c_1, a, \pi, N)..., (c_n, a, \pi, N))$
```

```
$\mathsf{Open}_{\mathsf{TL}}(\mathbf{c, a, \pi, N})$

Return $\psi \equiv_N a^{2^\pi}$

$\mathsf{Dec}_{\mathsf{TL}}\ (\mathbf{R||C, a, \pi, N, \psi})$

$K' = \tilde{H}(\psi)$
$(m', r', q') = \tilde{H}(K'||R) \oplus C$
$N' = r'q', \varphi(N') = (r'-1)(q'-1)$
$s' \equiv_{\varphi(N')} 2^\pi, \psi' \equiv_{N'} a^{s'}$
If $N' = N$ and $\psi \equiv_N \psi'$, return $m'$.
Otherwise, return $\perp$.
```

Lemma 1. *The time-locked encryption scheme Π^{TL} satisfies correctness, secrecy and uniqueness under Conjecture 1 in the random oracle model.*

We remark that adding a, π, and N to every ciphertext tuple is not necessary, but makes black-box use in our subsequent protocols more simple. This way every tuple can be decrypted *individually* and independently.

3 Constructing a Graded Public Key Infrastructure (GPKI)

Our goal is to build a broadcast protocol using POWs. We proceed in several modular steps. First, we will construct a protocol, inspired by [2], that achieves a slightly weaker property than a PKI, which we call a *Graded PKI* (GPKI). Then we use this GPKI to achieve gradecast, a preliminary version of broadcast.

3.1 Public Key Infrastructure from Proof of Work

As a first step, we show how to construct something similar to a public key infrastructure (PKI). The challenge that we face without an a-priori setup is that we cannot easily achieve a global consistent view on the participants running the protocol. In fact, as discussed in the last section, we do not know the exact number of parties but only an upper bound. We address this problem by building a so-called *graded PKI*, which shall satisfy the following properties:

Definition 5 (Graded Public Key Infrastructure). *Let $\tilde{g}_i$ be a local function (for party P_i) that assigns the public key of every other party a grade from* 0 *to* 2. *We say that n parties share a* Graded PKI *(GPKI) if the following properties hold for all honest parties P_i, P_j:*

- Graded Validity: *P_i's public keys have grade 2 for P_j: $\tilde{g}_j(\mathsf{pk}_i) = 2$.*
- Graded Consistency: *If P_i has $\tilde{g}_i(\mathsf{pk}_k) = 2$ then P_j has $\tilde{g}_j(\mathsf{pk}_k) \geq 1$ for all k (i.e., here we also consider keys of malicious parties).*
- Bounded Number of Identities: *Let $\mathcal{K} := \{\mathsf{pk} | \exists\ P_j : \tilde{g}_j(\mathsf{pk}) \geq 1\}$ be the global set of accepted identities. Then, $|\mathcal{K}| \leq n$.*

Note that a graded PKI differs from a "real" PKI since the local view of honest party P_i on the identity set $\widehat{\mathcal{K}_i} \subset \mathcal{K}$ can be different from the view of every other honest party. We give the description of our protocol below but defer the details of the proof to Appendix B. The protocol begins with a 2-round challenge phase in which parties exchange randomly chosen challenges among each other. This phase is followed by a PoW phase in which each party P_i computes a PoW on the combination of all the challenges that it has received, as well as on a freshly generated public key pk_i. In the two main rounds that follow, all parties P_i and P_j exchange their PoWs and public keys, along with a proof that P_i included P_j's challenge to compute its (correct) PoW and vice-versa. If P_i receives a valid proof from P_j by the end of the first round in this last phase, it sets $\tilde{g}_i(\mathsf{pk}_j) = 2$

and relays all of the information received by P_j to all other parties. If P_i receives P_j's message indirectly from P_k by the end of the second round and $\tilde{g}_i(\mathsf{pk}_k) = 2$ has already been set in the first round, but pk_j has not been assigned a grade yet, then P_i sets $\tilde{g}_i(\mathsf{pk}_j) = 1$. This ensures that all honest parties accept each others public keys with grade 2 and that if an honest party accepts any public key with grade 2 then all other parties accept it with grade at least 1.

In order to prove that a certain party's challenge has been used to compute a proof of work, we will use *Merkle commitments.* Commitments are a cryptographic building block, which allows a prover to commit to a value such that later the commitment cannot be opened to a different value. Merkle commitments have the additional feature that one can commit to a long value, while the opening proof remains short (logarithmic in the length of the committed value). For simplicity we will consider the case where we want to commit to a power of 2 elements. The main building block is the *Merkle Hash Tree* which allows to hash multiple values into one single hash root h and later verify for each element if it was included in the hash. A Merkle Tree MT is a a labeled binary tree which is computed from a set of $2^{\ell-1} < t \leq 2^{\ell}$ elements. The leaf nodes of MT are computed as the hash of these elements while all other nodes are computed as the hashes of their two children nodes.

On an high level a Merkle commitment scheme $(\mathsf{Mhash}, \mathsf{Mproof}, \mathsf{Mver})$ uses Merkle Trees and a hash function $H : \{0,1\}^* \to \{0,1\}^{\kappa}$, which we model as a random oracle. On input of $2^{\ell-1} < t \leq 2^{\ell}$ elements $x_1, \ldots, x_t$ the algorithm Mhash creates a Merkle Tree (using H to hash the values).ÂťThe algorithm outputs the Merkle Tree MT and its root hash h. The algorithm Mproof takes as input such a Merkle Tree MT, a value x_i and an index $i \in [2^{\ell}]$. If MT is a correct Merkle Tree and x_i is its i-th leaf the algorithm outputs an ℓ–tuple $\varphi = \mathsf{Mproof}(x_i, i, MT)$ which contains all elements necessary to compute the hashes from x_i to h. On input of an element x, an index i, a root hash h and a proof φ, one can call $\mathsf{Mver}(x_i, i, h, \varphi)$ to verify whether x_i was used to compute the Merkle Tree with root hash h by calculating the hash path from x_i using the elements stored in φ. If this results in a root hash $h' = h$, the algorithm outputs 1; otherwise it outputs 0. For more information on how to create Merkle Trees efficiently we refer to [31]. In addition to Merkle commitments our protocol below will use a $\pi_{\mathcal{A}}$-secure POW scheme $(\mathsf{Solve}, \mathsf{VerifyPow})$ with security parameter κ as defined in the previous section, and a UF-CMA-secure signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$. Our protocol runs in time $4\Delta + \delta$, where δ is chosen such that it is not possible for the adversary to compute more than one PoW solution per unit π of computing power under its control (after learning all challenges), until the end of the protocol. This guarantees that no more than a total of n public keys can be accepted by the honest parties even if the adversary controls all but one party.

Key Grading Protocol $\mathit{\Pi}^{\mathrm{KG}}$

1. Challenge Phase:

This phase consists of 2 rounds, each lasting a time interval $\mathit{\Delta}$ each with message complexity $\mathcal{O}(\theta)$ and bit complexity $\mathcal{O}(\theta\kappa)$ per party.

Round 1: Each party P_i draws $a^1 \overset{\$}{\leftarrow} \{0,1\}^\kappa$ and sends (Chal_1, a^1) to every party, where Chal_1 is a unique label referring to the first round of $\mathit{\Pi}^{\mathrm{KG}}$.

Round 2: After round 1, P_i received $\widehat{m} \leq \theta$ challenges $A_i^1 = (a_1^1, \ldots, a_{\widehat{m}}^1)$. P_i computes $(MT_i^1, a_i^2) = \mathsf{Mhash}(A_i^1)$ and sends (Chal_2, a_i^2) to every party.

Let $A_i^2 = (a_1^1, \ldots, a_{\widehat{m}}^1, a_1^2, \ldots, a_{\widehat{m}}^2)_{\widehat{m} \leq \theta}$ be all elements received by P_i in this phase, then a_i denotes its Merkle hash, i.e. $(MT_i^2, a_i) = \mathsf{Mhash}(A_i^2)$.

2. Proof of Work Phase:

This phase consists of 1 round, lasting time δ without any messages.

Round 3: Every party P_i samples a secret and public key pair $(\mathsf{sk}_i, \mathsf{pk}_i) \overset{\$}{\leftarrow} \mathsf{Gen}(1^\kappa)$ and computes $\psi_i = \mathsf{Solve}(a_i, \mathsf{pk}_i)$

3. Key Ranking Phase:

This phase consists of 2 rounds, each lasting time interval $\mathit{\Delta}$ with message complexity $\mathcal{O}(\theta)$ and bit complexity $\mathcal{O}(\theta(\log(\theta) + \ell + \kappa))$ per party.

Round 4: For every $a_j^2 \in A_i^2$, every party P_i creates the Merkle proof $\varphi_{i,j}^2 = \mathsf{Mproof}(a_j^2, j, MT_i^2)$ and sends message $(\mathsf{Key}_2, \mathsf{pk}_i, a_i, \varphi_{i,j}^2, \psi_i)$ to P_j, where Key_2 is a label referring to the first round of the key ranking phase.

Round 5: For every message of form $(\mathsf{Key}_2, \mathsf{pk}_k, a_k, \varphi_{k,i}^2, \psi_k)$ that P_i received at the end of round 4, it checks if $\mathsf{VerifyPow}((a_k, \mathsf{pk}_k), \psi_k) = 1$ and if $\mathsf{Mver}(a_i^2, i, a_k, \varphi_{k,i}^2) = 1$. If both checks pass, it assigns grade 2 to public key pk_k and adds pk_k to $\widehat{\mathcal{K}_i}$. For every $a_j^1 \in A_i^1$ party P_i creates $\varphi_{i,j}^1 = \mathsf{Mproof}(a_j^1, j, MT_i^1)$ and sends $(\mathsf{Key}_1, \mathsf{pk}_k, a_k, \varphi_{k,i}^2, a_i^2, \varphi_{i,j}^1, \psi_k)$ to P_j.

For every message of form $(\mathsf{Key}_1, \mathsf{pk}_j, a_j, \varphi_{j,k}^2, a_k^2, \varphi_{k,i}^1, \psi_j)$ that P_i received at the end of round 5 from P_k, it checks if pk_j has not yet been assigned a grade and if $\mathsf{VerifyPow}((a_j, \mathsf{pk}_j), \psi_j) = \mathsf{Mver}(a_k^2, k, a_j, \varphi_{j,k}^2) = \mathsf{Mver}(a_i^1, i, a_k^2, \varphi_{k,i}^1) = 1$. If all checks pass, P_i sets $\tilde{g}_i(\mathsf{pk}_j) = 1$. Any unranked key is assigned grade 0.

The upper bound on the number of challenges is θ by definition of the communication model. Therefore, the message complexity of our protocol is $\mathcal{O}(\theta)$ and the bit complexity $\mathcal{O}(\theta u)$, where $u = \ell + \kappa + \log(\theta)$ is comprised by the length of the POW ψ (length ℓ), the challenge (length κ) and the Merkle Proof φ (length $\log(\theta)$) that an element a was included in the POW challenge. We give a formal proof for Lemma 2 in Appendix B.1.

Lemma 2. *Protocol $\mathit{\Pi}^{\mathrm{KG}}$ achieves a GPKI among n parties in the random oracle model where the number of malicious identities is strictly less than $\frac{n}{2}$.*

3.2 From graded PKI to Gradecast

Now that we have a GPKI, we can use it to construct a protocol called *gradecast* (originally introduced by [13]). This protocol is used to distribute a message m from a dealer P_D to all parties under similar, but weaker conditions than for the full-fledged broadcast protocol.

Definition 6. *A protocol achieves* Gradecast *among n parties for some dealer P_D with input m iff the following properties hold for all honest parties P_i, P_j:*

Graded Validity: *If P_D is honest, then P_i outputs $m_i = m$ with $g_i = 2$.*
Graded Consistency: *If P_i outputs m_i with $g_i = 2$ then P_j outputs $m_j = m_i$ and $g_j \geq 1$.*

Next, we describe an n-party protocol that achieves gradecast for message $m \in \{0,1\}^\lambda$, which requires a GPKI. Let $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ denote a UF-CMA-secure signature scheme. Then, the 5-round protocol Π^{GC} achieves gradecast running in time 5Δ with message complexity $\mathcal{O}(n)$ and bit complexity $\mathcal{O}(n(\lambda + n(\kappa + |\sigma|)))$. Here, $|\sigma|$ denotes the length of a signature.

Gradecast Protocol Π^{GC}

Round 1: To gradecast $m \in \{0,1\}^\lambda$, P_D computes $\sigma = \mathsf{Sign}(\mathsf{sk}_D, m)$ and sends (m, σ) to all parties.
Round 2: Each party P_i does the following upon receiving (m_i, σ_i) from P_D: It checks if $\mathsf{Ver}(\mathsf{pk}_D, m_i, \sigma_i) = 1$ and if $g_i(\mathsf{pk}_D) \geq 1$. If both checks pass, P_i sends (m_i, σ_i) to every other party. Otherwise, it sets $m_i := \perp$ and sends nothing
Round 3: Let $(m_{j,i}, \sigma_{j,i}), j \in [\widehat{n}]$ be the message/signature pair that P_i received from P_j at the end of round 2. If $\exists\ j, k \in [\widehat{n}] : m_{j,i} \neq m_{k,i}$ and $\mathsf{Ver}(\mathsf{pk}_D, m_{j,i}, \sigma_{j,i}) = \mathsf{Ver}(\mathsf{pk}_D, m_{k,i}\sigma_{k,i}) = 1$, it sets $m_i := \perp$. If $m_i \neq \perp$, P_i sends (m_i, σ'_i) to each other party, where $\sigma'_i = \mathsf{Sign}(\mathsf{sk}_i, m_i)$.
Round 4: Each party P_i that received $l \geq \frac{n}{2}$ messages at the end of round 3 of the form $(m^*, \sigma^*_{j,i}), j \in [l]$ with $\tilde{g}_i(\mathsf{pk}_j) = 2$ and s.t. $\mathsf{Ver}(\mathsf{pk}_j, m^*, \sigma^*_{j,i}) = 1$, sends $(m^*, \sigma^*_{1,i}, \mathsf{pk}_1, ..., \sigma^*_{l,i}, \mathsf{pk}_l)$ to all other parties. It then outputs $m_i := m^*, g_i := 2$.
Round 5: Each party P_i that has not determined its output yet but received a message $(m^*, \sigma'_1, \mathsf{pk}_1, \ldots, \sigma'_l, \mathsf{pk}_l)$ at the end of round 4 such that $\forall j \in [l]$: $\mathsf{Ver}(\mathsf{pk}_j, m^*, \sigma^*_{j,i}) = 1$, $\tilde{g}_i(\mathsf{pk}_j) \geq 1$ and $l \geq \frac{n}{2}$, outputs (m_i, g_i) with $m_i := m^*, g_i := 1$. Otherwise, P_i sets $g_i = 0$ and outputs $\perp$.

The following lemma is proven in Appendix B.2.

Lemma 3. *Given a graded PKI, Π^{GC} is a constant round protocol achieving graded broadcast if strictly more than $\frac{n}{2}$ parties are honest.*

The idea of this protocol is to integrate the structure of the GPKI with the original gradecast protocol from [18] which relies on a full-fledged PKI. Informally, the original protocol does not include the checks in rounds 4 and 5 on the grades of the public keys under which the signatures were created. The main insight now is that the properties of the GPKI ensures graded consistency for the protocol Π^{GC}. Namely, we want to achieve that any message that is accepted by an honest party with grade 2 in round 4, is accepted with (at least) grade 1 by every other honest party by the end of round 5. To accept a message in round 4, we require that there were at least $n/2$ signatures under public keys of grade 2. By the properties of a GPKI, this guarantees that any accepted message (in round 4) has obtained at least this amount of signatures under public keys of grade at least 1 and is thus accepted in round 5 by every other honest party.

4 Electing a Leader with a Graded PKI

To lift a gradecast protocol to a full-fledged broadcast, we will follow the approach of Katz and Koo [18] and use a protocol for oblivious leader election. To this end, we adapt their approach (which uses a PKI instead of a GPKI) and build a leader election protocol using either moderated Verifiable Secret Sharing (VSS) or Time-Lock-Encryption (TLE). The authors of [18] show how to build a protocol for byzantine agreement from leader election and we can use their construction with only minor modifications. For this reason, we move this last step of constructing byzantine agreement from oblivious leader election to Appendix C. We begin by giving a formal definition of the properties of an oblivious leader election protocol.

Definition 7. *(Oblivious Leader Election): A two-phase protocol for n parties is an* Oblivious Leader Election protocol with fairness ε tolerating t malicious parties *if each honest party P_j outputs an index $i_j \in [n]$ at the end of the second phase and for any PPT adversary controlling at most t parties, with probability at least ε (over the random coins of the honest parties) there exists $i \in [n]$ s.t. the following conditions hold:*

- *Consistency: All honest parties P_j output $i_j = i$.*
- *Honesty: P_i followed the protocol specification at least for the first phase.*

Note in particular that if P_i was honest up to the end of the first phase, then it has followed all steps of the entire protocol according to the specification, up to this point. In the following sections we focus on how to build an oblivious leader election protocol from the building blocks that we have established thus far.

4.1 Oblivious Leader Election from Verifiable Secret Sharing (VSS)

The authors of [18] show how to build a constant-round oblivious leader election protocol from verifiable secret sharing (VSS) in a setting where a PKI is given. We show in detail how to translate the scheme of [18] into a setting where only a graded PKI (and hence a gradecast protocol) is given (c.f. Appendix C). While this protocol runs in constant rounds, the construction exhibits a very large communication complexity. This results from the fact that $\mathcal{O}(n^2)$ runs of a VSS scheme are executed in parallel, where each of them requires $\mathcal{O}(n)$ gradecast executions, i.e, $\mathcal{O}(n^3)$ runs of the gradecast protocol have to be executed in parallel. Note that the gradecast protocol has a cost of $\mathcal{O}(n^2(|\sigma| + \kappa))$ for any message up to a size of $\mathcal{O}(n^2)$. This comes from the fact that regardless of the message length, each user must send $\mathcal{O}(n)$ messages including $\mathcal{O}(n)$ signatures.

