Rewardable Naysayer Proofs*

Gennaro Avitabile [1], Luisa Siniscalchi [2], and Ivan Visconti [3]

[1]IMDEA Software Institute, Madrid, Spain. gennaro.avitabile@imdea.org
[2]Technical University of Denmark (DTU), Copenhagen, Denmark. luisi@dtu.dk
[3]Sapienza University of Rome, Rome, Italy. ivan.visconti@uniroma1.it

Abstract

Combining verifiable computation with optimistic approaches is a promising direction to scale blockchain applications. The basic idea consists of saving computations by avoiding the verification of proofs unless there are complaints.

A key tool to design systems in the above direction has been recently proposed by Seres, Glaeser and Bonneau [FC'24] who formalized the concept of a Naysayer proof: an efficient to verify proof disproving a more demanding to verify original proof.

In this work, we discuss the need of rewarding naysayer provers, the risks deriving from front-running attacks, and the failures of generic approaches trying to defeat them. Next, we introduce the concept of verifiable delayed naysayer proofs and show a construction leveraging proofs of sequential work, without relying on any additional infrastructure.

1 Introduction

In most blockchains the cost of a transaction for the end user depends on the amount of storage and computations of the invoked on-chain programs. To make an on-chain application successful and scalable, developers are therefore highly incentivized to minimize such costs for the end users. To this end, verifiable computation [20] has become a very popular tool. Verifiable computation outsources an expensive computation to an untrusted party which provides the result together with a *succinct* non-interactive proof (SNARK) that the result is correct. In the blockchain setting, this leads to a system where smart contracts verify proofs, which is much more efficient than executing the computations themselves.

An important application of this paradigm is zk-rollups (e.g., [22, 29]). A zk-rollup is a scaling solution that involves batching several transactions into a *single* one that updates the state of the blockchain to the one resulting from the execution of all the transactions. To guarantee that the state is not compromised, a succinct proof that the new state is correct is also provided by a rollup coordinator. However, verifying such proofs is often wasteful, as in blockchain applications an incentive mechanism can always be put in place to punish whoever publishes incorrect proofs. Therefore, optimistic paradigms [25, 5, 28] assuming that coordinators are unlikely to publish incorrect information on-chain seem a promising avenue to reduce costs.

Seres et al. [28] recently proposed a new paradigm called *naysayer proofs*. In naysayer proofs, the verifier (i.e., the smart contract) optimistically accepts a proof without verifying it, opening a dispute period in which the proof can be rejected and the state reverted. Instead, off-chain validators will verify the proof and, if it is incorrect, they will submit a *naysayer proof* certifying its *incorrectness*. The naysayer verifier checks the naysayer proof and, if it is correct, rejects the original proof. We call these off-chain validators

*This work will appear in the proceedings of Financial Cryptography and Data Security 2025 with the title: "Short Paper: Rewardable Naysayer Proofs" [6].

naysayer provers. The advantage of naysayer proofs is two-fold: first, in most cases, the naysayer proof will not be generated at all since the sole fact that they can be generated disincentivizes attempts of generating incorrect proofs; second, should the proof be challenged, naysayer proofs are more efficient to verify than the original proofs.

Naysayer proofs vs fraud proofs. A concept related to naysayer proofs is the one of *fraud proofs* that is used in the context of *optimistic rollups* [5, 25]. Fraud proofs operate under the optimistic assumption that a computation is correct unless someone challenges it. In case of a challenge, (some of) the steps of the computation are re-executed to resolve the dispute. Although naysayer proofs and fraud proofs are conceptually similar, there are some differences coming from the fact that what is disputed is a proof about a computation rather than the computation itself. In naysayer proofs, unlike fraud proofs, dispute resolution is *witness independent* [28]. Indeed, for fraud proofs, the entire input of the computation (i.e., the witness) must be made available at least to the challenger. On the other end, to verify a naysayer proof only the original statement and proof are required, which are usually much shorter than the witness.

