Round Efficient Byzantine Agreement from VDFs

Poulami Das\textsuperscript{1}, Lisa Eckey\textsuperscript{1}, Sebastian Faust\textsuperscript{1}, Julian Loss\textsuperscript{2}, and Monosij Maitra\textsuperscript{1}

\textsuperscript{1} Technical University of Darmstadt, Germany
\{firstname.lastname\}@tu-darmstadt.de
\textsuperscript{2} CISPA Helmholtz Center for Information Security, Germany
loss@cispa.de

Abstract. Byzantine agreement (BA) is a fundamental primitive in distributed systems and has received huge interest as an important building block for blockchain systems. Classical byzantine agreement considers a setting where \(n\) parties with fixed, known identities want to agree on an output in the presence of an adversary. Motivated by blockchain systems, the assumption of fixed identities is weakened by using a resource-based model. In such models, parties do not have fixed known identities but instead have to invest some expensive resources to participate in the protocol. Prominent examples for such resources are computation (measured by, e.g., proofs-of-work) or money (measured by proofs-of-stake). Unlike in the classical setting where BA without trusted setup (e.g., a PKI or an unpredictable beacon) is impossible for \(t \geq n/3\) corruptions, in such resource-based models, BA can be constructed for the optimal threshold of \(t < n/2\). In this work, we investigate BA without a PKI in the model where parties have restricted computational resources. Concretely, we consider sequential computation modeled via computing a verifiable delay function (VDF) and establish the following results:

1. Positive result: We present the first protocol for BA with expected constant round complexity and termination under adaptive corruption, honest majority and without a PKI. Earlier work achieved round complexity \(O(n\kappa^2)\) (CRYPTO'15) or \(O(\kappa)\) (PKC’18), where \(\kappa\) is the security parameter.

2. Negative result: We give the first lower bound on the communication complexity of BA in a model where parties have restricted computational resources. Concretely, we show that a multicast complexity of \(O(\sqrt{n})\) is necessary even if the parties have access to a VDF-oracle.

Keywords: Byzantine Agreement, Proof of Work, Verifiable Delay Functions, Impossibility

1 Introduction

In the Byzantine agreement (BA) problem, a set of \(n\) parties jointly run a distributed protocol to agree on a common output in the presence of some minority of \(t\) malicious parties. BA is a well-studied and fundamental problem in distributed computing and has recently garnered renewed interest in the context of blockchain protocols [1,9,26,8]. Traditionally, most existing protocols for BA assume a setting in which the parties’ identities are fixed and known at the beginning of the protocol. In the fixed identity setting, two types of protocols are studied: the first type requires setup, e.g., a public key infrastructure (PKI) or some form of correlated randomness. These protocols typically tolerate the (optimal) corruption threshold of \(t < n/2\). The second type does not require such assumptions but can tolerate only \(t < n/3\) corruptions.

More recently, a third type of protocol has emerged [5,25] that gives up on the fundamental assumption that parties know each other’s identities at the beginning of the protocol. Moreover, these protocols do not require setup in the classical sense, yet still
achieve the optimal corruption tolerance of $t < n/2$. Note that if identities are not fixed then without further measures, every party could pose as many parties and easily obtain a dishonest majority; this is commonly referred to as a sybil attack [12]. To avoid such attacks, parties must instead invest some expensive resources, such as computation or money, to participate in this type of protocol. A prominent example is the Proof-of-Work model (PoW) initially introduced by Bitcoin, where parties have limited access to a computational resource which they are forced to continuously expend in order to participate in the protocol.

In this work, we refine the PoW model by considering the effort it takes to evaluate verifiable delay functions (VDFs) [7,9] as the main computational resource. VDFs can be seen as a special type of proof-of-work whose computation cannot be sped up by much. This is in stark contrast to the typical lottery-type proofs-of-work, whose computation can be sped up almost arbitrarily, given sufficient parallel computation resources. We explore, for the first time, the implications of bounding the number of VDF evaluations that an (adaptive) adversary can compute in parallel: 1) We show an expected constant-round BA protocol that tolerates $t < n/2$ corrupted parties and does not rely on a PKI or known identities; 2) we give the first non-trivial communication lower bound by showing that any BA protocol in this setting requires at least $O(\sqrt{n})$ send-to-all steps.

1.1 The VDF Model

Our work introduces the VDF model as a refinement of the common PoW model to replace trusted setups and protect against Sybil attacks in permissionless consensus. Similar to the PoW model, we assume that the adversary only controls less than 1/2 of a computational resource to invest in producing proofs of computation. In contrast to the PoW model, however, we require a lower bound on the time it takes to create such proofs. This differs from the PoW model, where proofs can be computed almost arbitrarily fast, given sufficient parallel resources. We believe that the VDF model is a realistic alternative for the PoW model. Indeed, there exists various different constructions of VDFs [35,30,14] that leverage inherently sequential computation, and are used (or envisioned to be used) by blockchain projects for their consensus protocol (albeit not as an anti Sybil countermeasure as in our work). Examples include Chia, which relies on VDFs for leader election [9], and ETH 2.0, which plans to leverage VDFs for constructing a random beacon [8]. To make our model more realistic, we follow Wan et al. [34], and allow the adversary a small speedup in evaluating the VDF compared to the honest parties.

Let us describe our model with a concrete example. Suppose that the total amount computational power (over all protocol participants) over a fixed time period of length $t$ is 1000 VDF evaluations. Then, we demand that the total number of proofs produced by the adversary be at most 500 in the same time span. This is similar to the case of PoW model, where it is assumed that the majority of computational power belongs to honest parties. As mentioned above, we additionally give the adversary a small speedup, meaning that it can compute proofs a little bit faster than the honest parties.

The $\text{VDF}_\delta$ Oracle. We now explain our formalization of the VDF model in some more detail. At the center of our model, we introduce the oracle $\text{VDF}_\delta$, which parties can query on an input $s$ to obtain one evaluation $\phi$ of the VDF after $\delta$ time. Thus, $\delta$ denotes the difficulty parameter, specifying the number of sequential steps to be computed for one VDF evaluation. To make the model more realistic, we allow corrupted parties a $\varepsilon$-speedup
κ ≥ 1, meaning that they can obtain an output from VDFδ after δ/κ time. An adversary A in our model controls some q number of parties, where each party has κ-speed-up. For some i > 1, let us discuss how many proofs an adversary is able to compute within time t, where (i−1)·δ < t < i·δ. We can express t more concretely as t = (i−1)·δ + r·δ, where r ∈ (0, 1). With a corruption budget of q parties, A can invoke the VDFδ oracle q times concurrently (once per party). Since each proof is obtained after time δ/κ, at time (i−1)·δ, each party computes (i−1)·κ proofs. In the remaining r·δ time, each party can compute (at most) \( \lfloor \frac{r·δ}{κ} \rfloor \) proofs. Thus, in total A obtains at most \( ((i−1)·κ + \lfloor r·κ \rfloor)·q \) proofs at time t. Figure 1 illustrates our model with a small example. We refer to this

**Fig. 1.** Consider i = 3, δ = 5. We have i·δ = 15, (i−1)·δ = 10. Say an adversary A controls q parties \( \{P_i\}_{i \in [q]} \) with a speed-up κ = 3 compared to an honest party with κ = 1. Consider two time steps: t = 11 and t = 14. In both cases, each P_i can compute (i−1)·κ = 6 proofs in time 10 < i·δ. For the remaining time r·δ = 1 (for t = 11) and r·δ = 4 (for t = 14), no extra proofs can be computed in the first case, whereas \( \lfloor \frac{4·δ}{3} \rfloor \) = 2 extra proofs can be computed in the second case. Thus, A can compute in total 6q and 8q proofs for t = 11 and t = 14 respectively.

property of the VDFδ oracle as its **sequentiality** and give a formal definition in Section 2.

Such oracle abstraction of the VDF computation allows us to give a cleaner and more modular analysis of our main protocols. In support of our modelling approach, we conjecture that the VDFδ oracle can be instantiated in the standard model (Appendix A.1, Lemma 20) and we prove that it can be instantiated in the strong algebraic group model for constructions of Wesolowski [35] and Pietrzak [30] (Appendix A.2, Lemma 23).

1.2 Byzantine Agreement in the VDF Model.

As our main technical contribution, we show how to obtain an expected constant-round Byzantine agreement protocol without any additional trusted setup in the VDF+random oracle model. This is of particular significance, given that we can also instantiate the VDF model without any trusted setup, using Wesolowski’s construction. Given an upper bound
on the number of parties, our protocol tolerates $q$ adaptively corrupted parties with $\kappa$-speed-up, where $q(\lfloor \kappa \rfloor + 1) < n$. In particular, our protocols tolerate up to $\frac{q}{\kappa}$ corruptions when $\kappa = 1$. Our protocol combines several ideas from previous works in a novel way, as we now describe.

- **Step 1: Establishing a Graded Public key Infrastructure (GPKI).** We adopt the idea of Andrychowicz and Dziembowski [5] and start by setting up a precursor to a full PKI called graded PKI (GPKI) among the parties. Roughly speaking, a GPKI differs from a full PKI in that the keys of the parties are additionally associated with grades. These grades can differ between parties, but not by too much. As a first step, to reduce the round complexity of their protocol to $O(1)$ from $O(n\kappa^2)$ (where $\kappa$ is a security parameter), we make two modifications: 1) We set up a much weaker GPKI with only two possible grades, whereas [5] sets up $n$ possible grades. 2) We borrow a technique from Katz et al. [25] and rely on VDFs to make the round complexity of our protocol independent of $\kappa$. As a second step, the difficulty parameter $\delta$ of the VDFs must be adjusted to tolerate adversarial speedup in the VDF model.

- **Step 2: Graded Consensus from GPKI.** One of the ingredients needed in our BA protocol is a graded consensus protocol. Graded consensus is similar to a BA, with the difference that at the end of the protocol, every party outputs a value with some grade, where the grades of different parties must satisfy some consistency properties. We build upon the graded consensus protocol of Micali and Vaikuntanathan [28] where we modify their protocol such that it requires only a GPKI instead of a full PKI.

- **Step 3: Leader election protocol from VDFs.** It is well known that expected constant round protocols are inherently randomized, e.g., by electing a random leader in every protocol iteration. However, electing a random leader efficiently is challenging without prior setup. We overcome this issue by presenting a novel leader election protocol that leverages the oracle VDFs to efficiently elect a random leader that all honest parties agree on with high probability. Our protocol is inspired by leader elections based on verifiable random functions [2,3] and implements a leader election lottery with unique tickets. This makes the tickets hard to bias from the perspective of the adversary.

- **Step 4: BA protocol.** Finally, we combine all of the above components to adopt the expected constant round protocol of Katz and Koo [22] to our setting. We informally state our overall result in the following theorem.

**Theorem 1 (Informal).** Let $n$ denote an upper bound on the number of parties. Then, there exists an expected-constant round BA protocol in the VDF model that is secure against any adversary $A$ that controls at most $q$ parties with $\kappa$-speed-up, where $q \cdot (\lfloor \kappa \rfloor + 1) < n$.

1.3 A Lower Bound on Communication Complexity for BA

As a third contribution, we give the first lower bound for communication complexity of BA, assuming parties have bounded computational resources. Concretely, we show that in the VDF model without additional trusted setup, no protocol can realize BA with overwhelming probability by multicasting fewer than $O(\sqrt{n})$ messages in the presence of adaptive corruptions.

3 The construction in [25] uses only a proof of sequential as opposed to a VDF.
corruptions. In the multicast model of communication, honest parties are restricted to sending messages to all parties at once, whereas the adversary can send to only a subset of the parties. This models a setting in which parties communicate via a gossip network [17,5]. The cost of running the same protocol from the multicast model in the bilateral channel model [11] would be $O(n^{3/2})$. We remark, however, that the multicast restriction is crucially used in the lower bound, and thus, a better communication complexity might be possible in the bilateral channels model. Our lower bound builds on ideas of Abraham et al. [2] who show a bound for Byzantine broadcast in the multicast model without setup.

Our bound has to overcome several technical challenges that arise when parties have limited computational resources. The adversary in our attack has to carry out a simulation of the protocol in its head (this is a standard technique used in lower bounds), which may require to query the VDF oracle. At the same time, the adversary must also participate in an actual execution of the protocol it is attacking, which, of course, also results in queries to the oracle. Thus, the key difficulty in our lower bound is to carefully balance the adversary’s limited budget of queries to VDF over the two executions of the protocol (real and simulated).

Although our lower bound is relatively weak compared to most existing lower bounds in this area (which are quadratic, or of the form $O(n)$ in the multicast model, respectively), we argue that it is still meaningful. Namely, protocols that require significantly above $O(\kappa)$ multicasts are deemed impractical for large-scale settings with millions or even billions of users. This means that our bound essentially rules out efficient solutions in the VDF model unless further setup is assumed among the parties. Second, we point out that our lower bound actually holds in the relatively weak VDF-model and can likely be carried over to a less restrictive model (e.g., to the PoW model used by Bitcoin). It also leaves room for a tighter bound in such more general models.

1.4 Implications of Our Results and Related Work

Our model can be instantiated using Wesolowski’s VDF, which does not require trusted setup. Thus, our results show, for the first time, how to perform expected-constant round BA in a permissionless model with a simple honest majority and no trusted setup (beyond a random oracle). This has many important implications. For example, one could use our protocol to efficiently agree on a random string in a permissionless setting. This string could be used as a genesis block or as a uniform common reference string to perform an MPC protocol. Our results also significantly improves over the result of Andrychowicz and Dziembowski, who presented a protocol that achieves essentially the same, but requires $O(n \cdot \kappa^2)$ rounds to do so [5]. We also improve over a similar (slightly more efficient) version of this idea shown by Katz et al. [25]. Another closely related work is that of Garay et al. [19] who show how to bootstrap the classical Nakamoto consensus protocol [1,18,29] in the PoW model without trusted setup. However, it requires $O(\kappa)$ rounds, and therefore also does not constitute an expected constant round protocol.

Further Related Work. There is a large body of research on BA and related problems (sometimes colloquially referred to as “consensus”), and we focus here on the most closely related works. We have already mentioned the works of [5,25,19] who achieve BA in the PoW model without setup and run in $O(n \kappa^2)$, $O(n)$, and $O(\kappa)$ rounds, respectively. It should also be noted that we require stronger assumptions (namely a VDF and the random oracle model (ROM)) than the protocols in [5,19,20], who require only the ROM that can be
queried at a bounded rate by any party, but (possibly) weaker assumptions than required in the work of Katz et al. [25], who also require some form of an unpredictable beacon in their protocol. The more recent work of Aggarwal et al. [4] presents a setup-free solution in the PoW model that also runs in expected $O(1)$ rounds, but assumes a static adversary (while we consider the much stronger adversarial model of adaptive corruptions). Another related work that focuses on the PoW model is by Garay et al. [20]. They show how to achieve UC-secure BA and multi-party computation (MPC) protocols in the PoW model. Similar to [5], their BA takes $O(n\kappa^2)$ rounds.

Although the above-mentioned prior works achieve BA without a PKI [5,25,19], their techniques are not what we need to achieve $O(1)$-round BA. In fact, we notice that achieving $O(1)$-round BA protocols requires very particular techniques that have been studied [32,13,22,28,27,3,2] in the classical setting for many years and this round reduction is very challenging to achieve.

2 Definitions and Model

**Notation.** We write for the set of positive real numbers greater than some number $n \in \mathbb{R}$ as $\mathbb{R}_{>n}$, the set of natural numbers as $\mathbb{N}$. Throughout this paper, $\kappa$ will denote the security parameter. We write $x \leftarrow S$ to denote that $x$ is sampled uniformly at random from set $S$. Similarly, we write $y \leftarrow A(x)$ to denote that the output of a probabilistic algorithm $A$ on input $x$ is $y$.

2.1 Model

We consider a setting in which $n$ parties $P_1, \ldots, P_n$ engage in a distributed protocol $\Pi$. We assume that the exact number of parties is unknown but that there is some known upper bound $n$. Additionally, we assume that the majority of the parties follow the protocol honestly, and the remainder of the parties can be corrupted by the adversary (whose capabilities we will describe in the following). We emphasize that no public key infrastructure (PKI) needs to be shared among the parties, i.e., the parties do not initially know each other's public keys. Moreover, the parties are assumed to have synchronized clocks.

**Communication Model.** Inspired by [2], we consider a communication setting where parties multicast messages to other parties. In other words, a party may send the same message to everyone in the network (as opposed to possibly sending $n$ messages separately to $n$ parties). We say that a protocol has multicast complexity $\theta$, if the total number of multicasts (i.e., by all honest parties) in the protocol is at most $\theta$. This implies that in the classical communication model with bilateral channels, the (same) protocol requires sending $n\theta$ messages. We consider the synchronous model, where any message sent by an honest party over the multicast channel is received by all honest parties after at most time $\Delta$. As is usual in this model, any of our protocols proceeds in rounds of duration $\Delta$, where round $r$ of the protocol starts at time $(r-1) \cdot \Delta$ (assuming parties run the protocol at time 0).