Therefore, one idea to improve communication complexity is by batching multiple messages into one VSS execution. Unfortunately, due to technical reasons, this approach does not work in the protocol of [18]. Even when we apply further optimizations to their protocol, this results in an overall bit complexity of $\mathcal{O}(n^5(\log(n) + |\sigma| + \kappa))$ in the bilateral channels model. One of our main technical contributions is a method to replace the VSS approach with a time-locked encryption scheme. At a high level, the time-locked encryption gives us more flexibility, because parties do not need to interact after an initial phase of distributing commitments in order to open them. In contrast, in the VSS-based version of the protocol, parties must combine their shares to reconstruct the secret values. By removing the need for communication after an initial phase of setup and commitment distribution, we can batch many messages into a single invocation of the gradecast protocol per party. All in all, the resulting scheme has to run only $\mathcal{O}(n)$ gradecasts in parallel, which greatly reduces the bit complexity while adding only a small penalty to the running time of the protocol.

4.2 Oblivious Leader Election from Time-Locked Encryption (TLE)

We now present our protocol for oblivious leader election from time-locked encryption. We proceed using a two-step approach, introducing first a protocol Π^{BCE} which we call *Broadcast Emulation* protocol. Then, we show how to combine Π^{BCE} with time-locked encryption to elect a leader.

Broadcast Emulation. The goal of the protocol Π^{BCE} is to jointly distribute up to n messages per party, where we only require mild consistency guarantees for the message delivery. We will show that these mild consistency requirements suffice to build an oblivious leader election protocol when using time-locked encryption. In the Π^{BCE} protocol every party P_i starts with some set of $\widehat{n}$ messages $m_1, \ldots, m_{\widehat{n}}$, where each such message is assigned to one known party. Π^{BCE} now proceeds in two phases. In Phase 1, each P_i gradecasts the message vector $\boldsymbol{M} = (m_1, \ldots, m_{\widehat{n}})$ using a single invocation of the gradecast protocol from the previous section. Each message m_j is sent together with a public key, pk_j, which identifies the party P_j owning pk_j as *the relay* for the message m_j.

In Phase 2, each honest party P_i collects all messages for which he was assigned as the relay, i.e., all messages attached to its public key, and combines them into one message vector $\boldsymbol{M'}$. It then gradecasts $\boldsymbol{M'}$, again using only a single invocation of the gradecast protocol (thereby effectively relaying the messages in $\boldsymbol{M'}$). This time it includes for every message m' within $\boldsymbol{M'}$ the public key corresponding to the original sender, i.e., the party from which it received m' in Phase 1. At the end of Phase 2, every relay P_j is assigned a flag $f_j \in \{0,1\}$ by each honest party P_i. This flag indicates whether from P_i's view, P_j correctly relayed each message that it was supposed to relay.

Our protocol guarantees that P_i sets $f_j = 1$, if P_j honestly ran Π^{BCE}. Conversely, if any honest party sets $f_j = 1$, we are guaranteed that P_j has followed

Π^{BCE} as specified in the protocol description. At the end of the protocol, every honest party outputs its flags $f_1, ..., f_{\widehat{n}}$ and all the message vectors relayed by the other parties in Phase 2, i.e., $\boldsymbol{M}'_1, ..., \boldsymbol{M}'_{\widehat{n}}$. To allow to use the Π^{BCE} protocol in a modular way as a subprotocol in the leader election, we require that every party P_i additionally outputs the sets $\mathcal{T}_{i,1}, ..., \mathcal{T}_{i,\widehat{n}}$ and $\mathcal{T}_i, \tilde{\mathcal{T}}_i$, which are defined as follows: The set $\mathcal{T}_{i,j}$ contains all grade-two messages m intended to be relayed by P_j in Phase 2. This means that m was the first component of a tuple (m, pk_j) inside a message vector $\boldsymbol{M}$ which P_i received with grade 2 in Phase 1[6]. We then set $\mathcal{T}_i$, resp. $\tilde{\mathcal{T}}_i$, as the set of all message vectors $\boldsymbol{M}$ that P_i received in Phase 1 with grade 2 (resp. grade at least 1).

The protocol Π^{BCE} runs gradecast as a subprotocol twice and thus has a constant number of rounds. The message complexity for each party is $\mathcal{O}(n^2)$ the bit complexity is $\mathcal{O}(n^3(\lambda + \kappa + |\sigma|))$.

Broadcast Emulation Protocol Π^{BCE}

Phase 1:

Let $m_1, \ldots, m_{\widehat{n}}$ be P_i's messages with $m_j \in \{0,1\}^\lambda, j \in [\widehat{n}]$. P_i gradecasts $\boldsymbol{M} = (m_1, \mathsf{pk}_1, \ldots, m_{\widehat{n}}, \mathsf{pk}_{\widehat{n}})$.

Phase 2:

Let $\boldsymbol{M}_j = (m_1^j, \mathsf{pk}_1, \ldots, m_{\widehat{n}}^j, \mathsf{pk}_{\widehat{n}})$ be the message vector that P_i received with grade g_j from party P_j in Phase 1. Let $\mathcal{T}_i$ ($\tilde{\mathcal{T}}_i$, respectively) denote the set of all message vectors that P_i received in Phase 1 as part of a message with grade 2 (≥ 1, respectively). Let $\mathcal{T}_{i,j} \subset \mathcal{T}_i$ denote the set of all messages m that P_i received in Phase 1 as part of a message vector $\boldsymbol{M}$ with grade 2 and as the first component of a pair (m, pk_j). P_i gradecasts $\boldsymbol{M}' = (m_i^1, \mathsf{pk}_1, \ldots, m_i^{\widehat{n}}, \mathsf{pk}_{\widehat{n}})$.

Output Determination:

Let $\boldsymbol{M}'_j = (\tilde{m}_j^1, \mathsf{pk}_1, \ldots, \tilde{m}_j^{\widehat{n}}, \mathsf{pk}_{\widehat{n}})$ be the message vector that P_i received with grade g'_j from party P_j at the end of Phase 2. P_i assigns a flag f_j to every party P_j which is by default set to 1. It sets $f_j = 0$, if $g'_j < 2$, if P_j has relayed more than n messages or if there exists $(m_j^k, \mathsf{pk}_j) \in \boldsymbol{M}_k$ with $g_k = 2$ but $(m_j^k, \mathsf{pk}_k) \notin \boldsymbol{M}'_j$.[a] P_i outputs $f_1, \ldots, f_{\widehat{n}}, \boldsymbol{M}'_1, ..., \boldsymbol{M}'_{\widehat{n}}, \mathcal{T}_{i,1}, ..., \mathcal{T}_{i,\widehat{n}}, \mathcal{T}_i, \tilde{\mathcal{T}}_i$.

[a] This can be understood as P_j refusing to relay message m_j^k for which it was named the relay by P_k in Phase 1. Note that from P_i's perspective, P_k appears to be honest.

Lemma 4. *Let P_i and P_j be honest parties. Moreover, denote with m_k^j the message sent by P_j in Phase 1 for which party P_k was specified as the relay. Π^{BCE} satisfies the following conditions given that strictly more than $\frac{n}{2}$ of all parties are honest.*

[6] Since a pair (m, pk_j) gradecastet by some party in Phase 1 of Π^{BCE} means that P_j should relay m via gradecast in Phase 2.

1. *P_i outputs $f_j = 1$ at the end of Phase 2.*
2. *If P_i outputs $f_k = 1$, then P_i and P_j both received $\boldsymbol{M}'_k$ from P_k in Phase 2.*
3. *If P_i outputs $f_k = 1$ then P_k included m_k^j in its relay $\boldsymbol{M}'_k$ in Phase 2.*
4. *If $\boldsymbol{M} \in \mathcal{T}_i$ then $\boldsymbol{M} \in \tilde{\mathcal{T}}_j$.*

Proof. We show that the four properties as defined above hold. Let P_i, P_j be honest. We begin by showing Property 1, i.e., that P_i outputs $f_j = 1$ at the end of Phase 2. We show that all checks performed by P_i at the end of Π^{BCE} hold with respect to P_j. Suppose that P_i receives $\boldsymbol{M}_k$ from P_k in Phase 1 with grade $g_k = 2$. By the properties of gradecast, P_j receives $\boldsymbol{M}_k$ in Phase 1 with grade at least 1 by the graded consistency property of gradecast. Thus, P_j includes (m_j^k, pk_k) in its message vector $\boldsymbol{M}'_j$ for Phase 2 and by the graded validity property of gradecast, P_i receives $\boldsymbol{M}'_j$ in Phase 2 with grade $g'_j = 2$. Finally, P_j does not relay messages for more than n parties since $\widehat{n} \leq n$ by the bounded number of identities of the graded PKI. Therefore, P_i outputs $f_j = 1$.

For Property 2, we show that if P_i outputs $f_k = 1$, then P_j has received the same vector $\boldsymbol{M}'_k$ from P_k in Phase 2 as P_i. Since P_i only sets $f_k = 1$ if $g'_k = 2$, this follows directly from the graded consistency property of gradecast. For Property 3, we will show that if P_i outputs $f_k = 1$ then $(m_k^j, \mathsf{pk}_j) \in \boldsymbol{M}'_k$ given that $(m_k^j, \mathsf{pk}_k) \in \boldsymbol{M}_j$, i.e., P_k faithfully relays all messages in Phase 2 for which it is designated to relay by party P_j. By graded validity, P_j's message vector $\boldsymbol{M}_j$ in Phase 1 has grade $g_j = 2$ for P_i. Since P_i has set $f_k = 1$, it must hold that $(m_k^j, \mathsf{pk}_j) \in \boldsymbol{M}'_k$ as otherwise condition three in the checks of Π^{BCE} would be violated. Finally, Property 4 follows directly from the graded consistency property of gradecast. □

From Broadcast Emulation to Oblivious Leader Election. We present the oblivious leader election protocol Π^{OLE} based on Π^{BCE}. The idea of this protocol is to assign random values x_i to every party P_i and make the leader the party with the smallest x_i. To ensure that the consistency property of our protocol is satisfied, the honest parties must all choose the same value x_i as their local minimum with constant probability. We start by presenting a strawman solution using Π^{BCE} as follows. The protocol proceeds in two phases, both of which have two substeps. In Step 1 of the first phase, each party P_i selects uniformly random coins $x_{i,j} \in [n^4]$ for all $j \in [\widehat{n}]$ and computes corresponding commitments $c_{i,j}$. In Step 2 of Phase 1, the parties distribute the computed commitments by running Π^{BCE}. The commitment $c_{i,j}$ will be relayed by P_j in Phase 2 of Π^{BCE}. Thus, P_j relays a message of the form $(c_{1,j}, ..., c_{\widehat{n},j})$.

In Phase 2, the commitments of the parties are opened and x_j is computed as $x_j = \tilde{\mathsf{G}}(x_{1,j}||...||x_{n,j})$ by every party. Here, $\tilde{\mathsf{G}} : \{0,1\}^* \to [n^4]$ denotes a random oracle. By the properties of Π^{BCE}, if $f_j = 1$ from the view of at least one honest party, we know that all parties have received the same values $x_{k,j}$ (from the opened commitments) and that all the values chosen by honest parties were included in the call to $\tilde{\mathsf{G}}$. Since the honest parties' input to $\tilde{\mathsf{G}}()$ are themselves uniformly random values, one may conclude that x_j is a uniform value in $[n^4]$

and all honest parties have a consistent view of x_j[7].It would then follow from the uniformity of the values x_j that with probability roughly $1 - \frac{n-t}{n}$ (recall that t is the number of dishonest parties), the smallest value x_j will correspond to an honest party P_j, i.e., the honest parties elect an honest leader.

Unfortunately, the above approach does not work due to the following two challenges that we need to address. First, we must ensure that the commitments hide the random coins of the honest parties. Otherwise, the adversary could simply adjust its own coins so that the elected leader is chosen from the set of corrupted parties. This problem is easily solved by ensuring that the commitments are hiding. However, there is a another more challenging issue. The adversary, albeit not being able to adaptively choose its coins, can adaptively choose not to provide the opening information for its commitments. Even worse, he may distribute to some honest parties an opening, while he may refuse to send the opening to others. This completely breaks the security proof of the protocol because of two reasons. First, the adversary may bias the outcome of the protocol by opening its commitments in such a way that it becomes more likely that a dishonest party is elected leader. Second, the adversary may introduce inconsistencies between honest parties of which party being the elected leader.

Our solution to these problems is to use time-lock ciphertexts instead of commitments. We have already shown that our construction for time-lock encryption is hiding until time δ after the ciphertexts have been published. Moreover, the ciphertexts in our construction have the property that they can be opened to a unique value only. Concretely, Phase 2 of our OLE protocol will consist of the parties randomly choosing ciphertexts that they have received in Phase 1 and computing the opening tokens for them. They will then simply send these tokens to the network. The uniqueness property of our time-lock encryption protocol ensures that every ciphertext can either be opened, or that one can produce a proof (in the form of a token) that the ciphertext is invalid. We will show that this procedure guarantees that with constant probability all parties know the corresponding decryption to every necessary ciphertext by the end of Step 1 of Phase 2. In order to guarantee efficiency, our protocol includes checks which ensure that the honest parties need to compute only $\mathcal{O}(n)$ such decryption tokens in order to open all of the adversary's time-locked ciphertexts.

Before explaining the protocol, we will introduce all sets and flags which we will use in the following description of Π^{OLE}.

1. Recall, that every honest party P_i outputs $\mathcal{T}_{i,1}, ..., \mathcal{T}_{i,\widehat{n}}, \mathcal{T}_i, \tilde{\mathcal{T}}_i$ at the end of the Π^{BCE} protocol. To argue about a global set on all messages that were received with grade 2 (respectively with grade 1) by at least one honest party P_j, we additionally define the sets $\mathcal{T} := \bigcup_j \mathcal{T}_j$ (resp. $\tilde{\mathcal{T}} := \bigcup_j \tilde{\mathcal{T}}_j$), where the union operator iterates over the set of honest parties.
2. Additionally, each party needs to compute two sets $\mathcal{M}$ and $\mathcal{S}$, where $\mathcal{S}$ contains all ciphertexts which can be decrypted correctly. All other ciphertexts,

[7] We remark that the adversary cannot influence the values of the honest parties that are input to $\tilde{\mathsf{G}}()$. This follows from our model assumption, according to which the adversary cannot drop or modify messages sent by honest parties

which were not computed correctly are stored in $\mathcal{M}$. Notice that these both sets include additional information, necessary for verifying these claims.
3. At the end of the protocol each party P_i will assign a flag $\mathsf{trust}_{i,j}$ to every other party P_j if this party behaved honestly from P_i's view. The set trust_i contains all parties j that are assigned $\mathsf{trust}_{i,j} = 1$ by P_i.

Next, we present the formal description of our protocol individually for every phase. Overall, the protocol consists of $\mathcal{O}(r)$ rounds (where $r = \mathcal{O}(\log(n))$). It inherits the message and bit complexity from the underlying Π^{BCE}. In the setup and coin distribution phase all parties P_i choose $\widehat{n}$ random coins $x_{i,j}, j \in [\widehat{n}]$ and encrypt them. They distribute their time-locked-ciphertext using the Π^{BCE} protocol. At the end of this phase, P_i assigns a trust value $\mathsf{trust}_{i,j}$ to every party P_j from the Π^{BCE} protocol which equals to the value of its output flag f_j.

Oblivious Leader Election Protocol Π^{OLE}

Phase 1: Setup and Coin Distribution

Step 1: For all $j \in [\widehat{n}]$, P_i chooses random $x_{i,j} \in [n^4]$ and sets $\mathsf{trust}_{i,j} = 1$. It generates $\boldsymbol{c} = (c_1, \ldots, c_{\widehat{n}}) = \mathsf{Enc}_{\mathsf{TL}}(x_{i,1}, \ldots, x_{i,n})$.
Step 2: P_i runs Π^{BCE} on input $\boldsymbol{c}$ and outputs $(f_1, ..., f_{\widehat{n}}, \boldsymbol{M}'_1, ..., \boldsymbol{M}'_{\widehat{n}}, \mathcal{T}_{i,1}, ..., \mathcal{T}_{i,\widehat{n}}, \mathcal{T}_i, \tilde{\mathcal{T}}_i)$. $\forall j$ where $f_j = 0$, P_i sets $\mathsf{trust}_{i,j} = 0$.

In the first step of Phase 2 of Π^{OLE}, the time-locked ciphertexts need to be opened. In order to do so, every party P_i randomly chooses one vector $\tilde{\boldsymbol{M}} \in \tilde{\mathcal{T}}_i$ and computes its opening token $\tilde{\psi}$. Then it checks if $\tilde{\boldsymbol{M}}$ can be decrypted with token ψ[8]. If this is not possible, we call $\tilde{\boldsymbol{M}}$ *malformed*, i.e., if there are $c_i, c_j \in \tilde{\boldsymbol{M}}$ s.t. $\exists \psi : \mathsf{Dec}_{\mathsf{TL}}(c_i, \psi) = \perp, \mathsf{Dec}_{\mathsf{TL}}(c_j, \psi) \neq \perp$. Otherwise, we call $\tilde{\boldsymbol{M}}$ *wellformed* which means we can decrypt every component of $\tilde{\boldsymbol{M}}$ using the decryption token $\psi := \mathsf{Open}_{\mathsf{TL}}(\tilde{\boldsymbol{M}}[1])$. If $\tilde{\boldsymbol{M}}$ is wellformed, P_i sends $\tilde{\psi}$ (together with $\tilde{\boldsymbol{M}}$) to all other parties. This step is repeated r times to ensure that all ciphertexts are decrypted with high probability (c.f. Lemma 6)

In the second step, P_i verifies the validity of the ℓ received opening tokens $\tilde{\psi}_j, j \in [\ell]$ with respect to their corresponding time-locked ciphertext $\tilde{c}_j \in \tilde{\boldsymbol{M}}_{\boldsymbol{j}}$ s.t. $\tilde{c}_j \in \boldsymbol{M}'_i$, i.e., $\tilde{c}_j$ was one of the ciphertexts relayed by P_i in the second phase of the internal run of Π^{BCE}. If the opening is valid, P_i adds the pair $(\tilde{\psi}_j, \tilde{c}_j)$ to $\mathcal{S}$. Intuitively, at the end of the second step, P_i (from P_j's perspective) is responsible for including in the set $\mathcal{S}$ to every time-locked ciphertext c within $\mathcal{T}_{j,i}$ the pair (c, ψ) s.t. ψ is the unique opening token for c. If ψ does not exist, then instead P_i must provide a proof for this fact within the set $\mathcal{M}$. At the end of the second step, P_i gradecasts $\mathcal{S}$ and $\mathcal{M}$. Now, P_i adjusts the trust towards party P_j by setting $\mathsf{trust}_{i,j} = 0$ if any of these checks pass: (1) if the grade of the message is not 2, (2) if $\mathcal{S}_j$ contains invalid ciphertexts, (3) if $\mathcal{M}_j$ contains

[8] In the following, we will slightly abuse notation and say that $\tilde{\boldsymbol{M}}$ can be decrypted with a unique token ψ, if every component of $\tilde{\boldsymbol{M}}$ can be decrypted using ψ.