As observed in [28], witness-independent resolutions can lead to considerable savings for large witnesses such as transaction data of a rollup. For example, consider a zk-rollup batching several transactions in a succinct commitment (e.g., a Merkle tree) and a SNARK certifying that the committed transactions are correct and produce a certain final state. Due to the succinctness of these cryptographic tools, a single rollup transaction could batch hundreds of thousands of transactions. Such a proof can be extremely resource intensive and the rollup coordinator might actually implement it via hundreds of commitment/SNARK pairs where each pair deals with thousands of transactions and, according to the optimistic approach, is provisionally accepted. The advantage of naysayer proofs over fraud proofs becomes clear: every rollup update could contain a large number of transactions while a corresponding naysayer proof consisting of simply pointing to a single commitment/SNARK pair is fast to verify; nevertheless, unlike optimistic rollups, the corresponding transactions, thanks to witness independence, do not need to be made available to verifiers and validators.

Naysayer proofs and front-running attacks. To work in a permissionless setting, the naysayer paradigm requires an incentive mechanism ensuring that participation is profitable for the naysayer provers. The naysayer provers who post correct naysayer proofs should be rewarded and should be penalized in case they transmit incorrect proofs. Additionally, whoever posts a proof which is later proved incorrect by a naysayer prover must also be penalized[1].

Generally, all applications in which any user can obtain a reward by submitting a transaction before other participants are targets of *front-running* attacks [17, 19]. Here the front-runner uses knowledge of unprocessed transactions to get her transactions executed ahead of the unprocessed ones. Therefore, when considering systems rewarding naysayer provers, the attacker could steal the naysayer proof from the transactions pool and rush the publication of her naysayer proof to cash the reward, without doing the hard work of a naysayer prover (i.e., without validating huge amounts of original proofs).

Solutions to front-running. Several techniques with various trade-offs have been proposed to mitigate generic and specific front-running attacks [7]. A common approach consists of batching blinded transactions, so that all transactions submitted within a confirmation period are independent from each other, preventing any adversarial ordering of the transactions. For instance, *commit and reveal* approaches [8, 15, 4, 1] involve a first round where all inputs are committed and a second round where such commitments are opened so that rewards can be paid accordingly. When implemented with standard commitments, this technique requires multiple rounds of interaction between users and the blockchain. Additionally, this approach requires one to enforce the concept of *epoch*. That is, there must be a fixed time interval during which users are allowed to submit committed inputs. After this epoch has expired, no new committed inputs are allowed, and users can only submit openings for previously registered commitments. Some scenarios (e.g., auctions) have clearly

[1]The reward for posting correct naysayer proofs must be lower than the penalty given to users who post incorrect proofs. Otherwise, one could profit by posting an incorrect proof and shortly after a corresponding naysayer proof.

defined epochs, while for some others it might not be immediately clear how to define an epoch[2]. Finally, while preventing front-running attacks, the commit-and-reveal approach introduces selective opening attacks that can be mitigated through additional complexity such as the use of timed-release cryptography (e.g., timed commitments [10] or delay encryption [11]), ensuring that the inputs can always be recovered after a certain time, or by relying on threshold encryption [12] and a committee where the majority of the members is assumed to be honest, or by a multi-party computation of an anonymous committed broadcast functionality [23].

Ensuring delay to prevent front-running. Another method to prevent front-running does not blind the transactions' inputs, but it instead focuses on making the creation of a valid transaction slow [13, 27]. The central tool in this approach are verifiable delay functions (VDFs) [9]. A VDF takes a prescribed amount of time t (even in the presence of massive parallelism) to compute and can be verified in time much less than t. The overall idea is to tie the input of the VDF to the transaction as well as some user's identifying information. A transaction is then considered valid only if it comes with a proof of the VDF. Therefore, every time attackers want to front-run a transaction, they have to start a long-running computation. The time parameter t is set so that by the time the attacker finalizes her transaction, the transaction of the honest user will already be confirmed in the blockchain. The advantage of this approach is that it does not require a user to submit a second transaction. Unfortunately, the above generic use of a VDF to defeat front-running attacks suffers from the following technical caveat: the sequentiality of a VDF is only guaranteed when it is evaluated on a random input. This generally requires an external trusted infrastructure providing a publicly verifiable randomness beacon [14]. It is therefore interesting to study scenarios where no trusted external infrastructure (i.e., no additional points of failure) is available and still front-running scenarios can be defeated via delays.