**Random Oracle Model (ROM).** We model hash functions as random oracles [6]. The code of a hash function $H$ is defined as follows. On input $x$ from the domain of the hash function, $H$ checks whether $H(x)$ has been previously defined. If so, it returns $H(x)$. Else, it sets $H(x)$ to a uniformly random element from the range of $H$ and then returns $H(x)$.
Model of Computation. We consider the running time of parties in some fixed but unspecified model of computation, e.g., the Turing machine model or the arithmetic circuit model. This allows us to both simplify and generalize our results, as we discuss below. In addition to \( H \), parties in our model have access to oracle \( \text{VDF}_\delta \), which is used to restrict parties to performing certain computations sequentially. \( \text{VDF}_\delta \) has the following properties.

- **Evaluation:** On input \((\text{Eval}, S, \varepsilon)\), \(\varepsilon \geq 1\) from party \( P \) at time \( t \), \( \text{VDF}_\delta \) generates a proof of computation \( \phi \) according to some \( (\text{fixed}) \) output distribution \( D \) and returns \( \phi \) at time \( t + \frac{\delta}{2} \). It ignores any further inputs of the form \((\text{Eval}, S, \varepsilon)\) from \( P \) before time \( t + \frac{\delta}{2} \). We call the maximum value of \( \varepsilon \) over all queries of \( P \) to \( \text{VDF}_\delta \) the speedup of \( P \).

- **Verification:** On input \((\text{Verify}, \phi, S)\), \( \text{VDF}_\delta \) (immediately) returns 1, if \( \phi \) is a valid proof with respect to input \( S \) and 0 otherwise.

  Intuitively, \( \text{VDF}_\delta \) corresponds to a verifiable delay function (VDF) that takes \( \delta \) time to evaluate. Here, \( \delta \) can be set conservatively s.t. any honest party is able to compute an evaluation of the VDF within this timeframe. However, since not all parties run at the same speed, we allow for some speedup when evaluating the VDF. In our protocols, honest parties will always call \( \text{VDF} \) with \( \varepsilon = 1 \). The adversary, on the other hand, may set \( \varepsilon \) to some value above 1. We summarize this in the following definition.

**Definition 1 \(((q, t_p, \varepsilon))-Algorithm\).** \( \mathcal{A} \) is a \((q, t_p, \varepsilon)\)-adversary if it has full control over \( q \) parties with speedup at most \( \varepsilon \) and runs for at most \( t_p \) steps.

**Sequentiality of \( \text{VDF}_\delta \).** For a \((q, t_p, \varepsilon)\)-adversary \( \mathcal{A} \), we now define a natural notion of sequentiality. Intuitively, it should be impossible for \( \mathcal{A} \) to compute more evaluations of the VDF than its allotted budget of computation over a certain period of time. Here, \( \mathcal{A} \)'s budget spans over all \( q \) parties it controls and includes speedups quantified by \( \varepsilon \). Therefore, we expect \( \mathcal{A} \) to be unable to compute much more that \((i - 1) \cdot \varepsilon \cdot q \) proofs in less than \( i \cdot \delta \) time.

In a bit more detail, for any \( i \geq 1 \), when \( \mathcal{A} \) has less than \( i \cdot \delta \) time, with an \( \varepsilon \)-speedup it can compute exactly \( \lfloor (i - 1) \cdot \varepsilon \rfloor \) proofs at time \((i - 1) \cdot \delta \). The remaining time left is less than \( \delta (i \cdot \delta - (i - 1) \cdot \delta) \). \( \mathcal{A} \) can compute only a few more proofs in this time. In particular, denoting this remaining fractional time as \( r \cdot \delta \), where \( r \in (0, 1) \), \( \mathcal{A} \) can only compute at most \( \lfloor r \cdot \varepsilon \rfloor \cdot q \) proofs. Thus \( \mathcal{A} \) computes in total \( \lfloor (i - 1 + r) \cdot \varepsilon \rfloor \cdot q \) proofs at time \((i - 1 + r) \cdot \delta \).

Of course, such a notion only makes sense if \( \mathcal{A} \) can not start computing before a certain time \( T \). Hence, we begin by defining what it means for a random variable to be unpredictable in \( \mathcal{A} \)'s view. For convenience, we will say that \( S \) is determined at time \( T \) if its value is fixed in the view of at least one honest party at time \( T \).

**Definition 2 \(((k, T, \epsilon))-Unpredictable\).** Let \( S = (S_1, \ldots, S_k) \) be a vector of \( k \) random variables whose outcome is determined at time \( T \). We say that \( S \) is \((k, T, \epsilon)\)-unpredictable, if for all \((q, t_p, \varepsilon)\)-adversaries \( \mathcal{A} \), \( \Pr_{\hat{S}_1, \ldots, \hat{S}_k \leftarrow \mathcal{A}_{\hat{S}_1, \ldots, \hat{S}_k}^{\text{univ}}}[\exists j \in [k]: \hat{S}_j = \hat{S}_j] \leq \epsilon \), where the probability is over the random coins of \( \mathcal{A}, H, \text{VDF}_\delta \) and the random choice of \( S \).

We say, that \( S' \) depends on \( S \) if \( S' = H(\cdot ||S||) \), or if \( S' = H(\cdot ||S'||) \) with \( S'' = H(\cdot ||S'||) \). We now define the sequentiality property of \( \text{VDF}_\delta \) oracle relative to a sequence of inputs \( \{S'_1, \ldots, S'_r\} \), where each \( S'_j \) depends on some \((k, T, \epsilon)\)-unpredictable vector \( S \).
Definition 3 \((i, r, \beta)\)-Sequentiality. Let \(A\) be a \((q, t_p, \varepsilon)\)-adversary and fix \(T > 0\). Let \(S\) be \((k, T, \varepsilon)\)-unpredictable for some \(\varepsilon > 0\). For any \(i \in \mathbb{N}, r \in (0, 1)\), let \(T_{i,r} := T + (i - 1 + r) \cdot \delta\) and \(\tau_{i,r} := [(i - 1 + r) \cdot \varepsilon] \cdot q\). Let \(\{S'_1, \ldots, S'_{\tau_{i,r}+1}\}\) be a sequence of length \(\tau_{i,r}+1\), where each \(S'_j\) depends on at least one component of \(S\). We say that \(\text{VDF}_\delta\) is \((i, r, \beta)\)-sequential, if for all \(i \in \mathbb{N}, r \in (0, 1)\) s.t. \(A\) outputs \(\hat{\phi}_1, \ldots, \hat{\phi}_{\tau_{i,r}+1}\) before or at time \(T_{i,r}\), we have

\[
\Pr_{\hat{\phi}_1,\ldots,\hat{\phi}_{\tau_{i,r}+1} \leftarrow A_{\text{VDF}_\delta}^1}[\forall j \in [\tau_{i,r}+1]: \text{VDF}_\delta(\text{Eval}, S'_j, \varepsilon) = \hat{\phi}_j] \leq \varepsilon + \beta.
\]

The above probability holds over the random coins of \(A, H, \text{VDF}_\delta\) and the random choice of \(S\).

Relevance of our VDF\(_\delta\) oracle. As mentioned earlier, we consider the VDF construction due to Wesolowski [35], which is based on solving the classical RSW time-lock puzzle over class groups of an imaginary quadratic field. It uses a hash function \(H_G: \{0, 1\}^* \rightarrow G\), where \(G\) is a class group of an imaginary quadratic field. On input \(s \in \{0, 1\}^*\), the construction computes \(h := H_G(s)\) and outputs the value \(h^{2\delta}\). Verification can be done using \(\frac{\delta}{\log(\delta)}\) group elements and using a storage of \(\sqrt{\delta}\) group elements. This concrete instantiation would translate to our model by setting the output distribution \(D\) to the distribution that first draws \(s \leftarrow S\), computes \(h = H_G(s)\), and then evaluates the repeated squaring as described above. Note that the adversary \(A\) may guess the outcome of the random variable \(D\) before time \(T + \delta\), and we compensate for this in the definition via the factor \(\beta\). Moreover, the adversary may guess correctly the outcome of \(s\) of \(S\) and try to compute \(\text{VDF}_\delta\) before time \(T + \delta\). However, this is only possible with probability \(\varepsilon\) due to the unpredictability of \(S\).

We remark that for concrete constructions of VDFs such as Pietrzak’s or Wesolowski’s VDF [30,35], a simpler (and less idealized) way of modeling might be to consider adversaries \(A\) in the sequentiality definition from the class of arithmetic circuits of depth at most \(\delta\). Indeed, this would not require to model the VDF as an oracle, as one could simply bound the adversary’s probability of computing the VDF on unpredictable inputs according to the computational model. Since the focus of this work is on constructions and lower bounds of BA, we choose to model VDFs more abstractly to give a cleaner and more modular analysis of our main protocols. Moreover, as we use the VDF as a anti-Sybil protection, we need some way of quantifying the adversary’s resource budget, which is most commonly done via bounding oracle access. To support our modeling approach, we prove in Appendix A the sequentiality of our VDF\(_\delta\) oracle in the strong algebraic group model [33,24]. Although our proof holds for both constructions from [30,35], we choose to analyze Wesolowski’s VDF to obtain our results without additional trusted setup assumptions.

Protection against Homomorphic Computation of a VDF\(_\delta\). Some concrete constructions of VDFs (or of the related primitive time-lock encryption) may have an homomorphism, where for any two inputs \(s_1, s_2\), \(\text{VDF}(s_1) \cdot \text{VDF}(s_2) = \text{VDF}(s_1, s_2)\). This may enable an adversary to speed-up multiple evaluations of \(\text{VDF}_\delta\) on related inputs. In case of Wesolowski’s VDF [35], it already prevents such an homomorphism due to the use of the hash function \(H_G: \{0, 1\}^* \rightarrow G\). Recall that here, first, a group element \(h_1 = H_G(s_1)\) is computed, then output \(\phi_1\) is computed as \(\phi_1 = h_1^{2\delta}\) in \(\delta\) steps. Interestingly, this issue is also inherently prevented in our VDF model due to the following reason. Recall that in our definition of
sequentiality, we require that inputs to the oracle go via a hash function $H(\cdot)$. Even if the VDF itself exhibits a homomorphism, this effectively prevents any homomorphic evaluation for our concrete application.

**Adversary Model.** The adversary in our protocol is modelled as a $(q, t_p, \kappa)$-algorithm $A$ as defined above. $A$ can control at most $q$ parties each with a maximum speedup of $\kappa$, such that $q < \frac{n}{\lceil \kappa \rceil + 1}$ holds. In particular, the number of adversarial parties can be at most $< \frac{n}{2}$ (this is the case for $\kappa = 1$). We consider an adaptive adversary which can corrupt a party at any point during the protocol execution. Once a party has been corrupted, it can arbitrarily deviate from the protocol execution. Furthermore, it can deliver a message over the multicast channel only to a subset of honest parties. In this way, it can send different messages to different subsets of honest parties over the multicast channel. However, the adversary can not drop the messages of honest parties from the channel or delay them for longer than $\Delta$. Our adversary is *rushing*, which means it can observe all the messages that the honest parties send in any round of the protocol, and then choose its own messages for that round adaptively. We notice that we consider the standard notion of an adaptive, rushing adversary, as opposed to the stronger notion of a *strongly rushing* (or strongly adaptive) adversary (see for e.g., [3,2,10]) who can adaptively corrupt parties and then delete messages that they sent in the same round (prior to corruption).

**Lifting the Assumption on Clock Synchronization.** Here, we discuss how to overcome the assumption that parties have synchronized clocks. Since we focus on a setting, where parties do not share any PKI, it is crucial to relax this assumption, so that parties can work with their respective local clocks. Lets say, local clocks of individual parties can differ by at most time interval $\alpha$. When parties are instructed to send some messages in any round, they can always send messages at one particular time of the day, say 10:00 AM, local time. The duration of each round $\Delta$ can be set as $\Delta := 2\alpha$, to make sure that every party received messages sent back from all other parties before the current round ends and the next round begins.

**2.2 Definitions**

In this work, we focus on $n$-party protocols that reach consensus a.k.a. Byzantine agreement when up to $q$ out of $n$ parties are corrupted. All the following definitions consider a synchronous setting of communication.

**Definition 4 (Byzantine Agreement).** A protocol executed among $n$ parties where each party $P_i$ holds initial input $x_i$ and parties output upon terminating $y_i$ achieves Byzantine Agreement if the following properties hold whenever at most $q$ parties are corrupted:

- **Consistency:** If two honest parties $P_i$, $P_j$ output values $y_i$, $y_j$ respectively, then $y_i = y_j$.
- **Validity:** If all honest parties $P_i$ have the same input $x_i = x$, then all honest parties output $y_i = x$.
- **Termination:** All honest parties terminate.

We also study the sender-centric version of Byzantine Agreement, which is called Byzantine Broadcast.
Definition 5 (Byzantine Broadcast). A protocol executed among \( n \) parties, where a designated sender \( S \) holds an input \( x \) at the beginning of the protocol and parties output \( y_i \) upon terminating, achieves Byzantine Broadcast if the following holds whenever at most \( q \) parties are corrupted:

- **Consistency**: If two honest parties \( P_i, P_j \) output values \( y_i, y_j \) respectively, then \( y_i = y_j \).
- **Validity**: If the sender \( S \) is honest, then all honest parties output \( y_i = x \).
- **Termination**: All honest parties terminate.

Graded byzantine agreement (also known as graded consensus) is a weaker variant of byzantine agreement, while graded broadcast (or gradecast, for short) is a weaker variant of byzantine broadcast. They have been used in various previous works (cf. graded byzantine agreement in [16] and [15], gradecast in [28] and [23]). Both of them are very useful building blocks for byzantine agreement and byzantine broadcast protocols.

Definition 6 (Graded Byzantine Agreement). A protocol \( \Pi \) executed among \( n \) parties, where each party holds initial input \( x_i \) and parties output upon terminating a value \( y_i \) and a grade \( \zeta_i \), achieves Graded Byzantine Agreement if the following holds whenever at most \( q \) parties are corrupted:

- **Consistency**: If there is an honest party that outputs \( y_i \) with grade \( \zeta_i = 2 \), then every other honest party outputs \( y_j = y_i \) with grade \( \zeta_j \geq 1 \).
- **Validity**: If all honest parties input \( x_i = x \), then every honest party outputs \( y_i = x \) with grade \( \zeta_i = 2 \).
- **Termination**: All honest parties terminate.

Definition 7 (Graded Broadcast). A protocol \( \Pi \) among \( n \) parties, where a designated sender \( S \) holds an input \( x \) at the beginning of the protocol and parties output a value \( y_i \) and a grade \( \zeta_i \) upon terminating, achieves Graded Broadcast if the following holds whenever at most \( q \) parties are corrupted:

- **Consistency**: If there is an honest party that outputs \( y_i \) with grade \( \zeta_i = 2 \), then every other honest party outputs \( y_j = y_i \) with grade \( \zeta_j \geq 1 \).
- **Validity**: If \( S \) is honest, then every honest party \( P_i \) outputs \( y_i = x \) with grade \( \zeta_i = 2 \).
- **Termination**: All honest parties terminate.

For the following definition, let KeySet be the local key set of a party \( P \), which keeps tuples of the form \((pk_j, \zeta_j)\). \( pk_j \) is the public key of some party \( P_j \) and \( \zeta_j \in \{1, 2\} \) is the grade assigned to this key.

Definition 8 (Graded Public Key Infrastructure). A protocol \( \Pi \) among \( n \) parties, where parties output a set KeySet upon terminating achieves Graded PKI (tolerating \( q \) corrupted parties) if the following holds for any two honest parties \( P_i, P_j \):

- **Graded Validity**: \( P_i \)'s public key \( pk \) is assigned grade \( 2 \) by \( P_j \) (i.e., \((pk, 2) \in \text{KeySet}_j \)).
- **Graded Consistency**: If \( P_i \) assigns grade \( 2 \) to a public key \( pk \), i.e., \((pk, 2) \in \text{KeySet}_j \), then \( P_j \) assigns at least grade \( 1 \) to the same key, i.e., \((pk, \zeta) \in \text{KeySet}_j \), where \( \zeta \geq 1 \).
- **Bounded Number of Identities**: Let \( N := |\bigcup_{i \in [n]} \text{KeySet}_i| \) be the total number of keys in the combined key sets of all honest parties \( \mathcal{H} \), then the total number of keys that belong to corrupted parties is \( < \frac{1}{2} N \).


- **Termination:** All honest parties terminate.

We remark that we require the properties in the above definitions to hold with probability 
\(1 - 2^{-\kappa}\) (where \(\kappa\) is the security parameter), but we omit this detail from the definition for ease of presentation.

**Signature Schemes.** We use the following standard definition of digital signature schemes.

**Definition 9 (Signature Schemes).** A digital signature scheme is a triple of algorithms 
\((\text{KeyGen}, \text{Sign}, \text{VrfySig})\) with the following properties. On input the security parameter \(\kappa\), the randomized key generation algorithm \(\text{KeyGen}\) outputs a key pair \((sk, pk)\). On input a message \(m \in \{0,1\}^*\) and a secret key \(sk\), the randomized signing algorithm \(\text{Sign}\) outputs a signature \(\sigma\). On input a public key \(pk\), a message \(m \in \{0,1\}^*\), and a signature \(\sigma\), the deterministic verification algorithm \(\text{VrfySig}\) outputs 1 if the signature is correct or 0 otherwise.