any *valid*, i.e., wellformed ciphertext vectors (4) if P_j did not provide all of the necessary opening tokens to ciphertexts in $\mathcal{T}_{i,j}$ (5) if P_j provided an invalid proof for not being able to open $c \in \mathcal{T}_{i,j}$.

Oblivious Leader Election Protocol Π^{OLE} (cont.)

Phase 2: Decryption

Init: Every party sets $\mathcal{S} \leftarrow \emptyset, \mathcal{M} \leftarrow \emptyset$

Step 1: Each P_i repeats this step r times: Sample $\tilde{\boldsymbol{M}} \overset{\$}{\leftarrow} \tilde{\mathcal{T}}_i$, compute $\tilde{\psi} \leftarrow \mathsf{Open}_{\mathsf{TL}}(\tilde{\boldsymbol{M}}[1])$. If $\tilde{\boldsymbol{M}}$ is not malformed, send $(\tilde{\psi}, \tilde{\boldsymbol{M}})$ to every other party.

Step 2: Let $(\tilde{\psi}_1, \tilde{\boldsymbol{M}}_1, \ldots, \tilde{\psi}_\ell, \tilde{\boldsymbol{M}}_\ell)$ be the set of pairs that P_i received in the previous Step. P_i checks $\forall j \in [\ell]$ if there exists $(\tilde{c}_j, \cdot) \in \tilde{\boldsymbol{M}}_j \cap \boldsymbol{M}_i'^{\,a}$, $\tilde{\boldsymbol{M}}_j$ is not malformed, and $\mathsf{Dec}_{\mathsf{TL}}(\tilde{\psi}_j, \tilde{c}_j) \neq \perp$, then P_i adds $(\tilde{\psi}_j, \tilde{c}_j)$ to $\mathcal{S}$. If $\tilde{\boldsymbol{M}}_j$ is malformed, P_i instead adds $(\tilde{\boldsymbol{M}}_j, \tilde{\psi}_j, \tilde{c}_j)$ to $\mathcal{M}$. P_i gradecasts $(\mathcal{S}, \mathcal{M})$.

Step 3: We denote as $(\mathcal{S}_j, \mathcal{M}_j)$ the sets received from P_j in this manner and denote the assigned grade as g_j. P_i sets $\mathsf{trust}_{i,j} = 0$ if one of the following conditions holds:

(1) if $g_j < 2$,
(2) $\exists(\psi, c) \in \mathcal{S}_j : \mathsf{Dec}_{\mathsf{TL}}(c, \psi) = \perp$,
(3) $\exists(\boldsymbol{M}, \psi, \cdot) \in \mathcal{M}_j, \not\exists$ s.t. $(c_i, \cdot), (c_j, \cdot) \in \boldsymbol{M} : \mathsf{Dec}_{\mathsf{TL}}(c_i, \psi) \neq \mathsf{Dec}_{\mathsf{TL}}(c_j, \psi)$,
(4) $\exists c \in \mathcal{T}_{i,j}$ s.t. $(\cdot, c) \notin \mathcal{S}_j$ and $(\cdot, \cdot, c) \notin \mathcal{M}_j$,
(5) $\exists(\boldsymbol{M}, \cdot, c) \in \mathcal{M}_j : c \in \mathcal{T}_{i,j}, (c, \cdot) \notin \boldsymbol{M}$.

[a] Here, $\cdot$ is a placeholder for a public key.

At the end of Phase 2 each honest party P_i now knows the opening tokens necessary to decrypt the random values $x^i_{k,j} = \mathsf{Dec}_{\mathsf{TL}}(c_{k,j}, \psi_{k,j})$. Additionally, each party P_i has a trust value $\mathsf{trust}_{i,j} \in \{0, 1\}$ assigned to P_j, where $\mathsf{trust}_{i,j} = 1$ means that P_j behaved honestly during the protocol execution (so far). Now everyone can jointly elect a leader by querying a random oracle $\tilde{\mathsf{G}}$ on all random coins and using the result to determine the leader among the trusted parties.

Oblivious Leader Election Protocol Π^{OLE} (cont.)

Output determination:

For all $j \in [\widehat{n}], k \in [n]$, P_i does as follows: Let $(c_{k,j}, \cdot) \in \boldsymbol{M}_j'$ such that $(\psi_{k,j}, c_{k,j}) \in \mathcal{S}_j$. Let $x^i_{k,j} = \mathsf{Dec}_{\mathsf{TL}}(c_{k,j}, \psi_{k,j})$. If $x^i_{k,j}$ is outside of $[n^4]$, P_i sets $x^i_{k,j} = 0$. It then computes $x^i_j := \tilde{\mathsf{G}}(x^i_{1,j}||...||x^i_{\widehat{n},j})$ and adds j to trust_i iff $\mathsf{trust}_{i,j} = 1$. P_i then outputs the $j \in \mathsf{trust}_i$ which minimizes the value of x^i_j.

We proceed by proving a sequence of statements about Π^{OLE} which together prove Theorem 1. The idea of the proof is roughly as follows. Note that the set trust_i contains all parties j which have followed the protocol correctly from P_i's view. We now define $\mathsf{trusted} := \{k | \exists i$ s.t. $k \in \mathsf{trust}_i$ and P_i was honest at the end of Phase 2 of $\Pi^{\text{OLE}}\}$. Informally, $\mathsf{trusted}$ is the set of all parties that have followed Π^{OLE} correctly with respect to at least one honest party P_i (in other words, it is the union of all sets trust_i for all honest parties P_i). The basic structure of our proof is now as follows. We start with proving in Lemma 7 that with high probability for all honest P_i, P_j, $i \in \mathsf{trust}_j$, i.e., at the end of the protocol, honest parties trust each other. Thus, for each honest party P_i, we have that $i \in \mathsf{trusted}$ at the end of the protocol. Note that the sets trust_i and trust_j as defined in the protocol may differ in general for honest parties P_i, P_j. For this reason, the set $\mathsf{trusted}$ may not be explicitly known to any of the honest parties and is purely used as part of our proof strategy.

The next step for proving Theorem 1 is to show that if $k \in \mathsf{trusted}$, $x_k = x_k^i = x_k^j$ for all honest P_i, P_j, i.e., trusted parties have consistent coins among honest parties (cf. Lemma 9). Furthermore, we prove (also in Lemma 9) that these coins are uniformly random in the interval $[n^4]$. Combining these two observations, we prove in Lemma 10 that with high probability, the minimal value x_j for $j \in \mathsf{trusted}$ corresponds to an honest party P_j. Since the values are uniformly random in the interval $[n^4]$, they are also unique with suitable probability. Note that in this case, all honest parties will indeed choose the same (honest) P_j as their leader, because every honest party P_i chooses its leader locally from $\mathsf{trust}_i \subseteq \mathsf{trusted}$ and for all i, we have that $j \in \mathsf{trust}_i$, given that P_j is honest. Before moving on to proving Theorem 1, we start with the following simple technical lemma, whose proof can be found in the appendix.

Lemma 5. *For all $n > 2$, $(1 - \frac{1}{n})^n \leq \frac{1}{e}$.*

We now are ready to prove the following statement about the global consensus on the tokens for wellformed ciphertexts.

Lemma 6. *Let the number of honest parties at least $\frac{n}{2}$. At the end of Step 1 of Phase 2 of Π^{OLE}, w.p.r. at least $1 - ne^{-\frac{r}{2}}$, each honest party P_i knows for each wellformed $\boldsymbol{M} \in \mathcal{T}$ the unique token ψ s.t. $\forall c \in \boldsymbol{M} : \mathsf{Dec}_{\mathsf{TL}}(c, \psi) \neq \perp$.*

Proof. Note that there are at most n possible tokens for wellformed $\boldsymbol{M} \in \tilde{\mathcal{T}}$ (this follows from the uniqueness of the tokens and the bounded number of identities). On the other hand, there are at least $\frac{n}{2}$ honest parties and $\forall i, \mathcal{T} \subset \tilde{\mathcal{T}}_i$. In each round, P_i selects a message vector $\tilde{\boldsymbol{M}}$ at random from $\tilde{\mathcal{T}}_i$ and computes $\psi \xleftarrow{\$} \mathsf{Open}_{\mathsf{TL}}(\tilde{\boldsymbol{M}}[1])$. Clearly, P_i selects each $\tilde{\boldsymbol{M}}$ with probability at least $\frac{1}{n}$ in any given round. Therefore, the probability that in a single round, the unique token ψ to a wellformed vector $\boldsymbol{M} \in \mathcal{T}$ is *not* computed by any of the honest parties, is at most $(1 - \frac{1}{n})^{\frac{n}{2}} \leq \frac{1}{\sqrt{e}}$. This follows from Lemma 5 and the fact that the square root function is monotone. Whenever an honest party computes a token ψ, it sends it to every other party in the network. Thus, after r rounds and by the union bound, the probability that there exists a vector $\boldsymbol{M} \in \mathcal{T}$ s.t. $\tilde{\boldsymbol{M}}$ is not malformed and for which P_i does not know the unique decryption token ψ, is at most $ne^{-\frac{r}{2}}$. □

We proceed by proving that for any honest parties P_i, P_j, $i \in \mathsf{trusted}_j$ (and thus $i, j \in \mathsf{trusted}$).

Lemma 7. *Let P_i and P_j be honest parties and let $\mathcal{E}_{i,j}$ be the event that $j \in \mathsf{trusted}_i$. Then $\Pr[\bigwedge_{i,j} \mathcal{E}_{i,j}] \geq 1 - ne^{-\frac{r}{2}}$.*

Proof. Note that immediately after Phase 1, Property 1 of Lemma 4 guarantees that every honest player outputs $f_j = 1$, so $j \in \mathsf{trusted}_i$ at the end of Phase 1. We show that with probability at least $1 - ne^{-\frac{r}{2}}$ there exists no honest parties P_j, P_i s.t. P_j violates one of P_i's checks at the end of Phase 2. Clearly Check (1) holds, which follows directly from the properties of gradecast. Check (2) also holds, because by the protocol specification, P_j only includes (c, ψ) in $\mathcal{S}_j$ if $\mathsf{Dec}_{\mathsf{TL}}(c, \psi) \neq \perp$. For check (4), note that if $c \in \mathcal{T}_{i,j}$, P_j received (c, pk_j) as part of the *same* message vector $\boldsymbol{M} \in \tilde{\mathcal{T}_j}$ as P_i (by Property 4 of Lemma 4). By the protocol specification, P_j includes $(\boldsymbol{M}, \cdot, c) \in \mathcal{M}_j$ only if $(c, \cdot) \in \boldsymbol{M}$, and so this check also holds from P_i's perspective. It follows in a similar manner that Check (3) also holds, because P_j does not include vectors $\boldsymbol{M}$ in $\mathcal{M}_j$ that are not malformed. Finally, we argue that also the last check holds with probability at least $1 - ne^{-\frac{r}{2}}$. To see this, note that by Lemma 6, P_j knows the unique tokens to *every* wellformed vector $\boldsymbol{M} \in \mathcal{T}$ at the end of Step 1 of Phase 2 with probability at least $1 - ne^{-\frac{r}{2}}$. Since these vectors contain the ciphertexts within $\mathcal{T}_{i,j}$, P_j knows the unique tokens to *every* ciphertext in $\mathcal{T}_{i,j}$ at the end of Step 1 of Phase 2 with probability at least $1 - ne^{-\frac{r}{2}}$. By the protocol specification, for all $c \in \mathcal{T}_{i,j}$, P_j includes $(\psi, c) \in \mathcal{S}_j$ s.t. $\mathsf{Dec}_{\mathsf{TL}}(c, s) \neq \perp$. □

Next, we analyze the distribution of $x^i_j := \tilde{\mathsf{G}}(x^i_{1,j}||...||x^i_{n,j})$ for some honest party P_i and $j \in \mathsf{trusted}$. In the following, without loss of generality we denote as $x_{1,j}, ..., x_{l,j}$ the values corresponding to the honest parties (since honest parties do not send conflicting values). We will omit the superscript i since any honest party P_i will use these values to compute its respective view of x^i_j. We will denote the adversary A's contribution to x^i_j as x^i_{A}, i.e., $x^i_j = \tilde{\mathsf{G}}(x^i_{1,j}||...||x^i_{l,j}||x^i_{\mathsf{A}}) = \tilde{\mathsf{G}}(x_{1,j}||...||x_{l,j}||x^i_{\mathsf{A}})$. We remark that the view of x_{A} may be different for two honest parties. We prove the following claim:

Lemma 8. *Let $j \in \mathsf{trusted}$, let the indices in $[l]$ correspond to honest parties, and let $\mathcal{U}$ be uniform over $[n^4]$. Then, for $k \in [l]$, no PPT algorithm A can distinguish $x_{k,j}$ from $\mathcal{U}$ at the end of Phase 1 of Π^{BCE}, except with negl. probability.*

Proof. Since $x_{1,j}, ..., x_{l,j}$ are chosen uniformly at random from this range, it remains to argue that given the view up until the end of Phase 1, A can not distinguish any of them from $\mathcal{U}$. Since Π^{BCE} (and thus Phase 1 of Π^{OLE}) runs in less than δ real time and we are in the O_R model, A can make at most $\pi - 1$ calls to O_R within this time span. Therefore, we can apply Lemma 1. Consider the following experiment: We sample $b \xleftarrow{\$} \{0,1\}$ and set $u_0 \xleftarrow{\$} [n^4]$, $u_1 = x_{k,j}$ for some $k \in [l]$. A is given $(u_b, \boldsymbol{C})$. Here, $\boldsymbol{C}$ denotes the vector of time-locked ciphertexts of honest parties computed in Phase 1 of Π^{OLE}. Applying a standard hybrid argument and using the fact that in this experiment, A may not choose what challenge message u_b is encrypted (as opposed to our security notion of **SemSec**), we see that A's advantage in distinguishing the two cases is negligible. □

Finally, we obtain from the previous lemma the following corollary which states that the values $x_j, j \in \mathsf{trusted}$ obtained by the honest parties at the end of the protocol are uniformly random in the interval $[n^4]$.

Corollary 1. *Let $j \in \mathsf{trusted}$, let the indices in $[l]$ correspond to the honest parties, and let $\mathcal{U}$ be uniformly distributed in $[n^4]$. Then no PPT algorithm A can distinguish $x_j^i = \tilde{\mathsf{G}}(x_{1,j}^i||...||x_{l,j}^i||x_{\mathsf{A}}^i) = \tilde{\mathsf{G}}(x_{1,j}||...||x_{l,j}||x_{\mathsf{A}}^i)$ from $\mathcal{U}$ at the end of Phase 1 of Π^{BCE}, except with negligible probability.*

Lemma 9. *Let P_i, P_j be honest parties and let $k \in \mathsf{trusted}$. Then $x_k = x_k^i = x_k^j \mod n^4$ and x_k is uniformly distributed in $[n^4]$.*

Proof. Let $k \in \mathsf{trusted}$. Using Property 2 from Lemma 4 and the graded consistency property of gradecast, for $l \in [n]$, P_i and P_j obtain the same set $\mathcal{S}_k$ of time-locked ciphertexts/decryption tokens. Therefore, it immediately follows that both parties decrypt to the same unique values and for all $l \in [n], x_{l,k}^i = x_{l,k}^j$. This means that we can omit the superscript and simply write $x_{l,k}$ and thus also x_k. It now follows from Corollary 1 that x_k is uniformly distributed in $[n^4]$. $\square$

Lemma 10. *Let $i = \min_{k \in \mathsf{trusted}} x_k = \tilde{\mathsf{G}}(x_{1,k}||...||x_{\widehat{n},k})$. Then, with probability at least $\frac{n-t}{n} - \frac{1}{n^2}$, P_i is honest (here t denotes the number of malicious parties).*

Proof. Due to Lemma 9, we know that $\forall k \in \mathsf{trusted}$, the value x_k is uniformly random in the range $[n^4]$. Now, assume that all coins $x_k, k \in \mathsf{trusted}$ are distinct. Because $|\mathsf{trusted}| \leq n$ and using the union bound, this assumption holds with probability at least $1 - \frac{1}{n^2}$. According to Lemma 7, there are $n - t$ values in $\mathsf{trusted}$ corresponding to honest parties. Thus the probability that the unique, random, and minimal value $x_i, i \in \mathsf{trusted}$, corresponds to an honest party P_i, is at least $\frac{n-t}{n}(1 - \frac{1}{n^2}) \geq \frac{n-t}{n} - \frac{1}{n^2}$. $\square$

With Lemmas 9-10, we get that at the end of Π^{BCE}, every honest party elects the same leader with probability $\geq \frac{n-t}{n} - \frac{1}{n^2}$. Hence, Π^{OLE} is an $\mathcal{O}(\log n)$-round OLE protocol with constant fairness that satisfies honesty and consistency.