1.1 Our Contribution

In this paper, we study the problem of front-running attacks against rewarding mechanisms associated to naysayer proofs. We make the following contributions:

- *Front-running resistant naysayer proofs:* we define the notion of verifiable delayed naysayer proofs (VDNs). VDNs are naysayer proofs that when plugged in a natural reward-based system are resilient to front-running attacks. We design a compiler which, starting from proofs of sequential works (PoSWs) [24] and a naysayer proof with minimal additional requirements, gives a VDN. Our solution does not add any setup assumption and, being tied to the setting of naysayer proofs, it does not require randomness beacons (or any other type of external infrastracture).

- *Generic front-running protection approaches can be harmful:* We show that applying a generic front-running protection technique can hurt the requirements of the underlying reward-based system. In particular, we observe that naysayer proofs have a crucial efficiency requirement (i.e., they should be faster to verify than the original proofs) that is not necessarily preserved when a front-running protection approach is applied on top of them.

Verifiable delayed naysayer proofs. In VDNs, the naysayer prover takes as input an additional time parameter t and a prover-chosen watermark μ. The naysayer verifier also takes such additional values as input. Apart from completeness and soundness, we require the VDN to be sequential. By sequential we mean that given a sufficiently unpredictable statement/proof pair (stm, π) that can be naysayed, it is unfeasible to compute an accepting naysayer proof for (stm, π), with a watermark μ, in less than time t. This holds even when the adversary is provided with oracle access to fast generation of VDN proofs, except of course on (stm, π, μ)[3]. When using the VDN in a reward-based setting, it suffices to additionally verify that the

[2]In the context of naysayer proofs, an intuitive way to define an epoch is to set it identically to the dispute period. However, this restriction will delay the punishment of the misbehaving original prover for an unnecessary amount of time.

[3]This requirement is similar to the one of watermarkable VDFs [21, 31], with the difference that the output is not unique.

prover-chosen watermark matches with the public key which submits the transaction. Indeed, sequentiality guarantees that even if the front-runner $\mathcal{F}$ sees a VDN proof of an honest user $\mathcal{H}$ (w.r.t. her public key $\mathsf{pk}_{\mathcal{H}}$), $\mathcal{F}$ will not be able to produce a valid proof (w.r.t. her public key $\mathsf{pk}_{\mathcal{F}}$) before the transaction submitted by $\mathcal{H}$ is finalized in the blockchain.

Our VDN. We combine naysayer proofs with PoSWs to construct a VDN. PoSWs are publicly verifiable proofs attesting that a certain amount of sequential work was performed starting from a specific challenge. VDFs can be seen as a special case of PoSWs offering the additional guarantee of a unique output. We observe that PoSWs suffice for front-running prevention. Indeed, we do not need a unique output since such output is merely verified to ensure that a long sequential computation was carried out, but the result of the computation is not used elsewhere. Unlike VDFs, we know PoSWs that are unconditionally secure in the random oracle model and do not require any setup [24, 16, 3, 18, 2].

Our construction is very simple: the prover of the VDN runs the underlying naysayer prover over (stm, π) and gets a naysayer proof π_{nay}. Then, it calls a random oracle (RO) over $(\pi_{\mathsf{nay}}||\mathsf{stm}||\pi||\mu)$ to get the challenge χ for the PoSW. The PoSW π_{PoSW} is then computed w.r.t. χ and time parameter t. The VDN proof simply consists of both π_{nay} and π_{PoSW}, which are then individually verified by the verifier of the VDN. The caveat of this solution is that the security of PoSWs is formulated for uniformly random χ. Even if χ is derived from a RO, χ is a random string only if $(\pi_{\mathsf{nay}}, \mathsf{stm}, \pi, \mu)$ has high min-entropy. Generally, it is unclear how to find a high min-entropy source for all the users of a system.

Whenever a reward is connected to the punishment of another party, as in the naysayer setting, it is actually possible to find a source of high min-entropy. For example, in the rollup application discussed above, stm is a succinct representation (e.g., a cryptographic hash) of several transactions, which likely contain cryptographic material with high min-entropy such as digital signatures. Furthermore, there are generally many valid proofs π for the same stm. The malicious rollup coordinator is incentivized to use good entropy to create (stm, π), so that this pair is distributed similarly to honest proofs, and thus less likely to be verified by a naysayer prover. Moreover, we can include in π the digital signature used to authorize the rollup transaction. On an intuitive level, such a signature must be unpredictable; otherwise it would not be unforgeable and an attacker could damage the coordinators posting wrong proofs on their behalf.