Assume for convenience of notation that to each signed message implicitly has the public key of the signer and a fresh nonce appended. We require that the scheme satisfies (perfect) correctness, i.e., for all \(m \in \{0,1\}^*\) and \((sk, pk)\) output by \(\text{KeyGen}\) it holds that: \(\text{VrfySig}(pk, \text{Sign}(sk, m), m) = 1\). In line with most works in this area, we treat signature schemes as idealized objects satisfying unforgeability. This means that it is information-theoretically impossible to forge a signature under some public key \(pk\) without knowing the corresponding secret key \(sk\). It is easy to instantiate our schemes with a concrete scheme satisfying unforgeability under chosen message attack [21].

3 Graded Public Key Infrastructure (GPKI)

In a public key infrastructure (PKI) setting, parties have a globally consistent view of a keyset, containing \(n\) public keys, where \(n\) is the total number of parties. As a starting point, we present a construction which is similar to such a PKI. The challenge that we face without an a-priori setup is that we can not easily achieve a globally consistent view on the protocol participants. As discussed in the last Section 2.1, we do not know the exact number of parties, but a known upper bound \(n\). We address this problem by building a so-called graded PKI. A graded PKI differs from a "real" PKI since the local views of parties \(P_i, P_j\) on their respective key sets \(\text{KeySet}_i, \text{KeySet}_j\) can differ. Here we present \(H_{\text{KeyGrade}}\) (c.f. Figure 2), which achieves a graded PKI in the VDF + random oracle model. Our protocol builds on earlier works [25,5] which achieve a stronger notion of GPKI with \(n\) grades but require \(O(n^2)\) rounds to do so. In contrast, we run a 2-graded protocol requiring only a constant number of rounds. We begin by giving some intuition about our protocol and identifying the main challenges that it has to overcome. We describe the protocol from the view of a (honest) party \(P\) with \(\kappa = 1\).

**Challenge Phase.** The protocol begins with a two-round challenge phase. For \(i \in \{1, 2\}\), we denote \(i\)-th challenge message as \((\text{chal}|i, .)\).

- First round: \(P\) samples a uniform challenge \(c \leftarrow \{0,1\}^{\kappa + \log(t_p) + 2\log(\kappa) + 1}\). \(P\) then multicasts the message \((\text{chal}|1, c)\). Let \(c := (c_1, \ldots, c_\theta)\), where \(\theta \leq n\), denote the vector of challenges that \(P\) receives from parties (including its own) by the end of the first round.
Second round: $P$ computes its second round challenge $d$ as $d := H(c)$ and multicasts the message $(\text{chal}||2,d)$. Denote $d := (d_1,\ldots,d_{\theta'})$, where $\theta' \leq n$, as the challenges that $P$ receives over the course of the second round (including its own) and denote $\chi := H(d)$ as their hash.

**Proof of Computation Phase.** The challenge phase is immediately followed by a proof of computation phase. At the beginning of this round, $P$ generates a fresh pair of keys as $(sk,pk) \leftarrow \text{KeyGen}(1^\lambda)$. $P$ then calls $\text{VDF}_\delta$ on input $(\text{Eval}, s := \{\chi,pk\}, 1)$ to compute the proof $\phi$.

**Key Grading Phase.** After computing $\phi$ in the proof of computation phase, a party $P$ runs a phase of key grading, during which the public key of each other party will be assigned a grade in $\{1,2\}$. This phase consists of three rounds, where parties send around messages of the form $(\text{rank}||i,\cdot), i \in \{1,2\}$, for assignment of the $i$-th grade.

- **First round:** $P$ multicasts the message $(\text{rank}||2, pk, \chi, \phi, d)$.

- **Second round:** For each such message $(\text{rank}||2, pk_j, \chi_j, \phi_j, d_j)$ that was received by time $3\Delta + \delta$ from some party $P_j$, $P$ checks whether it was correctly formed. More precisely, $P$ first checks that $\phi_j$ is a proof that passes verification for the input $\{\chi_j,pk_j\}$, i.e., $\text{VDF}_\delta(\text{Verify}, \phi_j, \{\chi_j,pk_j\}) = 1$. Next, it checks that $\chi_j$ is consistent with $d_j$, i.e., $\chi_j = H(d_j)$. Lastly, $P$ checks if its second round challenge $d \in d_j$. If all these checks verify, $P$ is convinced that $P_j$ could not have started computing $\phi_j$ earlier than at round $2\Delta$, as $\phi_j$ depends on a value that $P$ chose uniformly at random at this time. Hence, it assigns the highest grade 2 to $pk_j$, i.e., it adds $(pk_j,2)$ to KeySet. To make sure that every other honest party assigns $pk_j$ at least grade 1, $P$ multicasts the message $(\text{rank}||1, pk_j, \chi_j, \phi_j, d_j, c)$. Note that, party $P_j$ also follows the same steps as above for each received message $(\text{rank}||2, pk_l, \chi_l, \phi_l, d_l)$ from $P_l$ and accordingly sends around $(\text{rank}||1, pk_l, \chi_l, \phi_l, d_l, c_l)$, if it has assigned grade 2 to $P_l$.

- **Third round:** Let $(\text{rank}||1, pk_l, \chi_l, \phi_l, d_l, c_l)$ (for $l \neq j$) be the message received by the end of the previous round from $P_j$, where $P_j$’s key $pk_j$ was already assigned grade 2 by $P$ in the first grading step. It assigns grade 1 to $pk_l$ (i.e., it adds $(pk_l,1)$ to KeySet), if it has not assigned grade 2 to $pk_l$ in the previous round but is convinced from $P_j$’s perspective (i.e., from the messages received from $P_j$) that $pk_l$ should have been marked with grade 2. In addition, $P$ needs to be convinced that the proof $\phi_l$ depended on some unpredictable value and could not have been computed earlier than at time $2\Delta$. To make sure of this, $P$ does the following. It first checks that the proof $\phi_l$ passes verification for the input $\{\chi_l,pk_l\}$, i.e., $\text{VDF}_\delta(\text{Verify}, \phi_l, \{\chi_l,pk_l\}) = 1$. Note, however, that since $P_j$ is a dishonest party (otherwise, $P$ would have already graded $pk_l$ in the first grading step), it may have computed $\chi_l = H(d_l)$ independently of $P_j$’s second round challenge $d$. In this case, it verifies instead that $\chi_l$ depends on $c_l$, by checking that both $H(c_l) \in d_l$ and $c \in c_l$. If all these checks pass, $P$ adds $(pk_l,1)$ to KeySet. Once all messages of this form have been processed, $P$ outputs KeySet.

It is easy to verify that each honest party’s key is assigned grade 2 by every honest party, thus proving graded validity. Additionally, the second step of the key grading ensures any key that an honest party has assigned grade 2 is assigned a grade of at least 1 by all honest parties – this implies graded consistency. Finally, an honest party only accepts keys if it is convinced that its corresponding proof could not have been precomputed prior to the
beginning of the proof of computation phase. The duration of the proof of computation phase is set to $\delta$ according to the VDF oracle. While an honest party (with no speed-up) computes one proof within $\delta$ time, an adversarial party with $\kappa$-speed-up of computational power computes $\kappa$ proofs within the same time $\delta$. Below, we will set the parameter $\delta$ such that no adversarial party can make more than $\kappa$ calls to VDF for the entire duration of the protocol, given that it calls VDF earliest at the beginning of the Challenge phase. Taken together, the total number of adversarial keys accepted by honest parties can not exceed half of the total number of keys. This implies bounded number of identities.

\begin{center}
Protocol $\Pi_{\text{KeyGrade}}$
\end{center}

We describe the protocol from the view of an honest party $P$ with no speed-up of computational power, where $P$ has access to a random oracle $H\colon \{0,1\}^* \to \{0,1\}^{\lambda+\log(p)+2\log(n)+1}$ and an oracle VDF. $P$ executes the following phases.

**Challenge Phase.**
- At time 0: Sample $c \leftarrow \{0,1\}^{\lambda+\log(p)+2\log(n)+1}$ and multicast the message $(\text{chal}\|1, c)$.
- At time $\Delta$: Add all $\theta(\leq n)$ challenges received by time $\Delta$ to the set $c_i = (c_1, \ldots, c_\theta)$. Compute $d = H(c)$ and multicast the message $(\text{chal}\|2, d)$.

**Proof of Computation Phase.**
- At time $2\Delta$: Add all $\theta'(\leq n)$ challenges received in round 2 to the set $d = (d_1, \ldots, d_{\theta'})$ and compute its hash $\chi = H(d)$. Then, sample key pair $(sk, pk) \leftarrow \text{KeyGen}(1^\lambda)$ and compute $\phi = \text{VDF}_\delta(\text{Eval}, s := (\chi, pk), 1)$.

**Key Ranking Phase.** Define the following Boolean conditions:
- $C_2(j)\colon \text{VDF}_\delta(\text{Verify}, \phi_j, \{\chi_j, pk_j\}) = 1$, $\chi_j = H(d_j)$, and $d \in d_j$.
- $C_1(j, l)\colon (pk_j, 2) \in \text{KeySet}$, $(pk_j, \cdot) \notin \text{KeySet}$, $\text{VDF}_\delta(\text{Verify}, \phi_j, \{\chi_l, pk_j\}) = 1$, $\chi_l = H(d_l)$, $H(c_j) \in d_j$, and $c \in c_j$.

Do:
- At time $2\Delta + \delta$: Multicast $(\text{rank}\|2, pk, \chi, \phi, d)$.
- At time $3\Delta + \delta$: For each received message $(\text{rank}\|2, pk_j, \chi_j, \phi_j, d_j)$ from $P_j$, such that $C_2(j)$, add $(pk_j, 2)$ to KeySet and multicast the message $(\text{rank}\|1, pk_j, \chi_j, \phi_j, d_j, c)$. 
- At time $4\Delta + \delta$: For each received message $(\text{rank}\|1, pk_i, \chi_i, \phi_i, d_i, c_i)$ from $P_j$ such that $C_1(j, l)$, add $(pk_i, 1)$ to KeySet. After processing each message, output KeySet.

Fig. 2. A protocol for establishing a GPKI with 2 grades run between $n$ parties, where parties have access to a random oracle $H$ and an oracle VDF. Duration of the $\Pi_{\text{KeyGrade}}$ protocol is $(5\Delta + \delta)$. For the subsequent Sections we set $\delta := 11\Delta$, which sets the duration of the protocol to $16\Delta$.

The following sequence of lemmas prove the security of $\Pi_{\text{KeyGrade}}$. Lemmas 1 to 3, respectively, show that $\Pi_{\text{KeyGrade}}$ satisfies the properties of graded validity, graded consistency and bounded number of identities as per Definition 8. Due to space limitation, we defer the proofs of Lemmas 1 and 2 to Appendix B.

**Lemma 1.** Protocol $\Pi_{\text{KeyGrade}}$ achieves graded validity.

**Lemma 2.** Protocol $\Pi_{\text{KeyGrade}}$ achieves graded consistency.

**Lemma 3.** Suppose that $A$ is a $(q, t_p, \kappa)$-algorithm where $q < \frac{n}{\kappa+1}$. For $k \in \mathbb{N}$, let $\delta = k\Delta$. If $k > 5\kappa$, and VDF is $(2, \frac{5}{\kappa}, \beta)$-sequential, then $\Pi_{\text{KeyGrade}}$ achieves bounded
number of identities with probability at least $1 - (2^{-\kappa - 2\log(\kappa)} - 1 + \beta)$. In particular, if $N := |\bigcup_{i \in [n]} KeySet_i|$ be the total number of keys in the combined key sets of all honest parties $H$, then the total number of keys that belong to corrupted parties is $< \frac{1}{2}N$.

**Proof.** Suppose that parties run $\Pi_{KeyGrade}$ at time $T$. Let $E$ be the event that $A$ predicts the first honest party $P_i$ at time $T' < T$. Since $A$ runs for at most $t_p$ steps and each honest party $P_i$ samples $c_i$ uniformly from the space $\{0, 1\}^{\kappa + \log(t_p) + 2\log(\kappa) + 1}$, $E$ occurs with probability at most $2^{-\kappa - 2\log(\kappa)}$. By Definition 2, this means that the sequence of honest challenges $\{c_i\}_{i \in H}$ is $((n - q), T, 2^{-\kappa - 2\log(\kappa)})$-unpredictable. Since the duration of $\Pi_{KeyGrade}$ is $\delta + 5\Delta$, the protocol terminates at $T + \delta + 5\Delta$. From $(2, \frac{5}{\kappa}, \beta)$-sequentiality of VDF, we have

$$T_2, \frac{5}{\kappa} = T + (\delta + 5\Delta) = T + (1 + \frac{5\Delta}{\delta}) \cdot \delta = T + (1 + \frac{5}{\kappa}) \cdot \delta$$

$$\tau_2, \frac{5}{\kappa} = \tau = \left(1 + \frac{5}{\kappa}\right) \cdot \delta = \left(1 + \frac{5}{\kappa}\right) \cdot \kappa \cdot q. \text{ (Since, } k > 5\kappa)$$

Let $\chi_1, \ldots, \chi_{\tau+1}$ depend on $\{c_i\}_{i \in H}$ and let $pk_1, \ldots, pk_{\tau+1}$ be pairwise distinct. $A$ is able to compute at most $q \cdot |\chi|$ VDFs within time $(T + \delta + 5\Delta)$ via honest invocations of the VDF oracle. It follows from $(2, \frac{5}{\kappa}, \beta)$-sequentiality of VDF that $A$ can output $\{\phi_i\}_{i \in [\tau + 1]}$ before or at time $(T + \delta + 5\Delta)$ such that for all $j \in [\tau + 1], VDF_{\delta}(\text{Verify}, \phi_j, (\chi_j, pk_j)) = 1$ with probability at most $2^{-\kappa - 2\log(\kappa)} + \beta$. (Note that if $\chi_j$ does not depend on $\{c_i\}_{i \in H}$, no honest party accepts $pk_j$.) As a result, the number of identities produced by $A$ is bounded by $\tau$ with probability at least $1 - (2^{-\kappa - 2\log(\kappa)} + \beta)$. The total number of keys from honest and corrupted parties can be given by $N = (n - q) + \tau = (n - q) + q \cdot |\chi|$, and the total number keys belonging to corrupted parties is $\tau = q \cdot |\chi| < (n - q)$ (given that $q < \frac{n}{\delta + 5\Delta}$ or, $q \cdot |\chi| < \frac{1}{5}N$).

$\square$

**Corollary 1.** Since $\delta > 5\kappa \cdot \Delta \geq (5\kappa + 1) \cdot \Delta$, the duration of $\Pi_{KeyGrade}$ is set as $\delta + 5\Delta := (5\kappa + 6) \cdot \Delta$.

**Corollary 2.** If $n$ be the number of parties in the $\Pi_{KeyGrade}$ protocol, $q$ be the number of adversarial parties, each with speedup at most $\kappa$, then the total number of keys in the combined key sets of all honest parties is given as

$$N := |\bigcup_{i \in [n]} KeySet_i| = (n - q) + q \cdot |\chi| = n + q \cdot (|\chi| - 1).$$

**Parameter Selection.** For the remaining sections of the paper we set $\kappa = 2$, which gives us: $\delta > 10 \cdot \Delta \geq 11\Delta$. We assign $\delta := 11\Delta$, and accordingly duration of $\Pi_{KeyGrade} := 16\Delta$. To tolerate a higher speed-up of adversarial parties the parameters need to be adjusted accordingly.

### 4 Construction of Graded Broadcast (GBC) from GPKI

As a next step, we construct a graded broadcast protocol that is an adaptation of the protocol in [28]. The resulting protocol $\Pi_{GBC}$ only requires a graded PKI whereas the
Lemma 5. If, at time $\Delta$, a set of signatures $\psi$ contains weakly valid countersignatures on $x$, then every honest party $P$ multicasts a set of signatures $\psi_P(x)$ at time $2\Delta$ which is consistent for $x$ in the view of every honest party.

Lemma 6. Protocol $\Pi_{GBC}$ achieves graded validity.

Lemma 7. Protocol $\Pi_{GBC}$ achieves graded consistency.

4.1 Construction of a Graded Byzantine Agreement from GBC

We define a simple graded byzantine agreement protocol $\Pi_{GBA}$ in Fig. 4 from $\Pi_{GBC}$. As a first step, each party runs $\Pi_{GBC}$ as the sender with their input. In the following step, a value is output with grade 2 or 1 depending on the message/grade outputs of the $\Pi_{GBC}$ initiated by other parties. The security of $\Pi_{GBA}$ follows from the following sequence of lemmas.

Lemma 8. Protocol $\Pi_{GBA}$ achieves graded validity.

Lemma 9. Protocol $\Pi_{GBA}$ achieves graded consistency.
Protocol $Π_{GBC}$

We describe the protocol from the view of party $P$ with a sender $S$. $P$ executes the following three phases.