Theorem 1. *If the majority of parties is honest, then Π^{OLE} is an $\mathcal{O}(r)$-round oblivious leader election protocol in the random oracle model with fairness $(1 - ne^{-\frac{r}{2}})(\frac{n-t}{n} - \frac{1}{n^2})$.*

A Quasi-Constant Round Protocol. In each round of Step 1 of Phase 2 of Π^{OLE}, P_i samples $\tilde{\boldsymbol{M}} \xleftarrow{\$} \tilde{\mathcal{T}}_i$ and computes $\mathsf{Open}_{\mathsf{TL}}(\tilde{\boldsymbol{M}}_1)$. One can easily improve the running time of the protocol by letting P_i only compute decryption tokens which *it does not already know*. Instead of sampling $\tilde{\boldsymbol{M}}$ uniformly at random, P_i could instead sample $\tilde{\boldsymbol{M}}$ from the set of vectors for which it has not already computed or received the openings. This saves much redundant work, since now the honest parties only compute new tokens instead of recomputing already available ones. Concretely, we reformulate Step 1 of Π^{BCE} as follows:

Quasi-Constant Round Version of Step 1:

P_i sets $\mathsf{Opened} = \{\}$. P_i repeats the following r times: It samples $\tilde{\boldsymbol{M}} \xleftarrow{\$} \tilde{\mathcal{T}}_i \setminus \mathsf{Opened}$, computes $\tilde{\psi} \leftarrow \mathsf{Open}_{\mathsf{TL}}(\tilde{\boldsymbol{M}}[1])$. If $\tilde{\boldsymbol{M}}$ is not malformed, it sends $(\tilde{\psi}, \tilde{\boldsymbol{M}})$ to every party (including itself). Let $(\tilde{\boldsymbol{M}}_1, \ldots, \tilde{\boldsymbol{M}}_\ell)$ be the set of wellformed vectors that P_i receives in this manner. P_i sets $\mathsf{Opened} = \mathsf{Opened} \cup \{\tilde{\boldsymbol{M}}_1, \ldots, \tilde{\boldsymbol{M}}_\ell\}$.

Let X_k be the number of decryption tokens which have not been computed by any honest party until the end of round k. We make the assumption that $\forall k : X_k = E[X_k]$,, i.e., in evthe above protocol is constant round for all practical purposes (see below). A similar heuristic has also been used in the analysis of Wagner's k-list algorithm [33].

Lemma 11. *If the majority of parties is honest, then under the assumption that* $\forall k : X_k = E[X_k]$ *the following holds:* $\forall k : X_k = E[X_k] \leq nl(k)$ *where* $l(k)$ *is defined for* $k \geq 1$ *as* $l(k) = \sqrt{e}^{-l(k-1)}$ *and* $l(1) = \sqrt{e}$.

Proof. (Sketch). As in the proof of Lemma 6, $X_1 = E[X_1] \leq n(1 - 1/n)^{n/2} \leq \frac{n}{\sqrt{e}}$. Since every party only decrypts ciphertexts from $\tilde{\boldsymbol{M}} \xleftarrow{\$} \tilde{\mathcal{T}}_i \setminus \mathsf{Opened}$ in the second round, where $|\tilde{\mathcal{T}}_i \setminus \mathsf{Opened}| = X_1 \leq \frac{n}{\sqrt{e}}$, the probability of a token ψ that opens $\boldsymbol{M} \in \tilde{\mathcal{T}}_i \setminus \mathsf{Opened}$ not being computed by any honest party P_i in the second round, is at most $(1 - \frac{1}{X_1})^{n/2}$. Thus, we can bound X_2 as $X_2 = E[X_2] \leq X_1(1 - \frac{1}{X_1})^{n/2} \leq \frac{n}{\sqrt{e}}(1 - \frac{\sqrt{e}}{n})^{n/2} \leq ne^{-\sqrt{e}/2} = n\sqrt{e}^{-\sqrt{e}}$. When repeating this argument, we get $X_3 \leq n(1 - \sqrt{e}^{\sqrt{e}}/n)^{n/2} \leq n\sqrt{e}^{-\sqrt{e}^{\sqrt{e}}}$. Iterating this argumentation over k rounds yields $X_k \leq nl(k)$. □

In other words, the above analysis gives us an r-round protocol where $r = \mathcal{O}(\log(\log(\cdots\log(n)\cdots)))$. Here, $\ldots$ stand for r iterative applications of log() to n. For all practical purposes, our protocol can be seen as being constant-round.

4.3 Achieving Consensus

So far we have seen how to construct a graded PKI (c.f. Lemma 2). From there we have shown how to elect a leader with Π^{OLE} which internally uses the gradecast and broadcast emulation protocols (c.f. Theorem 1). What remains to show is how to put these things together and achieve consensus using Π^{OLE}. Showing this last step requires us to slightly adapt the byzantine agreement protocol from [18] to work given only a GPKI rather than a PKI.

Lemma 12. *Let the number of corrupted parties be strictly less than* $\frac{n}{2}$. *If there exists a constant round gradecast protocol and an* $\mathcal{O}(r)$*-round OLE protocol with fairness* $\varepsilon = \mathcal{O}(1)$ *in the random oracle model, then there exists an* $\mathcal{O}(r)$*-round protocol for byzantine agreement.*

To construct a byzantine agreement protocol Π^{BA} from oblivious leader election we only add minor changes to the original description of [18], and defer the construction and proof to Appendix E. The following corollary summarizes the security properties of our final protocol Π^{BA}.

Corollary 2. *Suppose the adversary corrupts $< \frac{n}{2}$ parties. Then Π^{BA} achieves byzantine agreement in the random oracle model and has the following properties:*

- *Π^{BA} has an expected quasi-constant number of rounds (c.f. Lemma 11).*
- *Π^{BA} has a message complexity of $\mathcal{O}(n^2)$ and a bit complexity of $\mathcal{O}(n^3)$.*

From the close relation of byzantine agreement and broadcast protocols, it follows that Π^{BA} immediately gives rise to a broadcast protocol which runs in roughly the same time and is secure under the same assumptions (c.f. Section 2).

5 Broadcasting with a large number of parties

When n becomes large, consensus and broadcasting protocols suffer from increasing message and bit complexity. In this section, we propose an approach for efficiently running our broadcast protocol Π^{BC} when n becomes large. In contrast to the previous section, we now require that the adversary can only statically corrupt less than 1/3 of the computing power available in the system.

Informally, the idea of the protocol Π^{CP} is to divide the set of all players in a fixed number of k groups of size $\ell \ll n$, called cliques. This allows the parties to run protocols with a high message and bit complexity only *internally* in the cliques, while the overall, *global* communication between all parties is reduced. The cliques internally elect one member as its leader which will act as a relay for later broadcasts. We show that this protocol is secure as long as the adversary can corrupt strictly less than half of the leaders.

The first challenge of this protocol is to assign parties into cliques. We integrate this process in the Key Grading protocol Π^{KG}, which is run globally. Each party randomly assigns itself to one of the k cliques by including its clique identifier in the POW challenge. This commits all parties to their choice and prevents them from later changing it. After Π^{KG} terminates, a GPKI infrastructure has been created in which each public key is assigned to a clique identifier. Next, each clique internally elects a leader using the Π^{OLE} protocol. Note, that this step is the bottleneck of Π^{BC}, which is now only done internally within a clique (with $\ell \approx n/k$ parties) instead of running it globally (with n parties). The elected leaders can now each act as a relay in order to broadcast a message m for any party $P_i, i \in [n]$. Concretely, the party P_i sends m to all leaders of all cliques. The leaders will then agree on the message of every party P_i by running Π^{BC} (only among the leaders). After agreeing on m, every leader sends m to all parties in the system. If P_j receives the same message from at least $\frac{k}{2}$ leaders, it accepts m as P_i's message. It is easy to see that if at least $\frac{k}{2}$ of the leaders are honest, then all honest parties accept the same message which equals m if P_i was honest. We can only allow static corruption as otherwise an adversary could simply wait until the leaders are elected and then corrupt all of them.

To break the properties of the broadcast protocol an attacker needs to corrupt the majority (i.e., $\geq \lceil k/2 \rceil$) of the leaders. This is easy when it controls 1/3 or more of the parties. Approximately, all cliques will contain n/k honest parties. The attacker can force clique i to elect a malicious leader by overruling all honest votes. For this purpose, it needs to generate $t > n/k$ public keys (and corresponding proof of works) with clique identifier i. If the attacker controls 1/3 or more of the computational power, it can execute this attack in half of the cliques, thus forcing the election of $k/2$ corrupted leaders.

The main advantage of using the clique approach is that we are able to significantly decrease the communication complexity. Choosing the number of cliques as $k = \sqrt{n}$, each clique contains roughly $\sqrt{n}$ parties. This means that the leader election steps in all the cliques now result in communication complexities of only $O(n^{3/2})$. The same is true for the agreement steps among the leaders and so the overall communication complexity to broadcast a message is reduced from $O(n^3)$ to only $O(n^{3/2})$. On the downside of course, Π^{CP} only works in a more restricted attacker model. A proof for the following theorem including the specification of protocol Π^{CP} can be found in Section F.

Theorem 2. *Let n be the number of parties, $\alpha > 0, \beta > 0$, $k < n$ the number of cliques, $\epsilon > 2\sqrt{\frac{\ln(1/\alpha)}{k}}$ and $t = \frac{(1-\beta)(n-n\epsilon)}{3}$ the number of malicious parties, then Π^{CP} achieves broadcast with probability at least* $(1-\alpha)(1-k2e^{-(n-t)\beta^2/3k})$.

6 Discussion and Comparison

We compare our constant round protocols to the works of [2] and [19] which consider similar settings. [19] implicitly assume a random beacon as a form of trusted setup which makes their communication complexity independent of θ. More precisely, they use a random oracle which cannot be queried before the protocol has started. Moreover, their protocol requires knowledge of the exact number of parties running the protocol. Our work and [2] require only an upper bound on the number of participants. Both [2, 19] tolerate any number of corruptions, while our protocol requires an honest majority. While [2] has a linear running time in the number of parties, our protocol runs in expected constant number of rounds. We provide a more detailed comparison in Table 1. We use RB to indicate that there is *no* public random beacon, # denotes that knowledge of the number of participating parties is not required, and t denotes the maximal tolerable fraction of corrupted parties. Adp. denotes whether a protocol is secure against adaptive corruptions. $u := \log(\theta) + \kappa + \ell$ and $v = \log(n) + \kappa + |\sigma|$ where

Origin	RB	Adp.	#	t	Round Comp.	Msg. Comp.	Bit Comp.
[19]	–	✓	–	–	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2\ell)$
[2]	✓	✓	✓	–	$\mathcal{O}(n\kappa^2)$	$\mathcal{O}(n^2+\theta)$	$\mathcal{O}(n^2\kappa\log(\theta)+\theta u+n\ell)$
New (VSS)	✓	✓	✓	$\frac{1}{2}$	$\mathcal{O}(\kappa^2)$	$\mathcal{O}(n^4+\theta)$	$\mathcal{O}(n^5 v+\theta u+n\ell)$
Clique Version	✓	–	✓	$\frac{1}{3}$	$\mathcal{O}(\kappa^2)$	$\mathcal{O}(n^2+\theta)$	$\mathcal{O}(n^{\frac{5}{2}} v+\theta u+n\ell)$
New (TL Enc.)	✓	✓	✓	$\frac{1}{2}$	$\mathcal{O}(\kappa^2)$	$\mathcal{O}(n^2+\theta)$	$\mathcal{O}(n^3 v+\theta u+n\ell)$
Clique Version	✓	–	✓	$\frac{1}{3}$	$\mathcal{O}(\kappa^2)$	$\mathcal{O}(n+\theta)$	$\mathcal{O}(n^{\frac{3}{2}} v+\theta u+n\ell)$

Table 1. Comparison of round and communication complexities

κ denotes the length of the security parameter, ℓ denotes the length of the proof of work, $|\sigma|$ is the length of a signature and θ denotes the maximal number of messages that can be sent by all parties within Δ time.

7 Acknowledgements

Sebastian Faust and Lisa Eckey are partially funded by the Emmy Noether Program FA 1320/1-1 of the German Research Foundation (DFG).

References

1. Giulia Alberini, Tal Moran, and Alon Rosen. Public verification of private effort. Cryptology ePrint Archive, Report 2014/983, 2014. http://eprint.iacr.org/2014/983.
2. Marcin Andrychowicz and Stefan Dziembowski. PoW-based distributed cryptography with no trusted setup. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 379–399. Springer, Heidelberg, August 2015.
3. Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. Cryptology ePrint Archive, Report 2015/514, 2015. http://eprint.iacr.org/2015/514.
4. Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *ITCS 2016*, pages 345–356. ACM, January 2016.
5. John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 62–75. Springer, Heidelberg, August 2003.
6. Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source. Cryptology ePrint Archive, Report 2015/1015, 2015. http://eprint.iacr.org/2015/1015.
7. Gilad Bracha. An $O(\log n)$ expected rounds randomized byzantine generals protocol. *Journal of the ACM*, 34(4), 1987.
8. Benedikt Bünz, Steven Goldfeder, and Joseph Bonneau. Proofs-of-delay and randomness beacons in ethereum. Technical report, Princeton University and Stanford University, 2017.
9. Jeffrey Considine, Matthias Fitzi, Matthew K. Franklin, Leonid A. Levin, Ueli M. Maurer, and David Metcalf. Byzantine agreement given partial broadcast. *Journal of Cryptology*, 18(3):191–217, July 2005.
10. Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.
11. John R Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.
12. Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 139–147. Springer, Heidelberg, August 1993.
13. Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *20th ACM STOC*, pages 148–161. ACM Press, May 1988.

14. Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26, 1997.
15. Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, April 2015.
16. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. Cryptology ePrint Archive, Report 2017/454, 2017. http://eprint.iacr.org/2017/454.
17. Yves I. Jerschow and Martin Mauve. Modular square root puzzles: Design of non-parallelizable and non-interactive client puzzles. *Computers and Security*, 25, 2013.
18. Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 445–462. Springer, Heidelberg, August 2006.
19. Jonathan Katz, Andrew Miller, and Elaine Shi. Pseudonymous broadcast and secure computation from cryptographic puzzles. Cryptology ePrint Archive, Report 2014/857, 2014. http://eprint.iacr.org/2014/857.
20. Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.
21. Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.
22. Arjen K. Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Report 2015/366, 2015. http://eprint.iacr.org/2015/366.
23. Huijia Lin, Rafael Pass, and Pratik Soni. Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles. Cryptology ePrint Archive, Report 2017/273, 2017. http://eprint.iacr.org/2017/273.
24. Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 17–30. ACM Press, October 2016.
25. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
26. Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. Cryptology ePrint Archive, Report 2016/454, 2016. http://eprint.iacr.org/2016/454.
27. Rafael Pass and Elaine Shi. FruitChains: A fair blockchain. Cryptology ePrint Archive, Report 2016/916, 2016. http://eprint.iacr.org/2016/916.
28. Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. Cryptology ePrint Archive, Report 2016/917, 2016. http://eprint.iacr.org/2016/917.
29. Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.
30. Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, 1990.
31. Michael Szydlo. Merkle tree traversal in log space and time. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 541–554. Springer, Heidelberg, May 2004.
32. Sam Toueg. Randomized byzantine agreements. In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *3rd ACM PODC*, pages 163–178. ACM, August 1984.

33. David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, Heidelberg, August 2002.
34. Michael Waidner. *Byzantinische Verteilung ohne Kryptographische Annahmen trotz Beliebig Vieler Fehler (in German)*. PhD thesis, Karlsruhe, 1991.

Game $\mathbf{KDMSec}^{\mathsf{A}}(1^\kappa)$:	$\mathsf{O}_0(g, j)$	$\mathsf{O}_1(g, j)$
$\boldsymbol{K} = (K_0, ..., K_{n-1}) \overset{\$}{\leftarrow} \mathsf{Gen}_{\mathsf{KDM}}(1^\kappa)$	$c = \mathsf{Enc}_{\mathsf{KDM}}(g(\boldsymbol{K}), K_j)$	$c = \mathsf{Enc}_{\mathsf{KDM}}(0^{\lvert g(\boldsymbol{K})\rvert}, K_j)$
$b \overset{\$}{\leftarrow} \{0, 1\}$	Return c	Return c
$b' \leftarrow \mathsf{A}^{\mathsf{O}_b, \mathsf{O}_R, \tilde{H}}(1^\kappa)$		
Return $b' = b$		

Fig. 2. KDM Security Game in the random oracle model.

A Security of Our Time-Lock Encryption Scheme

In our time-locked encryption scheme presented in Section 2.4, we encrypt the trapdoor r, q s.t. $N = rq$ with the key $K = \tilde{H}(\psi)$, where $\psi \equiv_N a^{2^\pi}$. Thus, we must argue that our scheme remains secure when encrypting the values r and q with a key K that implicitly depends on them. To this end, we briefly recall the notion of *Key Dependent Message Security* (KDM Security for short), as introduced in [5]. In the KDM security game, the challenger first generates keys $\boldsymbol{K} = (K_0, ..., K_{n-1})$ using the key generation algorithm $\mathsf{Gen}_{\mathsf{KDM}}$. To formalize the KDM security game, we define two oracles $\mathsf{Real}^{\boldsymbol{K}, \tilde{H}}$ and $\mathsf{Fake}^{\boldsymbol{K}, \tilde{H}}$ which have access to $\boldsymbol{K}$ and $\tilde{H}$, and O_R. On input of (j, g) with $j \in [n]$ and where g is a fixed output-length function, $\mathsf{Real}^{\boldsymbol{K}, \tilde{H}}$ returns $\mathsf{Enc}_{\mathsf{KDM}}(g(\boldsymbol{K}), K_j)$ and $\mathsf{Fake}^{\boldsymbol{K}, \tilde{H}}$ returns $\mathsf{Enc}_{\mathsf{KDM}}(0^{\lvert g(\boldsymbol{K})\rvert}, K_j)$. Note that both encryptions are computed on strings of equal length (since g has a fixed-length output). g can be applied to $\boldsymbol{K}$ and can depend on $\tilde{H}$ in an arbitrary way. The adversary A gets oracle access to the random oracle $\tilde{H}$, O_R, and either the oracle $\mathsf{Real}^{\boldsymbol{K}, \tilde{H}}$ or $\mathsf{Fake}^{\boldsymbol{K}, \tilde{H}}$. It can issue queries of the form (j, g) to the oracle provided to it. The goal of the adversary is to distinguish whether it is interacting with $\mathsf{Real}^{\boldsymbol{K}, \tilde{H}}$ or $\mathsf{Fake}^{\boldsymbol{K}, \tilde{H}}$. We formalize the KDM security game in Figure 2. We refer to the oracle $\mathsf{Real}^{\boldsymbol{K}, \tilde{H}}$ as O_0 and to the oracle $\mathsf{Fake}^{\boldsymbol{K}, \tilde{H}}$ as O_1.