Front-running and efficiency in naysayer proofs. The verification algorithm of our VDN is slower than the one of the underlying naysayer proof system since it additionally verifies a PoSW. Therefore, to prove that our VDN is a naysayer proof system we have to assume that the verification algorithm of the PoSW is asymptotically faster than the verification algorithm of the original proof[4].

This is not a problematic assumption, since the verification time of state-of-the-art PoSWs [16, 18, 3] scales logarithmically in t and as a small polynomial in the security parameter. However, this highlights that not all front-running compilers are necessarily compatible with every reward-based system. For example, one could artificially create a PoSW whose verification time depends on a high-degree polynomial in the security parameter while still being logarithmic in t, making it asymptotically slower than the verification of the original proof. The resulting construction would still prevent front-running attacks but it would also make the naysayer reward-based system useless since verifying regular proofs would be less expensive than verifying naysayer proofs.

Practical considerations. Both VDFs and PoSWs are good candidates to enforce delays in our VDN. One can pick between the two depending on several trade-offs. VDFs usually require setups and number-theoretic assumptions [26, 31], but have more compact proofs [31]. PoSWs do not require setups and are based on symmetric primitives, but have larger proofs [16, 18, 3, 2]. Another measure to consider is the gap between the actual work done by the honest prover and the level of sequential security that the primitive gives. In VDFs, the honest prover has to do more work than t *sequential* computations, while in PoSWs this gap can be reduced to 0 [18, 2]. Obviously, as in all applications of timed primitives, estimating the right

[4]Recall that the verification of a naysayer proof must be faster than the verification of the original proof.

value of t is very challenging. However, we feel that this issue will naturally be better understood as these primitives get used in practice.

Future work. While we focused on defining and constructing a VDN, we think it is important to investigate the practicality of our solutions. We leave to future work a performance evaluation study on how verification gas costs would increase when implementing our construction with state-of-the-art PoSWs, as well as a study to quantify the concrete parameters of the timed primitives.

2 Definitions and Tools

We start with the definition of non-interactive proof, then we report the definition of naysayer proof of [28] and then we define PoSWs similarly to [18].

Definition 1 *A non-interactive proof system Π for some NP relation $\mathcal{R}$ consists of the following PPT algorithms:*

- $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$*: on input a security parameter λ, output* crs*.*
- $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{stm}, w)$*: on input* crs*,* stm*, and w, output π.*
- $0/1 \leftarrow \mathsf{Verify}(\mathsf{crs}, \mathsf{stm}, \pi)$*: on input* crs*,* stm*, and π output $1/0$ to accept/reject.*

We require Π to be correct and sound, we defer to [30] for formal definitions.

Definition 2 *Given a non-interactive proof system $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for some NP relation $\mathcal{R}$, the corresponding naysayer proof system Π_{nay} consists of the PPT algorithms defined as follows. We define τ to be a tuple $(\mathsf{crs}, \mathsf{stm}, \pi)$, where* crs*,* stm*, π are a common reference string, a statement, and a proof of the proof system Π. We call τ a naysayer tuple.*

- $\mathsf{crs}_{\mathsf{nay}} \leftarrow \mathsf{NSetup}(1^\lambda)$*: on input security parameter 1^λ, output* $\mathsf{crs}_{\mathsf{nay}}$*.*
- $\pi_{\mathsf{nay}} \leftarrow \mathsf{Naysay}(\mathsf{crs}_{\mathsf{nay}}, \tau)$*: on input* $\mathsf{crs}_{\mathsf{nay}}$*, and a naysayer tuple τ, output π_{nay}.*
- $0/1 \leftarrow \mathsf{NVerify}(\mathsf{crs}_{\mathsf{nay}}, \tau, \pi_{\mathsf{nay}})$*: on input* $\mathsf{crs}_{\mathsf{nay}}$*, a naysayer tuple τ, and π_{nay}, outputs 1 to accept and 0 to reject.*

We define the polynomial-time language $\mathcal{L}^{\Pi}_{\mathsf{nay}} := \{\tau = (\mathsf{crs}, \mathsf{stm}, \pi) : \Pi.\mathsf{Verify}(\mathsf{crs}, \mathsf{stm}, \pi) = 0\}$ and require the following properties to hold:

- *Naysayer Completeness: a naysayer proof system Π_{nay} w.r.t a non-interactive proof system Π is complete if for all $\tau \in \mathcal{L}^{\Pi}_{\mathsf{nay}}$ it holds that:*

$$\Pr\left[\mathsf{NVerify}(\mathsf{crs}_{\mathsf{nay}}, \tau, \pi_{\mathsf{nay}}) = 1 \;\middle|\; \begin{array}{l} \mathsf{crs}_{\mathsf{nay}} \leftarrow \mathsf{NSetup}(1^\lambda); \\ \pi_{\mathsf{nay}} \leftarrow \mathsf{Naysay}(\mathsf{crs}_{\mathsf{nay}}, \tau) \end{array}\right] = 1.$$

- *Naysayer Soundness: a naysayer proof system Π_{nay} w.r.t a non-interactive proof system Π is sound if for all PPT adversaries $\mathcal{A}$ it holds that:*

$$\Pr\left[\begin{array}{l} \mathsf{NVerify}(\mathsf{crs}_{\mathsf{nay}}, \mathsf{crs}, \tau) = 1 \wedge \\ \tau \notin \mathcal{L}^{\Pi}_{\mathsf{nay}} \end{array} \;\middle|\; \begin{array}{l} \mathsf{crs}_{\mathsf{nay}} \leftarrow \mathsf{NSetup}(1^\lambda); \\ (\pi_{\mathsf{nay}}, \tau) \leftarrow \mathcal{A}(\mathsf{crs}_{\mathsf{nay}}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

- *Efficiency:* $\mathsf{NVerify}$ *must be asymptotically faster than* Verify*.*

Definition 3 *A proof of sequential work (PoSW) is a pair of PPT oracle-aided algorithms* $\mathsf{Prove}, \mathsf{Verify}$*, executed by a prover* P *and a verifier* V *as follows:*

Common Input: P *and* V *get as common input* $\lambda, t \in \mathbb{N}$. *All parties have access to a random oracle* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$.

Statement: V *samples a random statement* $\chi \leftarrow \{0,1\}^\lambda$ *and sends it to* P.

Prove: P *computes* $\pi \leftarrow \mathsf{Prove}^{\mathsf{H}}(\chi, t)$ *and sends* π *to* V.

Verify: V *computes and outputs* $\mathsf{Verify}^{\mathsf{H}}(\chi, t, \pi)$.

We require the PoSW to be complete, sound, and succinct as follows:

- *Completeness: for all* $\lambda, t \in \mathbb{N}$, *and all* $\chi \in \{0,1\}^\lambda$ *it holds that*

$$\Pr\left[\mathsf{Verify}^{\mathsf{H}}(\chi, t, \pi) = 1 \;\middle|\; \pi \leftarrow \mathsf{Prove}^{\mathsf{H}}(\chi, t)\right] = 1.$$

- *Soundness: a PoSW is sound if for all* $\lambda, t \in \mathbb{N}$ *and* $\alpha > 0$, *for all* $\mathcal{A}$ *that make* $(1-\alpha)t$ *sequential*[5] *queries to* H, *it holds that*

$$\Pr\left[\mathsf{Verify}^{\mathsf{H}}(\chi, t, \pi) = 1 \;\middle|\; \chi \leftarrow \{0,1\}^\lambda, \pi \leftarrow \mathcal{A}^{\mathsf{H}}(\chi)\right] \leq \mathsf{negl}(\lambda).$$

- *Succinctness: for all* $\lambda, t \in \mathbb{N}$ *and every honestly generated proof* π *for parameter* t, *we have* $|\pi| \leq \mathsf{poly}(\lambda, \log t)$, $\mathsf{Prove}^{\mathsf{H}}$ *runs in time*[6] $\mathsf{poly}(\lambda, t)$, *and* $\mathsf{Verify}^{\mathsf{H}}$ *runs in time* $\mathsf{poly}(\lambda, \log t)$.