- At time 0: If $P$ is the sender $S$, it computes signature on message $x$ as $(x)_S$ and multicasts $(x, (x)_S)$ to all parties.
- At time $Δ$: If $P$ sees $(x, (x)_S)$, where $(x)_S$ is a valid signature on message $x$, then $P$ multicasts $(x, ⟨(x)_S⟩_P)$ to all parties.
- At time $2Δ$: Party $P$ collects all valid countersignatures of the form $(x', ⟨(x')_S⟩_S)$ from parties $P'_k$ with same input $x'$. $P$ constructs the signature set $ψ(x')$ with message $x'$ as

$$ψ(x') := \left\{ ⟨⟨(x')_S⟩_S⟩_k \right\}.$$ 

If $ψ(x')$ is a consistent signature set from $P$'s view and it has not received a countersignature of the form $⟨⟨x''⟩_S⟩_S$ where $x'' \neq x'$ then $P$ multicasts $ψ(x')$. (Otherwise, $P$ multicasts nothing.)

- At time $3Δ$: Let $ψ_k(x')$ be a weakly consistent signature set received from party $P'_k$ (note that this includes consistent sets). To determine its output, $P$ does as follows:
  - If $P$ sees at least $\frac{n}{2}$ consistent signature sets of the form $ψ_k(x')$ (for distinct $k$) on $x'$, then $P$ outputs $(x', 2)$.
  - If $P$ sees at least one weakly consistent signature set $ψ_k(x')$ for some $x'$, and no weakly consistent signature set for a different value $x'' \neq x'$, then it outputs $(x', 1)$.
  - Else, $P$ outputs $(⊥, 0)$.

Fig. 3. Gradecast protocol $Π_{GBC}$, where each party has a local KeySet from $Π_{KeyGrade}$ protocol. Duration of $Π_{GBC}$ is $4Δ$. $N = n + q \cdot (|x| - 1)$ (c.f Corollary 2).

5 Achieving Byzantine Agreement

As the final step, we build our Byzantine agreement protocol $Π_{BA}$ (c.f. Figure 5). Though our protocol is an adaptation of the same from [22], it uses as main ingredients the graded PKI protocol $Π_{KeyGrade}$ (Figure 2) and the graded byzantine agreement protocol $Π_{GBA}$ (Figure 4) from Sections 3 and 4.1 respectively. One subtle difference because of the use of a graded PKI is that each party $P$ relies on their local KeySet. To make the BA protocol work, we need one more ingredient - a leader election protocol. More precisely, as one of the steps of the $Π_{BA}$ protocol, it is necessary to elect one of the parties as a leader in an unpredictable manner such that an honest party will be elected with constant probability. For the description of $Π_{BA}$, we assume a protocol $Π_{Leader}$ that is run as a subroutine to $Π_{BA}$, in parallel with protocol $Π_{KeyGrade}$. As we will explain later, the subprotocols $Π_{KeyGrade}$ and $Π_{Leader}$ share a common state. For our purposes, we will require that the subprotocol $Π_{Leader}$ has the following properties for all $k \in N$ and all pairs of honest parties $P, P'$, assuming that parties initiate $Π_{BA}$ at time 0:

- $Π_{Leader}$ outputs a value $ℓ_P$ to $P$ at time $(12 \cdot k + 27) \cdot Δ$.
- With probability at least $\frac{n-q}{N} \geq \frac{1}{2}$, $ℓ_P = ℓ_{P'} = ℓ$ and $P_ℓ$ is an honest party at round $(12 \cdot k + 24) \cdot Δ$.

We present an instantiation $Π_{Leader}$ based on VDF in Section 5.1. The following theorem summarizes the properties of our $Π_{BA}$ protocol.

**Theorem 2.** Suppose that $Π_{KeyGrade}$ be a graded public key infrastructure protocol, $Π_{GBA}$ be a graded byzantine agreement protocol and $Π_{Leader}$ be a leader election protocol run between
Protocol $\Pi_{\text{GBA}}$

We describe the protocol from the view of party $P$, where $P$ executes the following two steps.

- At time 0: Initiate $\Pi_{\text{GBA}}$ as the sender with input $x$.
- At time $4\Delta$: Let $(x_j, g_j)$ be the message/grade that $P$ outputs in the gradecast initiated by $P$. For all $v$ that were received in at least one invocation of $\Pi_{\text{GBA}}$, $P$ sets $S^v := \{ j : x_j = v \land g_j = 2 \}$ and $\overline{S}^v := \{ j : x_j = v \land g_j \geq 1 \}$.
  - If there exists $v$ s.t. $|S^v| \geq \frac{n}{2}$, then prepare a tuple $t$ of value, grade pair $t := (v, 1)$, otherwise $t := (\bot, 0)$.
  - If $|S^v| > \frac{n}{2}$, then $t := (v, 2)$.
  - Output $t$.

Fig. 4. Graded Byzantine Agreement protocol $\Pi_{\text{GBA}}$, where each party has a local KeySet from $\Pi_{\text{KeyGrade}}$ protocol. Duration of $\Pi_{\text{GBA}}$ is $4\Delta$. $N = n + q \cdot (|x| - 1)$ (c.f Corollary 2).

$n$ parties. Then $\Pi_{\text{BA}}$ is a Byzantine agreement protocol tolerating $q$ corrupted parties with at most 2-speedup of computational power, where $q < \frac{n}{3}$. Moreover, it terminates within $O(1)$ rounds in expectation for all honest parties, and with probability $1 - 2^{-\kappa}$ within $16 + 12 \cdot (\kappa + 1)$ rounds.

Our proof for Theorem 2 is almost identical to [22, Theorem 13] which is why we defer it to Appendix E. After an initial setup phase in which parties agree on a graded PKI via $\Pi_{\text{KeyGrade}}$, $\Pi_{\text{BA}}$ runs a key leader election protocol. In every such iteration, we invoke two graded Byzantine agreement subroutines and one multicast subroutine, and update the values lock and $m$ until they converge on a common value for $m$. Here, the variable lock indicates the number of iterations after which the protocol may terminate. In particular, lock = $\infty$ means that the protocol could keep running for an unbounded number of iterations, lock = 1 means that termination will be reached after one more iteration and lock = 0 means that the protocol terminates in the next iteration. Overall, each iteration requires $12\Delta$ time.

To enter termination mode, i.e., to set lock = 1, parties rely on a randomly elected leader who is chosen in the last round of an iteration via $\Pi_{\text{Leader}}$. In our construction of $\Pi_{\text{Leader}}$, honest parties learn the identity of the leader for the $k$-th iteration at time $(12 - k + 27) \cdot \Delta$. On the other hand, dishonest parties learn it at round $(12 \cdot k + 25) \cdot \Delta$ – this is because dishonest parties, controlled by a rushing adversary can start the proof of computation phase of the $\Pi_{\text{KeyGrade}}$ at time 0 instead of at time $2\Delta$. Hence, the idea is to have all parties perform a multicast instruction in the protocol at round $(12 \cdot k + 24) \cdot \Delta$, i.e., one round prior to the adversary learning about the identity of the leader. If the elected leader executes this instruction honestly (which happens with probability at least $\frac{1}{2}$), the protocol enters termination mode, meaning that all honest parties set lock = 1. Overall, the protocol enters the termination routine after at most $\kappa$ many iterations with probability all but $2^{-\kappa}$.

**Generalization of Theorem 2.** Our Theorem 2 can be expressed more generically in the following Theorem 3, for any speedup parameter $\varkappa \in \mathbb{R}_{>0}$, where the number of rounds of $\Pi_{\text{KeyGrade}}$ is $(5\varkappa + 6) \cdot \Delta$ (c.f Corollary 1). In particular, for the lowest value of $\varkappa = 1$, our $\Pi_{\text{BA}}$ protocol tolerates at most $\frac{n}{3}$ corrupted parties.

**Theorem 3.** Suppose that $\Pi_{\text{KeyGrade}}$ be a graded public key infrastructure protocol, $\Pi_{\text{GBA}}$ be a graded byzantine agreement protocol and $\Pi_{\text{Leader}}$ be a leader election protocol run
between n parties. Then ΠBA is a Byzantine agreement protocol tolerating q corrupted parties with at most χ-speedup of computational power, where q < \frac{n}{(\lfloor \kappa \rfloor + 1)}$. Moreover, it terminates within O(1) rounds in expectation for all honest parties, and with probability $1 - 2^{-\kappa}$ within $(5\chi + 6) + 12 \cdot (\kappa + 1)$ rounds.

Proof of Theorem 3 follows similar to that of Theorem 2.

### Protocol ΠBA

We describe the following protocol from the view of party P with input $m_P$.

**Setup Phase.**
- At time 0: Participate in a run of $\Pi_{KeyGrade}$. In parallel, participate in a run of $\Pi_{Leader}$.  
- At time $16\Delta$: Denote $\text{KeySet}_P$ the set of keys output in $\Pi_{KeyGrade}$. (Hereafter, it is assumed that parties share a graded PKI.)

**Agreement Phase.** Initialize $\text{lock} := \infty, m := m_P$ and $k := 0$. Repeat the following loop forever:
- At time $(12k + 16) \cdot \Delta$: Initiate $\Pi_{GBA}$ on input $m$.
- At time $(12k + 20) \cdot \Delta$:
  - Let $(v, g)$ be the output of $P$. If lock = \infty then:
    - If $g = 2$, set $\text{lock} := 1$.
    - If $g = 1$, then set $m := v$.
    - If $g = 0$, then set $m = \bot$.
  - Initiate $\Pi_{GBA}$ on input $m$.
- At time $(12k + 24) \cdot \Delta$:
  - Let $(v, g)$ be the output of $P$. If lock = \infty then:
    - If $g = 1$, then set $m := v$.
    - If $g = 0$, then set $m = \bot$.
  - Multicast $m$.
- At time $(12k + 27) \cdot \Delta$:
  - Set $k := k + 1$ and denote $m_j$ the message received from $P_j$. Let $\ell$ denote the output obtained from $\Pi_{Leader}$.
  - If lock = \infty and $m = \bot$, then $P$ sets $m := m_\ell$.
  - If lock = 0, $P$ outputs $m$ and terminates.
  - If lock = 1, then $P$ sets lock := 0.

**Fig. 5.** A Byzantine agreement protocol $\Pi_{BA}$, where parties share a graded PKI by running the $\Pi_{KeyGrade}$ protocol. $\Pi_{BA}$ internally invokes two other subprotocols: graded byzantine agreement protocol $\Pi_{GBA}$ and a leader election protocol $\Pi_{Leader}$. $N = n + q \cdot (\lfloor \kappa \rfloor - 1)$ (cf Corollary 2). We set $\kappa = n + q$ (for adversarial speedup $\chi = 2$). All parties have access to random oracles $H : \{0, 1\}^* \rightarrow \{0, 1\}^{\kappa + \log(t_p) + 2 \log(\kappa) + 1}$, $H_N : \{0, 1\}^* \rightarrow N$ and an oracle VDF,.$\delta$.

### 5.1 The Protocol ΠLeader

In this section, we present our construction of the leader election protocol $\Pi_{Leader}$ (cf. Figure 6) that relies on a VDF accessed as an oracle. The idea of our protocol is for each party to compute a proof of (sequential) computation on some value that could not not have been predicted before commencing the current run of $\Pi_{Leader}$. More concretely, parties are asked to present an evaluation of VDF at periodic steps in $\Pi_{Leader}$ and spend the time interval between these steps in computing of the next evaluation. Such a chain of sequential evaluations of VDF is only accepted in round $r$ if it accounts for $r$ sequential steps of
computation. In addition, it must be possible to ensure that the computation could not have started before the onset of the first round.

The key observation is that per party, there can be at most one such sequence of evaluations that accounts for the entire duration of $H_{\text{Leader}}$ up to that round (since evaluations of a VDF can not be parallelized). Moreover, the adversary can not predict the sequences produced by honest parties due to its limited budget of computation. Hence, parties can use (hashes of) these sequences as unique (per party) and unpredictable sources of randomness, which can be verified by everyone efficiently. In each election, the party who produces the \emph{smallest} hash, computed according to hash function $H_{\Pi} : \{0,1\}^* \rightarrow \mathbb{N}$, will be elected as the leader in $H_{\text{Leader}}$. Since an honest party sends its hash to everyone and each party produces the smallest hash with the same (uniform) probability, parties agree on the hash of an honest leader as the minimal hash with probability at least $\frac{1}{2}$ in every election. We note that this idea closely mimics the standard approach of electing leaders in Byzantine agreement protocols using verifiable random functions (VRFs). However, to use VRFs, it is required that a trusted dealer generates and distributes the keys or some unpredictable random string at the beginning of the protocol. Since we can not rely on either of these setup assumptions, we choose to instead rely on the above approach that uses VDFs.

![Protocal $H_{\text{Leader}}$](image)

We describe the protocol from the view of an honest party $P$ with $\varepsilon = 1$. Initialize a set $L := \{\}$. $P$ participates (in parallel) in a run of $H_{\text{KeyGrade}}$. (We make this explicit below).

\textbf{Setup Phase}.

- At time 0: In parallel, participate in a run of $H_{\text{KeyGrade}}$.
- At time $13\Delta$: Denote $\phi^0 = \text{VDF}_{13\Delta}(\text{Eval}, \chi, 1)$ the proof computed during the proof of computation phase of $H_{\text{KeyGrade}}$. Call $\text{VDF}_{13\Delta}(\text{Eval}, H_{\Pi}(\phi^0), 1)$ to compute $\phi^1$.
- At time $16\Delta$: Denote $\text{KeySet}$ the set of keys output in $H_{\text{KeyGrade}}$.

\textbf{Leader Election Phase.} Initialize $k := 1$ and $\text{bad}_j := \text{false}$, for all $j \in [n]$ s.t. there exists $(pk_j, g_j) \in \text{KeySet}$. (We refer to the owner of $pk_j$ as $P_j$ below). Denote $\phi_j^0$ the proof of computation received together with $pk_j$ during $H_{\text{KeyGrade}}$. Repeat the following sequence of steps forever:

- At time $(26 + 12(k - 1) \cdot \Delta)$: Upon completing computation of $\phi^k$, call $\text{VDF}_{12\Delta}(\text{Eval}, H_{\Pi}(\phi^k), 1)$ to compute $\phi^{k+1}$. Multicast $\phi^k$.
- At time $(27 + 12(k - 1) \cdot \Delta)$: For all $j \in [n]$, denote $\phi_j^k$ the proof received from party $P_j$ and do:
  - Set $\text{bad}_j := \text{true}$ if nothing was received from $P_j$ in this iteration.
  - If $k = 1$ and $\text{VDF}_{13\Delta}(\text{Verify}, \phi_j^0, H_{\Pi}(\phi_j^0)) = 0$, set $\text{bad}_j := \text{true}$.
  - If $k > 1$ and $\text{VDF}_{13\Delta}(\text{Verify}, \phi_j^{k-1}, H_{\Pi}(\phi_j^{k-1})) = 1$, and $\text{bad}_j$, then $L := L \cup \{H_{\Pi}(\phi_j^k)\}$.
  - If $k > 1$ and $\text{VDF}_{13\Delta}(\text{Verify}, \phi_j^k, H_{\Pi}(\phi_j^{k-1})) = 0$, set $\text{bad}_j := \text{true}$.
  - If $k > 1$ and $\text{VDF}_{12\Delta}(\text{Verify}, \phi_j^k, H_{\Pi}(\phi_j^{k-1})) = 1$ and $\text{bad}_j$, then $L := L \cup \{H_{\Pi}(\phi_j^k)\}$.
- If $k = 1$: Output $\ell$, s.t. $H_{\Pi}(\phi_j^0) = \min L$. Set $k := 2$.
- If $k > 1$: Output $\ell$, s.t. $H_{\Pi}(\phi_j^k) = \min L$. Set $k := k + 1$.

\begin{figure}[h]
\centering
\caption{Leader election protocol $H_{\text{Leader}}$, where all parties have access to random oracles $H : \{0,1\}^* \rightarrow \{0,1\}^{\kappa + \log(p) + 2 \log(n) + 1}$, $H_{\Pi} : \{0,1\}^* \rightarrow \mathbb{N}$ and an oracle VDFs. $H_{\text{Leader}}$ is run for $k \geq 1$ iterations when the byzantine agreement protocol $H_{\text{BA}}$, that invokes $H_{\text{Leader}}$ internally terminates after $k$ iterations. $N = n + q \cdot (|\varepsilon| - 1)$ (c.f Corollary 2). We set $N = n + q$ (for adversarial speedup $\varepsilon = 2$).
\end{figure}
Lemma 10. Let $A$ be a $(q, t_p, 2)$-algorithm, where $q < \frac{n}{3}$ and suppose that $\text{VDF}_{11\Delta}$, $\text{VDF}_{12\Delta}$ and $\text{VDF}_{13\Delta}$ are respectively $(2, \frac{5}{11}, \beta)$, $(2, \frac{1}{11}, \beta)$ and $(2, \frac{2}{11}, \beta)$ sequentiality of $\text{VDF}_{11\Delta}$, $\text{VDF}_{12\Delta}$ and $\text{VDF}_{13\Delta}$ respectively in presence of a $(q, t_p, 2)$ adversary. We defer the corresponding proofs to Appendix D due to space limitations.