Definition 8. *(KDM Security). Let* $\mathsf{KDM} = (\mathsf{Gen}_{\mathsf{KDM}}, \mathsf{Enc}_{\mathsf{KDM}}, \mathsf{Dec}_{\mathsf{KDM}})$ *be a symmetric encryption scheme. The key generation algorithm* $\mathsf{Gen}_{\mathsf{KDM}}$ *on input* 1^κ *outputs a key* $K \overset{\$}{\leftarrow} \mathcal{K}$. *The encryption algorithm* $\mathsf{Enc}_{\mathsf{KDM}}$ *takes as input a message* $m \in \mathcal{M}$ *and a key* $K \in \mathcal{K}$ *and outputs a ciphertext* $c \in \mathcal{C}$. *The decryption algorithm* $\mathsf{Dec}_{\mathsf{KDM}}$ *takes as input a ciphertext* $c \in \mathcal{C}$ *and a key* $K \in \mathcal{K}$ *and outputs a message* $m \in \mathcal{M}$. *We require that the scheme be correct:* $\forall K \in \mathcal{K}, \forall m \in \mathcal{M}$: $\mathsf{Dec}_{\mathsf{KDM}}(\mathsf{Enc}_{\mathsf{KDM}}(m, K), K) = m$. *We say that* KDM *satisfies* KDM Security *if for all PPT adversaries* A *we have that* $\Pr[\mathbf{KDMSec}^{\mathsf{A}} = 1] = \mathsf{negl}(\kappa)$.

We begin by recalling the following lemma from [5]. It presents a simple symmetrical encryption scheme satisfying KDM security which we will use as a starting point in the security proof of our time-lock encryption scheme.

Lemma 13. *[5] Let* $\Pi^{\mathrm{KDM}} := (\mathsf{Gen}_{\mathsf{KDM}}, \mathsf{Enc}_{\mathsf{KDM}}, \mathsf{Dec}_{\mathsf{KDM}})$ *be defined as follows. Let* $\mathsf{Gen}_{\mathsf{KDM}}(1^\kappa)$ *be the algorithm that samples a uniform* $K \in \mathcal{K} = \mathcal{M} = \{0, 1\}^\kappa$,

```
Game $\mathbf{G}_0^{\mathsf{A}}(1^\kappa)$ :
00 $r_1, q_1, ..., r_n, q_n \xleftarrow{\$} \mathsf{PrimeGen}(1^\kappa)$
01 $\forall i : N_i = r_i q_i$
02 $\forall i : a_i \xleftarrow{\$} \mathbb{Z}_{N_i}$
03 $\forall i : \psi_i \equiv_{N_i} a_i^{2^\pi}$
04 $\forall i : K_i := \left(\tilde{H}(\psi_i), a_i, \pi, N_i\right)$
05 $\boldsymbol{K} = (K_0, ..., K_{n-1})$
06 $b \xleftarrow{\$} \{0,1\}$
07 $b' \leftarrow \mathsf{A}^{\mathsf{O}_b, \tilde{H}, \mathsf{O}_R}(1^\kappa)$
08 Return $b' = b$
```

```
$\mathsf{O}_0(g, j)$
09 $c = \mathsf{Enc}_{\mathsf{KDM}}(g(\boldsymbol{K}), K_j)$
10 Return $c$
```

```
$\mathsf{O}_1(g, j)$
11 $c = \mathsf{Enc}_{\mathsf{KDM}}(0^{|g(\boldsymbol{K})|}, K_j)$
12 Return $c$
```

Fig. 3. Game $\mathbf{G}_0$.

let $\mathsf{Enc}_{\mathsf{KDM}}$ *be defined as* $\mathsf{Enc}_{\mathsf{KDM}}(m, K) := R|| \left(\tilde{H}(K||R) \oplus m\right)$*, where* R *is a uniformly random string from* $\{0,1\}^\kappa$*. Let* $\mathsf{Dec}_{\mathsf{KDM}}$ *be a decryption algorithm defined as* $\mathsf{Dec}_{\mathsf{KDM}}(R'||C', K') := \tilde{H}(K'||R') \oplus C'$*. Then* Π^{KDM} *is KDM secure.*

We will consider the slightly altered scheme $\Pi_2^{\mathrm{KDM}} := (\mathsf{Gen}^2_{\mathsf{KDM}}, \mathsf{Enc}^2_{\mathsf{KDM}}, \mathsf{Dec}_{\mathsf{KDM}})$ in which $\mathsf{Gen}_{\mathsf{KDM}}$ and $\mathsf{Enc}_{\mathsf{KDM}}$ are replaced by the algorithm $\mathsf{Gen}^2_{\mathsf{KDM}}$ and $\mathsf{Enc}^2_{\mathsf{KDM}}$, which work as follows. $\mathsf{Gen}_{\mathsf{KDM}}$ samples primes $q, r \xleftarrow{\$} \mathsf{PrimeGen}$ and sets $N = rq, \varphi(N) = (r-1)(q-1), a \xleftarrow{\$} \mathbb{Z}_N, s \equiv_{\varphi(n)} 2^\pi, \psi \equiv_N a^s$. It then returns $K = (\tilde{H}(\psi), N, a)$. $\mathsf{Enc}^2_{\mathsf{KDM}}$ is defined as $\mathsf{Enc}_{\mathsf{KDM}}(m, K) := \left(R|| \left(\tilde{H}(K||R) \oplus m\right), a, \pi, N\right)$. In the following lemma, we prove the KDM security of Π_2^{KDM} with respect to an adversary who can make at most $\pi - 1$ calls to O_R.

Lemma 14. *Let* $\mathbf{KDMSec}_{\Pi_2^{\mathrm{TL}}}$ *be the KDM security game instantiated with* Π_2^{KDM}*. Then, for any PPT adversary* A *making at most* $\pi - 1$ *queries to* O_R *(either directly or indirectly) and at most* q *further queries (encryption queries, direct RO queries, and indirect RO queries),* $\Pr[\mathbf{KDMSec}^{\mathsf{A}}_{\Pi_2^{\mathrm{TL}}}(1^\kappa) = 1] = \mathsf{negl}(\kappa)$*, given that Conjecture 1 is true.*

Proof. Assume throughout the proof that A does not ask $\tilde{H}$ on the same query twice. Cleary, this is w.l.o.g., since there is no need to repeat a query to $\tilde{H}$ for which A already knows the answer. For the first part of the proof, let us assume that A does not query $\tilde{H}$ on any of the values ψ_i, either directly or via an encryption query. We prove the statement via a sequence of games. Let $\mathbf{G}_0 = \mathbf{KDMSec}_{\Pi_2^{\mathrm{TL}}}$. $\mathbf{G}_0$ is depicted in Figure 3.

We begin by introducing a second, auxiliary game $\mathbf{G}_1$ which is depicted in Figure 4. In $\mathbf{G}_1$, direct queries on a point s to $\tilde{H}$ are answered by setting $\tilde{H}(s) \xleftarrow{\$} \{0,1\}^\kappa$. On the other hand, calls to $\tilde{H}$ that result from calls to the oracles O_0 and O_1 are simulated by choosing $R, C \xleftarrow{\$} \{0,1\}^\kappa$ and then defining $\tilde{H}(s)$

```
Game $\mathbf{G}_1^{\mathsf{A}}(1^\kappa)$ :
    $bad \leftarrow \mathsf{false}$
    $\forall s \in \{0,1\}^* : \tilde{H}(s) := \perp$
    $\mathcal{X} \leftarrow \emptyset$
    $r_1, q_1, ..., r_n, q_n \xleftarrow{\$} \mathsf{PrimeGen}(1^\kappa)$
    $\forall i : N_i = r_i q_i$
    $\forall i : a_i \xleftarrow{\$} \mathbb{Z}_{N_i}$
    $\forall i : \psi_i \equiv_{N_i} a_i^{2^\pi}$
    $\forall i : K_i := \left(\tilde{H}(\psi_i), a_i, \pi, N_i\right)$
    $\boldsymbol{K} = (K_0, ..., K_{n-1})$
    $b \xleftarrow{\$} \{0,1\}$
    $b' \leftarrow \mathsf{A}^{\mathsf{O}_b, \tilde{H}, \mathsf{O}_R}(1^\kappa)$
    Return $b' = b$
```

```
$\tilde{H}(s)$ :
    If $s \in \mathcal{X}$ then
        $bad \leftarrow \mathsf{true}$
    If $\tilde{H}(s) = \perp$
        $\tilde{H}(s) \xleftarrow{\$} \{0,1\}^\kappa$
    Return $\tilde{H}(s)$
```

```
$\mathsf{O}_R(a, b, \mathsf{op}, N)$ :
    Return $a$ $\mathsf{op}$ $b$[a]
```

[a] $\mathsf{op} \in \{=, \times, +, -, \div\}$ is performed over the ring $\mathbb{Z}_N$

```
$\mathsf{O}'_0(g, j)$ :
    Compute $M \leftarrow g(\boldsymbol{K})$
    To do this, run $g$. When $g$ calls the RO on $s$ and $\tilde{H}(s) = \perp$, set $\tilde{H}(s) \xleftarrow{\$} \{0,1\}^\kappa$ and return $\tilde{H}(s)$ [a]
    $R \xleftarrow{\$} \{0,1\}^\kappa$
    $C \xleftarrow{\$} \{0,1\}^{|M|}$
    If $\tilde{H}(K_j || R) \neq \perp$
        $bad \leftarrow \mathsf{true}$
        $C \leftarrow \tilde{H}(K_j || R) \oplus M$
    $\tilde{H}(K_j || R) \leftarrow C \oplus M$
    $\mathcal{X} \leftarrow \mathcal{X} \cup \{K_j || R\}$
    Return $(R || C, a_j, \pi, N_j)$
```

[a] When g calls O_R on (a, b, op, N_j), reply with $\mathsf{O}_R(a, b, \mathsf{op}, N_j)$.

```
$\mathsf{O}'_1(g, j)$ :
    Compute $M \leftarrow g(\boldsymbol{K})$.
    $R \xleftarrow{\$} \{0,1\}^\kappa$
    $C \xleftarrow{\$} \{0,1\}^{|M|}$
    If $\tilde{H}(K_j || R) \neq \perp$
        $bad \leftarrow \mathsf{true}$
        $C \leftarrow \tilde{H}(K_j || R) \oplus 0^{|M|}$
    $\tilde{H}(K_j || R) \leftarrow C \oplus 0^{|M|}$
    $\mathcal{X} \leftarrow \mathcal{X} \cup \{K_j || R\}$
    Return $(R || C, a_j, \pi, N_j)$
```

Fig. 4. Game $\mathbf{G}_1$.

```
$\mathsf{O}''(g,j)$ :
37 Compute $M \leftarrow g(\boldsymbol{K})$
38 To do this, run $g$. When $g$ calls the RO on $s$ and $\tilde{H}(s) = \perp$, set $\tilde{H}(s) \xleftarrow{\$} \{0,1\}^{\kappa}$ and return $\tilde{H}(s)$
39 $R \xleftarrow{\$} \{0,1\}^{\kappa}$
40 $C \xleftarrow{\$} \{0,1\}^{|M|}$
41 If $\tilde{H}(K_j||R) \neq \perp$
42     $bad \leftarrow \mathsf{true}$
43 $\mathcal{X} \leftarrow \mathcal{X} \cup \{K_j||R\}$
44 Return $(R||C, a_j, \pi, N_j)$
```

Fig. 5. Oracle $\mathsf{O}''(g,j)$

accordingly. As both games return identically distributed values, it immediately follows that $\Pr[\mathbf{G}_0^{\mathsf{A}}(1^\kappa) = 1] = \Pr[\mathbf{G}_1^{\mathsf{A}}(1^\kappa) = 1]$.

Consider now the oracle procedure $\mathsf{O}''(\cdot)$ depicted in Figure 5. Denote with $\mathbf{G}_2$ the game resulting from $\mathbf{G}^1$, if $\mathsf{O}'_0(\cdot)$ is replaced with $\mathsf{O}''(\cdot)$. We argue that $|\Pr[\mathbf{G}_1^{\mathsf{A}}(1^\kappa) = 1] - \Pr[\mathbf{G}_2^{\mathsf{A}}(1^\kappa) = 1]| \leq \Pr[bad_2 \mid b = 0]$, where bad_2 denotes the event that bad becomes true at any point during game $\mathbf{G}_2^{\mathsf{A}}(1^\kappa)$. To this end, we first argue that bad is set with the same probability in both games. The claim then follows, because the games can clearly only differ if at some point in either game, bad becomes true. In order to obtain $\mathsf{O}''(\cdot)$ from $\mathsf{O}_0(\cdot)$, we have deleted the statements $C \leftarrow \tilde{H}(K_j||R) \oplus M$ and $\tilde{H}(K_j||R) \leftarrow C \oplus M$ from the code of $\mathsf{O}'_0(\cdot)$. Clearly, deleting the statement $C \leftarrow \tilde{H}(K_j||R) \oplus M$ does not change the probability that bad is set to true in game $\mathbf{G}_1$, since by the time that the statement is executed, bad has already been set. Similarly, deleting the statement $\tilde{H}(K_j||R) \leftarrow C \oplus M$ does not change the probability that bad is set in $\mathbf{G}_1$. To see why this is true, note that whenever this statement is executed in $\mathbf{G}_1$ then subsequently, the statement $\mathcal{X} \leftarrow \mathcal{X} \cup \{K_j||R\}$ is also executed. Now, either there is a subsequent call $\tilde{H}(K_j||R)$ or $\tilde{H}(K_j||R)$ is never queried for the remainder of $\mathbf{G}_1$. In the former case, $\mathbf{G}_1$ sets bad. In the latter case, bad is not set as a result of deleting the statement $\tilde{H}(K_j||R) \leftarrow C \oplus M$ in $\mathbf{G}_1$. We now define $\mathbf{G}_3$ as the game resulting from $\mathbf{G}_2$ when also $\mathsf{O}'_1(\cdot)$ is replaced with $\mathsf{O}''(\cdot)$. By the same argumentation as above, we have that $|\Pr[\mathbf{G}_2^{\mathsf{A}}(1^\kappa) = 1] - \Pr[\mathbf{G}_3^{\mathsf{A}}(1^\kappa) = 1]| \leq \Pr[bad_3 \mid b = 1]$, where bad_3 denotes the probability that bad gets set to true in $\mathbf{G}_3$. We finally define the game $\mathbf{G}_4$ in which there exists only a single copy of $\mathsf{O}''(\cdot)$ which is called regardless of the value of b. Clearly, $\Pr[\mathbf{G}_3^{\mathsf{A}}(1^\kappa) = 1] = \Pr[\mathbf{G}_4^{\mathsf{A}}(1^\kappa) = 1]$. Summing up, we have that $|\Pr[\mathbf{G}_1^{\mathsf{A}}(1^\kappa) = 1] - \Pr[\mathbf{G}_2^{\mathsf{A}}(1^\kappa) = 1]| \leq \frac{1}{2}(\Pr[bad_2 \mid b = 0] + \Pr[bad_3 \mid b = 0]) = \Pr[bad_4.]$ We argue that in $\mathbf{G}_4$, the probability that bad_4 is set in a query to $\mathsf{O}''(\cdot)$ is at most $q^2/2^\kappa$, since at all times $\tilde{H}$ is defined on at most q points and before testing whether $\tilde{H}(K_j||R) \neq \perp$, we sample $R \xleftarrow{\$} \{0,1\}^\kappa$. Similarly, the probability that bad_4 is set in a call to $\tilde{H}$ is at most $qn/2^\kappa \leq q^2/2^\kappa$. This follows since in $\mathbf{G}_4$, setting bad_4 to true while calling $\tilde{H}$ is only possible if one guesses correctly a value in the set $\mathcal{X}$. Since the values returned by both $\mathsf{O}''(\cdot)$ and $\tilde{H}$ are independent of $\boldsymbol{K}$ in $\mathbf{G}_4$, the values in $\mathcal{X}$ which are of the form $K_j||\cdot$ are uniformly random from A's perspective in $\mathbf{G}_4$. This follows from the assumption that A has not queried any of the values ψ_i to $\tilde{H}$ and thus $\boldsymbol{K}$ is a uniformly random vector from A's perspective.