3 Verifiable Delayed Naysayer Proofs

Definition 4 *Given a non-interactive proof system* $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for some NP relation* $\mathcal{R}$, *the corresponding verifiable delayed naysayer proof system* Π_{VDN} *consists of the oracle-aided PPT algorithms defined as follows:*

- $\mathsf{crs}_{\mathsf{VDN}} \leftarrow \mathsf{VDNSetup}(1^\lambda)$*: on input security parameter* 1^λ, *output a common reference string* $\mathsf{crs}_{\mathsf{VDN}}$.
- $\pi_{\mathsf{VDN}} \leftarrow \mathsf{VDNaysay}^{\mathsf{H}}(\mathsf{crs}_{\mathsf{VDN}}, t, \tau, \mu)$*: on input common reference string* $\mathsf{crs}_{\mathsf{VDN}}$, *time parameter* t, *naysayer tuple* τ, *and watermark* μ, *output a proof* π_{VDN}.
- $0/1 \leftarrow \mathsf{VDNVerify}^{\mathsf{H}}(\mathsf{crs}_{\mathsf{VDN}}, t, \tau, \mu, \pi_{\mathsf{VDN}})$*: on input common reference string* $\mathsf{crs}_{\mathsf{VDN}}$, *time parameter* t, *naysayer tuple* τ, *watermark* μ, *and a proof* π_{VDN}, *output* 1 *to accept or* 0 *to reject.*

Completeness, soundness, and efficiency are analogous to the ones of Def. 2. while $\mathcal{Z}$-Watermarked Sequentiality *is defined as follow: For all* $\lambda, t \in \mathbb{N}$ *and* $\alpha > 0$, *and for an auxiliary input distribution* $\mathcal{Z}$, *we define the following game applied to an adversary* $\mathcal{A} := (\mathcal{A}_0, \mathcal{A}_1)$*:*

$$\mathsf{crs}_{\mathsf{VDN}} \leftarrow \mathsf{VDNSetup}(1^\lambda), \mathsf{st} \leftarrow \mathcal{A}_0^{\mathsf{H}}(\mathsf{crs}_{\mathsf{VDN}}, t), \tau \leftarrow \mathcal{Z}(\mathsf{crs}_{\mathsf{VDN}}, t)$$

$$(\pi^*_{\mathsf{VDN}}, \mu^*) \leftarrow \mathcal{A}_1^{\mathsf{H}(\cdot), \mathsf{VDNaysay}^{\mathsf{H}}(\mathsf{crs}_{\mathsf{VDN}}, t, \cdot, \cdot)}(\mathsf{st}, \tau)$$

We say that $\mathcal{A}$ *wins the game if* $\mathsf{VDNVerify}^{\mathsf{H}}(\mathsf{crs}_{\mathsf{VDN}}, t, \tau, \mu^*, \pi^*_{\mathsf{VDN}}) = 1$ *and* $\mathcal{A}$ *did not query the oracle over* (τ, μ^*). Π_{VDN} *satisfies* $\mathcal{Z}$*-watermarked sequentiality if no pair of randomized algorithm* $\mathcal{A}_0$ *which runs in total time* $\mathsf{poly}(t, \lambda)$, *and* $\mathcal{A}_1$ *that makes at most* $(1-\alpha)t$ *sequential queries to* H, *can win the above game with probability greater than* $\mathsf{negl}(\lambda)$.

[5] Notice that no restriction is posed on the number of *parallel* queries $\mathcal{A}$ can make. In a nutshell, the budget of sequential queries gets reduced whenever a new query includes the output of a previously asked RO query as a sub-string.

[6] To simplify the notation, in accordance to [18], we give t as input to all the algorithms. However, we implicitly mean that, whenever required to comply with the efficiency requirements, the time parameter t is passed in unary.

Our PoSW-based construction. Our construction works as follows:

- $\mathsf{crs}_{\mathsf{VDN}} \leftarrow \mathsf{VDNSetup}(1^\lambda)$: $\mathsf{crs}_{\mathsf{nay}} \leftarrow \Pi_{\mathsf{nay}}.\mathsf{Setup}(1^\lambda), \mathsf{crs}_{\mathsf{VDN}} := \mathsf{crs}_{\mathsf{nay}}$.
- $\pi_{\mathsf{VDN}} \leftarrow \mathsf{VDNaysay}^{\mathsf{H}}(\mathsf{crs}_{\mathsf{VDN}}, t, \tau, \mu)$: parse $\mathsf{crs}_{\mathsf{VDN}} := \mathsf{crs}_{\mathsf{nay}}$, $\pi_{\mathsf{nay}} \leftarrow \Pi_{\mathsf{nay}}.\mathsf{Naysay}(\mathsf{crs}_{\mathsf{nay}}, \tau)$, $\chi = \mathsf{H}(\pi_{\mathsf{nay}}||\tau||\mu)$, $\pi_{\mathsf{PoSW}} \leftarrow \mathsf{PoSW}.\mathsf{Prove}^{\mathsf{H}}(\chi, t)$, $\pi_{\mathsf{VDN}} := (\pi_{\mathsf{nay}}, \pi_{\mathsf{PoSW}})$.
- $0/1 \leftarrow \mathsf{VDNVerify}^{\mathsf{H}}(\mathsf{crs}_{\mathsf{VDN}}, t, \tau, \mu, \pi_{\mathsf{VDN}})$: parse $\mathsf{crs}_{\mathsf{VDN}} := \mathsf{crs}_{\mathsf{nay}}$ and $\pi_{\mathsf{VDN}} := (\pi_{\mathsf{nay}}, y, \pi_{\mathsf{PoSW}})$, $\chi = \mathsf{H}(\pi_{\mathsf{nay}}||\tau||\mu)$, output $\mathsf{PoSW}.\mathsf{Verify}^{\mathsf{H}}(\chi, t, \pi_{\mathsf{PoSW}}) \wedge \Pi_{\mathsf{nay}}.\mathsf{NVerify}(\mathsf{crs}_{\mathsf{nay}}, \tau, \pi_{\mathsf{nay}})$.

Theorem 1 *If Π_{nay} is a naysayer proof system w.r.t. Π, if PoSW is a proof of sequential work, if $\mathcal{Z}$ is the distribution of naysayable proofs generated with high min-entropy, and if $\mathsf{PoSW}.\mathsf{Verify}^{\mathsf{H}}$ is asymptotically faster than $\Pi.\mathsf{Verify}$, then the above construction is a VDN.*

Proof sketch 1 *We give a short proof sketch for all the required properties of the VDN.*

Completeness: *it follows by inspection.*

Soundness: *it follows from the soundness of Π_{nay}.*

$\mathcal{Z}$-Watermarked Sequentiality: *it follows from the fact that a $\pi_{\mathsf{nay}}||\tau||\mu$ with high min-entropy gives a uniformly distributed χ when hashed down with a random oracle, and from the soundness of the PoSW. Intuitively, all the polynomially many queries of $\mathcal{A}_0$ do not give it any insight on χ, as well as the polynomially many queries done by $\mathcal{A}_1$ to $\mathsf{VDNaysay}^{\mathsf{H}}(\mathsf{crs}_{\mathsf{VDN}}, \mathsf{crs}, t, \cdot, \cdot)$ over $(\tau', \mu') \neq (\tau, \mu^*)$. Therefore, one can simply reduce watermarked sequentiality to the soundness of PoSW.*

Efficiency: *the running time of $\mathsf{VDNVerify}^{\mathsf{H}}$ is the sum of the running times of the verification algorithms of Π_{nay} and PoSW, which are both asymptotically faster than $\Pi.\mathsf{Verify}$, the same holds for $\mathsf{VDNVerify}^{\mathsf{H}}$.*

4 Acknowledgments

We thank Hamza Abusalah for insightful discussions and the anonymous reviewers of FC 2025 for their feedback. This work has been supported by the grants PID2022-142290OB-I00 (ESPADA project co-funded by FEDER, EU), JDC2023-050791-I (co-funded by the ESF+), and TED2021-132464B-I00 (PRODIGY project, co-funded by the European Union NextGenerationEU/PRTR), all funded by MCIN/AEI/10.13039/501100011033. It has also received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (PICOCRYPT project, Grant No. 101001283). Ivan Visconti is member of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica and his contribution on this work was in part supported under the National Recovery and Resilience Plan (NRRP), Mission 4, Component 2, Investment 1.1, Call for tender No. 104 published on 2.2.2022 by the Italian Ministry of University and Research (MUR), funded by the European Union -NextGenerationEU - Project Title "PARTHENON" - CUP D53D23008610006 - Grant Assignment Decree No. 959.