Lemma 10. Let $A$ be a $(q, t_p, 2)$-algorithm, where $q < \frac{n}{3}$ and suppose that $\text{VDF}_{11\Delta}$, $\text{VDF}_{12\Delta}$ and $\text{VDF}_{13\Delta}$ are respectively $(2, \frac{5}{11}, \beta)$, $(2, \frac{1}{11}, \beta)$ and $(2, \frac{2}{11}, \beta)$ sequentiality of $\text{VDF}_{11\Delta}$, $\text{VDF}_{12\Delta}$ and $\text{VDF}_{13\Delta}$ respectively in presence of a $(q, t_p, 2)$ adversary. We defer the corresponding proofs to Appendix D due to space limitations.

Our $H_{\text{Leader}}$ protocol is formally described in Figure 6. Below, we elaborate on the two phases of the protocol from the view of (an honest) party $P$ in a bit more detail and give some intuition about them.

Setup Phase. The protocol begins at time 0 with a one-time setup phase which is executed in parallel with a run of $H_{\text{KeyGrade}}$. Recall that $H_{\text{KeyGrade}}$ begins with two rounds of exchanging (unpredictable) challenges among parties. At the end of the second round of exchanges, $P$ creates a hash $\chi$ from these values. During the subsequent proof of computation phase, it then computes $\phi^0 = \text{VDF}_{11\Delta}(\text{Eval}, \{\chi||pk\}, 1)$. When $\phi^0$ becomes available at time $13\Delta$, $P$ immediately starts computing $\phi^1$ as $\text{VDF}_{13\Delta}(\text{Eval}, H_N(\phi^0), 1)$. At time $16\Delta$, $P$ outputs KeySet in $H_{\text{KeyGrade}}$.

Leader Election Phase. The setup phase is followed by a leader election phase, which begins at time $26\Delta$ and is repeated until $P$ terminates $H_{\text{Leader}}$ from within the invocation of $H_{\text{BA}}$. (The time interval between these two phases is spent computing $\text{VDF}_{13\Delta}(\text{Eval}, H_N(\phi^0)), 1)$. $P$ keeps a flag $\text{bad}_j$ (initialized to false) for each key $pk_j$ that it has previously accepted during $H_{\text{KeyGrade}}$. The purpose of $\text{bad}_j$ is to indicate whether $P_j$ (the owner of $pk_j$) has ever stopped investing computational effort during this run of $H_{\text{Leader}}$. Note that since $P$ has accepted $pk_j$ in $H_{\text{KeyGrade}}$, it has already received a proof $\phi^0_j$ associated with $pk_j$. This ensures that computational effort was invested with respect to $pk_j$ up to time $13\Delta$. Below, $k$ denotes the current iteration of the protocol $H_{\text{BA}}$ and is initialized as $k := 1$. The leader election phase now proceeds in following two rounds:

- During the first round, $P$ completes the computation of $\phi^k$ and immediately commences computation of $\phi^{k+1}$ by calling $\text{VDF}_{12\Delta}(\text{Eval}, H_N(\phi^k), 1)$. It multicasts $\phi^k$.
- In the second round, upon receiving $\phi^k_j$ from party $P_j$, $P$ verifies that $P_j$ has continuously been investing computational effort in $H_{\text{Leader}}$. To do so, it checks that $\neg \text{bad}_j$ holds and that $\text{VDF}_{12\Delta}(\text{Verify}, \phi^k_j, H_N(\phi^{k-1}_j)) = 1$. If either of these conditions is violated (or nothing was received from $P_j$), $P$ sets $\neg \text{bad}_j$ to indicate that $P_j$ has broken the chain of continuous computation from the beginning of $H_{\text{Leader}}$ and can never again be trusted as an honest leader.
- $P$ completes the iteration by computing the hash of every proof $\phi^k_j$ that it has received for which $\neg \text{bad}_j$ still holds. It elects the party $P_\ell$ to be the leader if $H_N(\phi^k_\ell)$ was the minimal value among all hashes. (Ties can be resolved by fixing some arbitrary rule in the protocol description).

We say that a party, upon receiving a proof $\phi^k_j$ from another party $P_j$, accepts it, provided it does not set $\text{bad}_j = \text{true}$. With this, we proceed to state Lemmas 10 to 12 below formally. To prove our results, we consider $(2, \frac{5}{11}, \beta)$, $(2, \frac{1}{11}, \beta)$, $(2, \frac{2}{11}, \beta)$ sequentiality of $\text{VDF}_{11\Delta}$, $\text{VDF}_{12\Delta}$ and $\text{VDF}_{13\Delta}$ respectively in presence of a $(q, t_p, 2)$ adversary. We defer the corresponding proofs to Appendix D due to space limitations.

Using the above lemma, we can now view the entire set of proofs that are accepted by at least one honest party in any iteration $k$ of $H_{\text{Leader}}$ as a vector of random variables.
Lemma 11. Suppose that $VDF_{11,\Delta}, VDF_{12,\Delta},$ and $VDF_{13,\Delta}$ are $(2, \frac{5}{11}, \beta), (2, \frac{1}{6}, \beta), (2, \frac{2}{13}, \beta)$ sequential respectively. Then for all $k \geq 1$, the vector $h_k = (\phi_1^k, \ldots, \phi_n^k)$, where $p \leq n$. We now prove that $h_k$ is unpredictable from the view of $A$ before time $((25 + 12(k - 1)) \cdot \Delta)$ for all $k \geq 1$.

Lemma 12. Assume that the conditions of Lemma 11 hold. Set $\beta = 2^{-2\log \kappa - \nu - 2}$. Then with probability at least $\frac{1}{2}$ and for all $1 \leq k \leq \kappa$, all honest parties output $\ell$ in $\Pi_{Leade}$ such that $P_i$ is honest at time $(12(k - 1) + 24) \cdot \Delta$.

6 Communication Complexity in the VDF Model

In this section, we provide the first lower bound for the communication complexity of Byzantine broadcast in the multicast model (c.f. Definition 10) in presence of a VDF oracle. More concretely, inspired by [2, Section 7], we consider a setting without a PKI where parties are connected via multicast channels. In addition – and in contrast to previous work [2] – we assume that parties have access to a VDF oracle. This adds additional technical challenges to the analysis. To state our theorem, we refine our definition of Byzantine broadcast to make failure probabilities and communication complexity of the protocol explicit.

Definition 10 (Byzantine Broadcast in the Multicast Model). Consider a protocol that is executed between $n$ parties, where a designated sender $S$ holds an input $x_S$ at the beginning of the protocol and all parties output upon terminating. We call this a $(q, p)$-secure protocol for Byzantine broadcast with multicast complexity $\Theta$, if the following properties hold (simultaneously) with probability at least $\frac{1}{2}$ when at most $q$ parties are adaptively corrupted:

- **Consistency:** Every honest party $P_i$ outputs the same value $x_i = x$.
- **Validity:** If the sender $S$ is honest, then all honest parties output $x_i = x_S$.
- **Termination:** All parties terminate.
- **Multicast Complexity:** The multicast complexity of the protocol is at most $\Theta$.

Theorem 4. Let $c = O(1)$ and $n \geq (64c^2 + 2c)$. Then there is no $(q, p)$-secure protocol for Byzantine broadcast (among $n$ parties) with $\left[\frac{2\sqrt{3}}{\kappa}\right]$ multicast complexity (relative to a VDF oracle VDF), when $p > \frac{10}{n^2}$ and $q = \frac{n}{2} - c$.

Our proof is inspired by the lower bound of [2] (Section 7) for Byzantine broadcast without a PKI in the multicast model. (Note that our lower bound for Byzantine broadcast implies a lower bound for Byzantine agreement.) The goal of our proof is to show that the view of a special (honest) party $P$ that is not the sender $S$ can be made identical in a protocol execution where the input bit of $S$ is either 0 or 1. This leads to a violation of the consistency property of Byzantine agreement. For the formal proof, we define four worlds: $\text{World}_{c,b}, \text{World}_{c,1-b}, \text{World}_{h,b}$ and $\text{World}_{h,1-b}$. In $\text{World}_{c,s}$, we consider a protocol execution where the special party $P$ is statically corrupted, whereas in $\text{World}_{h,s}$ this party remains honest throughout the protocol. Moreover, in $\text{World}_{c,b}$, the sender has input bit $b$.

To show the violation of the consistency property, we then proceed as follows. In both worlds, $\text{World}_{c,b}$ and $\text{World}_{h,b}$, we show that honest parties have the same view. Moreover,
the special party $P$ acts in both worlds as if it receives messages according to world $\text{World}_{c,b}$ and $\text{World}_{c,1-b}$. Notice that in world $\text{World}_{c,b}$, this can be done since $P$ is statically corrupted, and hence it can be instructed by the adversary to behave accordingly. In world $\text{World}_{h,b}$, this is done by adaptively corrupting parties that multicast messages in a certain round, and instructing the freshly corrupted party to also multicast messages according to the (honest) party $P$'s view of world $\text{World}_{h,1-b}$. (This is also the reason for restricting the number of multICASTs in the theorem, since each multicast requires to corrupt the party that multicast). This confuses $P$ as to which world it is actually being run in, and hence the honest $P$ in worlds $\text{World}_{h,1-b}$ and $\text{World}_{h,b}$ behaves as the malicious party in worlds $\text{World}_{c,1-b}$, $\text{World}_{c,b}$. It is now possible to show that with high probability, $P$'s confusion leads to it outputting an inconsistent bit in one of these worlds.

A crucial difference between our setting and the setting of [2] is the way in which the adversary collapses the views of worlds $\text{World}_{h,1-b}$ and $\text{World}_{h,b}$ from that of $P$. In a nutshell, this requires the adversary to simulate the execution of the protocol in one of these worlds. Unfortunately, when parties have access to a VDF oracle, a simple simulation strategy ceases to work. At a high level, in VDF-based protocols, the simulation depends on oracle queries to the VDF oracle, and hence can only be completed if the adversary has sufficient query budget for the VDF. In our simulation, we achieve this by letting the adversary statically corrupt some set of parties which do not participate in the protocol (the adversary crashes them in every one of the worlds). We can then use their VDF oracle budget to complete the simulation for those parties who do participate.

We defer some details of the proof of Theorem 4 to Appendix F, but provide an overview below.

**Proof (Of Theorem 4).** Suppose for the sake of contradiction, there exists a $(q, p)$-secure protocol $\Pi$ for Byzantine broadcast with $\frac{\sqrt{2q}}{8}$ multicast complexity such that $p > \frac{19}{20}$, $q = \frac{n}{2} - c$, and $n \geq (64c^2 + 2c)$. We proceed by presenting the strategy of an adversary $A$ that violates consistency of $\Pi$ with probability at least $\frac{1}{20}$. Throughout the rest of the description, we denote $q = \left\lfloor \frac{\sqrt{2q}}{8} \right\rfloor$.

We explain $A$’s strategy separately for each of the worlds $\text{World}_{c,b}, \text{World}_{h,b}$ as introduced above. In each of the worlds, the adversary statically corrupts an arbitrary set of $(q - g)$ parties $R$ that do not include $P$ or the sender $S$ at the beginning of the execution of $\Pi$ in that world. These parties behave as if they are crashed (i.e., they never send any messages). We remark that $R$ is fixed through all worlds. Furthermore, let us denote $C$ as the event that two distinct (but possibly dependent) executions of $\Pi$ satisfy validity and consistency and have multicast complexity at most $g$. The following lemma lower bounds the probability of the event $C$.

**Lemma 13.** Let the event $C$ be defined as above. Then $\Pr[C] \geq 2p - 1$.

**Behavior in $\text{World}_{c,b}$:** $A$ statically corrupts the parties in $R$ and uses their computational resources in the simulation of $\text{World}_{c,1-b}$. It corrupts one additional special party $P$ (which is not the sender $S$) and directs $P$ to behave honestly in $\text{World}_{c,b}$ and the simulation as if it were receiving messages from two executions of the protocol in which the sender holds either 0 or 1. $P$’s precise strategy is described below. The remaining $(n + q - g) - 1$ parties remain honest throughout the execution $\text{World}_{c,b}$ (including the sender $S$). Denote this set of parties as $L$.

In more detail, $A$’s strategy is as follows.
- $\mathcal{A}$ chooses random coins for all parties in $\mathcal{L}$ and simulates an execution of $\Pi$ where the sender holds input $1 - b$, the parties in $\mathcal{R}$ are crashed throughout the execution of $\Pi$, and the only other corrupted party is $P$. (In other words, $\mathcal{A}$ simulates $\text{World}_{c,1-b}$.)
- It selects $(q - q)$ parties in the set $\mathcal{L}$ uniformly at random.
- If the simulation directs a party $Q \in \mathcal{L}$ to query VDF, $\mathcal{A}$ instructs a party $Q' \in \mathcal{R}$ to make the same query (unless that party is already waiting VDF to reply to a prior query). When VDF returns $\phi$ to $Q'$, the adversary returns $\phi$ to $Q$ in the simulation.
- The party $P$ behaves as if it receives messages from both $\text{World}_{c,b}$ and the simulation that it is running in its head. It reacts to these messages as an honest party $P$ would do in an execution of $\Pi$ where everybody holds input $b$.
- If $P$ sends a message in $\text{World}_{c,b}$ or in the simulation, $\mathcal{A}$ delivers this message to all honest parties in $\text{World}_{c,b}$ and in the simulation.

Observe that for a small set of $(n - 2q + 2q)$ parties in $\mathcal{L}$, $\mathcal{A}$ is not able to simulate the VDF calls in the simulation (it can only simulate such calls for $|\mathcal{R}|$ many parties in $\mathcal{L}$). Denote this set of parties by $\mathcal{U}$. Clearly, the simulation of the adversary fails if any party from $\mathcal{U}$ attempts to multicast a message in $\Pi$ within the first $q$ multicasts. Let $F_1$ denote the event that simulation of the adversary fails in $\text{World}_{c,b}$ (conditioned on the event $C$). Lemma 14 below bounds the probability $\Pr[F_1|C]$.  

**Lemma 14.** Let $F_1$ denote the event that $A$’s simulation fails in $\text{World}_{c,b}$. Then $\Pr[F_1|C] < \frac{1}{6}$.

Note that since the number of corrupted parties is strictly less than $\frac{n}{2}$, by the validity property of Byzantine broadcast, all the honest parties in $\text{World}_{c,b}$ output bit $b$ with probability greater than $p$ in case the failure event $F_1$ does not occur.

**Behaviour in $\text{World}_{h,b}$:** Initially, $\mathcal{A}$ statically corrupts the parties in $\mathcal{R}$ which will be used as the resource to simulate $\text{World}_{h,1-b}$. The remaining $(n + q - q)$ parties (excluding crashed parties in $\mathcal{R}$) is denoted by set $\mathcal{L}'$. Note that the special party $P$ (which is not the sender $S$) is not among the aforementioned statically corrupted parties and remains honest throughout the protocol, i.e., $P \in \mathcal{L}'$. Note that the sender $S \in \mathcal{L}' \setminus P$.

$A$ now simulates the world $\text{World}_{h,1-b}$ for all parties in set $\mathcal{L}' \setminus \{P\}$ as follows.

- $\mathcal{A}$ chooses random coins for all parties in $\mathcal{L}' \setminus \{P\}$ and simulates $\text{World}_{h,1-b}$. More precisely, it simulates an execution of $\Pi$ where the sender holds input $1 - b$ and the parties in $\mathcal{R}$ are crashed throughout the execution of $\Pi$ (the remaining parties act honestly).
- It selects $(q - q)$ parties in set $\mathcal{L}'$ uniformly at random.
- When the simulation directs a party $Q \in \mathcal{L}' \setminus \{P\}$ to multicast in some round $r$, $\mathcal{A}$ adaptively corrupts $Q$ in round $r$ of the real execution of $\Pi$ (i.e., in $\text{World}_{h,b}$), unless $Q$ is already corrupted. Note that the designated sender $S$ might get corrupted in this step.
- A corrupted party $Q$ continues to send honest messages in $\text{World}_{h,b}$ to all remaining parties in $\mathcal{L}'$ but it forwards to $P$ all messages from both $\text{World}_{h,b}$ and $\text{World}_{h,1-b}$.
- To produce the simulated messages of $\text{World}_{h,1-b}$, when the simulation directs a party $Q \in \mathcal{L}' \setminus P$ to query VDF, $\mathcal{A}$ acts as follows: it instructs a party $Q' \in \mathcal{R}$ to make the same query (unless that party is already waiting for the VDF to reply to a prior query). When the VDF returns $\phi$ to $Q'$, the adversary returns $\phi$ to $Q$ in the simulation.
- When $P$ multicasts a message in $\text{World}_{h,b}$, that message is also multicast in the simulation.