Now assume instead that for some i, A calls $\tilde{H}(\psi_i)$ s.t. $\psi_i \equiv_{N_i} a_i^{2^\pi}$ at some point during the game $\mathbf{G}_0^{\mathsf{A}} = \mathbf{KDMSec}_{\Pi_2^{\mathrm{TL}}}^{\mathsf{A}}$, either directly or via an encryption query. W.l.o.g., assume that this is the first query of this form that A provokes in $\mathbf{G}_0^{\mathsf{A}}$. Then one can build an adversary B that breaks Conjecture 1. B simulates $\mathbf{G}_1$ to A. It begins by sampling $K'_0, ..., K'_{n-1}$ uniformly at random from $\{0,1\}^\kappa$ and $i \xleftarrow{\$} \{1, ..., q\}$. It samples $\forall j \in [n], i \neq j : r_i, q_i \mathsf{PrimeGen}(1^\kappa)$ and sets $N_j = r_j q_j, a_j \leftarrow \mathbb{Z}_{N_i}$. It sets $\boldsymbol{K}[j] = \big(K'_j, a_j, \pi, N_j\big)$ and sets $\boldsymbol{K}[i] = (K'_i, a, \pi, N)$, where a and N are the values B obtained from its own game. It answers all subsequent encryption queries by A of the form (g, j) by computing $g(\boldsymbol{K})$. On input a query s to $\tilde{H}$, B checks if $\tilde{H}(s)$ has previously been defined. If so, it returns $\tilde{H}(s)$. Otherwise, it sets $\tilde{H}(s) \xleftarrow{\$} \{0,1\}^\kappa$ and returns $\tilde{H}(s)$. When A asks a query of the form $\mathsf{O}_R(a, b, \mathsf{op}, N)$:, it answers the query by forwarding it to O_R in its own experiment. When $\tilde{H}$ is queried for the ith time on input ψ, then B aborts its simulation and outputs ψ as a solution. Since A has asked at most $\pi - 1$ queries to O_R, so has B. Moreover, it is easy to see that B's simulation of $\mathbf{G}_0$ is perfect for A (up to the point of aborting) from a real execution of $\mathbf{G}_1$ if B guesses correctly the value of i for which A first queries $\tilde{H}$ on ψ_i. Therefore, B returns the correct value $\psi \equiv_N \psi_i$, thus breaking Conjecture 1. □

A.1 Proof of Lemma 1

Note that in game **SemSec**, the adversary A essentially obtains weakened versions of the oracles $\mathsf{Real}^{\boldsymbol{K},\tilde{H}}$ or $\mathsf{Fake}^{\boldsymbol{K},\tilde{H}}$ seen in $\mathbf{KDMSec}_{\Pi_2^{\mathrm{TL}}}$: Namely, it obtains either

$$\mathsf{Enc}_{\mathsf{TL}}(\mathbf{0}) := \big(\mathsf{Enc}^2_{\mathsf{KDM}}(\tilde{\mathbf{0}}_i, K_0)\big)_{i\in[n]} = \Big(R_i||(\tilde{H}(K_0||R_i) \oplus \tilde{\mathbf{0}}_i), a_0, \pi, N_0\Big)_{i\in[n]}$$

or

$$\mathsf{Enc}_{\mathsf{TL}}(\boldsymbol{m}) = \big(\mathsf{Enc}^2_{\mathsf{KDM}}(\tilde{\boldsymbol{m}}_i, K_0)\big)_{i\in[n]} = \Big(R_i||(\tilde{H}(K_0||R_i) \oplus \tilde{\boldsymbol{m}}_i), a_0, \pi, N_0\Big)_{i\in[n]}$$

(for some random $R_i \in \{0,1\}^\kappa$), where $\tilde{\boldsymbol{m}}[i] = (m_i, r_0, q_0)$ and $\tilde{\mathbf{0}}$ is defined analogously. It may issue no further encryption queries. Furthermore, A is not allowed to choose what function of K_0 is encrypted in game **SemSec** nor can it see any ciphertexts that depend on $K_j, j \geq 1$. Using this intuition, we now prove Lemma 1.

Proof. Correctness and uniqueness of the scheme are easily verified. We prove that the scheme satisfies secrecy. We show that from any PPT adversary A that wins game **SemSec**, one can construct a PPT B against $\mathbf{KDMSec}_{\Pi_2^{\mathrm{TL}}}$ with the same advantage as A by simulating **SemSec** to A as follows. B runs A on input 1^κ. B answers all queries made by A to the oracles $\tilde{H}$ or O_R by forwarding them to its own version of these oracles and providing A with the answers that it obtains in this manner. When A outputs its challenge $\boldsymbol{m}$, B queries O_b on the tuple $(g_{N_0}, 0)$ where g_{N_0} is a function that maps $\boldsymbol{m}$ to the vector $(m_i, r_0, q_0)_i$. It obtains the answer $\boldsymbol{c} = (c_i, a_0, \pi, N_0)_i$ and returns it to A. When A returns a bit b', B returns b'. Clearly, B provides a prefect simulation of the game **SemSec** to A. Therefore, B's advantage in $\mathbf{KDMSec}_{\Pi_2^{\mathrm{TL}}}$ is equal to A's advantage in **SemSec**. Using Lemma 14, we obtain that A's advantage must be negligible. $\square$

B Details on the Graded Public Key Infrastructure

What remains to show for our protocol is why the protocol Π^{KG} which we construct in Chapter 3 achieves the properties of a graded public key infrastructure (GPKI) and that the protocol Π^{GC} is a gradecast protocol.

B.1 Proof of Lemma 2

In Lemma 2 we claim that protocol Π^{KG} achieves a graded PKI among n parties. We prove this lemma by showing that the adversary cannot create more public keys than the number of parties that it controls (sybil attack). To prevent a pre-computation attack by the adversary, every honest party needs to ensure, that the challenge used for the Proof of Work Phase is fresh. For this reason a valid Proof of Work ψ_k from party P_k is only accepted by an honest party P_i if a fresh challenge a_i^2 (generated in the Challenge Phase) was included in the puzzle. If this is the case, it assigns the grade 2 to P_k's public key. This resembles the highest confidence in trust. But since there might exist another honest party P_j who's challenge a_j^1 was not included in P_k's puzzle, P_i needs to guarantee that P_j accepts pk_k with grade 1 in this case. For this reason P_i's second challenge a_i^2 depends on all challenges P_i learned in the first round of the Challenge Phase. This also includes a_j^1. Therefore when P_j learns P_k's Proof of Work on challenge a_i^2, and receives a proof that a_i^2 includes a_j^1, then P_j knows that P_k's Proof of Work was created freshly. Nevertheless, since it did not include a_j^2, P_j only assigns grade 1 to pk_j.

Proof. Let P_i, P_j be honest parties. First we show that the protocol achieves Graded Validity, i.e., P_i's public key is accepted by P_j with grade 2. Following the protocol, P_i includes P_j's second challenge a_j^2 in A_i^2 in round 2 and thus includes it in M_i^2. In round 4, P_i sends pk_i along with a valid proof of work in form of a message $(\mathsf{Key}_2, \mathsf{pk}_i, a_i, \varphi_{i,j}^2, \psi_i)$ to P_j. Thus, every check in round 5 by P_j will pass on this message and P_j will accept P_i's key pk_i with grade 2. Next we show Graded Consistency. Suppose that P_i accepts a key pk_k with grade 2. Then it must have received a valid message $(\mathsf{Key}_2, \mathsf{pk}_k, a_k, \varphi_{k,i}^2, \psi_k)$ in round 4 by P_k. Valid means, that the proof of work ψ_k is correct and $\varphi_{k,i}^2$

proves that ψ_k depended on P_i's challenge a_i^2. Therefore, it will send message $(\mathsf{Key}_1, \mathsf{pk}_k, a_k, \varphi_{k,i}^2, a_i^2, \varphi_{i,j}^1, \psi_k)$ to P_j in round 5. It is clear that P_j will accept pk_k with grade (at least) 1, since the set A_i^1 of challenges on top of which the hash a_i^2 of P_i was computed, included P_j's challenge a_j^1 and P_i computed a correct proof $\varphi_{i,j}^1$ for this fact. Moreover, P_j can check just as P_i before that ψ_k depended on a_i^2 and $\mathsf{VerifyPow}((\mathsf{pk}_k, a_k), \psi_k) = 1$.

It remains to show that the number of identities is bounded by n, which follows directly from the fact that an adversary that controls less than half of the hashing power can generate strictly less than $\lceil \frac{n}{2} \rceil$ identities pk which are accepted by at least one honest party. Every honest party P_i will accept every pk that comes with a proof of work which includes a challenge a_i^1 or a_i^2 indirectly. From the fact that these challenges are chosen randomly by the honest parties, we know that the adversary cannot predict them and thus must invest some number computing steps to compute a proof of work on a given public key pk. Let $\mathcal{I}$ denote the set of public keys for which the adversary made at least π calls to the random oracle during the POW phase of the protocol. We follow the analysis of [2] to bound $|\mathcal{I}|$. By our definition of π-secure POW schemes, we know that any PPT prover succeeds only with negligible probability in computing identities outside of $\mathcal{I}$. Let $\delta = 4\kappa^2 \Delta$ be the time to compute a POW and $\delta' = \frac{\kappa \log(\delta\pi)}{\pi}$ be the POW verification time. Then the total time T of the protocol is computed as

$$T = (\delta + 2\Delta + 2(\Delta + \theta\delta')) = \delta(1 + 4\Delta/\delta + \theta\delta'/\delta) = \delta(1 + \epsilon(\kappa)).$$

From the values of δ and δ' it is clear that the term $\epsilon(\kappa) := 4\Delta/\delta + \theta\delta'/\delta$ vanishes for large enough κ. Note that we have $|\mathcal{I}|\pi \leq \frac{T}{\delta}\pi_{\mathcal{A}} = (1 + \epsilon(\kappa))\pi_{\mathcal{A}}$. This is true since the (minimal) number of random oracle calls $|\mathcal{I}|\pi$ that the adversary can use to compute keys within $\mathcal{I}$ is bounded from above by the total number of calls the adversary can make during time T. Since it can make at most $\pi_{\mathcal{A}}$ calls to the random oracle during any δ time steps, it can make at most $\frac{T}{\delta}\pi_{\mathcal{A}}$ calls to the random oracle in total over the duration of the protocol. Thus we have

$$|\mathcal{I}| \leq (1 + \epsilon(\kappa))\pi_{\mathcal{A}}/\pi.$$

Since $\epsilon(\kappa)$ vanishes for suitably large κ, we have that

$$|\mathcal{I}| \leq \pi_{\mathcal{A}}/\pi < \lceil \frac{n}{2} \rceil$$

and so the number of identities that the adversary can create is bounded in particular by $\lceil \frac{n}{2} \rceil - 1$. □

B.2 Proof of Lemma 3

In Lemma 3 we claim that given a graded PKI, Π^{GC} is a constant round protocol achieving graded broadcast if at least $\frac{n}{2}$ parties are honest. We will show this by proving that Π^{GC} satisfies graded validity and graded consistency.

First we show the graded validity property.

Proof. Let P_i and P_j be honest parties. Suppose P_D is honest. In round 2, all honest parties P_i received the same message/signature pair $m_i = m, \sigma_i = \sigma$ from P_D where $g_i(\mathsf{pk}_D) = 2$ by graded validity of the PKI. Additionally, since P_D is honest $\mathsf{Ver}(\mathsf{pk}_D, m, \sigma) = 1$. Therefore, P_i sends (m, σ) to every other party in round 2. Since signatures cannot be forged, no honest party could have received two different valid signatures (on distinct messages) under pk_D. In round 3, every honest party P_j sends message (m, σ'_j) where $\sigma'_j = \mathsf{Sign}(\mathsf{sk}_j, m)$. This means in round 4, P_i received $l \geq \frac{n}{2}$ message/signature tuples $(m, \sigma'_{1,i}), \ldots, (m, \sigma'_{l,i})$ from at least every honest party P_j with $g_i(\mathsf{pk}_j) = 2$. This means that P_i outputs m with grade 2.

Next, we show the graded consistency property. We first argue that P_i, P_j, either send the respective messages $(m^*, \sigma'_i), (m^*, \sigma'_j)$ in round 3, or at least one of them does not send a message, i.e., sets $m_j = \bot$ or $m_i = \bot$. To see this, we perform a case distinction over the possible misbehavior of the adversary.

– In round one, the adversary can send a message m with an invalid signature under pk_D to P_i (or with $g_i(\mathsf{pk}_D) = 0$). In this case, P_i sets $m_i = \bot$ in round 2 and thus sends no message in round 3.
– In round one or two, the adversary can send messages $m_i \neq m_j$ to parties P_i, P_j, respectively, with valid signatures under public key pk_D. If so, then at least one of these parties will set $m_i = \bot$ or $m_j = \bot$.

From this we argue that

$$\nexists\ \tilde{m} : g_i(\tilde{m}) = g_j(m^*) = 2 \land \tilde{m} \neq m^*.$$

This is true, since there cannot be a message $\tilde{m} \neq m^*$ in round 4, which is received by a P_i along with $\frac{n}{2}$ or more valid signatures under distinct public keys. This follows from the bounded number of identities property of the graded PKI and the fact that at least $\frac{n}{2}$ parties are honest. Therefore, there cannot be $\tilde{m} \neq m^*$ such that $g_i(\tilde{m}) = g_j(m^*) = 2$ in round 4. Now suppose P_i accepts m^* with grade 2 in round 4. This means that P_i received $l \geq \frac{n}{2}$ messages of the form $(m^*, \sigma'_{k,i}), k \in [n]$ from P_k with $g_i(\mathsf{pk}_k) = 2$ and $\mathsf{Ver}(\mathsf{pk}_k, m^*, \sigma'_{k,i}) = 1$. Then it sends $(m^*, \sigma^*_{1,i}, \mathsf{pk}_1, ..., \sigma^*_{l,i}, \mathsf{pk}_l)$ to P_j in round 4. Suppose P_j has not decided yet on an output. From the graded consistency of the graded PKI we know that $\forall r \in [l] : g_j(\mathsf{pk}_r) \geq 1$. Therefore, P_j outputs m^* with grade 1 in round 5. Note, that m^* is the only possible message that P_j accepts in round 5. This follows from the above equation and the fact that the adversary cannot come up with $\frac{n}{2}$ valid signatures on $\tilde{m} \neq m^*$. □

C Oblivious Leader Election via Verifiable Secret Sharing

In this section we show how to construct a protocol for Oblivious Leader Election using Gradecast (c.f. 3.2) and Verifiable Secret Sharing. For this we adapt the protocol of [18] which originally works in a setting with a public key infrastructure to work in a setting with a graded PKI.

C.1 Moderated Verifiable Secret Sharing

One important building block for this scheme is Verifiable Secret Sharing (VSS) in which a dealer (denoted as P_D) distributes shares of his secret s to all participating parties $P_1, \ldots, P_n$, whenever at most t parties are dishonest.

Definition 9. *(Verifiable Secret Sharing/VSS). Let Π^{VSS} be a 2-phase protocol run among n parties and some dealer P_D with input s. In the first phase, the* Sharing Phase*, shares of s are distributed to all parties and at the end of the* Reconstruction Phase *all parties P_i output a reconstructed secret s_i. Π^{VSS} is called* VSS secure for $t < n$ malicious parties *if the following conditions holds for all honest parties P_i, P_j:*

- ***Validity:*** *If P_D is honest, $s_i = s$.*
- ***Secrecy:*** *If P_D is honest during the Sharing Phase, then the joint view of all malicious parties is independent of s at the end of the Sharing Phase.*
- ***Reconstruction:*** *All honest parties P_i, P_j will output $s_i = s_j$.*

We require an adaption of the VSS scheme which is called *Moderated VSS* in which we have not just a designated dealer P_D, but also a party which acts as a moderator P_M and relays the messages of the dealer. Formally a moderated VSS scheme is defined as:

Definition 10. *(Moderated VSS) In a moderated VSS protocol Π^{mVSS} which is run among n parties and some dealer P_D with input s and a moderator P_M, each party P_i outputs a bit f_i at the end of the Sharing Phase and s_i at the end of the Reconstruction Phase. Let P_i, P_j be two honest parties. We say that Π^{mVSS} is* moderated VSS secure for $t < n$ malicious parties *if the following conditions hold whenever at most t parties are malicious.*

- ***Graded Sharing:*** *If P_M is honest, then P_i outputs $f_i = 1$.*
- ***Graded Validity:*** *If P_D is honest and P_i outputs $f_i = 1$, then P_j will output $s_j = s$.*
- ***Graded Secrecy:*** *If P_D is honest during the Sharing Phase and P_i outputs $f_i = 1$, then the joint view of all malicious parties is independent of s at the end of the Sharing Phase.*
- ***Graded Reconstruction:*** *If P_i outputs $f_i = 1$ then P_i, P_j output $s_i = s_j$.*

We use the moderated VSS in a setting where each honest party $P_i \in \{P_1, \ldots, P_n\}$ *in parallel* acts as dealer and distributes one distinct secret for every one of all $\widehat{n}$ parties known to P_i. To share these $\widehat{n}$ secrets, P_i will choose each known party (including itself) as the moderator exactly once; thereby sharing each of its secret using a different moderator. To allow this parallel invocation of n^2 VSS schemes, we will beforehand execute a few steps in the setup phase to assign moderators to secrets. Let $(\alpha_1, ..., \alpha_{\widehat{n}})$ denote some public evaluation points from a finite field $\mathbb{F}$ where λ is the size of a field element.

Setup protocol for moderated VSS $\Pi^{mVSS}_{\text{setup}}$

The protocol has 5 rounds, each lasting a time interval Δ. The message complexity is $\mathcal{O}(1)$, the bit complexity is $\mathcal{O}(n(|\sigma| + \lambda))$. Every honest party P_i runs $\Pi^{mVSS}_{\text{setup}}$ in parallel.

Step 1: Every honest party P_i sets $f_i = 1$

Step 2: P_i assigns a field element to every known public key and gradecasts the resulting tuples $m = (\alpha_1, \mathsf{pk}_1), ..., (\alpha_{\widehat{n}}, \mathsf{pk}_{\widehat{n}})$.

Step 3: Let m' be the message that P_i receives from P_j in the previous gradecast. P_i sets $f_j = 0$ if $g_i(m') < 2$, if m' contained more than n tuples or if P_i knows an identity of grade 2 which is not included in m'. Otherwise, we say that P_i *accepts the identities from message* m' for P_j. This set of identities is denoted as ID^j_i with size $n^i_j := |\text{ID}^i_j|$.

We will refer to the execution of the moderated VSS in which P_i is the dealer and P_j is the moderator, as $\Pi^{mVSS}_{i,j}$ or *run* (i, j) of the moderated VSS interchangeably.