References

[1] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder - scalable, robust anonymous committed broadcast. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1233–1252. ACM Press, November 2020.

[2] Hamza Abusalah and Valerio Cini. An incremental PoSW for general weight distributions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 282–311. Springer, Cham, April 2023.

[3] Hamza Abusalah, Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. Reversible proofs of sequential work. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 277–291. Springer, Cham, May 2019.

[4] Miguel Ambrona, Marc Beunardeau, and Raphaël R. Toledo. Timed commitments revisited. Cryptology ePrint Archive, Report 2023/977, 2023.

[5] Arbitrum. Docs. https://docs.arbitrum.io.

[6] Gennaro Avitabile, Luisa Sinischalchi, and Ivan Visconti. Short paper: Rewardable naysayer proofs. In Christina Garman and Pedro Moreno-Sanchez, editors, *FC 2025, April 2025, Revised Selected Papers*, volume to appear of *Lecture Notes in Computer Science*.

[7] Carsten Baum, James Hsin-yu Chiang, Bernardo David, Tore Kasper Frederiksen, and Lorenzo Gentile. Sok: Mitigation of front-running in decentralized finance. In Shin'ichiro Matsuo, Lewis Gudgeon, Ariah Klages-Mundt, Daniel Perez Hernandez, Sam Werner, Thomas Haines, Aleksander Essex, Andrea Bracciali, and Massimiliano Sala, editors, *FC 2022 International Workshops*, volume 13412 of *LNCS*, pages 250–271. Springer, 2022.

[8] Joseph Bebel and Dev Ojha. Ferveo: Threshold decryption for mempool privacy in BFT networks. Cryptology ePrint Archive, Report 2022/898, 2022.

[9] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Cham, August 2018.

[10] Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 236–254. Springer, Berlin, Heidelberg, August 2000.

[11] Jeffrey Burdges and Luca De Feo. Delay encryption. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 302–326. Springer, Cham, October 2021.

[12] Ran Canetti and Shafi Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 90–106. Springer, Berlin, Heidelberg, May 1999.

[13] Eric Chen and Albert Chon. Injective protocol: A collision resistant decentralized exchange protocol, 2018.

[14] Kevin Choi, Aathira Manoj, and Joseph Bonneau. SoK: Distributed randomness beacons. In *2023 IEEE Symposium on Security and Privacy*, pages 75–92. IEEE Computer Society Press, May 2023.

[15] Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.

[16] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 451–467. Springer, Cham, April / May 2018.

[17] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy*, pages 910–927. IEEE Computer Society Press, May 2020.

[18] Nico Döttling, Russell W. F. Lai, and Giulio Malavolta. Incremental proofs of sequential work. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 292–323. Springer, Cham, May 2019.

[19] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: Front-running attacks on blockchain. In Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala, editors, *FC 2019*, pages 170–189, Cham, 2020. Springer International Publishing.

[20] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482. Springer, Berlin, Heidelberg, August 2010.

[21] Charlotte Hoffmann and Krzysztof Pietrzak. Watermarkable and zero-knowledge verifiable delay functions from any proof of exponentiation. Cryptology ePrint Archive, Paper 2024/481, 2024.

[22] Linea. Documentation. https://docs.linea.build/.

[23] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 887–903, New York, NY, USA, 2019. Association for Computing Machinery.

[24] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In Robert D. Kleinberg, editor, *ITCS 2013*, pages 373–388. ACM, January 2013.

[25] Optimism. Docs. https://docs.optimism.io/.

[26] Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, January 2019.

[27] Emrah Sariboz, Gaurav Panwar, Roopa Vishwanathan, and Satyajayant Misra. FIRST: frontrunning resilient smart contracts. *CoRR*, abs/2204.00955, 2022.

[28] István András Seres, Noemi Glaeser, and Joseph Bonneau. Short paper: Naysayer proofs. In Jeremy Clark and Elaine Shi, editors, *FC 2024, March 2024, Revised Selected Papers, Part II*, volume 14745 of *Lecture Notes in Computer Science*, pages 22–32. Springer, 2024.

[29] StarkEx. Documentation. https://docs.starkware.co/starkex/overview.html.

[30] Justin Thaler. *Proofs, Arguments, and Zero-Knowledge*. 2023.

[31] Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Cham, May 2019.