Observe that there is a gap of $(n - 2q + 2\varrho) - 1$ parties for which the adversary could not simulate, but it might be the case that one of these parties want to speak in the protocol. Denote the set of these parties as $U'$. Let $F_2$ denote the event that the simulation of the adversary fails. For $q = (\frac{n}{2} - c)$, Lemma 15 below proves that $\Pr[F_2|C] < \frac{1}{6}$.

**Lemma 15.** Let $F_2$ denote the event that $A$’s simulation fails in $\text{World}_{h,b}$. Then $\Pr[F_2|C] < \frac{1}{6}$.

We now state two technical lemmas below and defer their proofs to Section F so as not to deviate from describing the main proof structure. We first give an indistinguishability lemma about the worlds $\text{World}_{c,b}$ and $\text{World}_{h,b}$ for forever honest parties.

**Lemma 16.** Conditioned on the events $C$ and $\neg F_1$, $\text{World}_{c,b}$ is indistinguishable from $\text{World}_{h,b}$ for parties that are forever honest in both $\text{World}_{c,b}$ and $\text{World}_{h,b}$.

Next, we give an indistinguishability lemma about the worlds $\text{World}_{h,b}$ and $\text{World}_{h,1-b}$ for party $P$.

**Lemma 17.** Conditioned on $C$ and $\neg F_2$, $\text{World}_{h,b}$ is indistinguishable from $\text{World}_{h,1-b}$ for party $P$.

In $\text{World}_{c,b}$, since the sender $S$ is honest, the validity property of Byzantine broadcast implies that all honest parties output $b$. Using the indistinguishability between $\text{World}_{c,b}$ and $\text{World}_{h,b}$ guaranteed by Lemma 16, we can use consistency to ensure that party $P$ which is honest in $\text{World}_{h,b}$, outputs $b$ in world $\text{World}_{h,b}$. We formalize this intuition in the following Lemma (and defer its proof also to Section F).

**Lemma 18.** Let $Y$ denote the event that the forever honest parties in $\text{World}_{h,b}$ output $b$. Then $\Pr[Y|C \cap \neg F_1] = p$.

Similarly, using indistinguishability between $\text{World}_{h,b}$ and $\text{World}_{h,1-b}$ guaranteed by Lemma 17, we obtain:

**Lemma 19.** Let $X$ denote the event that $P$ does not output 1 in $\text{World}_{h,1}$. Then $\Pr[X|C \cap \neg F_2] = \frac{1}{2}$.

Thus, the probability that consistency of the Byzantine broadcast is violated in $\text{World}_{h,1}$ is at least $\Pr[X \cap Y]$ which we bound from below as follows:

$$
\Pr[X \cap Y] = \Pr[X] + \Pr[Y] - 1 \\
\geq \Pr[X \cap C \cap \neg F_2] + \Pr[Y \cap C \cap \neg F_1] - 1 \\
= \Pr[X|C \cap \neg F_2] \cdot \Pr[C \cap \neg F_2] + \Pr[Y|C \cap \neg F_1] \cdot \Pr[C \cap \neg F_1] - 1 \\
= \frac{1}{2} \cdot \frac{5}{6} \cdot (2p - 1) + p \cdot \frac{5}{6} \cdot (2p - 1) - 1 = \frac{20p^2 - 17}{12} \\
= \frac{7}{80} > \frac{1}{20} = 1 - p.
$$

This contradicts the supposition that $II$ achieves consistency with probability more than $\frac{19}{20}$ and within complexity $\varrho$. 


Acknowledgments

This work is supported by the German Research Foundation DFG - SFB 1119 - 236615297 (CROSSING Project S7), by the German Federal Ministry of Education and Research (BMBF) iBlockChain Project (grant nr. 16KIS0902), by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

References

Supplementary Material
A Sequentiality of Our VDF<sub>δ</sub> Oracle

In this section, we will instantiate the VDF<sub>δ</sub> oracle and argue its sequentiality as per Definition 3. To this end, we first recall the definition of Abelian group family from [33] and then introduce some new definitions.

Definition 11 (Abelian Group Family). An Abelian group family \{G_κ\}_{κ∈Z>0}, is a family of probability distributions over finite Abelian groups defined with:

1. An efficient sampling algorithm GGen that, on input 1<sup>κ</sup>, samples uniformly at random a group \( G ∈ G_κ \) and outputs a group description of \( G \) and the identity element \( 1_G \).
2. An efficient sampling algorithm GSample which, given a group description of \( G \), outputs a group element \( x ∈ G \) sampled uniformly at random.
3. Efficient algorithms GMul and GInv that, respectively, multiplies two group elements, and inverts a group element.
4. A group order upper bound \( U(κ) : ∀κ ∃G ∈ G_κ : U(κ) ≥ |G| \), such that \( \log U(κ) ∈ \text{poly}(κ) \) and \( 1/U(κ) ∈ \text{negl}(κ) \).
5. A random group generator count \( n(κ) ∈ Z_{>0} \) and \( n(κ) ∈ \text{poly}(κ) \) such that

\[
\Pr[\mathbf{g} \neq \mathbb{G} | \mathbb{G} \leftarrow G_κ; \mathbf{g} \leftarrow \mathbb{G}^{n(κ)}] \in \text{negl}(κ).
\]

Definition 12 (q-fold Group Algorithm relative to GGen). For any \( κ ∈ N \), we say that \( A \) is a q-fold group algorithm for \( G_κ \), if \( ∀G ∈ G_κ \) (i.e., for all \( G \) in the support of GGen(1<sup>κ</sup>)) it can compute up to \( q \) group operations over \( G \) in parallel. Every group operation is assumed to take unit time. Further, \( A = (A_1,A_2) \) is expressed as a stateful algorithm operating in two phases, where \( A_1 \) denotes the first “preprocessing” phase that outputs \( A_2 \) computing the second “online” phase.

Definition 13 (Generalized One-More Time-Lock Experiment). Fix parameters \( Γ,q,i,δ \), where \( Γ > (i − 1)q \) and \( δ ∈ N \) denotes an arbitrary time. For any \( κ ∈ Z_{>0} \), an Abelian group family \( G_κ \) and any q-fold group algorithm \( A = (A_1,A_2) \), we define the experiment General-TL<sub>Γ,q,i,δ</sub> as follows:

1. Sample \((G,1_G) \leftarrow \text{GGen}(1^κ)\) and \( g := (g_1, \ldots, g_n) \leftarrow G^n \).
2. Run \( A_1 \) in a preprocessing phase on inputs \((G,1_G,g)\). When \( A_1 \) outputs an algorithm \( A_2 \), set \( τ = (i − 1) · q \).
3. Sample a sequence of \( Γ \) elements \( \{X_1, \ldots, X_Γ\} \leftarrow G^Γ \).
4. Run \( A_2 \) in an online phase on inputs \((g,\{X_1, \ldots, X_Γ\})\).
5. When \( A \) outputs the set \( \{(Y_1, X_ℓ_1), \ldots, (Y_ℓ+1, X_ℓ+1)\} \), where \( \{X_1, \ldots, X_ℓ+1\} ⊆ \{X_1, \ldots, X_Γ\}\), return 1 if \( ∀j ∈ [τ + 1], Y_j = X_{j+1}^δ \). Return 0 otherwise.

Denote \( \text{Gen-TL}_{Γ,q,i,δ}^{A,G_κ} \) the random variable associated to the output of General-TL<sub>Γ,q,i,δ</sub>. For some \( β > 0 \), we say that the experiment General-TL<sub>Γ,q,i,δ</sub> is \((t_p, β)\)-hard, if for all q-fold group algorithms \( A = (A_1,A_2) \), running for time at most \( t_p \) in the preprocessing phase and time less than \( i · δ \) in the online phase

\[
\Pr[\text{Gen-TL}_{Γ,q,i,δ}^{A,G_κ} = 1] ≤ β.
\]
Denote $1_G$ as the identity of $G$. Initialize $x := 1_G$ and $r := 1$. Repeat the following $\delta$ many times:
- $b := \lfloor 2r/l \rfloor \in \{0, 1\}$ and $r := 2r \pmod{l}$
- $x := x^b g^b$

Output $\pi := x$.

Fig. 7. Procedure $\text{Alg}(g, l, \delta)$

Wesolowski’s VDF [35]

Parameters:
- $\text{Primes}(2\kappa)$ is the set of the first $2^{2\kappa}$ prime numbers.
- $H_{\text{prime}} : \{0, 1\}^* \rightarrow \text{Primes}(2\kappa)$ is a random oracle.
- $d$ is a (large) negative square free integer such that $d \equiv 1 \pmod{4}$.
- $G = \text{Cl}(d)$ is a class group of an imaginary quadratic field $\mathbb{Q}(\sqrt{d})$.
- $H_G : \{0, 1\}^* \rightarrow G$ is a random oracle.

Proof Generation. On input $(\text{Eval}, s)$ do:
- $g \leftarrow H_G(s), y := g^2, l := H_{\text{prime}}(g, s)$.
- Compute $\pi := g^{2^\delta b} = \text{Alg}(g, l, \delta)$, as defined in Figure 7.
- Output $\phi := (y, \pi)$.

Proof Verification. On input $(\text{Verify}, \phi = (y, \pi), s)$ do:
- $g \leftarrow H_G(s), l := H_{\text{prime}}(g, s)$. Set $r = 2^\delta \pmod{l}$.
- If $\pi^r g^r = y$, output 1. Output 0 otherwise.

Fig. 8. Construction of $\text{VDF}_\delta$ via Wesolowski’s VDF.

A.1 Proof of VDF Sequentiality

We recall Wesolowski’s VDF construction based on class groups [35] in Figure 8 and show that it satisfies the sequentiality property as stated in Definition 3. To this end, we first instantiate the group family $G_\kappa$ in Definitions 11 to 13 above with the class groups as described in Figure 8. In particular, we define $G_\kappa := \{\text{Cl}(d) : d \in d_\kappa\}$, where $d_\kappa$ is family of (sufficiently large) negative square free integers $d$ with $d \equiv 1 \pmod{4}$ and $\text{Cl}(d)$ are class groups of an imaginary quadratic field $\mathbb{Q}(\sqrt{d})$ based on some $d \in d_\kappa$. We now proceed to describe the proof of VDF sequentiality in Lemma 20 below.

Lemma 20. If General-TL$^{R, G_\kappa}_{\Gamma, \delta}$ is $(\Gamma, \beta)$-hard, then the VDF$_\delta$ construction from Figure 8 is $(i, \beta)$-sequential as per Definition 3.

Remark 1. Before proving Lemma 20, we want to highlight that we choose to instantiate the group family with class groups, and the VDF itself from [35] as in Figure 8. This is because it does not require any trusted setup assumptions which is beneficial for our main protocols in Figures 2, 5 and 6. However, we also emphasize that the same proof for Lemma 20 with minor changes will also hold for the RSW based VDF from [31], though this requires a trusted setup assumption to generate the initial modulus $N$ for our main protocols.
Proof. Assume \( A \) to be a \((q, \Gamma, 1)\)-algorithm (as per Definition 1)\(^4\) that breaks the \((i, \beta)\)-sequenceability of \( \text{VDF} \) from Figure 8 (as per Definition 3). We then construct a \( q \)-fold group algorithm \( \mathcal{R} = (\mathcal{R}_1, \mathcal{R}_2) \) that internally executes \( A \) and breaks the \((\Gamma, \beta)\) hardness of the General-TL\(^6\) experiment. In particular, the \( q \)-fold group algorithm \( \mathcal{R} = (\mathcal{R}_1, \mathcal{R}_2) \) simulates \( \text{VDF}_\delta \) (as per Figure 8) and the oracles \( H_G, H_{\text{prime}} \) to \( A \). Note that since \( \mathcal{R} \) can compute \( q \) group operations in parallel, it can indeed run the \((q, \Gamma, 1)\)-algorithm \( A \) internally. Here, we assume \( A \) is able to evaluate \( \text{VDF}_\delta \) at most \( q \) many times in parallel.

In more detail, for an arbitrary time \( T^5 \) and some \( k \in \mathbb{Z}_{>0} \), consider a \((k, T, \epsilon)\)-unpredictable vector \( S \) (as per Definition 2). For a security parameter \( \kappa \), the General-TL\(^6\) experiment samples \( (G, 1_G) \leftarrow \text{GGen}(1^n), g \leftarrow G^n \) and sends it to \( \mathcal{R}_1 \). On input \((G, g)\), \( \mathcal{R}_1 \) fixes the oracles \( H_G, H_{\text{prime}} \) implicitly and simulates them along with the VDF\(_\delta \) oracle to \( A \). When \( \mathcal{R}_1 \) outputs \( \mathcal{R}_2 \), General-TL\(^6\) sends \((g, \{X_1, \ldots, X_T\})\) to \( \mathcal{R}_2 \) where \( \{X_1, \ldots, X_T\} \leftarrow G^T \). \( \mathcal{R}_2 \) sends \( S \) to \( A \) at time \( T \). Note that \( A \) can issue hash oracle queries both before and after time \( T \). Accordingly, \( \mathcal{R} \) simulates such queries as follows.

**Simulation of** \( H_G \) **and** \( H_{\text{prime}} ** Simulation of** \( H_{\text{prime}} \) is trivial. For all oracle calls to \( H_G \) before time \( T \), simulate \( H_G \) in a straightforward manner – on any input, return a uniform output and maintain a table of all input/output tuples to answer consistently to repeated queries. \( \mathcal{R}_1 \) executes at most \( T \) many steps in the preprocessing phase, since \( A \) can execute the same by definition. For oracle calls after time \( T \), proceed as follows. \( \mathcal{R}_2 \) gets its challenges in the online phase as \((G, \{X_1, \ldots, X_T\})\) from General-TL\(^6\) and stores them in a list as \( L := \{X_1, \ldots, X_T\} \). On some input \( S' \) to \( H_G \), sample a generator \( X \leftarrow L \), set \( X := H_G(S') \) and update the table of \( H_G \) accordingly. Update list \( L \) as \( L := L \setminus X \). There are two possible cases to consider:

**Case 1:** \( A \) outputs \( S_j \in \mathcal{S} \) for some \( j \in [k] \) before time \( T \). In this case, \( H_G \) has not been programmed to output an element \( X \in \{X_1, \ldots, X_T\} \) in case \( A \) queries it on an input that depends on \( S \). Hence, when \( A \) returns output \( \{\phi_1, \ldots, \phi_{T+1}\} \), \( \mathcal{R} \) cannot use it to win its own game. Since \( S \) is considered to be \((k, T, \epsilon)\)-unpredictable, this happens with probability at most \( \epsilon \).

**Case 2:** \( A \) outputs some \( S_j \in \mathcal{S} \) for some \( j \in [k] \) after time \( T \). In this case, \( \mathcal{R} \) has used one of the elements \( X \in \{X_1, \ldots, X_T\} \) to “program” the oracle \( H_G \) on input any component of \( S \). When \( A \) outputs \( \{\phi_1, \ldots, \phi_{T+1}\} \), where \( \forall j \in [\tau+1], \text{VDF}_\delta(\text{Verify}, \phi_j, S'_j) = 1 \) using less than \( i \cdot \delta \) time, \( \mathcal{R} \) can return \( \{(Y_{\tau+1}, X_{\tau+1}) : (Y_1, X_1), \ldots, (Y_\tau, X_\tau) \} := \{(\phi_1, X_1), \ldots, (\phi_{\tau+1}, X_{\tau+1})\} \) as an answer to its own game, where \( \{X_1, \ldots, X_{\tau+1}\} \subseteq \{X_1, \ldots, X_T\} \). Since (parallel) time used in the online phase is less than \( i \cdot \delta \), \( \mathcal{R} \) wins the experiment General-TL\(^6\).

Thus, we have in case 2
\[
\Pr_{\phi_1, \ldots, \phi_{T+1} \leftarrow \mathcal{A}_{H_G, H_{\text{prime}}, \text{VDF}_\delta}} \left[ \forall j \in [\tau + 1]: \text{VDF}_\delta(\text{Eval}, S'_j, 1) = \phi_j \right] \leq \Pr_{\text{Gen-TL}_{G, \delta}} \leq \beta
\]

Since case 1 occurs with probability at most \( \epsilon \), combining the two cases we have
\[
\Pr_{\phi_1, \ldots, \phi_{T+1} \leftarrow \mathcal{A}_{H_G, H_{\text{prime}}, \text{VDF}_\delta}} \left[ \forall j \in [\tau + 1]: \text{VDF}_\delta(\text{Eval}, S'_j, 1) = \phi_j \right] \leq \beta + \epsilon.
\]
A.2 Generalized One-More Time-Lock Experiment in SAGM

Katz et al. [24] introduced and defined the strong algebraic group model (SAGM) as a stronger variant of the algebraic group model (AGM). Baarsen and Stevens [33] recently extended this model to include their setting of finite Abelian groups, where they provided several results on connections between various hardness assumptions. In particular, they identified a generic template for showing connections between various problems on any finite Abelian group family in any given Abelian hidden-order group model (AHO-GM), where GM can be instantiated with standard model (SM) or AGM or SAGM.