Lemma 15. *If at least one honest party P_i has accepted ID^i_j in the above protocol, then every other honest party P_k obtained the same set of identities, i.e., $\text{ID}^k_j = \text{ID}^i_j$ and $n^i_j = n^k_j \leq n$ as well as the same set of evaluation points $\boldsymbol{\alpha}^i_j = \boldsymbol{\alpha}^k_j$. Furthermore, ID^k_j contains the identities of all honest parties and if P_j is honest, then every honest party accepts the message received by P_j.*

Proof. If P_k, has accepted ID^k_j, then $g_k = 2$ and $n^k_j \leq n$. This means that any honest party P_i has received the same set of identities, by the properties of gradecast, and in particular that $n^i_j \leq n$. Furthermore, all honest identities have grade 2 for P_k, so since it accepted ID^k_j, all of them must be included in this set, and therefore also in ID^i_j. Finally, note that if P_j is honest, all the conditions for acceptance are satisfied, because every identity of grade 2 from $P'_k s$ view has grade at least 1 for P_j and will thus be included in m_j. The statement follows in a similar way for the set of evaluation points, $\boldsymbol{\alpha}^l_j = \boldsymbol{\alpha}^k_j = \boldsymbol{\alpha}_j$. □

Lemma 15 allows us to omit the super index from the set of accepted identities for run (i, j) as long as an honest party has accepted this set (which is the only relevant case). We henceforth refer to the set of identities chosen by party P_j in the above protocol as ID_j and say that this set is *accepted*. This set allows all parties in the system to check which moderator was assigned to moderate every share.

Next, we show how adapt any existing VSS protocol Π^{VSS} as defined in [18] to meet the requirements for our protocol. Let Π^{VSS} be a constant round protocol that tolerates $t < \frac{n}{2}$ parties and that uses a broadcast channel only during the Sharing Phase. The number of broadcasts that any honest party has to perform in the worst case is $\mathcal{O}(1)$, the message complexity per party is $\mathcal{O}(n)$ and he bit complexity is $\mathcal{O}(n(\lambda + |\sigma|))$. We show how to replace some steps in Π^{VSS}

to arrive at $\mathit{\Pi}^{mVSS}_{i,j}$. Then the following protocol shows an emulation of $\mathit{\Pi}^{VSS}$ for dealer P_i and moderator P_j.

Moderated VSS Protocol $\mathit{\Pi}^{mVSS}_{D,M}$

Whenever some dealer P_D broadcasts a message m in $\mathit{\Pi}^{VSS}$, we perform the following 10-round subprotocol, each lasting a time interval $\mathit{\Delta}$. For every such broadcast we will use a moderator P_M which has been assigned in the setup phase.

Step 1: P_D gradecasts m.
Step 2: P_M gradecasts the message it received from P_D if its public key $\mathsf{pk}_D \in \mathrm{ID}_M$.
Step 3: Let m and m' be the output of party P_i from the two gradecasts, respectively, then P_i will consider m' as the broadcast message from P_D. It sets $f_i = 0$ if $g_i(m') < 2$, if both $m' \neq m$ and $g_i(m) = 2$ holds or if the moderator P_M has relayed a message from an identity outside of ID_i.

Whenever P_j sends a message m to P_i in $\mathit{\Pi}^{VSS}$, P_i accepts m iff $\mathsf{pk}_j \in \mathrm{ID}_i$ and it is sent along with a valid signature σ_j under pk_j.
Each party P_i outputs f_i at the end of the Sharing Phase. The message complexity is $\mathcal{O}(n^2)$. The bit complexity is $\mathcal{O}(n^2(\lambda + |\sigma|))$ per party.

Lemma 16. *Let ID_M be an accepted set of identities. Let $\mathit{\Pi}^{VSS}$ be a constant round secure VSS that relies on a PKI, uses a broadcast channel only in the Sharing Phase and tolerates $t < \frac{n}{2}$ malicious parties. Then there exists a constant round moderated VSS $\mathit{\Pi}^{mVSS}_{D,M}$ tolerating $t < \frac{n}{2}$ malicious parties in which each honest party P_k uses the set ID_j as its set of identities.*

Proof. Let $i, j \in [n]$ be fixed. Clearly, if $\mathit{\Pi}^{VSS}$ is a constant round protocol, then so is $\mathit{\Pi}^{mVSS}_{D,M}$. We proceed by showing that $\mathit{\Pi}^{mVSS}_{D,M}$ satisfies the properties of graded VSS.

Assume that P_M is honest and that P_i receives the gradecast message m from P_D in step 1 of the protocol. Let m' be the message that P_i receives from P_M in the first round of the gradecast (step 2). By the validity property of gradecast, $g_i(m') = 2$. Now, say that P_i outputs $g_i = 2$ in step 1. From the graded consistency property, we know that the moderator P_M receives the same message m as P_i in the first gradecast. Because P_M is honest, it will correctly moderate this exact message and we have $m' = m$. Therefore, P_i does not set $f_i = 0$ at the end of this invocation. Repeating this argument over all invocations of this protocol, $f_i = 1$ for any honest party P_i at the end of the Sharing Phase.

Now we consider the case where P_M might not be honest. We show that whenever a honest party P_i sets $f_i = 1$, then broadcast was achieved for this emulation and $\mathit{\Pi}^{mVSS}_{D,M}$ achieves all the guarantees of 'regular' VSS. If $f_i = 1$ at the end of an emulation, then $g_i(m') = 2$. From the graded consistency of gradecast we know that every honest party received m' and considers this message as the *broadcast message by P_D*. This proves the consistency property needed for broadcast. If additionally P_D is honest, then in step 1, every honest party P_i receives the same message m with $g_i(m) = 2$. Because $f_i = 1$, it must be that $m = m'$ and thus, the validity property of this broadcast is also met.

Consider now private messages sent between parties. As we argue about an accepted set of identities ID., each honest party accepts private messages from the same identities. Finally, as all honest identities are included in ID. and there can be at most t malicious parties in ID., secrecy is preserved just as in Π^{VSS} and the honest parties can execute the reconstruction phase as specified. This completes the proof. □

To make the n^2 runs of the moderated VSS run in parallel, assume that each party includes a session identifier in its messages containing the public key of the dealer and the moderator.

C.2 From Moderated VSS to Oblivious Leader Election

Now that we can utilize the Π^{mVSS} protocol defined above, we will show how to construct a leader election protocol from it.

Lemma 17. *Given a constant round secure moderated VSS protocol Π^{mVSS} tolerating at most $t < \frac{n}{2}$ malicious parties, then there exists an OLE protocol Π^{OLE} with fairness $\delta = \frac{n-t}{n} - \frac{1}{n^2}$ tolerating t malicious parties.*

Oblivious Leader Election Protocol Π^{OLE}

The protocol exhibits the same (constant) running time as Π^{mVSS} and has total message complexity $\mathcal{O}(n^3)$ per party. The total bit complexity per party is $\mathcal{O}(n^4(\log n + |\sigma|))$.

Proof. **Phase 1:** Each party P_i chooses random $c_{i,j} \in [\widehat{n}^4], j \in [\widehat{n}]$ and sets $\mathsf{trust}_{i,j} = 1, j \in [\widehat{n}]$. The parties now run the Sharing Phase of a moderated VSS with each party P_i using $c_{i,1}, ..., c_{i,\widehat{n}}$ as its secrets. To share $c_{i,j}$, P_i uses P_j as the moderator. If P_k outputs $f_k = 0$, it sets $\mathsf{trust}_{k,j} = 0$. Once the Sharing Phase is over, each party P_k sets $\mathsf{trust}_k := \{j : \mathsf{trust}_{k,j} = 1\}$.

Phase 2: The parties now run the Reconstruction Phase of the moderated VSS. Let $c^k_{i,j}$ denote $P_k's$ view of $c_{i,j}$ (if this lies outside of $[n^4]$, P_k uses a default value). Each party P_k sets $c^k_j := \sum_{i=1}^n c^k_{i,j} \pmod{n^4}$. It then outputs a $j \in \mathsf{trust}_k$ minimizing c^k_j.

Let **trusted**$:= \{k | \exists i : k \in \mathsf{trust}_i$ and P_i was honest at the end of phase 1$\}$. We prove that the above protocol satisfies consistency and validity. Clearly, if P_i was honest up to the end of Phase 1, then $i \in \mathsf{trusted}$. Also, if $k \in \mathsf{trusted}$, then, using the properties of moderated VSS we have for $1 \le l \le n$ that

$$c^i_{l,k} = c^j_{l,k},$$

for any honest P_i, P_j. This means that $c^i_k = c^j_k = c_k$. As

$$c_k = \sum_i^n c_{i,k} \pmod{n^4},$$

c_k for $k \in \mathsf{trusted}$ is uniformly distributed in $[n^4]$, since all honest parties P_i have picked $c_{i,k}$ uniformly at random from $[n^4]$. Now, assume that all the coins $c_k, k \in \mathsf{trusted}$ are distinct. As $|\mathsf{trusted}| \leq n$ and using the union bound, this is true with probability at least $1-\frac{1}{n^2}$. The are $n-t$ values in trusted corresponding to honest parties. Thus the probability that the unique, random, and minimal value $c_k, k \in \mathsf{trusted}$, corresponds to an honest party P_k, is at least $\frac{n-t}{n}$. Note that $k \in \mathsf{trust}_i$, whenever P_k is honest and thus the view among all honest parties on the elected leader will also be consistent with at least this probability. □

D Proof of Lemma 5

Proof. Consider the function $f(n) := (1 - \frac{1}{n})^n$ and its derivative

$$f'(n) = \frac{(1-\frac{1}{n})^n \cdot ((n-1) \cdot \ln \frac{n-1}{n} + 1)}{n-1}.$$

We will show that $f'(n) > 0$ for all $n > 2$ and thus $f(n)$ is monotonically increasing in this range. It is well known that $\lim_{n\to\infty} f(n) = \frac{1}{e}$. Since $f(2) = \frac{1}{4} < \frac{1}{e}$, it follows that $f(n)$ must be upper bounded by $\frac{1}{e}$ for all $n > 2$.

Since $\frac{(1-\frac{1}{n})^n}{n-1} > 0$ for all $n > 1$, it is enough to show that $g(n) := \ln(\frac{n-1}{n})(n-1) \geq -1$ for all $n > 2$. The Taylor Series expansion yields $g(n) = \ln(\frac{n-1}{n})(n-1) = -\left(\frac{1}{n} + \frac{1}{2n^2} + \mathcal{O}(\frac{1}{n^3})\right)(n-1)$. Taking the limit of n to infinity, we obtain -1. We show that $g(n)$ is monotonically decreasing for all $n > 2$. Since $g(2) < 0$, it then follows that $g(n)$ must be lower bounded by -1. Thus, we consider $g'(n) = \left(\ln(\frac{n-1}{n})(n-1)\right)' = \frac{1}{n}+\ln(\frac{n-1}{n})$ and $g''(n) = \frac{-1}{n^2}+\frac{1}{n-n^2}$. Since $g''(n) < 0$ for $n > 2$, $g'(n)$ is monotonically decreasing for $n > 2$. Furthermore, $g'(2) = \frac{1}{2} + \ln(\frac{1}{2}) < 0$ and so for all $n > 2$, $g'(n) < 0$. This proves that also $g(n)$ is monotonically decreasing in this range. □

E From Leader Election to Byzantine Agreement and Broadcast

We will show this by proving lemma 12, which states that an OLE protocol can be used to construct a constant round protocol for byzantine agreement and thus broadcast. To prove this statement, we will proceed similar to the authors of [18] who show that in a PKI setting OLE can be used to achieve byzantine agreement, which itself implies broadcast.

Proof. We describe a protocol similar to [18] which achieves byzantine agreement for messages of message space $\mathcal{M}$ using internally a OLE protocol tolerating t malicious parties. Let $\mathcal{M}$ be the set of possible input values and $\perp$. Let further $\mathsf{lock}_i := \infty$ for each P_i and let $m_i \in \mathcal{M}$ be $P_i's$ input. The parties execute the following round in parallel.

Byzantine Agreement Protocol Π^{BA}

Step 1: Each party P_i gradecasts m_i. Let $(m_{j,i}, g_{j,i})$ be the message/grade that P_i received from P_j.
Step 2: For all m that it received P_i sets $\mathcal{S}_i^m := \{j : m_{j,i} = m \wedge g_{j,i} = 2\}$ and $\tilde{\mathcal{S}}_i^m := \{j : m_{j,i} = m \wedge g_{j,i} \geq 1\}$. If $\mathsf{lock}_i := \infty$, then set $m_i := m$ if there exists m s.t. $|\tilde{\mathcal{S}}_i^m| \geq \frac{n}{2}$. Otherwise, set $m_i := \Phi$. If $|\mathcal{S}_i^m| \geq \frac{n}{2}$, set $\mathsf{lock}_i := 1$.
Step 3: Each party P_i gradecasts m_i. Let $(m_{j,i}, g_{j,i})$ be the message/grade that P_i received from P_j.
Step 4: For all m that it received P_i sets $\mathcal{S}_i^m := \{j : m_{j,i} = m \wedge g_{j,i} = 2\}$ and $\tilde{\mathcal{S}}_i^m := \{j : m_{j,i} = m \wedge g_{j,i} \geq 1\}$. If $\mathsf{lock}_i := \infty$, then set $m_i := m$ if there exists m s.t. $|\tilde{\mathcal{S}}_i^m| \geq \frac{n}{2}$. P_i sends m_i to all parties. $m_{j,i}$ be the message that party P_i receives from P_j. A message is only accetped if it comes from an identity with grade at least one.
Step 5: The parties run the OLE protocol. Let ℓ_i be the leader as chosen by P_i.
Step 6: If $\mathsf{lock}_i := \infty$ and $|\tilde{\mathcal{S}}_i^{m_i}| \leq \frac{n}{2}$, then P_i sets $m_i := m_{\ell_i,i}$.
Step 7: If $\mathsf{lock}_i = 0$, P_i outputs m_i and terminates. If $\mathsf{lock}_i := 1$, then P_i sets $\mathsf{lock}_i := 0$ and goes to step 1. If $\mathsf{lock}_i := \infty$, P_i goes to step 1.

Let P_i be an honest party and let us refer to an execution of the above seven-round protocol as an *iteration*. It is easy to see that lock_i is a monotonically decreasing value. Moreover, once P_i has set $\mathsf{lock}_i \neq \infty$, it terminates the protocol by the end of the next iteration and the value which was held by m_i at the point in time where $\mathsf{lock}_i \neq \infty$ was set, remains unchanged until the protocol terminates. Suppose that immediately prior to a parallel iteration, each honest party P_i holds the same value $m_i = m$. Then, each party gradecasts m in Round 1. This means that $|\tilde{\mathcal{S}}_i^m| \geq |\mathcal{S}_i^m| \geq \frac{n}{2}$ in Round 2 and therefore each honest P_i sets $\mathsf{lock}_i = 1$ (unless of course, it has already set $\mathsf{lock}_i = 0$) and $m_i = m$. By the above argumentation, each honest party terminates outputs m by the end of the following iteration at the latest.

Next, we argue that if an honest P_i sets $\mathsf{lock}_i = 1$ in Round 2 of iteration I, then each honest party P_j terminates at the end of iteration $I + 2$, at the latest, with output $m_j = m_i = m$. Consider the first iteration I in which some honest P_i sets $\mathsf{lock}_i = 1$ in Round 2. This means that $|\tilde{\mathcal{S}}_i^{m_i}| \geq |\mathcal{S}_i^{m_i}| \geq \frac{n}{2}$. By the properties of gradecast, $\mathcal{S}_i^{m_i} \subseteq \tilde{\mathcal{S}}_j^{m_j}$, and therefore also $|\tilde{\mathcal{S}}_j^{m_j}| \geq \frac{n}{2}$, so $m_j = m$. Therefore, every honest P_j gradecasts m in Round 3 and in Round 4, each party receives m at least $\frac{n}{2}$ and so sets $m_j = m$ in Round 4. Also, because I is the first such iteration, no honest party will terminate at the end of I. Observe now that each honest party holds the same value $m_i = m$ immediately prior to iteration $I + 1$. By what we have argued above, each honest party will terminate at the latest at the end of iteration $I + 2$.

Now, consider the scenario where an honest leader P_ℓ is elected an each honest P_i has still set $\mathsf{lock}_i = \infty$ in some iteration I. If all honest parties have $|\mathcal{S}_i^m| \leq \frac{n}{2}$ in step 6, then each party P_i holds the same value $m_i = m_{\ell_i,i} = m_{\ell,i} = m_\ell = m$ at the end of iteration I, because P_ℓ was honest in Round 4. So suppose instead that for at least one honest P_i, $|\mathcal{S}_i^{m_i}| > \frac{n}{2}$ in step 6 and let P_j be another honest party. Either $|\mathcal{S}_j^{m_j}| > \frac{n}{2}$ or $|\mathcal{S}_j^{m_j}| \leq \frac{n}{2}$. In the former case, $\mathcal{S}_j^{m_j} \cap \mathcal{S}_i^{m_i} \neq \emptyset$ and thus by the properties of gradecast, $m_i = m_j$ (no honest party may gradecast distinct messages to parties P_i, P_j both with grade 2). In the latter case, $|\mathcal{S}_j^{m_j}| \leq \frac{n}{2}$, and P_j sets $m_j := m_{\ell,j}$. Since P_ℓ was honest in Round 4 and because

$$\mathcal{S}_i^{m_i} \subseteq \tilde{\mathcal{S}}_\ell^{m_i} \implies \tilde{\mathcal{S}}_\ell^{m_i} > \frac{n}{2},$$

P_ℓ set $m_\ell := m_i$ in Round 4. For this reason, P_j sets $m_j := m_{\ell,j} = m_\ell = m_i$. Thus, by the end of iteration I, each honest party P_i has set $m_i = m_\ell = m$ and by the above argumentation, this concludes the proof. □

F Formal Discussion of Clique Protocol

We show in this section how to build the clique-based protocol described informally in the main body of the paper.

F.1 The Clique Protocol Π^{CP}

Let Π^{OLE} be a an oblivious leader protocol Π^{BC} be a broadcast protocol.

Protocol Π^{CP}

Setup Phase

This phase consists of 3 rounds and lasts time $2\Delta + \delta$ with message complexity $\mathcal{O}(\theta)$ and bit complexity $\mathcal{O}(\theta\kappa)$. All parties run the Challenge and Proof of Work Phase of Π^{KG} with a slight modification in the Proof of Work Phase: Every party P_i samples a clique identifier $\Gamma_i \leftarrow [k]$ and adds it to the puzzle p_i.