In this section, we analyse the hardness of the generalized one-more time-lock problem from Definition 13 in the SAGM. In particular, we reduce the hardness of the multiple of group order (MO) problem to that of the General-TL$_{\Gamma,q,i,\delta}^{A,G}$ in the AHO-SAGM for any general Abelian group family $\mathcal{G} = \{G_\kappa\}_{\kappa \in \mathbb{Z}_{>0}}$. To this end, we first recall some necessary definitions and lemmas from [33] below.

**Definition 14 (Strongly algebraic algorithm).** An algorithm $A$ over a group $G$ is called strongly algebraic if it has one or more output rounds (between which it may perform arbitrary local computation). An output round is called algebraic if it contains one or more group elements. For each group element $X$ it outputs, it must also output a tuple of one of the following forms:

1. $(X,X_1,X_2) \in G^3$ such that $X = X_1 \cdot X_2$, where $X_1,X_2$ were either provided as input to $A$ or were output by $A$ in some previous step(s).
2. $(X,X_1) \in G^2$ such that $X = X_1^{-1}$, where $X_1^{-1}$ was either provided as input to $A$ or were output by $A$ in some previous step(s);

The algebraic running time of $A$ is the number of algebraic steps it takes, and is denoted by aTime($A$).

**Definition 15 (Multiple of Group Order (MO)).** For any $\kappa \in \mathbb{Z}_{>0}$, an Abelian group family $\mathcal{G}_\kappa$, consider the experiment MO$_{\kappa}^{A,G}$ w.r.t. any algorithm $A$:

1. Sample $(G,1_G) \leftarrow \text{GGen}(1^\kappa)$ and $g := (g_1,\ldots,g_n) \leftarrow G^n$.
2. Run $A(G,1_G,g)$ to get an output $N$.
3. Return 1, if $(N \neq 0 \land N \equiv 0 \pmod{|G|})$. Else, return 0.

Then, MO$_{\kappa}^{A,G}$ is $(t,\epsilon)$-hard, if for all $A$ running in time $t$, $\Pr[\text{MO}_{\kappa}^{A,G} = 1] < \epsilon$.

**Notation.** We introduce some notations for the lemmas that we describe next. Let $g = (g_1,\ldots,g_n) \in G^n$ where $G \in \mathcal{G}_\kappa$ is some Abelian group family. For any vector $v = (v_1,\ldots,v_n) \in \mathbb{Z}^n$ and matrix $A = [a_1,\ldots,a_n] \in \mathbb{Z}^{n \times n}$, define $g^v = \prod_{i=1}^{n} g_i^{v_i}$ and $g^A := (g^{a_1},\ldots,g^{a_n})$ respectively. By det($A$), we mean the determinant of the square matrix $A$.

We now recall the definition of a relation from [33]. Given a system of generators $g = (g_1,\ldots,g_n) \in G^n$ from a finite Abelian group $(G,1_G)$, we call any vector $e = (e_1,\ldots,e_n) \in \mathbb{Z}^n$ a “relation” for $g$ when $g^e = 1_G$. Baarsen and Stevens [33] provided a generic template for reducing the MO problem (Definition 15) to another computational problem (say $G$) on any finite Abelian group in the AHO-GM. We recall this template in the SAGM in Lemma 21 below.

**Lemma 21 ([33, Lemma 5.4]).** Let $\mathcal{G} = \{G_\kappa\}_{\kappa \in \mathbb{Z}_{>0}}$ be an Abelian group family. Let $G$ be some computational game which, given $\kappa$, is based on sampling a group $G \leftarrow G_\kappa$ and
elements \( g = (g_1, \ldots, g_n) \leftarrow \mathbb{G}^n \) uniformly at random. Let \( \text{Rel}^A \) be a relation sampler that takes as input a group \( G \in \mathbb{G}_n, g = (g_1, \ldots, g_n) \in \mathbb{G}^n \), and has oracle access to an adversary \( A \) for game \( G \). Assume \( \text{Rel}^A \) satisfies the following properties for any given adversary \( A \) in the SAGM:

1. \( \text{Rel}^A(G, g) \) outputs \( \perp \) (failure) or a relation \( e \) such that \( g^e = 1_G \).
2. When \( G = \langle g \rangle \), each execution of \( \text{Rel}^A(G, g) \) has an independent and identical success probability \( p'_{G, g} \) with \( |p'_{G, g} - p_G| \in \text{neg}(\kappa) \), where \( p_G := \text{Adv}^G_{G, A} |G(\kappa) \) denotes the advantage of \( A \) in \( G \) conditioned on the event \( G \leftarrow \mathbb{G}_n \).
   Essentially, it says that every execution of \( \text{Rel}^A \) outputs with a probability \( p'_{G, g} \) that is independent of all other executions. Further, \( p'_{G, g} \) is separated negligibly apart from the advantage of \( A \) winning in \( G \), conditioned on the random choice of \( G \leftarrow \mathbb{G}_n \).
3. For \( G = \langle g \rangle \), given \( n \) relation outputs \( \{e_1, \ldots, e_n\} \) from \( n \) independent and successful executions of \( \text{Rel}^A(G, g) \), it holds that \( \Pr[\det(E) = 0] \in \text{neg}(\kappa) \), where \( E := [e_1, \ldots, e_n] \in \mathbb{Z}^{n \times n} \).
4. The (expected) runtime of \( \text{Rel} \) is asymptotically equivalent to that of \( A \).

Then the following holds: \( \text{MO}^\text{Rel}_{\mathbb{G}_n} \) is \((t, \epsilon)-\text{hard}\) implies \( G^A \) is \( (t', \epsilon')-\text{hard} \), where \( t, t' \) and \( \epsilon, \epsilon' \) are polynomially related.

Remark 2. The original statement of Lemma 21 in [33, Lemma 5.4] relates the hardness of \( \text{MO} \) to that of the group-theoretic problem \( G \) with some more parameters: a constant \( S \geq 4 \), \( C_S \geq 1 \) (is a function of \( S \)) and \( p = \text{Adv}^G_{G, A} \) is the probability that defines the adversary’s advantage in \( G \). Based on this, the underlying relation sampler \( \text{Rel} \) needs to be invoked \( [Sn/p] \) times in order to sample \( n \) linearly-independent relations with overwhelming probability. However, our final proof for Lemma 23, that reduces the hardness of \( \text{MO} \) to that of \( \text{General-TL}_{F, q, \delta} \), follows mostly from that of the [33, Theorem 8.2] with similar arguments. Thus, we avoid stating these parameters explicitly in Lemma 21 for brevity and will only provide an informal statement with a proof sketch for Lemma 23.

Before showing the final reduction in Lemma 23, we recall [33, Lemma 8.1] that helps to bound the size of the representation coefficients of the output elements of strongly algebraic algorithms.

**Lemma 22 ([33, Lemma 8.1]).** Let \( G \) be a finite Abelian group and let \( g = (g_1, \ldots, g_n) \) be a tuple of elements of \( G \). Let \( A \) be any strongly algebraic algorithm running in at most \( t \) rounds on input \( g \) and \( X = g^r = \prod_{i=1}^n g_i^{r_i} \) for \( r = (r_1, \ldots, r_n) \in \mathbb{Z}_{\geq 0}^n \) (i.e., \( \text{ATime}(A(g, X)) \leq t \)). Let \( Y \) be any output of \( A \) and let \( (Y_s, Y_{s, 1}, Y_{s, 2}) \) or \( (Y_s, Y_{s, 1}) \) be the corresponding tuples for each element \( Y_s \) being output at round \( s \in [t] \). (Note that \( A \) is in fact allowed to output arbitrary many tuples in each round, but we can always pick a path of sequential computation leading to \( Y \).) Then the following two statements hold.

1. The generalized discrete logarithm \( \text{DLog}_A(g, Y) \) w.r.t. \( g \) and \( A \), can be recursively computed as follows:
   - \( \text{DLog}_A(g, g_i) = 1_i \) (the vector with a 1 in the \( i \)-th place and 0 on all others) for \( i \in [n] \), \( \text{DLog}_A(g, X) = r \);
   - For \( s = 1, \ldots, t \), let
     \[
     \text{DLog}_A(g, Y_s) = \begin{cases} 
     \text{DLog}_A(g, Y_{s, 1}) + \text{DLog}_A(g, Y_{s, 2}), & \text{if } Y_s = Y_{s, 1}Y_{s, 2} \\
     -\text{DLog}_A(g, Y_{s, 1}), & \text{if } Y_s = Y_{s, 1}^{-1} 
     \end{cases}
     \]

2. The generalized discrete logarithm $d = (d_1, \ldots, d_n) := \text{DLog}_A(g, Y)$ satisfies $|d_i| < 2^k_r, \forall i \in [n]$.

**Lemma 23 (Informal).** $\text{MO}^{\text{Rel}}_{\Gamma, \delta}$ is $(t, \epsilon)$-hard implies $\text{General-TL}^{\text{A}, \Gamma, \delta}$ is $(t', \epsilon')$-hard, where $t, t'$ and $\epsilon, \epsilon'$ are polynomially related.

**Proof (Sketch).** Let $A = (A_1, A_2)$ be a $q$-fold group algorithm (adversary) breaking the $\text{General-TL}^{\text{A}, \Gamma, \delta}$ experiment, where $A_1$ runs in the standard model on input $(G, g)$ during the preprocessing phase and produces $A_2 \leftarrow A_1(G, g)$ which runs in the strong algebraic group model during the online phase. In particular, $A_2$ uses $i \cdot \delta - 1$ parallel algebraic steps (or less) in the online phase. Using the generic template from Lemma 21, we then construct an adversary $B^{A_1}$ against $\text{MO}^{\text{Rel}}_{\Gamma, \delta}$, that takes as input $(G, g)$ for $G \in \Gamma, g \in G^n$. Accordingly, we define a relation sampler $\text{Rel}^{A_1}$ similar to [33] in Figure 9 below, where we assume that the variable state keeps an account of the internal variables across all the subroutines in all the invocations and denote $t_k := \text{ATime}(A_2(\hat{g}_k, X_k))$.

### Fig. 9. Relation sampler $\text{Rel}$ for Lemma 23

For any $k \in [[Sn/p]]$, let $(\hat{g}_k, X_k = \{X^k_1, \ldots, X^k_T\})$ be the input to $A_2$ in the online phase. $A_2$, being a $q$-fold group algorithm, can compute $q$ algebraic steps in parallel. Thus, to generate $\tau + 1 = (i - 1) \cdot q + 1$ proofs in $i \cdot \delta - 1$ or less steps, $A_2$ must evidently compute...
at least one of these proofs in a way that deviates from the honest computation (i.e., as prescribed via the interface of the VDF).

When \( \mathcal{A}_2 \) outputs \( \left( X^k_j, Y^k_j, \left( Y^k_j|\ell|\right)_{s=1}^{t_j} \right) \), the reduction \( B^{A_1} \) picks \( j \in [\tau + 1] \) as the first such index w.l.o.g. for which \( t_j = \text{ATime}(A_2(\mathbf{g}_k, X_k)) < \delta \). In this case, the correctness of DLog algorithm from Lemma 22 enforces that the vector \( z_k := (2^dA_k r^k_j - A_k d_k) \) returned by Ext forms a relation w.r.t. \( \mathbf{g} \), thus proving Lemma 21-(1). Further, conditioned on the event that \( G = \langle \mathbf{g} \rangle \) (that holds by our assumption in Lemma 21), the instances \((\tilde{g}_k, X_k = \{X^k_\gamma\}_{\gamma \in \Gamma})\) have negligible statistical distance to the uniform distribution on \( \mathbb{G}^{n+\Gamma} \). This ensures Lemma 21-(2), i.e., each execution of Rel\( ^d \)(\( G, \mathbf{g} \)) has independent and identical success probability. Note that the time complexity for each call to Rel\( ^d \)(\( G, \mathbf{g} \)) is asymptotically dominated by that of \( \mathcal{A} \) up to a factor of at most \( \Gamma \). Thus, so long as \( \Gamma \) is some \( \text{poly}(\kappa) \), their runtimes are asymptotically equivalent, and this proves Lemma 21-(4).

Lemma 21-(3) follows similar to that of [33, Theorem 8.2]. That is, we assume w.l.o.g. that \( \mathcal{A} \) succeeds on all instances \( k \in [n] \). Then, for the \( k \)-th successful run, there exists a smallest index \( j \in [\tau + 1] \) with \( t_j < \delta \) and \( Y^k_j = (X^k_{\ell_j})^2 \) for some \( \ell_j \in [\Gamma] \). As stated before, \( \text{Ext}(G, \mathbf{g}, \text{state}) \) returns the relation vector \( z_k = (2^dA_k r^k_{\ell_j} - A_k d_k) \) accordingly.

Recall \( r^k_{\ell_j} = \{r^k_v, \ell_j\}_{v \in [n]} \), \( d_k = (d_{kv})_{v \in [n]} \) are vectors of length \( n \). We drop the coefficients \( j \in [\tau + 1], \ell_j \in [\Gamma] \) from now on for brevity. Thus, for \( A_k = [a^k_1, \ldots, a^k_n] = [a^k_{u,v}]_{u,v \in [n]} \) we express \( a^k_{u,v} = a_{u,v}^\kappa \cdot O_v + a_{u,v}^\kappa \) where \( O_v := |\langle g_v \rangle| \) is defined as the order of the Abelian group generated by \( g_v \). With this, we expand the coefficients of \( z_k = (z_{k,u})_{u \in [n]} \) as:

\[
z_{k,u} = \sum_{v=1}^{n} a_{u,v}^\kappa \cdot \zeta_{k,v} \cdot O_v + \sum_{v=1}^{n} a_{u,v}^\kappa \cdot \zeta_{k,v}, \quad \text{where} \quad \zeta_{k,v} = (2^\delta r^k_v - d_{kv}).
\]

Lemma 22 and \( \text{ATime}(A_2(\tilde{g}_k, X_k)) < \delta \) ensures that \( \zeta_{k,v} \neq 0, \forall k \in [n] \). We can now split the above coefficients as \( \tilde{z}_{k,u} + \hat{z}_{k,u} \), with

\[
\tilde{z}_{k,u} := a_{1,v}^\kappa \cdot \zeta_{k,1} \cdot O_v \quad \text{and} \quad \hat{z}_{k,u} := a_{1,v}^\kappa \cdot \zeta_{k,1} + \sum_{u \neq 1} a_{u,v}^\kappa \cdot \zeta_{k,v}
\]

The rest of the proof follows exactly similar to [33, Theorems 7.2, 6.5].

\[\square\]

B Deferred Proofs for \( \Pi_{\text{KeyGrade}} \) Protocol

Proof of Lemma 1.

**Proof.** It is easy to verify that all honest parties’ keys are assigned the highest grade 2 by all other honest parties from the protocol description. Hence, the protocol satisfies graded validity (for any number of corrupted parties). \[\square\]

Proof of Lemma 2.

**Proof.** Let \( P_i \) and \( P_j \) be honest parties. We show that if \( P_j \) assigns grade 2 to some party \( P_i \)'s public key \( pk_i \), then \( P_i \) assigns \( pk_i \) at least grade 1. Since \( (pk_i, 2) \in \text{KeySet}_j \) in the second round of the Key Grading Phase, \( P_j \) must have received a correctly formed message.
(rank[|2, pk_i, \chi_i, \phi_i, d_i]) from P_i, i.e., such that VDF_{\delta}(\text{Verify}, \phi_i, \{\chi_i, pk_i\}) = 1, and that \chi_i depends on P_i's first round challenge d_j, i.e., \chi_i = H(d_i) and d_j \in d_i. Thus, P_j multicasts the message (rank[|1, pk_i, \chi_i, \phi_i, d_i, c_j]) to all parties at the beginning of the third round of the Key Grading Phase. P_i receives this message by the end of the third round of the Key Grading Phase and has, at that point, already assigned pk_j grade 2. It is then able to verify that P_j's message is correctly formed and depends on c_i (by performing P_j's checks and the additional checks c_i \in c_j and d_j = H(c_j)). Hence, it assigns pk_j the grade 1 at this point, unless it has previously assigned it the grade 2.

C Deferred Proofs for \Pi_{GBC}, \Pi_{GBA} Protocols

Proof of Lemma 4.
Proof. If the sender S is honest, then it multicasts x with a valid signature \langle x \rangle_S at time 0. Let P be an honest party. From the protocol description, P generates a valid countsignature as \langle \langle x \rangle_S \rangle_P and multicasts it at time \Delta. At time 2\Delta, P collects valid countsignatures of the form \langle \langle x \rangle_S \rangle_k from all parties P_k with g_k = 2 into its set of signatures \psi_P(x). Note that due to the graded validity property of the underlying graded PKI, if P_i is honest, then P_i's public key has grade 2 for P_i, i.e., (pk_i, 2) \in KeySet_P. As there are at least \frac{N}{2} honest parties, this implies that P's signature set \psi_P(x) contains at least \frac{N}{2} countsignatures from honest parties. Moreover, since honestly generated countsignatures are valid in the view of each honest party, this implies that \psi_P(x) is a consistent signature set in the view of every honest party. Due to the unforgeability of underlying digital signature scheme, there is no valid countsignature of the form \langle \langle x' \rangle_S \rangle_k, where g_k = 2 and x' \neq x. Hence, P multicasts a set \psi_P(x) at time 2\Delta which consistent in the view of every honest party.

Proof of Lemma 5.
Proof. Let us assume that at time 3\Delta, honest party P_i receives at least \frac{N}{2} signature sets which are all consistent with some value x. This implies that among them, there is at least one set \psi_P(x) which was sent by an honest party P at time 2\Delta. Now, towards contradiction, suppose honest party P_j receives the set \psi_P'(x') of countsignatures from party P' such that \psi_P'(x') is weakly consistent with x' \neq x. By definition, \psi_P'(x') contains at least \frac{N}{2} weakly valid countsignatures on x' from distinct outer signers. Among these \frac{N}{2} outer signers, there is at least one honest party P'. Since P' would have sent the countsignature \langle \langle x' \rangle_S \rangle_P, at time \Delta, all honest parties would have received it by time 2\Delta. Hence, no honest party would have sent a consistent set of signatures for x at time 2\Delta. This contradicts that P sends such a set at time 2\Delta.

Proof of Lemma 6.
Proof. If S is honest, then by Lemma 4 every honest party P multicasts a signature set \psi_P(x) at time 2\Delta which is consistent with x. At time 3\Delta, every honest party hence receives at least \frac{N}{2} such signature sets. Moreover, by Lemma 5, no honest party receives a set of signatures that is weakly consistent with some x' \neq x. This implies that all honest parties output (x, 2).

Proof of Lemma 7.
Proof. Let $P$ be a honest party that outputs $(x, 2)$. We want to show that every honest party $P'$ outputs $(x, 2)$. Since $P$ outputs grade 2 for value $x$, it receives at least $\frac{N}{2}$ signature sets from distinct parties, which are all consistent with $x$. One of these sets must have been sent by an honest party $P^*$ and received by time $3\Delta$. Therefore, $P'$ must have also received this set $\psi_{P^*}(x)$ from $P^*$ by time $3\Delta$. From the graded consistency property of the underlying graded PKI, each countersignature that is valid in $P^*$'s view, is weakly valid in each honest parties view. This implies that $\psi_{P^*}(x)$ is a weakly consistent signature set in view of party $P'$. By Lemma 5, $P'$ receives no signature set that is weakly consistent with $x' \neq x$. Hence, $P'$ outputs $(x, 2)$ or $(x, 1)$.

\hspace{1cm} \blacksquare

Proof of Lemma 8

Proof. If $P_i$ output $v_i$ with grade 2, this means $P_i$ must have set $|S_{v_i}| \geq \frac{N}{2}$. Due to the graded consistency property of $\Pi_{\text{GBC}}$, $P_j$ must have $|S_{v_j}| \geq \frac{N}{2}$, such that $v_j = v_i$. This implies $P_j$ outputs value $v_j = v_i$ with grade at least 1.

\hspace{1cm} \blacksquare

Proof of Lemma 9

Proof. Due to the graded validity property of $\Pi_{\text{GBC}}$, the result follows immediately.

\hspace{1cm} \blacksquare

D Deferred Proofs of $\Pi_{\text{Leader Protocol}}$

Proof of Lemma 10.

Proof. From the arguments of Lemma 3 and due to $(2, \frac{s}{13}, \beta)$ sequentiality of $\text{VDF}_{11\Delta}$, $A$ computes at most $[(2 + \frac{5 \cdot \beta}{13})]q = 2q$ proofs of the form $\phi_i^0 = \text{VDF}_{11\Delta}(\text{Eval}, \chi_i, \tau)$, $(\tau \leq 2)$, earliest at time $11\Delta$. Next, $A$ can make at most $2q$ different calls of the form $\text{VDF}_{13\Delta}(\text{Eval}, H_{Q}(\phi_i^0), \chi)$ at $11\Delta$, which returns $2q$ proofs of the form $\phi_i^1$ at $24\Delta$. An honest party computes its $\phi_i^0$ and $\phi_i^1$ at time $13\Delta$ and $26\Delta$ respectively. Due to the $(2, \frac{2}{13}, \beta)$ sequentiality of $\text{VDF}_{13\Delta}$, $A$ cannot complete any additional proof $\phi_i^1$ within the remaining time $(26 - 24)\Delta = 2\Delta$. Applying the same argument inductively for $k \geq 2$ and $\text{VDF}_{12\Delta}(\text{Eval}, H(\phi_i^{k-1}), \tau)$ completes the proof.

\hspace{1cm} \blacksquare

Proof of Lemma 11.

Proof. We prove this statement by induction on $k$. From the arguments of Lemma 3, we know that the probability of a $(q, t_p, 2)$-algorithm guessing any element of $h_0$ is at most $2^{-\kappa - 2\log(\kappa) - 1} + \beta$ before (earliest) time $11\Delta$. Thus, $h_0$ is $(n - q, 11\Delta, 2^{-\kappa - 2\log(\kappa) - 1} + \beta)$-unpredictable and the base case $k = 1$ follows directly from combining $(2, \frac{2}{13}, \beta)$-sequentiality of $\text{VDF}_{13\Delta}$ with $(n - q, 11\Delta, 2^{-\kappa - 2\log(\kappa) - 1} + \beta)$-unpredictability of $h_0$. For the step case, assume that $h_k$ is $(n - q, (25 + 12(k - 1)) \cdot \Delta, 2^{-\kappa - 2\log(\kappa) - 1} + (k + 1) \cdot \beta)$-unpredictable. Combining this with $(2, \frac{1}{5}, \beta)$-sequentiality of $\text{VDF}_{12\Delta}$ immediately yields that $h_{k+1}$ is $(n - q, (25 + 12(k - 1)) \cdot \Delta, 2^{-\kappa - 2\log(\kappa) - 1} + (k + 2) \cdot \beta)$-unpredictable.

\hspace{1cm} \blacksquare

Proof of Lemma 12.
Proof. For \( k \geq 1 \) denote \( E_k \) the event that \( A \) queries \( H \) on an element of \( h_k \) before time \((12(k - 1) + 25) \cdot \Delta \). By Lemma 11, \( E_k \) occurs with probability at most \( 2^{-\kappa-2 \log(k) - 1} + (k + 1) \cdot \beta \). By a union bound, \( E := \bigcup_{1 \leq k \leq \kappa} E_k \) occurs with probability at most \( 2\kappa^2 \cdot \beta + \kappa \cdot 2^{-\kappa-2 \log(k) - 1} \leq 2 \cdot 2^{-\kappa - 1} \leq 2^{-\kappa} \). Unless \( E \) occurs, the values \( H_N(h_{k,i}) \) are uniformly random values in the range \([N]\) from the view of \( A \) for all \( 1 \leq i \leq N \) at time \((12(k - 1) + 25) \cdot \Delta \). Hence, conditioned on \( \neg E \), the component \( \ell \) which minimizes \( H_N(h_{k,i}) \) corresponds to an honest party with probability at least \( \frac{n - q}{N} \), where \( N = n + q \). Overall, \( \ell \) corresponds to an honest party at time \((12(k - 1) + 24) \cdot \Delta \) with probability at least \( \frac{n - q}{N} = \frac{n - q}{n + q} \geq \frac{1}{2} \) (since \( q < \frac{n}{3} \)).

\( \square \)

E Proof of Theorem 2.

Proof. Let \( P \) be an honest party with input \( m_P \) and let us refer to an execution of the main 12-round protocol loop as an iteration. It is easy to see that \( lock \) (initially set to \( \infty \)) is a monotonically decreasing value. Moreover, once \( P \) sets \( lock := 1 \) in iteration \( I \), it terminates the protocol by the end of iteration \( I + 2 \) and the value of \( m \) which \( P \) holds at this point in time (i.e., when setting \( lock = 1 \)) remains unchanged until the protocol terminates. Now let us proceed with the proof of validity, consistency and termination properties of \( \Pi_{GBA} \) protocol below.

Proof of Validity. Suppose that immediately prior to an iteration \( I \), each honest party \( P_i \) holds the same value \( v \). Then, each honest party runs first iteration of \( \Pi_{GBA} \) on input \( v \). Due to the validity property of \( \Pi_{GBA} \), each honest party \( P_i \) outputs \((v, 2)\), hence sets \( lock := 1 \) (unless of course, it has already set \( lock := 0 \)) and \( m := v \). By the above argumentation, each honest party outputs \( v \) by the end of iteration \( I + 2 \) at the latest. This already suffices to prove validity of the protocol, since if all honest parties start the protocol with the same value \( v \), then the first iteration is indeed such an iteration.

Proof of Consistency. To prove consistency, we argue that if an honest party \( P \) is the first party to set \( lock := 1 \) after the first invocation of \( \Pi_{GBA} \) in some iteration \( I \), then every honest party terminates at the end of iteration \( I + 2 \), at the latest, with output \( v \). Consider the first iteration \( I \) in which some honest \( P \) sets \( lock := 1 \) after outputting \((v, 2)\) in the first invocation of \( \Pi_{GBA} \). Due to the consistency property of \( \Pi_{GBA} \), every other honest party must output \( v \) with at least grade 1, setting \( m := v \). Hence, every honest \( P_j \) inputs \( v \) to the second invocation of \( \Pi_{GBA} \) in iteration \( I \). Again due to the validity property of \( \Pi_{GBA} \), each honest party outputs \( v \) with grade 2 after second invocation of \( \Pi_{GBA} \) in iteration \( I \). Since \( I \) is the first iteration in which some honest party sets \( lock := 1 \), no honest party terminates at the end of \( I \). Observe now that each honest party holds the same value \( v \) 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 \( I + 3 \) (and with the same output \( v \)). The above ensures consistency of the protocol.

Proof of Termination. It remains to argue that the protocol terminates within expected constant rounds and with overwhelming probability after \( \kappa \) rounds. Consider the scenario where an honest leader \( P_i \) is elected and each honest \( P_i \) has still set \( lock := \infty \) in some iteration \( I \). If all honest parties have \( m = \bot \) after the second round of \( \Pi_{GBA} \) invocations,
then each party $P_i$ holds the same value $m$ at the end of iteration $I$, because $P_i$ was honest at the time where it multicast and all honest parties use $P_i$’s value. Now let’s say, there is an honest party $P$ that has lock = $\infty$ and set $m = v$ after the second $\Pi_{GBA}$ invocation. Consider another honest party $P_j$ that either has $m_j = v_j$ or $m_j = \bot$. Due to the graded consistency property of $\Pi_{GBA}$, if $m_j = v_j$, then $v_j = v$. If $m_j = \bot$, then later $P_j$ sets $m_j = m$. But again from the graded consistency property of $\Pi_{GBA}$, since $P$ was honest when it multicast $m$, $m = m$. Thus, by the end of iteration $I$, each honest party has updated their value to $m^\ell = m$.

It is easy to see that the expected number of iterations until an honest leader is elected is 2 (by the properties of $\Pi_{Leader}$). Thus by the above, the protocol terminates within an expected two iterations and terminates with probability $1 - 2^{-\kappa}$ after $\kappa$ iterations.

\[\square\]

F Deferred Proofs of Theorem 4

Proof of Lemma 13.

**Proof.** Let $A_1$ and $A_2$ denote the events that two (possibly dependent) executions (labeled one and two for the purpose of this lemma) of protocol $\Pi$ (in any of the worlds) achieve Byzantine Broadcast and have $\varrho$ multicast complexity. By assumption, $\Pr[A_1] \geq p$, and hence $\Pr[C] = \Pr[A_1 \cap A_2] \geq \Pr[A_1] + \Pr[A_2] - 1 = 2p - 1$.

Proof of Lemma 14.

**Proof.** To bound the probability of event $F_1$ (conditioned on $C$), we first define the following events. Let $S$ denote the event that a uniformly chosen party $P_i \in \mathcal{L}$ ever multicasts in the simulation. Conditioned on $C$, the simulation directs parties to multicast at most $\varrho$ many times. Since party $P_i$ is chosen uniformly from the set $\mathcal{L}$, $\Pr[S|C] = \frac{\varrho}{|\mathcal{L}|}$. Now, observe that the set $\mathcal{U}$ is a uniformly created subset of set $\mathcal{L}$ chosen by the adversary according to its simulation strategy. By the previous calculation, the probability that any particular party in set $\mathcal{U}$ ever multicasts in the simulation coincides with the probability of event $S$. Now, let $T_1$ denote the event that at least party uniformly chosen from set $\mathcal{U}$ ever multicasts in the protocol. By a union bound, we see that

$$\Pr[T_1|C] \leq \sum_{i=1}^{\left|\mathcal{U}\right|} \Pr[S|C].$$

Hence, with multicast complexity up to $\varrho$, $\Pr[T_1|C] \leq \sum_{i=1}^{\left|\mathcal{U}\right|} \Pr[S|C]$. Since $F_1$ can only occur as a result of an unsimulated party attempting to multicast when we have conditioned on $C$, we can infer that $\Pr[F_1|C] = \Pr[T_1|C]$.

Now,

$$\Pr[F_1|C] \leq \sum_{i=1}^{\left|\mathcal{U}\right|} (\Pr[S|C]) = \varrho \cdot |\mathcal{U}| \cdot \frac{n - q + \varrho}{4} \cdot \frac{n - q + \varrho}{n - q + \varrho} \cdot \frac{n - q + \varrho}{n - q + \varrho} \cdot \frac{n - q + \varrho}{n - q + \varrho} \leq \frac{\sqrt{\varrho}(n - 2t + \sqrt{\varrho})}{(n - q + \varphi)}.$$
By setting $t = \frac{n}{2} - c$, for $c = O(1)$, we bound \((n-2t+\sqrt{\frac{n}{2}})\) as
\[
\frac{(2c + \sqrt{n-2c})}{\frac{n}{2} + c + \sqrt{n-2c}} = \frac{(2c + \sqrt{n-2c})}{\frac{n}{2} - c + \sqrt{n-2c} + 2c} = \frac{\sqrt{n-2c}}{\frac{n}{2} + \sqrt{n-2c} + 2c}
\]
\[
= \frac{\sqrt{n-2c}}{(4\sqrt{n} - 2c + 1 + \frac{16c}{\sqrt{n-2c}})} < \frac{2(\frac{8c}{\sqrt{n-2c}} + 1)}{(4\sqrt{n} - 2c + 1 + \frac{4c}{\sqrt{n-2c}})} < \frac{2}{3\sqrt{n}} = \frac{2}{3\sqrt{n}},
\]
where the last inequality holds for $n \geq (64c^2 + 2c)$. By substituting $q = \left(\frac{n}{2} - c\right)$ in $\sqrt{\frac{n}{2}}$, we finally obtain $\Pr[F_1|C] \leq \frac{\sqrt{n-2c}}{8} \cdot \frac{4}{3\sqrt{n}} = \frac{\sqrt{n-2c}}{6\sqrt{n}} < \frac{1}{6}$.

The proof of Lemma 15 is similar to that of 14.

**Proof of Lemma 16.**

*Proof.* The statement holds due to the following reasons. 1) In $\text{World}_{c,b}$, the forever honest parties always receive honest messages. 2) In $\text{World}_{c,b}$, parties that are adaptively corrupted by $A$ send correct messages to the honest parties except for party $P$. 3) The behavior of the corrupted party $P$ in $\text{World}_{c,b}$ is exactly like that of honest party $P$ in $\text{World}_{b,b}$. 4) The multicast complexity in both worlds $\text{World}_{c,b}$ and $\text{World}_{b,b}$ is at most $\varrho$, according to the definition of the event $C$. Therefore, conditioned on events $C$ and $\neg F_1$, the views of the parties that are forever-honest in $\text{World}_{c,b}$ and $\text{World}_{b,b}$ are identically distributed.

**Proof of Lemma 17.**

*Proof.* The statement of the lemma holds due to the following reasons. 1) In $\text{World}_{b,b}$, $P$ always receives messages of both worlds ($\text{World}_{b,0}$ and $\text{World}_{b,1}$) from adaptively corrupted parties. 2) Both the worlds $\text{World}_{b,0}$ and $\text{World}_{b,1}$ have multicast complexity at most $\varrho$ according to the definition of the event $C$. Therefore, conditioned on events $C$ and $\neg F_2$, the views of the party $P$ in the worlds $\text{World}_{b,0}$ and $\text{World}_{b,1}$ are identically distributed.

**Proof of Lemma 18.**

*Proof.* Since the sender $S$ is honest in $\text{World}_{c,b}$, by the validity property of the Byzantine Broadcast, all the forever honest parties output $b$ with at least probability $p$ in $\text{World}_{c,b}$. From Lemma 16, $\text{World}_{c,b}$ is indistinguishable from $\text{World}_{b,b}$ for forever honest parties. We infer that $\Pr[Y|C \cap \neg F_1] \geq p$.

**Proof of Lemma 19.**

*Proof.* By Lemma 17, conditioned on events $C$ and $\neg F_2$, the views of the party $P$ in the worlds $\text{World}_{b,0}$ and $\text{World}_{b,1}$ are identically distributed. Therefore, conditioned on the events $C$ and $\neg F_2$, the probability of $P$ not outputting 1 in $\text{World}_{b,1}$ is given as $\Pr[X|C \cap \neg F_2] \geq \frac{1}{2}$.