Clique Phase

This phase lasts time $3\Delta + t_{\mathrm{OLE}}$ with message complexity $\mathcal{O}(\theta + \mathsf{msg}_{\mathrm{OLE}})$ and bit complexity $\mathcal{O}(\theta(\log(\theta) + \kappa + \ell) + \mathsf{bit}_{\mathrm{OLE}})$.

Key Ranking All parties run the Key Ranking Phase of Π^{KG} but append Γ_i in the first message of this phase. The rest of Π^{KG} is executed between parties with the same identifier Γ_i.

Elect Leader All parties within one clique run Π^{OLE} and elect one member P_i^L per clique Γ_i. Then, all parties in the clique sign pk_i^L and send it all other parties.

A party accepts P_i^L as leader for Γ_i if it received valid signatures from the majority of Γ_i's members on pk_i^L and if P_i^L committed to Γ_i with a correct Proof of Work.

Broadcast Phase

This phase lasts time $2\Delta + t_{\mathrm{BC}}$ with message complexity $\mathcal{O}(n + \mathsf{msg}_{\mathrm{BC}})$ and bit complexity $\mathcal{O}(n(\kappa + |\sigma| + \lambda) + \mathsf{bit}_{\mathrm{BC}})$ for a leader.

– If P_j wants to broadcast message $m \in \{0,1\}^\lambda$ it sends $(\mathsf{pk}_j, m, \sigma)$ to all leaders, where pk_j is its own public key and $\sigma = \mathsf{Sign}(\mathsf{sk}j, m)$.
– Upon receiving message $(\mathsf{pk}_j, m_j, \sigma_j)$ every leader $P_i^L, i \in [k]$ checks if $\mathsf{Ver}(\mathsf{pk}_j, m_j, \sigma_j) = 1$ and $g_i^L(\mathsf{pk}_j) \geq 1$. If these checks pass, it broadcasts $(\mathsf{pk}_j, m_j, \sigma_j)$ to all other leaders.
– For every distinct message $(\mathsf{pk}_j, m_j', \sigma')$ that an honest leader $P_i^L, i \in [k]$ received at in this phase with $\mathsf{Ver}(\mathsf{pk}_j, m_j', \sigma_j') = 1$ and $g_i^L(\mathsf{pk}_j) \geq 1$ it checks, if it also received another message $(\mathsf{pk}_j, m_j'', \sigma_j'')$ such that $m_j'' \neq m_j'$, $\mathsf{Ver}(\mathsf{pk}_j, m_j', \sigma_j') = 1$ and $\mathsf{Ver}(\mathsf{pk}_j, m_j'', \sigma_j'') = 1$. If this is the case, P_i^L does nothing. Otherwise, it creates a signature $\sigma_i^L = \mathsf{Sign}(\mathsf{sk}i^L, m_j')$ and sends $(\mathsf{pk}_j, m_j', \sigma_i^L)$ to every party.

Every party P_ℓ that received a message $(\mathsf{pk}_j, m^*, \sigma_{i,j}^L)$ from at least $\frac{k}{2}$ distinct leaders P_i^L (that P_ℓ accepted at the end of the Clique Phase) where $\mathsf{Ver}(\mathsf{pk}_i^L, m*, \sigma_{i,j}^L) = 1$, outputs m^* as the message of pk_j.

We describe the above protocol built on a Π^{OLE} with run time t_{OLE}, message complexity $\mathsf{msg}_{\mathrm{OLE}}$, and bit complexity $\mathsf{bit}_{\mathrm{OLE}}$ when run in the largest clique. Let Π^{BC} have run time t_{BC}, message complexity $\mathsf{msg}_{\mathrm{BC}}$, and bit complexity $\mathsf{bit}_{\mathrm{BC}}$ for messages of length λ when run with k players. The total message complexity of the protocol is $\mathcal{O}(\theta + \mathsf{msg}_{\mathrm{OLE}} + n + \mathsf{msg}_{\mathrm{BC}})$ and the total bit complexity of the protocol is $\mathcal{O}(\theta(\log(\theta) + \kappa + \ell) + \mathsf{bit}_{\mathrm{OLE}} + n(\kappa + |\sigma| + \lambda) + \mathsf{bit}_{\mathrm{BC}})$ per party. The total round complexity is $7\Delta + t_{\mathrm{OLE}} + t_{\mathrm{BC}}$.

F.2 Proof for Theorem 2

Theorem 2 states that the above protocol Π^{CP} achieves broadcast. Let n be the number of parties, t the number of malicious parties, k the number of cliques and $\beta > 0$. We structure our proof in the following three statements (Lemma 18, 19, 21).

Lemma 18. *With probability at least* $k2e^{-(n-t)\frac{\beta^2}{3k}}$, *each clique* C_j *contains* $X^j \in [\mu(1-\beta), \mu(1+\beta)]$ *honest parties.*

Proof. For this proof we will need the Chernoff bound:
Chernoff's Inequality. Let $X_1, ..., X_n$ be i.i.d. Bernoulli variables with probability parameter p and let $X = \sum_i X_i, \beta \in [0,1]$. Then, $\Pr[|X - pn| \geq \beta] \leq 2e^{-\frac{\beta^2}{3}pn}$.

Since the honest parties choose their clique membership at random, $\mu = \frac{n-t}{k}$ is the expected number of honest parties in every clique. Let X_i^j be the indicator variable that equals one whenever party P_i chooses clique C_j and zero otherwise. Clearly, for any honest party P_i, $E[X_i^j] = \Pr[X_i^j = 1] = \frac{1}{k}$. Now, let $X^j = \sum_i X_i^j$. By linearity of expectation $\forall j : \mu = \mu^j := E[X^j] = \frac{n-t}{k}$. By the Chernoff bound, for any $\beta > 0$ we have that

$$\forall j : \Pr[|X^j - \mu| \geq \beta\mu] = \Pr[|X^j - \mu| \geq (n-t)\frac{\beta}{k}] \leq 2e^{-(n-t)\frac{\beta^2}{3k}}.$$

The probability that $\exists j : |X^j - \mu| \geq (n-t)\beta$ is thus bounded by $k2e^{-(n-t)\frac{\beta^2}{3k}}$. $\square$

Some simplifying assumptions. Using Lemma 18, we make some mild assumptions that will greatly simplify the rest of the argumentation for this chapter. First, we assume that each clique contains the same number $\mu' = (1-\beta)\mu$ of honest parties. Clearly, this is a lower bound on the number of honest parties in each clique, as was shown in Lemma 18 to be true with high probability. Additionally, we assume that the adversary's computational power is sufficient to take over *exactly* s cliques. It is not hard to generalize the argumentation to the case where the adversary can additionally corrupt a fraction of another clique, but we feel that this would deter from the main arguments in the proof below. For an example of such a situation, see Figure F.2. Note that in this example, any $s \geq 4$ would already be enough to take over 4 cliques and therefore the majority of the elected leaders. Lastly, let us assume that the number μ' of honest parties in every clique is sufficiently large such that $\left(\frac{1}{\mu'}\right)^2 \approx 0$. This assumption is also without much loss of generality, since for this entire section we assume that the number of parties is very large. The fairness of the Π^{OLE} protocol then becomes $\frac{\mu'-t'}{\mu'}$ rather than $\frac{\mu'-t'}{\mu'} - \frac{1}{(\mu'+t')^2}$ for any number $t' > 0$ of corrupted parties within some clique, where for simplicity, we also assume that the parties run Π^{OLE} for sufficiently many rounds r to ensure that $1 - e^{r/2} \approx 1$.

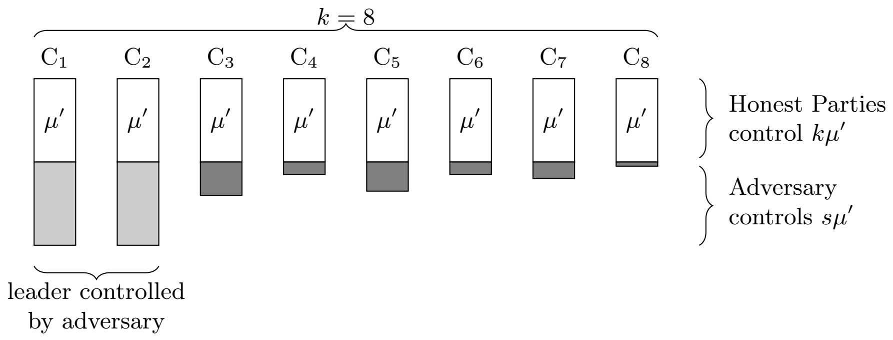
$k = 8$
C1
C2
C3
C4
C5
C6
C7
C8
μ'
Honest Parties control $k\mu'$
Adversary controls $s\mu'$
leader controlled by adversary

Fig. 6. Proof model for $n = \mu'(k+s) = 11\mu'$ parties, where $s\mu'$ parties are controlled by the adversary and $k\mu'$ parties are honest.

Let us provide some intuition on the number of corrupted parties that we can possibly tolerate. As corrupting $\frac{k}{2}$ of the cliques would require

$$\frac{k}{2}(1-\beta)\mu = \frac{k(n-t)}{2k}(1-\beta) = \frac{n-t}{2}(1-\beta)$$

corrupted parties, i.e., exactly half as many corrupt parties as there are honest parties, we require that the adversary corrupts strictly less than $\frac{1-\beta}{3}n$ parties. We prove that under these assumptions, the majority of the cliques will elect an honest leader with high probability.

Lemma 19. *Let $s < \frac{k}{2}$ and let $\mu' s$ be the number of parties controlled by the adversary. With probability at least $1 - e^{-\frac{2w^2}{k}}$, the number of dishonest leaders which are elected is at most $s + w$.*

Proof. Let $s = \tilde{s} + \widehat{s}$. Suppose that the adversary corrupts exactly μ' parties in $\tilde{s}$ cliques and spreads the remaining $\mu'\widehat{s}$ parties which it controls arbitrarily among the the remaining $l = k - \tilde{s}$ cliques. Let $X_i, \in [l]$ be the indicator variable that is 1 iff a malicious leader is elected in clique C_i and let $X := \sum_i X_i$. As none of the resulting l cliques elects an honest leader with probability 0, clique C_i elects a dishonest leader with probability

$$\Pr[X_i = 1] = \frac{k_i}{\mu' + k_i},$$

where $k_i, i \in [l]$ is the number of corrupted parties that C_i contains. We first show that the expected number $E[X] = \sum_i E[X_i]$ of cliques in which the adversary elects a leader using the $\mu'\widehat{s}$ parties is at most $\widehat{s}$. Clearly,

$$E[X] = \sum_{i=1}^{l} \frac{k_i}{\mu' + k_i}.$$

Using the method of Lagrange multipliers we maximize this sum via the expression

$$\sum_{i=1}^{l} \frac{k_i}{\mu' + k_i} - \lambda(\widehat{s}\mu' - \sum_i k_i).$$

Calculating the derivatives, we obtain the equations

$$\forall i \in [l] : -\frac{\mu'}{(\mu + k_i)^2} \Leftrightarrow k_i = \mu' \pm \sqrt{\frac{\mu'}{\lambda}}$$

and $\widehat{s}\mu' = \sum_i k_i$. Combining this yields Let $s = \tilde{s} + \widehat{s}$. Suppose that the adversary corrupts exactly μ' parties in $\tilde{s}$ cliques and spreads the remaining $\mu'\widehat{s}$ parties which it controls arbitrarily among the the remaining $l = k - \tilde{s}$ cliques. Let $X_i, \in [l]$ be the indicator variable that is 1 iff a malicious leader is elected in clique C_i and let $X := \sum_i X_i$. Clique C_i elects a dishonest leader with probability $\Pr[X_i = 1] = \frac{k_i}{\mu' + k_i}$, where $k_i, i \in [l]$ is the number of corrupted parties that C_i

contains. We first show that the expected number $E[X] = \sum_i E[X_i]$ of cliques in which the adversary elects a leader using the $\mu'\widehat{s}$ parties is at most $\widehat{s}$. Clearly, $E[X] = \sum_{i=1}^{l} \frac{k_i}{\mu'+k_i}$. It is well known that this sum is maximized when $k_1 = ... = k_l = \frac{\widehat{s}\mu'}{l}$. In other words, the best strategy to maximize this expectation is to distribute the $\mu'\widehat{s}$ parties as evenly as possible among the l cliques. The sum then becomes

$$\widehat{s}\mu' = l(\mu' \pm \sqrt{\frac{\mu'}{\lambda}}) \Leftrightarrow \widehat{s} = l(1 \pm \sqrt{\frac{1}{\mu'\lambda}}) \tag{1}$$

$$\Leftrightarrow \lambda = (\frac{\widehat{s}}{l} - 1)^{-2}\frac{1}{\mu'} \tag{2}$$

and thus $\forall i \in [l] : k_i = \sqrt{(\mu'(\frac{\widehat{s}}{l} - 1))^2} + \mu' = \mu'(\frac{\widehat{s}}{l} - 1) + \mu' = \frac{\widehat{s}\mu'}{l}$. $E[X]$ is maximal when the $\mu'\widehat{s}$ parties are distributed evenly among the l cliques. The sum then becomes

$$\frac{l\frac{\widehat{s}\mu'}{l}}{\mu'(1+\frac{\widehat{s}}{l})} = \frac{l\frac{\widehat{s}}{l}}{(1+\frac{\widehat{s}}{l})} = \frac{l\frac{\widehat{s}}{l}}{(\frac{\widehat{s}+l}{l})} = \frac{l\widehat{s}}{\widehat{s}+l} \tag{3}$$

Thus, the following statement holds $E[X] \leq \frac{l\widehat{s}}{\widehat{s}+l} \leq \widehat{s}$ which can be seen as follows.

$$\frac{l\widehat{s}}{\widehat{s}+l} - \widehat{s} = \frac{l\widehat{s} - \widehat{s}^2 - \widehat{s}l}{\widehat{s}+l} - \widehat{s} = -\frac{\widehat{s}^2}{\widehat{s}+l} \leq 0, \forall \widehat{s} > 0. \tag{4}$$

We remind the reader of the Hoeffing bound:

Lemma 20. *(Hoeffding's Inequality). Let $X_1, \ldots, X_n$ be independent Bernoulli variables and let $X = \sum_i X_i, t > 0$. Then,* $\Pr[X - \mathbb{E}[X] \geq t] \leq e^{-2\frac{t^2}{n}}$.

As the variables $X_i, i \in [l]$ are independent, we apply the Hoeffding bound to obtain

$$\Pr[E[X] - X \geq w] \leq e^{-\frac{w^2}{l}} \leq e^{-\frac{w^2}{k}},$$

i.e., with probability at least $1 - e^{-\frac{w^2}{k}}$, $X \leq \widehat{s} + w$. Thus, with at least this probability, the number $X + \tilde{s}$ of dishonest leaders elected in all cliques is smaller than $\tilde{s} + \widehat{s} + w = s + w$. $\square$

Theorem 2 states that the above protocol Π^{CP} achieves broadcast with probability $(1-\alpha)(1 - k2e^{-(n-t)\frac{\beta^2}{3k}})$ when using k cliques. We will now show that this holds when $\epsilon > 2\sqrt{\frac{\ln(\frac{1}{\alpha})}{k}}$ and $t = \frac{(1-\beta)(n-n\epsilon)}{3}$.

Proof. By Lemma 18 we have that with probability at least $k2e^{-(n-t)\frac{\beta^2}{3k}}$ each clique contains at least $\mu' = (1-\beta)\frac{2n}{3k}$ honest parties. By Lemma 19, with probability at most $e^{-\frac{w^2}{k}} = \alpha \Leftrightarrow w = \sqrt{\ln(\frac{1}{\alpha})k}$ the adversary can get a leader elected in more than $s + w$ cliques, where $s = \frac{t}{\mu'} = \frac{(1-\epsilon)k}{2}$. As we require that

$$s + w < \frac{k}{2} \Leftrightarrow s < \frac{k}{2} - \sqrt{\ln(\frac{1}{\alpha})k} \Leftrightarrow \frac{(1-\epsilon)k}{2} < \frac{k}{2} - \sqrt{\ln(\frac{1}{\alpha})k},$$

we obtain that $\epsilon > 2\sqrt{\frac{\ln(\frac{1}{\alpha})}{k}}$. It is now easy to see that with probability at least $(1-\alpha)(1-k2e^{-(n-t)\frac{\beta^2}{3k}})$ we can apply Lemma 21, which concludes the proof. □

What remains to show is, that any party in the protocol can broadcast when following this protocol.

Lemma 21. *Let Π be the protocol described above and assume that a majority of the elected leaders are honest. Assume that there exist an authenticated broadcast protocol* BC *which tolerates strictly less than half of the parties to be corrupted. Then Π achieves broadcast.*

Proof. We only sketch how to prove the statement under the above assumptions; indeed, it is not hard to see that Π achieves consistency and validity in this case. We first show consistency. Suppose that P_i is an honest party and outputs m at the end of the protocol. In this case, it has received at least $\frac{k}{2}$ signatures from the elected leaders. As we have assumed that the majority of the leaders is honest, this means that the honest leaders have signed the same message at the end of Round 10 of Π. But this means that the majority of the leaders also send the same message along with a signature to every other honest party P_j (note that all honest parties see pk_j with grade 2 by the properties of the key Ranking Phase). Thus P_j also outputs m at the end of Π. We now show validity of Π. Assume that P_i is honest and sends m to all leaders that it knows along with a valid signature on m. The leaders now all broadcast m to each other. Since signatures are unforgeable, none of the leaders receives conflicting messages $m \neq m'$ along with valid signatures in this step. Therefore, at the end of Round 10, all honest leaders send m to every honest party P_j. As P_j receives at least $\frac{k}{2}$ valid signatures on m at the end of Round 10, it outputs m, thus satisfying validity. Note that the communication complexity complexity in Π only depends on $\mathsf{bit}_{\mathrm{BC}}$ and $\mathsf{msg}_{\mathrm{BC}}$ rather than $k \cdot \mathsf{bit}_{\mathrm{BC}}$ and $k \cdot \mathsf{msg}_{\mathrm{BC}}$. Informally, this improvement can be achieved by electing only a single leader for the k parallel broadcasts, rather than k leaders. See [18] for further details. □